



Gerald V. Post

Database Management Systems

**SQL Server 2005/2008
Visual Studio 2005/2008**

**Designing & Building Business Applications
Fourth Edition**

Database Management Systems

Designing and Building
Business Applications
With

SQL Server 2005/2008
Visual Studio 2005/2008

Version 4.0.3

Gerald V. Post

University of the Pacific

Database Management Systems
Designing and Building Business Applications
With SQL Server 2005/2008 and Visual Studio 2005/2008

Copyright © 2007,2008 by Gerald V. Post

All rights reserved. No part of this publication may be reproduced or distributed in any form or stored in any database or retrieval system without the prior written consent of Gerald V. Post.

Students:

Your honesty is critical to your reputation. No company wants to hire a thief—particularly for jobs as critical as application development and database administration. If someone is willing to steal something as inexpensive as an e-book, how can that person be trusted with billions of dollars in corporate accounts?

You are not allowed to “share” this book in any form with anyone else. You cannot give or sell any information from this publication in any form to anyone else.

To purchase this book or other books: <http://JerryPost.com/books>

If you want a detailed step-by-step discussion of how to create applications with the 2005 versions, you can buy a print copy of the SQL Server book from Prentice Hall: Introduction to SQL Server 2005/ Perry and Post

Brief Contents

1 Introduction

Part One: Systems Design

2 Database Design

3 Data Normalization

PART TWO: QUERIES

4 Data Queries

5 Advanced Queries and Subqueries

PART THREE: APPLICATIONS

6 Forms and Reports

7 Database Integrity and Transactions

8 Applications

9 Data Warehouses and Data Mining

PART FOUR: DATABASE ADMINISTRATION

10 Database Administration

11 Distributed Databases

12 Physical Data Storage

Introduction, 1

- Case: All Powder Board and Ski Shop, 2
 - Inventory*, 2
 - Bindings and Boots*, 3
 - Sales*, 4
 - Rentals*, 5
- Lab Exercise, 6
 - Project Outline*, 6
 - Project Plan*, 7
 - Feasibility*, 8
 - The Database Management System*, 9
- Exercises, 15
- Final Project, 16

Database Design, 17

- Database Design, 18
- SQL Server Data Types, 18
- Case: All Powder Board and Ski Shop, 20
 - Business Objects: First Guess*, 20
 - Relationships*, 21
- Lab Exercise, 21
 - Database Design System*, 21
 - All Powder Design*, 23
- Exercises, 30
- Final Project, 32

Data Normalization, 33

- Database Design, 34
- Generated Keys: Identities, 34
- Case: All Powder Board and Ski Shop, 35
- Lab Exercise, 36
 - All Powder Board and Ski Database Creation*, 36
 - Relationships*, 41
- Exercises, 46
- Final Project, 47

Database Queries and SQL, 48

- Database Queries, 49
- Case: All Powder Board and Ski Shop, 49
- Lab Exercise, 50
 - All Powder Board and Ski Data*, 50
 - Computations and Subtotals*, 60
- Exercises, 65
- Final Project, 66

Advanced Queries, 67

- Advanced Database Queries, 68
- Case: All Powder Board and Ski Shop, 69
- Lab Exercise, 69
 - All Powder Board and Ski Data*, 69
 - SQL Data Definition and Data Manipulation*, 78
- Exercises, 84
- Final Project, 86

Forms and Reports, 87

- Forms and Reports, 88
- Case: All Powder Board and Ski Shop, 89
- Lab Exercise, 90
 - All Powder Board and Ski Shop Forms*, 90
 - All Powder Basic Reports*, 109
- Exercises, 116
- Final Project, 117

Database Integrity and Transactions, 118

- Program Code in SQL Server, 119
- Case: All Powder Board and Ski Shop, 120
- Lab Exercise, 121
 - All Powder Board and Ski Data*, 121
 - Database Cursors, Keys, and Locks*, 136
- Exercises, 146
- Final Project, 148

Applications, 149

- Applications, 150
- Case: All Powder Board and Ski Shop, 150
- Lab Exercise, 151
 - All Powder Board and Ski Shop Application*, 151
- Exercises, 164
- Final Project, 165

Data Warehouses and Data Mining, 166

- Data Warehouse, 167
- Case: All Powder Board and Ski Shop, 168
- Lab Exercise, 168
 - All Powder Board and Ski Shop*, 168
 - Introductory Data Analysis*, 184
- Exercises, 196
- Final Project, 198

Database Administration, 199

Database Administration Tasks, 200

Case: All Powder Board and Ski Shop, 202

Lab Exercise, 202

All Powder Board and Ski Shop, 202

Security and Privacy, 210

Exercises, 218

Final Project, 220

Distributed Databases, 221

Location, Location, Location, 222

Case: All Powder Board and Ski Shop, 223

Lab Exercise, 223

All Powder Board and Ski Shop, 223

The Internet, 231

Exercises, 239

Final Project, 241

Physical Database Design, 242

Storing Data, 243

Lab Exercise, 243

All Powder Board and Ski Shop, 243

Data Clusters, 246

Exercises, 247

Final Project, 247

Introduction

Chapter Outline

Case: All Powder Board and Ski Shop, 2

Inventory, 2

Bindings and Boots, 3

Sales, 4

Rentals, 5

Lab Exercise, 6

Project Outline, 6

Project Plan, 7

Feasibility, 8

The Database Management System, 9

Exercises, 15

Final Project, 16

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 have 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, sidecut 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										
		<table border="1"> <thead> <tr> <th>Size</th> <th>QOH</th> </tr> </thead> <tbody> <tr> <td>34</td> <td></td> </tr> <tr> <td>35</td> <td></td> </tr> <tr> <td>36</td> <td></td> </tr> <tr> <td>...</td> <td></td> </tr> </tbody> </table>		Size	QOH	34		35		36		...				
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 customers who could use those special sizes. The catch is that the owner is concerned

Figure 1.3

Sales						
Customer					Sale Date	
First Name	Last Name				Salesperson	
Phone	E-Mail				Department	
Address			Shipping Address			
City, State ZIP			City, State ZIP			
Male/Female		Ski/Board		Skill Level		
Age/Date of Birth		Style				
Item	Description	New/Used	Size	Quantity	Price	Subtotal
Item Total						
Tax						
Total Due			Method of Payment			

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 to mark each row separately, the store managers have determined that about 20

Figure 1.4

Rentals						
Customer First Name		Last Name		Rental Date		
Phone		E-Mail		Expected Return		
Address City, State ZIP			Shipping Address City, State ZIP			
Male/Female		Ski/Board		Skill Level		
Age/Date of Birth		Style				
Item	Description	Size	Fee	Return Date	Condition	Charges
Item Total						
Tax						
Total Due		Added Charges				
Method of Payment		Signature				

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 exercise. You should begin by reviewing the description of the company. You should also use the Internet to check out some

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:

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

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 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.

Action

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

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

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.



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.

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			

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 SQL Server, 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 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

SQL Server is one of the easiest large-scale DBMSs to install. It can be installed on a server, and accessed via client software. Note that if you want to work on the analysis exercises in Chapter 8, you will have to install the Analysis Services. This tool is included on the same installation disk, but you have to go back and specifically select it after you have installed the main database.

SQL Server is small enough to install on individual computers and run as a standalone development database.

Several versions of the DBMS exist, but SQL Server 2000 is probably the most common at the moment. Microsoft is working on a newer version with substantial changes, with the codename Yukon. You can probably obtain beta copies of this software now, but it is too early to cover in this book.

As shown in Figure 1.8, Microsoft provides a considerable amount of online information for SQL Server. The most cost-effective way to obtain the software is to get it through the MSDN Academic Alliance program. For a small annual

Action

Start SQL Server Management Studio.
Open the SQL Server Group and log in.
Expand the database assigned to you, or create it if necessary.
Right click on Tables, select New Table.
Enter column names and data types.
Click the CustomerID row.
Set the Identity value to Yes
Click the key icon in the main toolbar.
Close the form and name the table Customer.

Figure 1.8

<http://msdn.microsoft.com>
Microsoft developer's network
<http://www.microsoft.com/sql>
SQL Server home site
<http://www.msdnaa.net>
Academic Alliance educational pricing

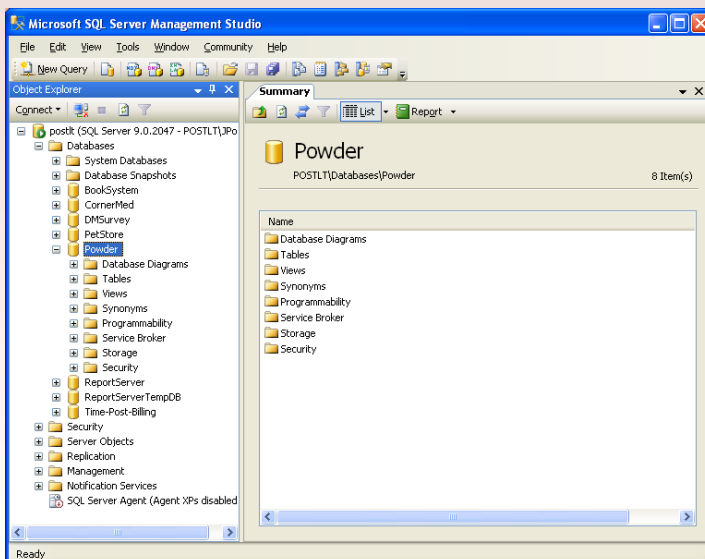
fee, a university can obtain licensed copies for the computer labs and individual students. As an individual developer, you can join MSDN for an annual fee and obtain a full developer's copy of the software. Commercial use licenses are considerably more expensive. The MSDN Web site also enables you to register free of charge for access to information and help files.

SQL Server installs fairly easily. By default, it uses standard Internet protocols to connect to other machines. In most cases, you can accept the default installation values. It is also easy to install on your own computer, so you can run the entire database and examples from one machine. If you install the main database on a shared server, you will have to install the client pieces on the individual machines. You will also have to configure security controls on the database to give desired people access to the database. Security issues are covered in more detail in Chapter 10.

As shown in the SQL Server Management Studio in Figure 1.9, you can create multiple databases on one copy of SQL Server. Each database can have separate security conditions. For the labs in this workbook, you should create the Powder database. Right-click the Database icon and select the option to create a New Database, and fill in the name. You might also want to create a new user (powder) and give it a SQL Server authentication password. Make this user the database owner of the new Powder database. Some situations may arise later where you need this connection.

To get a quick perspective of the various components of the DBMS, you need to build a simple database. SQL Server has several graphical tools to help you manage the database. You can also use SQL commands, but those are covered in a later chapter. Most administrators perform all tasks by writing SQL statements. However, for one or two simple tables, it is faster to use the graphical tools. Start the SQL Server Management Studio. Expand the databases section and expand the main database you have been assigned to use (Powder). As you select the various icons (tables, indexes, views, and so on), you will see that your schema contains several system tables. Do not alter or delete any of these tables.

Figure 1.9



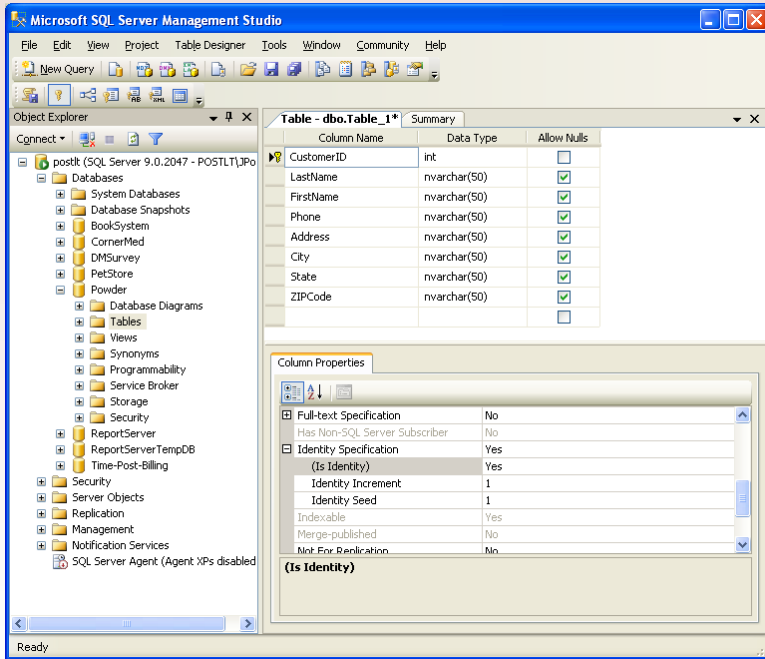
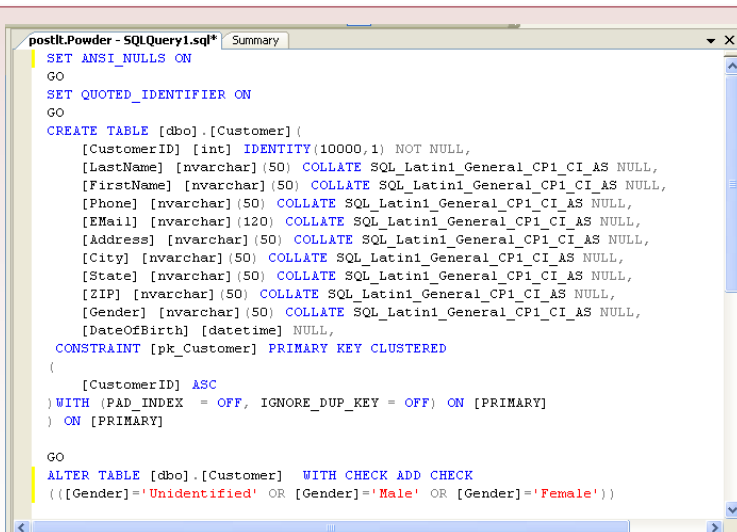


Figure 1.10

Relational databases consist of a collection of tables, so the first step is to create a new table. Figure 1.10 shows the definition of a customer table. As explained in Chapter 2, each table must have a primary key. In this case, select the CustomerID row by clicking the gray box on the left side. Make sure the Identity value in the lower box is set to Yes. This option will cause values for the CustomerID to be generated automatically, which ensures that each value is unique. Next, click

Figure 1.11



the key icon in the main toolbar for the form. This option places the key icon next to the CustomerID row which indicates that the CustomerID is the primary key for this table.

As shown in Figure 1.11, SQL Server can generate the underlying SQL command used to create the table. The command is slightly more complex than necessary because it specifies the character set for each column. You can generate the SQL script by clicking the Generate Change Script on the main toolbar. If you have already saved the table, you can right-click the table in the Tables list and choose the Script Table As/Create To option to generate the script. If you memorize the syntax of this statement, you could have skipped the design screen and typed the statement directly. More importantly, you can copy this statement and put it into a text file that you can execute later if you ever need to re-create this table.

The next step is to open the table and enter some data for fake customers. Right-click on the table name and select the option to Open Table. Again, you could use SQL to insert rows; however, when you first work with SQL Server, it is easiest to enter the data directly using the table editor. Figure 1.12 shows some sample data you can use, or make up three rows of data on your own. Click the Apply button to save the data.

SQL Server provides several tools in the table window to examine the data. You can sort by columns or even filter the rows to just see customers that meet some criteria.

In practice, you will rarely enter data directly into tables. Instead, you will build forms that users can run to enter and edit data. SQL Server does not provide tools to build forms and reports. Instead, you will need a tool that is designed to create forms that attach to a database. Several choices exist. This workbook will use Microsoft Visual Studio .NET with Windows forms.

Visual Studio .NET has a Wizard to help build forms for database applications. However, it requires several steps to get started. First, make sure you have Visual Studio installed on the client/development computer. Then, create a new project based on Windows and choose a programming language of either C# or Visual Basic. The examples in this book will use Visual Basic. The C# code is similar, but the syntax is different.

When you start a new project, Visual Studio automatically adds a Form1. You should change the Text property to Customer, and you might want to rename the entire form to Customer.vb. You need to add data items onto the form that will connect to the Customer table. To do that, you need to create a Data Source that tells Visual Studio exactly which tables and columns you want to use. Along the way,

Action

- Right click the table name.
- Select Open Table.
- Enter sample data.
- Close the table.

Figure 1.12

Table - dbo.Customer		Summary							
CustomerID	LastName	FirstName	Phone	EMail	Address	City	State	ZIP	
0	Walk-in	NULL	NULL	NULL	NULL	NULL	NULL	NULL	
1	Jones	Jack	111-222-3333	JonesJ202@msn.com	123 Main	Sacramento	CA	95838	
2	Sanchez	Paul	111-444-9999	SanchezP844@msn.com	777 Oak	Sacramento	CA	95838	
3	Garner	Chad	213-080-4599	GarnerC73@msn.com	555 Trident Place	Chicago	IL	60601	

you will create a connection string that tells Visual Studio how to connect to your database server. Choose Data/Add New Data Source from the main menu (or open the Data Sources window and click the link found there). Click the button to add a New Connection. If necessary, change the database type to Microsoft SQL Server. On the standard connection form, enter the name of the database server (which might be your local workstation). For teaching environments, you generally want to choose SQL Server authentication, then enter your username and password for the database. Check the box to save your password. Pick the database that holds the Customer table and click the button

to test the connection. Close the connection form and choose the wizard option to include sensitive data (username and password) in the connection string. On the next page, expand the table list and place a checkmark next to the Customer table to select the entire table.

When you are returned to Visual Studio, open the Data Sources window by clicking the Tab or choosing it in the Data menu. You might have to click the Refresh button to display your newly-created Data Source. For the Customer form, you want to see and edit data for one customer at a time. Expand new Data Source

Action

Start the Visual Studio and select File/ New Project to choose a VB Windows Application.

Choose Data/Add New Data Source.

Add a New Connection.

Choose Microsoft SQL Server.

Enter the server name, and use your SQL Server Authentication.

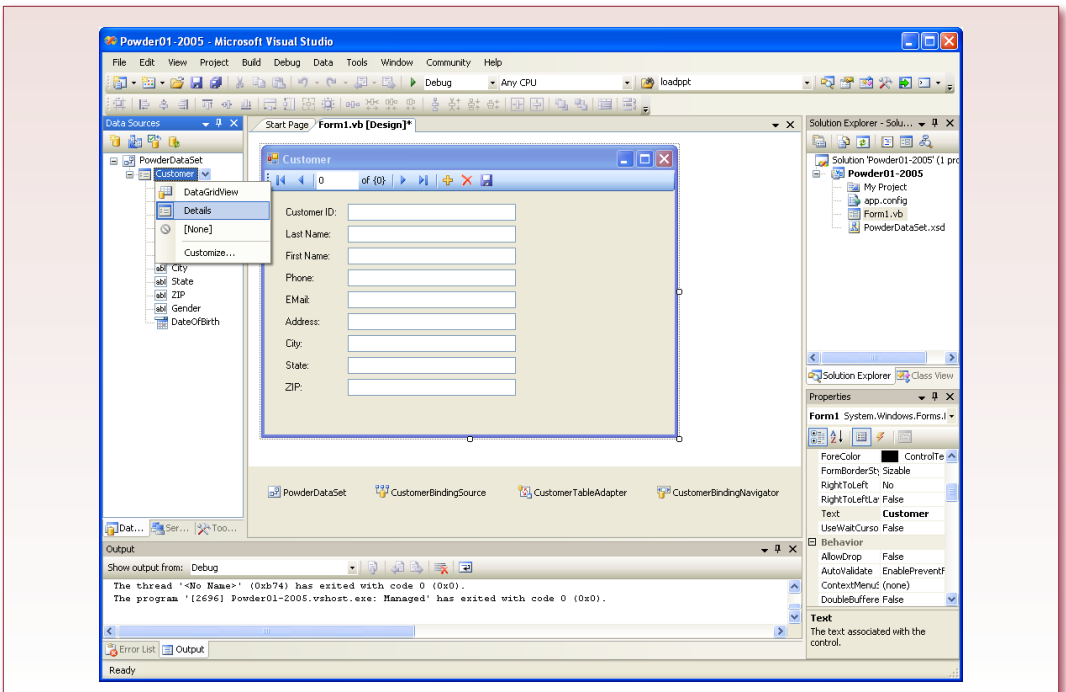
Check the box to save your password.

Pick your database and test the connection.

Choose the option to include sensitive data.

Expand the Table list and select the Customer table with a check mark.

Figure 1.13



and select the Customer table. Click the drop-down-list arrow and choose the Details layout. The default Grid layout shows multiple rows of customers at one time, but there are too many columns to use the Grid for this table. Finally, drag the Customer table onto the main form, and Visual Studio will automatically create the labels and text boxes for all of the columns. Figure

1.13 shows that the system also places a navigation bar at the top of the form. You will use this form to scroll through the list of customers and to insert, delete rows, and save any changes.

The Wizard automatically connects the form data to the database, using your Data Source and the connection data you specified. Click the Start Debugging button to run the form. Figure 1.14 shows the sample form. You can use the navigation buttons to scroll through the list of customers. Add a new row or change data if you want to see how the form works. You will have to click the Save button to write any changes to the database. If you close the form without saving changes, they will be discarded.

Action

View the Data Sources window and click the Refresh button.

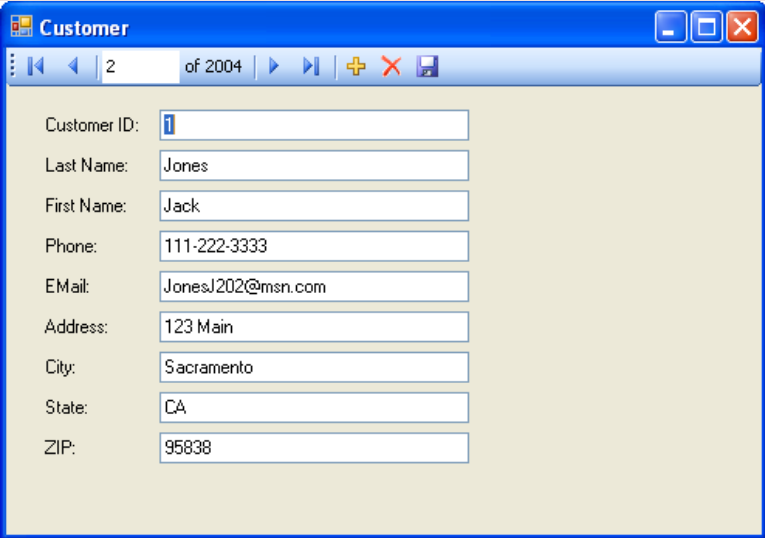
Select the Customer table and click the drop-down arrow.

Change the layout to Details.

Drag the Customer table onto the form.

Click the Debug button to run the form.

Figure 1.14



The screenshot shows a Windows application window titled "Customer". At the top, there is a navigation bar with buttons for back, forward, and other actions, along with a status bar indicating "2 of 2004". The main area contains a form with the following fields:

Customer ID:	<input type="text" value="1"/>
Last Name:	<input type="text" value="Jones"/>
First Name:	<input type="text" value="Jack"/>
Phone:	<input type="text" value="111-222-3333"/>
EMail:	<input type="text" value="JonesJ202@msn.com"/>
Address:	<input type="text" value="123 Main"/>
City:	<input type="text" value="Sacramento"/>
State:	<input type="text" value="CA"/>
ZIP:	<input type="text" value="95838"/>

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 current-

ly 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.

Database Design

Chapter Outline

Database Design, 18

SQL Server Data Types, 18

Case: All Powder Board and Ski Shop, 20

Business Objects: First Guess, 20

Relationships, 21

Lab Exercise, 21

Database Design System, 21

All Powder Design, 23

Exercises, 30

Final Project, 32

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, using pencil and paper will be relatively easy. However, when you are learning, using pencil and paper is 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 will not work with SQL Server because SQL Server limits the changes you can make to tables—particularly after relationships have been built and data has been added.

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. SQL Server does have a designer to build entity-relationship diagrams. This system is useful because you can generate the diagrams from the list of tables. But it has limited advice and design checking facilities.

There is a better tool to learn database design. 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 available when you return to a lab or 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.

SQL Server 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 identify specify constraints (such as 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. But, in the future, 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 numeric data type is variable length in SQL Server, because you can specify the number of significant digits. A full 38 digits requires 17 bytes of storage.

Figure 2.1 shows the main data types available in SQL Server 2000. The types you will use most often are nvarchar, datetime, int, float, and money. When you need to store date or time values, be sure to use the datetime type. It supports date arithmetic so users can subtract two dates to obtain the number of days between them. The image type can hold pictures or even spreadsheets or documents. You can generally use the ANSI SQL keywords as well. Sometimes it is easier to use those instead of the SQL Server types, but ultimately SQL Server converts them into native types. For instance, SQL defines the integer data type, which SQL Server converts to int.

The issue of precision and scale is sometimes confusing in fixed decimal and currency data types. Precision represents the total number of significant digits supported in the value—regardless of any decimal points or size of the number. For example, a number with a precision of 5 digits would include 12345 as well as 12.345. If the scale is specified, it indicates a fixed number of decimal points and controls round off to that value. It is particularly useful for handling currency values.

The other confusing issue in modern databases is the use of Unicode or “national” character sets. The older varchar data type assigns one character to one byte and can only handle ASCII codes or essentially English-language characters. If your database needs to store text in additional languages, it will have to use Unicode character sets that typically assign two bytes to any character or ideogram. In this case, use the nvarchar data type, but note that it cuts the maximum length of text in half. varchar can handle strings up to 8,000 bytes. Nvarchar can also handle 8,000 bytes, but that is only 4,000 Unicode characters.

Figure 2.1

	Name	Data	Bytes
Text (Characters)			
Fixed	char or nchar	8000 bytes	Fixed
Variable	varchar	8000 bytes	Variable
National/Unicode	nvarchar	4000 characters	Variable
Memo	text or ntext	2 gigabytes	Variable
Numeric			
Byte (8 bits)	tinyint	0 to 255	1
Integer (16 bits)	smallint	-2 ¹⁵ to 2 ¹⁵ - 1	2
Long (32 bits)	int	-2 ³¹ to 2 ³¹ - 1	4
(64 bits)	bigint	-2 ⁶³ to 2 ⁶³ - 1	8
Fixed precision	decimal(p,s)	p: 1...38, s: 0...p	5-17
Float	real	7 digits	4
Double	float	15 digits	4, 8
Currency	money	38 digits, 4 decimal	8
Yes/No	bit	0, 1	variable
Date/Time	datetime, smalldatetime	1/1/-4712 to 12/31/9999 (sec.)	7/11/13
Interval			
Image	image	2 gigabytes, 4 gigabytes	Variable
Generated Key	identity	Long (+/- 2 billion)	4

Case: All Powder Board and Ski Shop

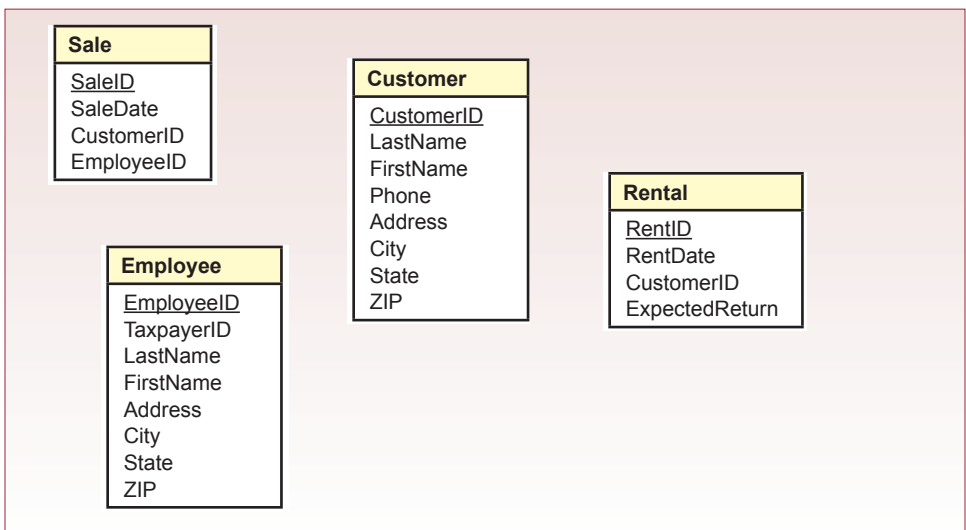
With any database project, the first step is to understand the various elements of the organization and the components that will become part of the database application. This knowledge is critical, because the database design must reflect the business rules. In real life, you can ask workers about the 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 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 SQL Server, you have to assign this column an int data type, and choose the option to specify it as an identity column. In terms of the design, it is useful to indicate that this key is internally generated by the DBMS, so the database design system refers to it as an AutoNumber data type. 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

Figure 2.2



and think about it. A few other classes should be relatively obvious for this case. In particular, several support tables are used to provide look up 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.

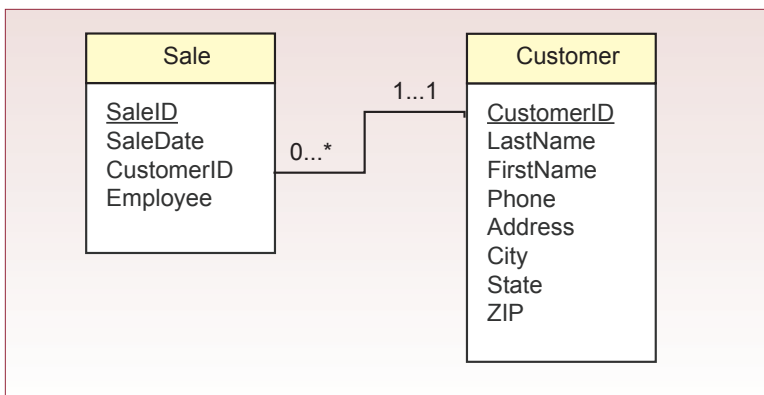
Relationships are displayed on the diagram by drawing connecting lines between the two tables involved. The business rules are shown as annotations at the end of each connection. Each side of the connection displays minimum and maximum values. Figure 2.3 shows the association between the Sale and Customer 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 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

Figure 2.3



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

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 instructor 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.

Action

Browser: <http://JerryPost.com/dbdesign>

New student who has two key numbers.

Figure 2.4

Enter the key numbers you received

Create a username and password

Enter your correct name, e-mail address and StudentID

Microsoft Internet Explorer
Address: <http://time-post.com/dbdesign/NewStudentKeys.asp>

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 SH5 - ENZE - HXC
Key2 N9G - M8OJ - GLL

Username studenty
Password ●●●●
E-mail Jerry@JerryPost.com
First Name Sample
Last Name Student
Student ID 123456

Submit

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.

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 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.

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.

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 right-clicking the table heading, typing “Sale” as the new name, and pressing the Enter key.

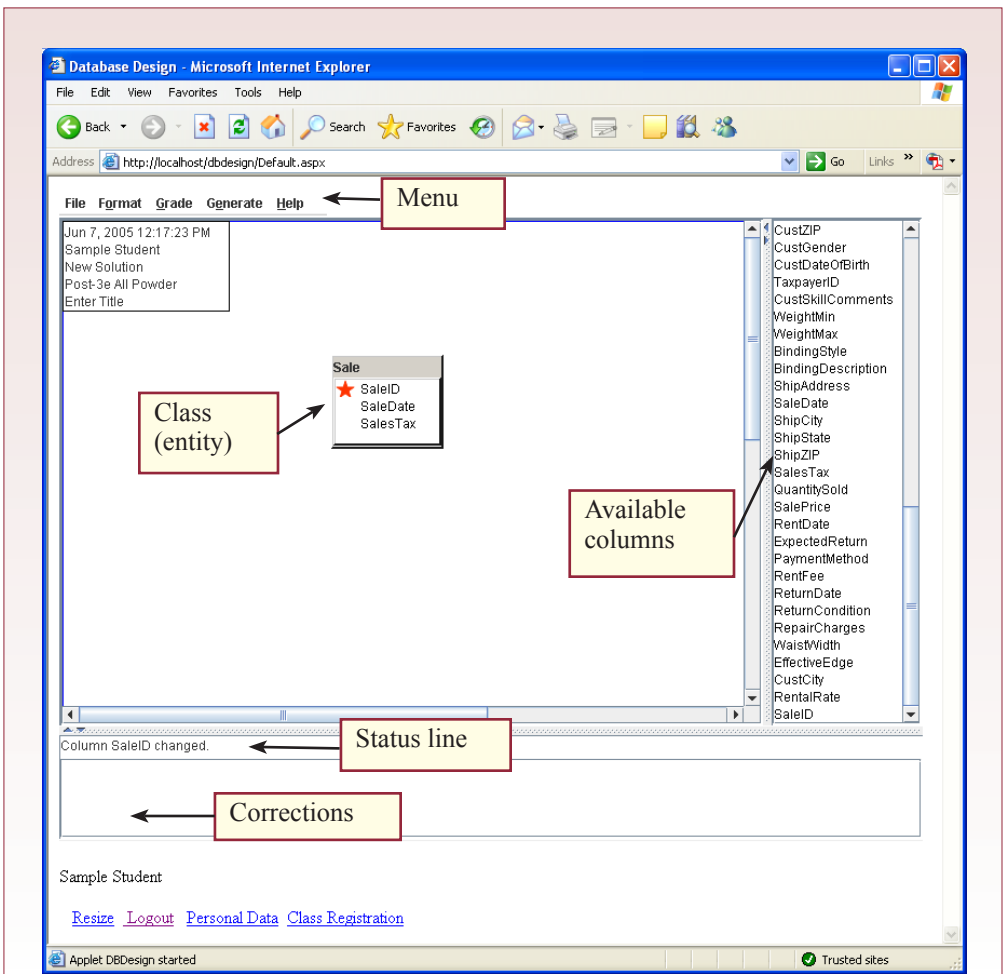
Figure 2.5

The screenshot shows a web browser window titled "Student Register for Class - Microsoft Internet Explorer". The address bar shows the URL "http://jerrypost.com/dbdesign/StudentRegisterClass.aspx". The main content area is titled "Register for Class" and contains the following text: "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." Below this text are four dropdown menus: "Country/State" (set to "USA All States"), "School" (set to "University of the Pacific"), "Class" (set to "Open Database Management Fall 2005 Open - 1"), and "Admit Code" (empty). At the bottom of the form are three buttons: "Register", "Close", and "Cancel". Three callout boxes with arrows point to the form elements: the first points to the "Country/State" dropdown, the second points to the "School" dropdown, and the third points to the "Admit Code" input field.

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 right-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. 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. Right-click on the column name within the table and the current data

Figure 2.6



type is displayed at the bottom of the pop-up menu. Move the cursor to that item and a complete list of data type choices pops up. Choose the desired data type by highlighting it and clicking the left button. Be sure to save your work every few minutes in case you lose the Internet connection or the server times out.



Activity: Create Relationships

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 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.

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.

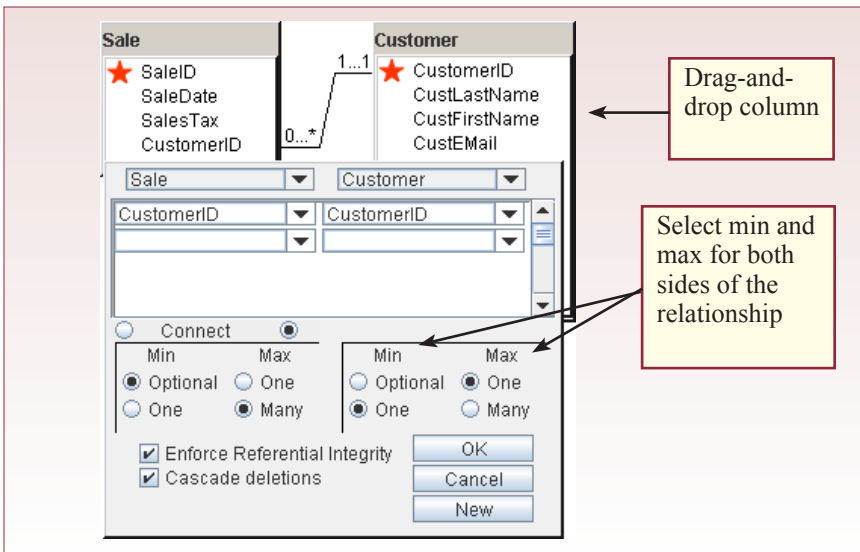


Figure 2.7

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

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.

Action
 Choose Grade/Grade and Mark.
 Click messages in window.
 Fix errors by removing columns and adding new tables.

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 “Does SKU in table Sale really

depend on SaleID?" and double 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. 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 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

Figure 2.8

The screenshot shows the Database Design interface in Microsoft Internet Explorer. The main window displays a diagram with three tables: Inventory, Sale, and Customer. The Sale table has columns: SaleID (primary key), SaleDate, SalesTax, CustomerID, and SKU. The Inventory table has columns: SKU, Size, and QOH. The Customer table has columns: CustomerID (primary key), CustFirstName, CustLastName, CustPhone, CustEmail, CustCity, CustState, CustZIP, CustGender, and CustDateOfBirth. Relationships are shown: Inventory (1..1) to Sale (0..*), and Sale (1..1) to Customer (0..*). The SKU column in the Sale table is highlighted in red. A diagnostic message at the bottom states: "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?". Annotations with red boxes and arrows point to the SKU column, the diagnostic message, and the relationship between the Inventory and Sale tables.

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 table 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.

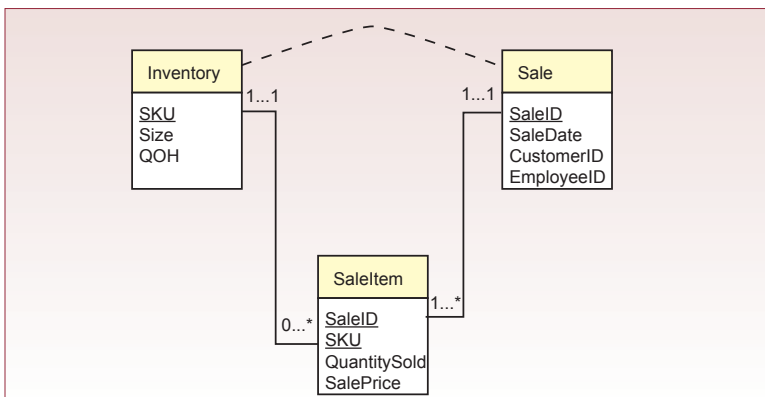
The new SaleItem table corresponds to the repeating lines of items that you would see listed on a paper sale form. Examining the two new relationships reveals how the table works. Reading from the Sale to the SaleItem table, each sale can contain from one to many items, and in reverse, each SaleItem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

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 (un-

Figure 2.9



filled 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.

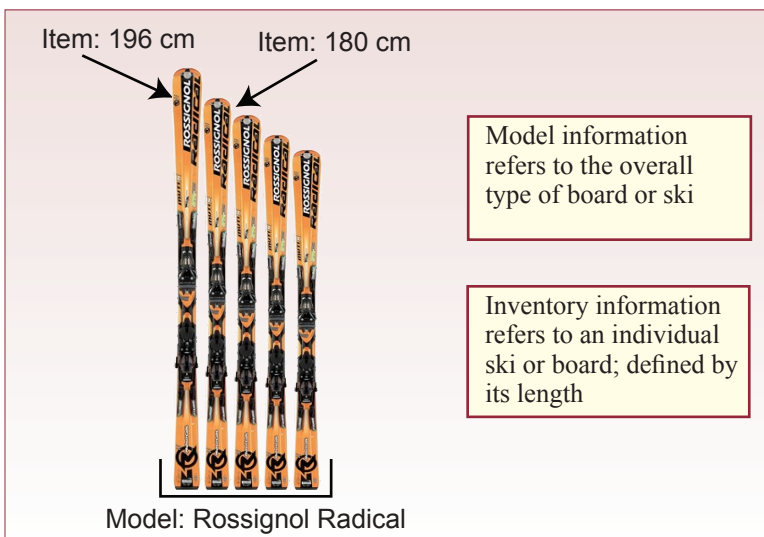
If you grade this version, you will see that the detail issues have been corrected. However, some design issues still exist in terms of handling inventory. The inventory for a ski shop is somewhat more complicated than for a typical retail store. In particular, snowboards and skis are sold in varying 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 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.

Action

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

Figure 2.10



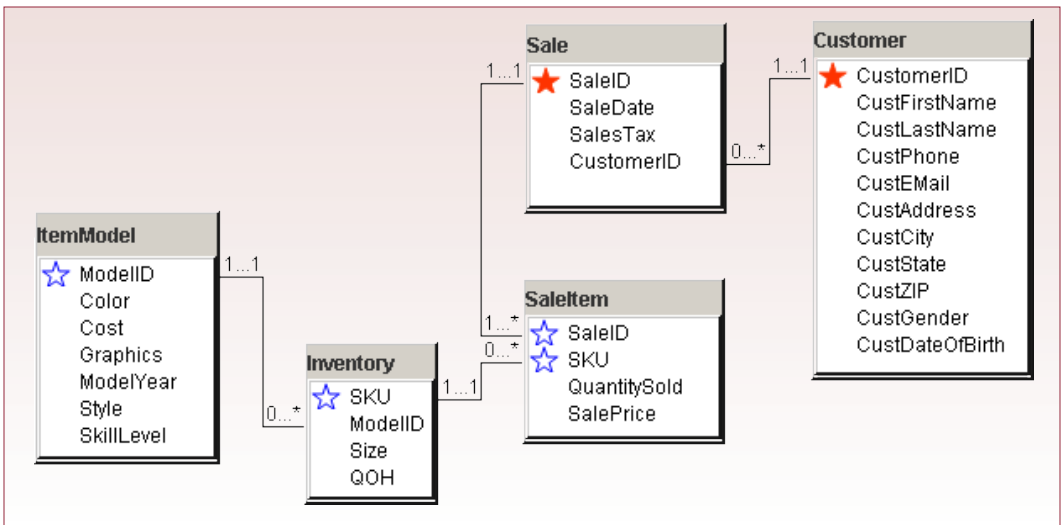


Figure 2.11

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.

Last Name, First Name Phone, Cell Phone Address City, State, ZIPCode	Year	Position/Title	Comment	
Event title Start Date End Date Charity Charity contact Phone Amount raised	Member Activities for Event			
	Date	Hours	Activity	Comment

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.

Data Normalization

Chapter Outline

Database Design, 34

Generated Keys: Identities, 34

Case: All Powder Board and Ski Shop, 35

Lab Exercise, 36

All Powder Board and Ski Database Creation, 36

Relationships, 41

Exercises, 46

Final Project, 47

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.

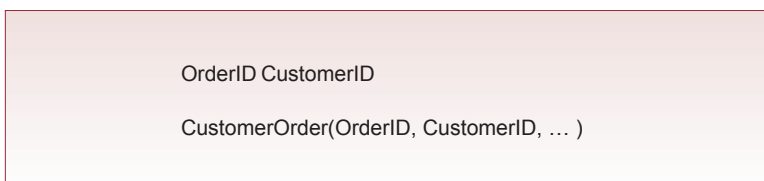
If you are uncertain about which columns should be keyed, write them down separately and evaluate the business rules between the two objects. Figure 3.1 shows a typical situation with orders and customers. First ask yourself: For a given order, can there ever be more than one customer? If the answer 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: Identities

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



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.

SQL Server has a sequence process to generate new key values. You assign an int type to the primary key in a table where you want the key value created. This data type does not actually create the number. To create numbers, you also declare the column to be an identity. The sequence generator is relatively flexible and you can specify a starting value and an increment. This system generates an identity value whenever a new row is inserted into the table. The process is relatively automatic, but it is slightly tricky to obtain the generated value if you need to use it in a second table. This step is covered in a later lab. However, sequences really should be set up when you define the table so that you remember to do it. One of the activities in this lab will show you how to set up an automatically generated key value; you can copy the process for your other projects.

For now, you must carefully identify the key columns that might need generated values. For instance, the CustomerID column in the Customer table, or the OrderID in the Order table might be assigned a generated value. But the CustomerID column in the Order table would never be a generated key. It would be given the same numeric data type, although the actual key generation can take place only in the original (Customer) table. Make sure you understand the difference. The CustomerID is the only column that is a primary key in the Customer table, and it is the source table for customers. Consequently, it is acceptable to generate key values for CustomerID in the Customer table. On the other hand, the CustomerID is a placeholder in the Order table—it represents the customer placing the order. The customer is not created in the Order table, so the CustomerID value cannot be generated in the Order table. The CustomerID must already exist in the Customer table before it can be assigned to a row in the Order 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 3.2, 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 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 assumptions about the underlying business rules, you can create slightly different tables than those recommended by the design system.



Activity: Create Tables

Once you have determined the overall database design, you have to select between the two methods for creating tables in SQL Server. The enterprise manager console contains a visual editor that makes it easy to enter column

Action

If necessary, create a new database.

Create Customer table with enterprise manager.

Enter column names.

Select data types.

Assign the primary key.

Exit and save the table.

Figure 3.2

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.

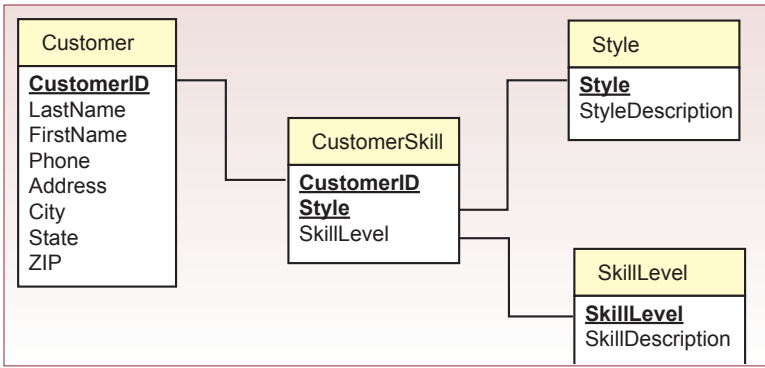
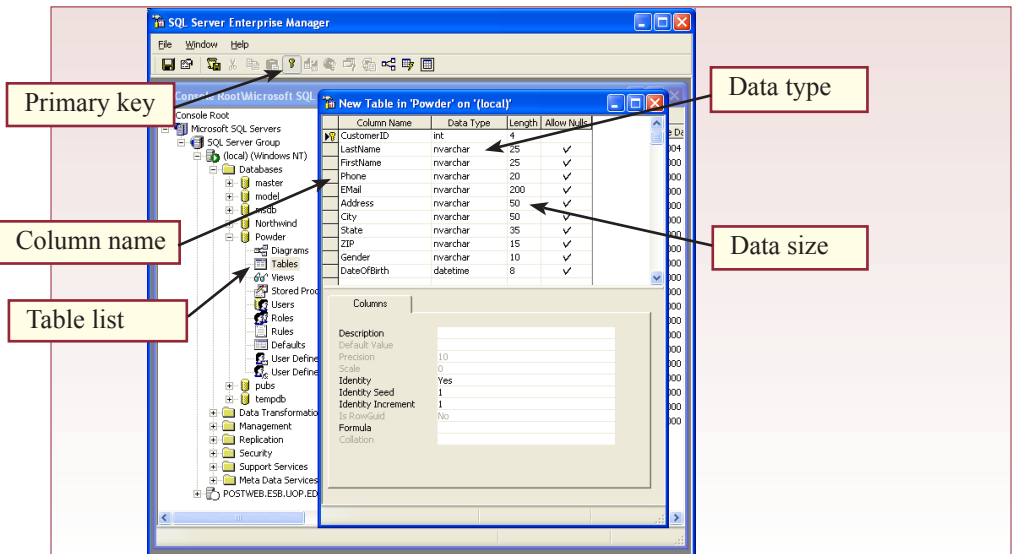


Figure 3.3

names and select the data types. However, ultimately, the editor converts your choices into an SQL statement that actually creates the table. For some things, the designer is easy to use, but ultimately, you will want to learn the SQL syntax. For now, begin with the designer. Later, you can generate the SQL script for the table. This way, if you ever need to re-create the tables, you simply have to execute the text file as a set of SQL statements.

Figure 3.4 shows the basic elements of the table design screen. Note that if you still have the Customer table created in Chapter 1, you can either edit that table or simply drop it and start over. As you enter the column (field) names, you select the data type from a drop-down list. For text data you should generally use nvarchar instead of the older varchar. You never know when someone will want to store names or other data using a different language alphabet. Some data types have size limits. For example, you should specify the maximum number of characters expected in a text column. SQL Server will efficiently store the data even if it takes less than the specified number of characters, but it will not allow anyone to enter a value with more than the number entered. SQL Server will allow up to 4,000 characters for the nvarchar data type, but try to be somewhat conservative

Figure 3.4



because those values might be used to set default format widths. For most columns, you want to make sure that the Allow Nulls column remains checked.

Primary keys are straightforward in SQL Server. Simply select the fields that will be keyed and then click the key icon in the toolbar. When you need multiple columns as part of the key, you can select the multiple items by dragging the mouse over all of them before releasing the button, or by holding down the Ctrl or Shift key while clicking each column. The Customer table has only one key column (CustomerID).

In SQL Server, generated keys are specified with the Identity keyword. In the Customer table, you want the database to generate a new key value each time a customer row is added. Click the CustomerID field and examine the properties. When you set the Identity property to Yes, the DBMS will automatically generate new values. You can also specify a starting seed and an increment.

Although they are not needed in the Customer table, be careful when selecting among the many numeric data types. Remember that integers do not have fractional values. In the All Power case, 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.

When you close the table remember to give it an appropriate name (Customer).



Activity: Create Constraints and Default Values

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 3.5 shows the statements for setting a condition to ensure that cost values are always positive. Pay particular attention to the commas—there are no commas within a column definition, only at the end of each column. The condition must be entered in parentheses and must represent a valid SQL WHERE statement. Almost any SQL condition can be used and they will be explained in detail in Chapter 4.

As a more complex example, you might want to return to the Customer table and add a constraint that limits the values that can be entered for Gender. The easiest approach would be to drop the existing table (DROP TABLE Customer;) and create it again; adding a new CHECK constraint. However, you could also use the ALTER TABLE command. In either case, you want people to enter data from a fixed list of items (female, male, and unidentified). You could probably get by without the “unidentified” option by using null values for that purpose, but it is a little easier for users if you specify it as a possibility. The condition that enforces this constraint is UPPER(Gender) IN ('FEMALE', 'MALE', 'UNIDENTIFIED'). The UPPER function converts whatever text is entered into all uppercase

Action
 Right-click Gender and select Check constraint.
 Select the Check tab.
 Enter check condition:
 Upper(Gender) IN ('FEMALE', 'MALE', 'UNIDENTIFIED').
 Enter unique name: CK_Customer_Gender.
 Close the form and save the changes.
 Test the constraint with sample data.

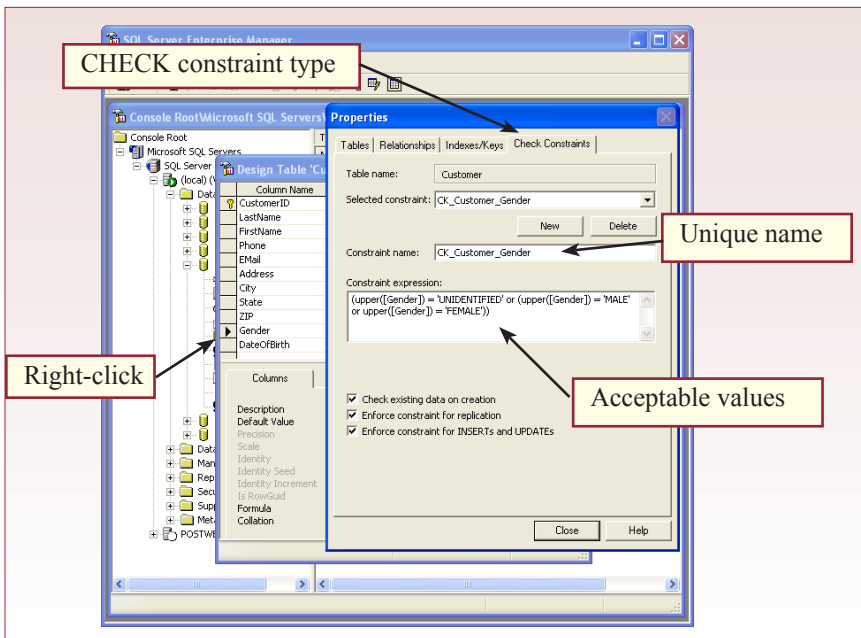


Figure 3.5

characters because the comparison is case sensitive. The three acceptable items are entered in the list with single quotes around each word or phrase and separated by commas.

Notice that it is straightforward to specify default values. These are values that you want entered whenever the user does not provide a value for the specified column. 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 entering data. For example, a SaleDate can be set to the SYSDATE function so that the current date is automatically entered. For example, to specify a default value of 1 for Cost, simply add the line; DEFAULT 1 (with no commas or equal sign).



Activity: Create Tables with SQL

It is relatively easy to create tables in SQL Server with the design screen in the enterprise manager. However, right click the table name, choose All Tasks, and Generate SQL Script. You will see that the table is actually being created in SQL. The design screen simply takes your choices and converts them into the proper SQL syntax. Eventually, you will find several advantages to simply creating the tables in SQL yourself. For starters, all you need is a simple system to execute SQL commands instead of the visually oriented design screen. You can quickly build and execute SQL commands and send them across the Internet when you are thousands of miles away from the main database machine. Also, some options are easier to control by entering them directly into the SQL—instead of trying to figure out how to get the designer to understand what you want. But probably the most important reason

Action

- Start the Windows Wordpad program
- Type the CREATE TABLE commands
- Save the file as ProductCategory.sql
- Start the SQL Query Analyzer program
- Select the Powder database
- Log in as the All Powder developer
- Open the ProductCategory file
- Click the Execute Query button to run it.

```
CREATE TABLE Customer
(
    CustomerID          int IDENTITY(1,1) NOT NULL,
    LastName            nvarchar(25),
    FirstName           nvarchar(25),
    Phone               nvarchar (25),
    Email               nvarchar (120),
    Address             nvarchar (50),
    City                nvarchar (50),
    State               nvarchar (25),
    ZIP                 nvarchar (15),
    Gender              nvarchar (15),
    DateOfBirth         datetime,
    CONSTRAINT pk_Customer PRIMARY KEY (CustomerID),
    CONSTRAINT ck_Customer_Gender
        CHECK (Upper(Gender) IN
            ('FEMALE', 'MALE', 'UNIDENTIFIED'))
)
```

Figure 3.6

for using plain SQL is that you can create and store the statements in a simple text file. Whenever you need to rebuild the tables, you simply execute the file and all of the tables will be created. Figure 3.6 shows the SQL statement that creates the Customer table with the primary key and gender constraints. Notice that the syntax for the columns is straightforward. Simply list the column name followed by its data type. The columns are separated by commas. If the column names are reserved words or contain special characters you must put square brackets around the name.

The primary key constraint is straightforward. It is identified with the CONSTRAINT keyword followed by the name of the constraint (pk_Customer) and the type of constraint (PRIMARY KEY). The key columns are then listed in parentheses. If there are multiple columns, they are separated with commas. The check constraint on the gender column is similar. Some people prefer to write it directly beneath the Gender column, but it is equally easy to read if all check constraints are listed at the end of the definition. Again, it is straightforward: begin with the CONSTRAINT keyword and its unique name. Add the CHECK keyword and follow it by the condition to be evaluated. Check constraints are used to specify one type of business rules and to ensure that the database retains consistent data.

Figure 3.7

```
CREATE TABLE ProductCategory
(
    Category            nvarchar(50),
    CategoryDescription nvarchar(250),
    CONSTRAINT pk_ProductCategory PRIMARY KEY (Category)
)
```

For practice, you should create the ProductCategory table shown in Figure 3.7 using the SQL statements. The most powerful aspect of using SQL is that you can have a file of commands that you can execute on a different machine to create the tables. To illustrate the process, start a text editor (Wordpad) and type in the commands to create the ProductCategory table. You do not have to worry about tab spacing, but the alignment does make the command easier to read in the file. You do have to be extremely careful about commas and parentheses. Save the file as “ProductCategory.sql.”

If you have a large file to create many tables, it is easiest to use the command-line program isql to execute the file. Using line commands also enables you to script the database build process within a batch file. However, for shorter queries with one or two tables, it is straightforward to run the query within the query analyzer. You will use the query analyzer often, so you might want to place a shortcut icon to it on your desktop. Start the SQL Query Analyzer. Choose the Powder database and log in if requested. Retrieve the query by opening the file. Click the Execute Query arrow to run the command. The command will execute in a couple of seconds and it will tell you that the table has been created. However, if you receive any error messages, return to the text file and make sure the commas and parentheses are correct.

If you want to remove the table, simply type `DROP TABLE ProductCategory;` at the prompt and the table will be removed. Close the query analyzer when you are finished.

Relationships



Activity: Define Relationships

SQL Server has a graphical tool to create and display table relationships. However, relationships can also be formed within the SQL `CREATE TABLE` command. Figure 3.8 shows a typical relationship between the Department and Employee tables. Employees are assigned to a department, but the department comes from a list in the Department table. In this example, the Department column in the Employee table is a foreign key because it refers to a primary key in a second table. The Department table is the reference table because it supplies the data to the Employee table.

You can use the SQL Server visual designer to create the relationship. First, in the Enterprise Manager, create the Department table and then the Employee table. Next, create a new diagram and add the new Department and Employee tables. Drag the Department column from the Department table and drop the cursor on the Department column in the Employee table. As shown in Figure 3.9, a relationship box will pop up.

Be sure that you check the Cascade Delete option. With this option enabled, if anyone deletes a department from the Department table, the DBMS will automatically remove all employees in that department. Why? To ensure consistency of

Action

- Create the Department table.
- Be sure the Department column is keyed.
- Create the Employee table.
- Set EmployeeID as a primary key constraint.
- Create a new diagram and add both tables.
- Drag the Department column from the Department to the Employee table.
- Verify the columns match.
- Check the update boxes.

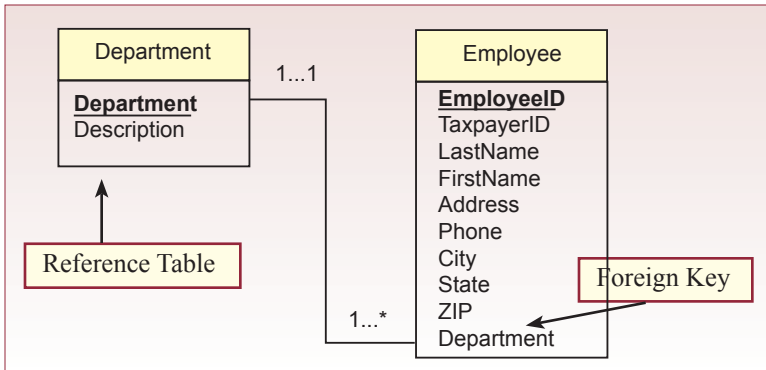
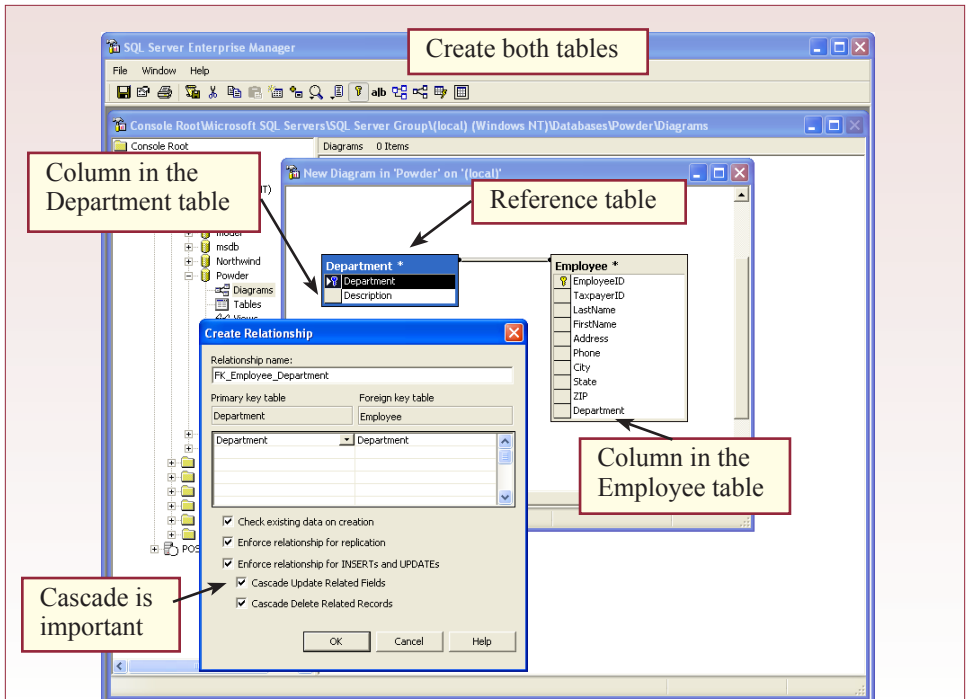


Figure 3.8

the data. If the department no longer exists, then you cannot say that employees belong to that department. Cascade deletions save you an enormous amount of grief in the long run because they keep the data accurate. However, deleting one row of data in a high-level table could result in the deletion of huge chunks of the database. Later, you will learn how to assign security permissions to the tables so that only a few people will be allowed to delete departments.

You can also create foreign key relationships within SQL. If you generate the SQL script for the Employee table, you can see the syntax. Foreign key relationships are the main reason that it is often easier to create tables within a text file first and then execute the text file. Note that the tables must be created in a specific order. In this example, the Department table has to be defined before the Employee table. You could go back in the designer and alter a table to set a new relationship, but it is much safer to define everything at one time. With a text file,

Figure 3.9



you save the entire database structure and re-create it almost instantly. Figure 3.10 shows the SQL to create the Department and Employee tables. The foreign key constraint is straightforward, but you have to enter the keywords in the specified order. You begin with the CONSTRAINT keyword followed by the name of the constraint as usual. The FOREIGN KEY phrase specifies the type of constraint, and it is followed by the name of the column (or columns) in the Employee table that is affected by the constraint. The keyword REFERENCES is followed by the name of the reference table (Department), and the column referred to is listed in parentheses. Note the use of the ON DELETE CASCADE command to set the Cascade option. One table can have several relationships with other tables. You simply list each one as a new foreign key constraint.

Figure 3.10 also shows how to specify a default value for the Department column. In this case, employees will be assigned to the Sales department if no other value is entered. Of course, you should make sure that the Sales department is listed in the Department table.

At this point, you should create all of the All Powder tables in SQL Server. You can use the table designer to help you get started, but you should ultimately generate the SQL scripts and review them. Then, if anything goes wrong or you need to make substantial changes to the design, you can drop all of the existing tables and start over. Make any necessary changes to the text file, then return to SQL Plus and execute the file to rebuild all of the tables.

You can create a SQL Server diagram to work directly with the graphical representation of the tables. Figure 3.11 shows that you can modify tables and create relationships directly. Changes made to the diagram are transferred to the underlying table definitions. However, when you are finished, you should return to the

Figure 3.10

```
CREATE TABLE Department
(
    Department    nvarchar(50),
    Description    nvarchar(150),
    CONSTRAINT pk_Department PRIMARY KEY (Department)
)
Go
CREATE TABLE Employee
(
    EmployeeID    int,
    TaxpayerID    nvarchar(50),
    LastName       nvarchar(25),
    FirstName      nvarchar(25),
    Address        nvarchar(50),
    Phone         nvarchar(25),
    City          nvarchar(50),
    State         nvarchar(15),
    ZIP           nvarchar(15),
    Department     nvarchar(50)
    DEFAULT 'Sales',
    CONSTRAINT pk_Employee PRIMARY KEY (EmployeeID),
    CONSTRAINT fk_DepartmentEmployee FOREIGN KEY (Department)
    REFERENCES Department(Department)
    ON DELETE CASCADE
)
Go
```

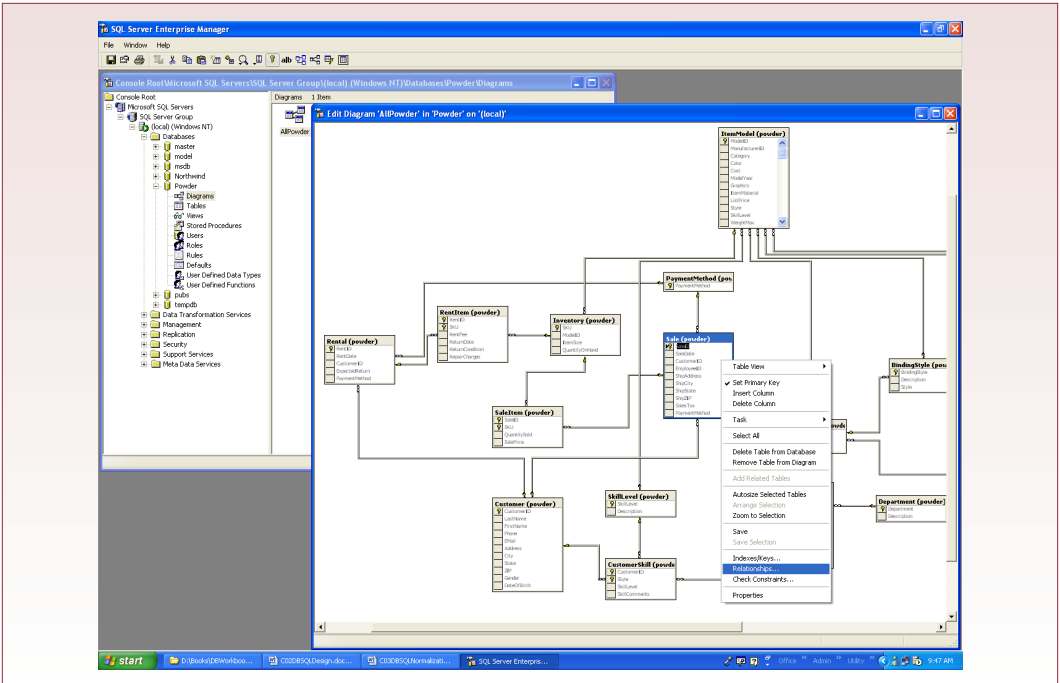


Figure 3.11

table list and generate the SQL definitions to a text file so that you can quickly recreate the tables later.

You can create a relationship diagram from scratch by starting a new diagram and adding each table individually. Alternatively, if you already have the tables defined, you can create a new diagram and then drag-and-drop all of the tables onto the diagram. The system will automatically arrange the tables and show the relationships.



Activity: Estimate the Database Size

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 how much disk space you need to purchase, and whether you will need more servers and faster processors.

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 3.12 shows the process for the Customer table. Some of the column size estimates are straightforward. Look back to Chapter 2 for a reminder 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 of customer last names. You could use existing data, or perhaps a sample

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.

CustomerID	NUMBER(12)	8
LastName	NVARCHAR(50)	30
FirstName	NVARCHAR(50)	20
Phone	NVARCHAR(50)	24
Email	NVARCHAR(150)	100
Address	NVARCHAR(50)	50
State	NVARCHAR(50)	4
ZIP	NVARCHAR(15)	20
Gender	NVARCHAR(15)	20
DateOfBirth	Date	7
Average bytes per customer		283
Customers per week (winter)		*200
Weeks (winter)		*25
Bytes added per year		1,415,000

Figure 3.12

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 there are about 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 to be about 1 million bytes.

You need to follow a similar process for all of the tables in the case. Figure 3.13 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, SQL Server can easily handle this database on a PC-based server.

Figure 3.13

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.

Database Queries and SQL

Chapter Outline

Database Queries, 49

Case: All Powder Board and Ski Shop, 49

Lab Exercise, 50

All Powder Board and Ski Data, 50

Computations and Subtotals, 60

Exercises, 65

Final Project, 66

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 efficiently store data. 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. At one time, SQL Server includes a visually oriented QBE system, but you really need to learn and understand the straight-text SQL. The logic of SQL is the same as for QBE, but it can be cumbersome because you have to type more text. Also, the JOIN statements are a little more confusing in SQL.

SQL Server has three methods to enter SQL commands: (1) the query analyzer, (2) create a new view, and (3) command-line `isql`. In general, you will find the QBE editor for views to be the easiest to use. However, if you want to run a chain of SQL commands, you will generally need to save them as a text file and run them with the `isql` command. The query analyzer is useful when you need to improve the performance of long-running queries. Note that when you want to execute several SQL commands within one file, you often separate them with the `GO` command.

There is one other important issue you need to know about SQL Server. You often need to issue a commit command to ensure that your changes are written to the database. It is part of the transaction processing system that is explained in more detail in Chapter 7.

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, when you are writing queries, it helps to have a copy of the class (relationship) diagram handy. 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. 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, generate data for items with appropriate attributes, and 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.

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, sample data files are provided on the main textbook CD for each of these tables, plus the common supporting tables. The files for each of these tables are stored in the standard comma-separated values (CSV) format. You could import these tables manually. Rather than force you to enter all of the commands by hand, you can run a batch file that will create the tables and load the data into the tables. You need to run a batch file from command line that handles these tasks.



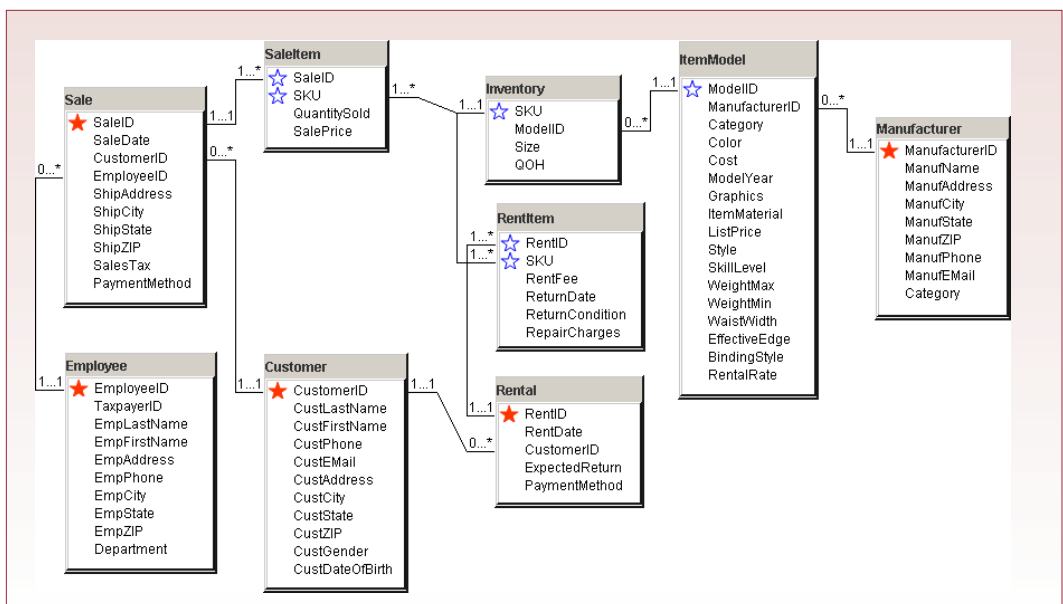
Activity: Import Data

To begin, you should copy all of the files in the BuildAllPowder folder (including the subfolder) to a folder on the computer running SQL Server. In particular, the batch files run the isql and bcp utilities. To simplify the process,

Action

- Copy files from BuildAllPowder.
- Drop or rename conflicting tables.
- Switch to command mode.
- Run the batch file BuildAllPowderSQL server username password.
- Check the tables.

Figure 4.1



the best approach is to start with a clean schema. Notably, you should delete any tables that might conflict with the new tables. The easiest way to delete the tables is to run the included SQL script file `DropAllPowderTablesSQL.sql`. Your other option is to check the names in that file and see if any conflict with tables that you already have, then rename your tables.

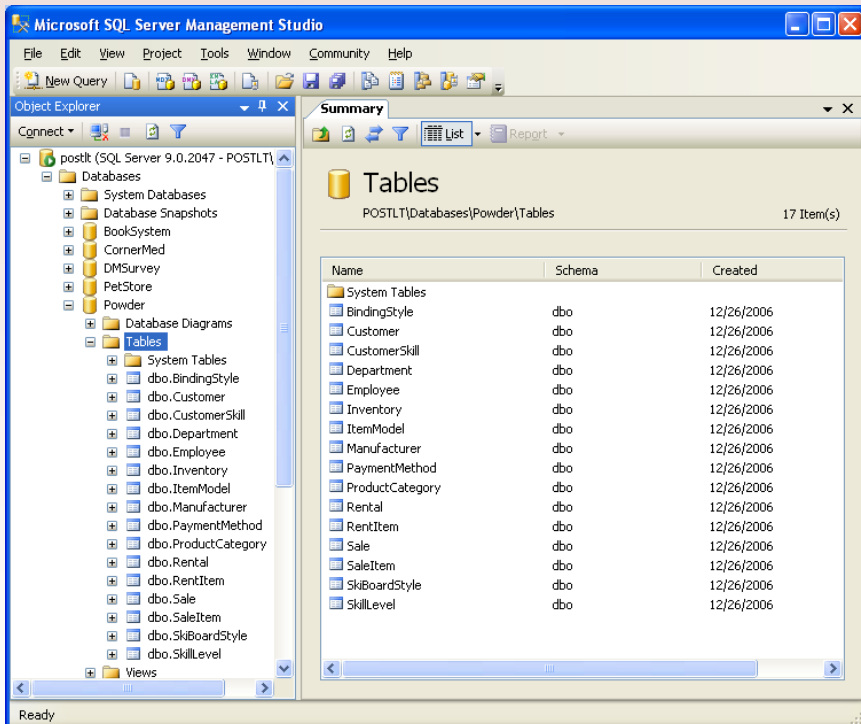
The rest of the installation is relatively automatic. You need to switch to command mode, using `Start/All Programs/Accessories/Command Prompt`. Change to the directory holding the data files. Run the batch file `BuildAllPowderSQL` and specify the server, username, and password: `BuildAllPowderSQL server username password`.

After a minute or two, the tables should be created, the data copied, and the tables analyzed to improve performance. You should not receive any error messages. If you do see errors, they are probably due to an incorrect server name or password. As you can see in Figure 4.2, after the SQL script has completed with no errors, the tables that were created will be displayed in the enterprise manager console.

You should also consider creating the relationship diagram for the project. Using the SQL Server Enterprise Manager, open the database and add a new diagram. Pick all of the new tables that were created, and the wizard will build the initial diagram.

If you ever need to transfer files and the database structure to a different computer, this approach is a useful solution. As long as you can export the data as a CSV file, you can use a similar set of scripts to transfer the data. At some point

Figure 4.2



you should examine the batch files to see how the transfer is accomplished. But it is best to wait until after the next couple of labs.



Activity: Create Basic Queries

Creating a query requires that you translate a business question into a format the query system can process. Sometimes this step is straightforward; at 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?

SQL Server provides two primary methods to create a query. Figure 4.3 shows the SQL Query editor that opens by default and uses straight SQL. Experienced users will probably prefer this tool. Figure 4.4 shows that within the Enterprise Manager, you can create a View to help you create a query using a point-and-click interface. This approach will often be preferred by beginners because it builds the SQL command as you select items on the screen. Although it is relatively easy to use, you ultimately must learn the SQL syntax.

To avoid doing all of the labs twice, this chapter will focus on the QBE approach used in the Enterprise Manager View. Notice that the tool also shows the SQL syntax as the query is created, so it is a good way to learn SQL.

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

Action

Start a new View.

Right-click and Add the ItemModel table.

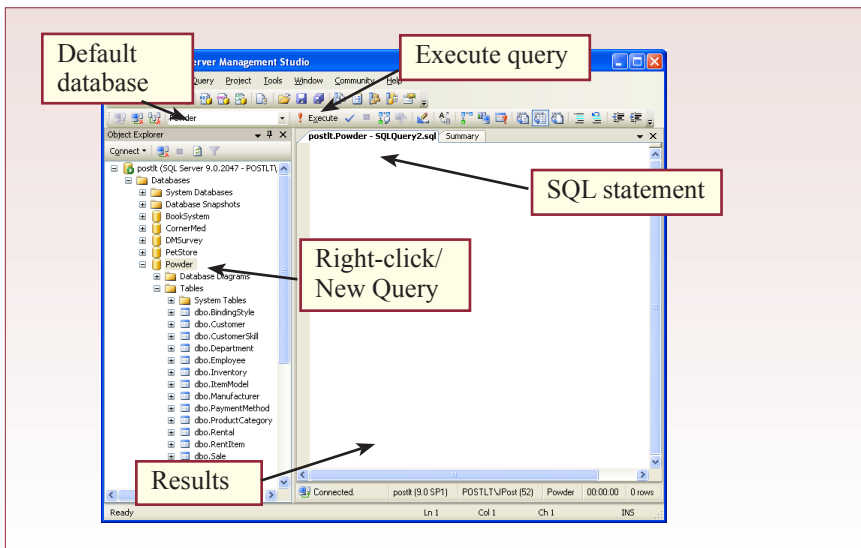
Select columns: Category, ListPrice, WeightMax, Color, and Graphics.

Enter conditions: Category='Board'
AND ListPrice<300 AND
WeightMax>150.

Check the SQL text.

Run the query.

Figure 4.3



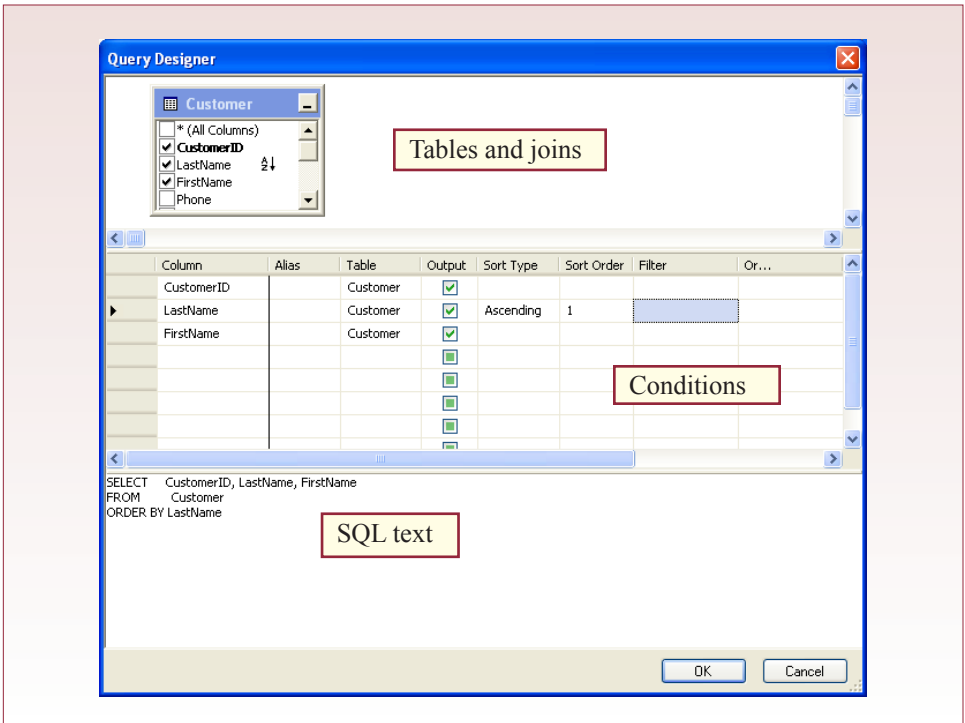


Figure 4.4

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 4.5 shows the basic query and the results. To create the view, first right-click the table area and choose the option to Add Table. Select the ItemModel table and close the selection box. Next, choose the columns that you want to display in the query. For this example, pick: Category, ListPrice, WeightMax, Color, and Graphics. The column data will be displayed in the order they are selected.

Figure 4.5

Question	Display snowboards with a list price under \$300 and max weight over 150 pounds.			
SQL	<pre>SELECT Category, ListPrice, WeightMax, Color, Graphics FROM ItemModel WHERE Category=N'Board' AND ListPrice < 300 AND WeightMax > 150;</pre>			
CATEGORY	LISTPRICE	WEIGHTMAX	COLOR	GRAPHICS
Board	292	188	Orange	Fade
Board	263	181	Magenta	Geometric
Board	262	179	Purple	Space
Board	290	194	Blue	Abstract
Board	294	158	Red	Sunrise
Board	270	191	Yellow	Landscape
Board	255	239	Red	Gothic
Board	256	171	Magenta	Sunrise
Board	283	226	Blue	Gothic
Board	277	163	White	Gothic
Board	259	223	Magenta	Linear

You can edit the SQL text if you want to quickly change the order. The next step is to enter the selection criteria. In the row for Category, enter Board. Notice that the tool will automatically change the text to =N'Board' which is SQL Server's method of converting the national (Unicode) character set. Then add the conditions for the ListPrice and WeightMax. Notice that all three conditions are entered in the same column, which means that all three are connected with AND clauses. That is, a row will match and be displayed only if it meets all three 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 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

Action

Start a new query.

Add the ItemModel table.

SELECT Category, Color, ItemMaterial, Style, ListPrice.

Enter conditions: Category='Ski'
And Style='Jump' And
ItemMaterial='Composite'.

Run the query to ensure it works.

Add the conditions for Color='Yellow'
and ListPrice<300.

Test the query.

Add the conditions for Color='Red' and
ListPrice<400.

Add the correct parentheses.

Run the query and test it.

Figure 4.6

The screenshot shows the Query Designer window for the ItemModel table. The Filter column contains three conditions: = 'Ski', = 'Composite', and = 'Jump'. A red box labeled "Three main conditions" points to these three filter entries. Below the designer, the SQL text is displayed as follows:

```
SELECT Category, Color, ItemMaterial, Style, ListPrice
FROM ItemModel
WHERE (Category = 'Ski') AND (ItemMaterial = 'Composite') AND (Style = 'Jump')
```

At the bottom, a results grid shows the following data:

Category	Color	ItemMaterial	Style	ListPrice
Ski	Magenta	Composite	Jump	372.00
Ski	Turquoise	Composite	Jump	248.00
Ski	Red	Composite	Jump	294.00
Ski	Magenta	Composite	Jump	425.00
Ski	Red	Composite	Jump	137.00
Ski	Red	Composite	Jump	223.00
Ski	Orange	Composite	Jump	388.00
Ski	Blue	Composite	Jump	229.00
...

Question	List jumping skis, made from composite materials. And Yellow And ListPrice < 300			
SQL	SELECT Category, Color, ItemMaterial, Style, ListPrice FROM itemModel WHERE Category=N'Ski' AND ItemMaterial=N'Composite' AND Style=N'Jump' AND Color='Yellow' AND ListPrice<300;			
CATEGORY	COLOR	ITEMATERIAL	STYLE	LISTPRICE
Ski	Yellow	Composite	Jump	\$70.00

Figure 4.7

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 4.6 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. Yellow skis are required to cost less than \$300, so what happens if you add both conditions to the query? Figure 4.7 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. So, the query returns only yellow skis.

To see the red skis, you have to add the option of Red as a color, but you also have to establish the higher acceptable price for red skis. The QBE solution is a little tricky. You can add the Red color and its price limit in a second column.

Figure 4.8

The screenshot shows the Query Designer window for a query named 'Query1'. The table selected is 'ItemModel'. The criteria table is as follows:

Column	Alias	Table	Output	Sort Type	Sort Order	Filter	Or...
Category		ItemModel	<input checked="" type="checkbox"/>			= 'N'Ski'	= 'N'Ski'
Color		ItemModel	<input checked="" type="checkbox"/>			= 'Yellow'	= 'Red'
ItemMaterial		ItemModel	<input checked="" type="checkbox"/>			= 'N'Composite'	= 'N'Composite'
Style		ItemModel	<input checked="" type="checkbox"/>			= 'N'Jump'	= 'N'Jump'
ListPrice		ItemModel	<input checked="" type="checkbox"/>			< 300	< 400

The SQL view shows the following query:

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

The results table shows the following data:

Category	Color	ItemMaterial	Style	ListPrice
1	Ski	Red	Composite	Jump 284.00
2	Ski	Red	Composite	Jump 137.00
3	Ski	Red	Composite	Jump 223.00
4	Ski	Yellow	Composite	Jump 70.00

The status bar at the bottom indicates: Query executed s... postt (9.0 SP1) POSTLT\NPost (52) Powder 00:00:00 4 rows

Remember that each additional condition column is connected with an OR statement. Figure 4.8 shows the catch that you have to duplicate all of the other AND conditions as well. Notice that the generated SQL text includes the duplicated conditions. If you were writing the query in straight SQL (without the QBE grid), you could use parentheses for the two color conditions and then write the base conditions only once.

Anytime you encounter a query that contains both And and Or connectors, you will have to use parentheses to specify how the conditions are grouped. Remember from algebra that conditions inside the innermost parentheses are evaluated first. The key in this example is to group the color yellow with its price condition and group the color red with its price condition. However, using the QBE grid, the system will rewrite your query and add the duplicate columns.

The final query shows that four skis match the conditions. Check them carefully to ensure that all conditions are met. Even if all of the skis in the result are acceptable, how do you know if the query found all of the matches? This question highlights one of the difficulties of any query language. The only way you know if the query is right is if you carefully build it step-by-step and test the individual steps. In this example, the first query was straightforward and ignored color and price constraints. It returned 20 matches, so the four matches returned by the final query seems like a reasonable number. In this case, the two sets are small enough that you can check the results by hand.



Activity: Use Multiple Tables in a 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. SQL Server 2000 supports the SQL standard for joining tables. As it is the easiest to understand, it will be used here. The older SQL Server syntax is shown at the end of this section because you will still see many queries that use it.

Action

Start with a blank query.

Use only the Sale table.

Set SaleID, SaleDate, CustomerID, and PaymentMethod.

Set the SaleDate between 1/5/2004 AND 5/31/2004.

Set PaymentMethod to Cash.

Run the query to test it.

To understand the join process, create a new query using just the Sale table. The objective is to find all of the sales in May that were made with a cash payment. Figure 4.9 shows the initial query. Note the use of the Between clause to specify the month of May. The date format should follow the style specified in your Windows system. Windows enables you to change the default date style using the Regional settings in the Control Panel. SQL Server adopts this format as the default style.

Observe that the query returns the CustomerID. But no one is going to memorize CustomerID numbers. Instead, you need to look up the 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. Now you could take each of the ID values returned by the Sale query and create a new query on the Customer table and manually enter the values to find

Question	List customers (ID) with sales in May who paid with Cash.																				
SQL	<p>COLUMN PaymentMethod Format A15</p> <pre>SELECT SaleID, SaleDate, CustomerID, PaymentMethod FROM Sale WHERE SaleDate Between '01-May-2004' AND '31-May-2004' AND PaymentMethod='Cash';</pre>																				
	<table border="1"> <thead> <tr> <th>SALEID</th> <th>SALEDATE</th> <th>CUSTOMERID</th> <th>PAYMENTMETHOD</th> </tr> </thead> <tbody> <tr> <td>1495</td> <td>13-MAY-04</td> <td>645</td> <td>Cash</td> </tr> <tr> <td>1304</td> <td>07-MAY-04</td> <td>1309</td> <td>Cash</td> </tr> <tr> <td>1356</td> <td>02-MAY-04</td> <td>314</td> <td>Cash</td> </tr> <tr> <td>1376</td> <td>10-MAY-04</td> <td>69</td> <td>Cash</td> </tr> </tbody> </table>	SALEID	SALEDATE	CUSTOMERID	PAYMENTMETHOD	1495	13-MAY-04	645	Cash	1304	07-MAY-04	1309	Cash	1356	02-MAY-04	314	Cash	1376	10-MAY-04	69	Cash
SALEID	SALEDATE	CUSTOMERID	PAYMENTMETHOD																		
1495	13-MAY-04	645	Cash																		
1304	07-MAY-04	1309	Cash																		
1356	02-MAY-04	314	Cash																		
1376	10-MAY-04	69	Cash																		

Figure 4.9

the names. However, the table JOIN command is much easier and more powerful to use.

In the SQL query, add the Customer table. Because the foreign key relationships were established when the tables were created, the designer automatically shows the connection between the two tables. If this connection was incorrect or unnecessary for the query, you could delete it and create a different one by dragging a column from one table and dropping on the desired matching column in the second table.

Check the FROM statement in the SQL syntax to see the effect of adding the table. Notice the INNER JOIN line that adds the Customer table and specifies how it is connected to the Sale table: INNER JOIN Customer ON Sale.CustomerID = Customer.CustomerID. The command must use the table prefix on the CustomerID column in the SELECT statement: Sale.CustomerID..

Action

Add the Customer table.

Check the join condition:

ON Sale.CustomerID = Customer.
CustomerID.

Add Customer LastName and FirstName to the SELECT statement.

Run the query to test it.

Figure 4.10

The screenshot shows the Query Designer interface with two tables, Sale and Customer, joined together. The Sale table columns are SaleID, SaleDate, CustomerID, PaymentMethod, and EmployeeID. The Customer table columns are CustomerID, LastName, FirstName, and Phone. The join condition is ON Sale.CustomerID = Customer.CustomerID. The SELECT statement includes Sale.SaleID, Sale.SaleDate, Sale.CustomerID, Sale.PaymentMethod, Customer.LastName, and Customer.FirstName. The WHERE clause filters for dates between 5/1/2006 and 5/31/2006 and PaymentMethod = 'N\Cash'. The results grid shows 4 rows of data.

Additional columns

Join conditions

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 2/29/2006
AND ItemModel.Category="Ski"

```

Figure 4.11

Figure 4.10 shows the basic query design. Once the tables are joined correctly, you can add any column to the other clauses. In this case, place the Customer LastName and FirstName columns in the SELECT clause. 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. Just be sure to specify the table name (Customer.CustomerID).

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 SQL, it is

Figure 4.12

The screenshot shows the Query Designer window with the following tables and fields selected:

- Manufacturer:** Name, City
- ItemModel:** Category
- Inventory:** SKU, ModelID, ItemSize, QuantityOnHand
- Customer:** LastName, FirstName
- Sale:** SaleDate, CustomerID, EmployeeID
- SaleItem:** SKU, QuantitySold, SalePrice

The design grid below shows the following configuration:

Column	Alias	Table	Output	Sort Type	Sort Order	Filter	Or...
LastName		Customer	<input checked="" type="checkbox"/>				
FirstName		Customer	<input checked="" type="checkbox"/>				
Category		ItemModel	<input checked="" type="checkbox"/>			= 'N'Ski'	
Name		Manufacturer	<input checked="" type="checkbox"/>			= 'N'Atomic'	
SaleDate		Sale	<input checked="" type="checkbox"/>			BETWEEN '1/1/2006'	

The SQL view at the bottom of the window shows the following query:

```

SELECT Customer.LastName, Customer.FirstName, ItemModel.Category, Manufacturer.Name, Sale.SaleDate
FROM
  Manufacturer INNER JOIN
  ItemModel ON Manufacturer.ManufacturerID = ItemModel.ManufacturerID INNER JOIN
  Inventory ON ItemModel.ModelID = Inventory.ModelID INNER JOIN
  SaleItem ON Inventory.SKU = SaleItem.SKU INNER JOIN
  Sale ON SaleItem.SaleID = Sale.SaleID INNER JOIN
  Customer ON Sale.CustomerID = Customer.CustomerID
WHERE (ItemModel.Category = 'N'Ski') AND (Manufacturer.Name = 'N'Atomic') AND (Sale.SaleDate BETWEEN '1/1/2006' AND '2/28/2006')

```


best to think about the query and look at the relationship screen for a minute. As shown in Figure 4.11, 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. The join statements are the key to creating this query. Begin with one table, then add each new table after an INNER JOIN command. If you write the SQL by hand, be sure to specify the table links using a collection of ON conditions. Once the tables and links have been defined, you can use columns from any of the tables. Just remember that if a column by the same name exists in more than one table, you refer to that column with its full Table.Column name.

Older SQL Server queries are based on the older SQL syntax. Join conditions represent one of the greatest differences in this syntax. To see the difference, the Sale/Customer query will be rebuilt. Figure 4.13 shows the difference. Begin the query with the SELECT, FROM, and WHERE clauses. Enter the columns to be displayed, then the date condition in the WHERE clause. List the Sale and Customer tables in the FROM clause separated by a comma. Finally, add the join condition (Sale.CustomerID = Customer.CustomerID) to the WHERE clause. There are no INNER JOIN or ON statements. When you run the query, you should receive the same results as earlier. Notice that because of the way the tables were

Figure 4.13

The screenshot shows a SQL query editor window titled "postIt.Powder - SQLQuery3.sql*" with a "Summary" tab. The query text is as follows:

```

SELECT LastName, FirstName, ItemModel.Category, Name, SaleDate
FROM Manufacturer, ItemModel, Inventory, SaleItem, Sale, Customer
WHERE Manufacturer.ManufacturerID = ItemModel.ManufacturerID
AND ItemModel.ModelID = Inventory.ModelID
AND Inventory.SKU = SaleItem.SKU
AND SaleItem.SaleID = Sale.SaleID
AND Sale.CustomerID = Customer.CustomerID
AND (ItemModel.Category = N'Ski')
AND (Name = N'Atomic')
AND (Sale.SaleDate BETWEEN '1/1/2006' and '2/28/2006')

```

Two yellow callout boxes with red borders provide annotations:

- The first box, labeled "List tables separated by commas", has an arrow pointing to the `FROM` clause of the query.
- The second box, labeled "Place join condition in the WHERE clause", has an arrow pointing to the `WHERE` clause of the query.

At the bottom of the window, the "Results" tab is active, displaying a table with the following data:

	LastName	FirstName	Category	Name	SaleDate
1	Mahoney	Francis	Ski	Atomic	2006-01-23 00:00:00.000
2	Patterson	Gene	Ski	Atomic	2006-02-15 00:00:00.000

created, SQL Server insists that you include the owner name as well as the table name. For example, powder.Manufacturer. In fact, you can go to one more level and specify the name of the database as well. For instance, the full name of the SaleDate column is: Powder.powder.Sale.SaleDate.



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 acquisition cost. The profit is simply the difference between the list price and the cost. In the SELECT clause you enter the calculation and give it a name using the AS keyword: ListPrice-Cost AS Profit. Notice that the query is sorted by Category and ListPrice. Simply add an ORDER BY clause at the end of the command with the columns you want sorted. The DESC option specifies a descending order.

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

Action

Create a new query using only the ItemModel table.

In the SELECT row, add a new pseudo column to compute ListPrice-Cost As Profit.

Add the ORDER BY line to sort by Category and List Price descending.

Run the query.

Figure 4.14

The screenshot shows the Query Designer interface. The table list includes ItemModel with columns ModelID, ManufacturerID, Category, and Color. The query grid shows the following configuration:

Column	Alias	Table	Output	Sort Type	Sort Order	Filter
Category		ItemModel	<input checked="" type="checkbox"/>	Ascending	1	
ItemMaterial		ItemModel	<input checked="" type="checkbox"/>			
ListPrice		ItemModel	<input checked="" type="checkbox"/>	Descending	2	
ListPrice - Cost	Profit		<input checked="" type="checkbox"/>			

The SQL statement shown is:

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

The results table below shows the data returned by the query, with the Profit column highlighted in red:

	Category	ItemMaterial	ListPrice	Profit
1	Board	Wood	649.00	227.15
2	Board	Wood	647.00	226.45
3	Board	Wood	646.00	226.10
4	Board	Wood	644.00	225.40
5	Board	Fiberglass	642.00	224.70
6	Board	Wood	642.00	224.70
7	Board	Composite	633.00	221.55
8	Board	Wood	633.00	221.55
9	Board	Fiberglass	629.00	220.15

Lower	To lowercase
Len	Length/number of characters
Substring	Get substring
LTrim, RTrim	Remove leading and trailing spaces
Upper	To uppercase
GetDate	Current date
DateAdd	Add days, months, years to a date
DateDiff	Subtract two dates
Convert	Highly detailed formatting
Day, Month, Year, DatePart	Parts of a date
Abs	Absolute value
Cos	Cosine, all common trig functions
Floor	Integer, drop decimal values
Round	Round-off

Figure 4.15

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 practice. The Convert 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 LEFT(CONVERT(nvarchar, SaleDate, 120),

Figure 4.16

The screenshot shows the Query Designer window for a query named 'Sale'. The table 'Sale' is selected, and the columns 'SaleID', 'SaleDate', and 'LEFT(CONVERT(nvarchar, SaleDate, 120), 7)' are included. The 'SaleMonth' column is highlighted in red in the data preview.

Column	Alias	Table	Output	Sort Type	Sort c
SaleID		Sale	<input checked="" type="checkbox"/>		
SaleDate		Sale	<input checked="" type="checkbox"/>		
LEFT (CONVERT (nvarchar, SaleDate, 120), 7)	SaleMonth		<input checked="" type="checkbox"/>		

```

SELECT SaleID, SaleDate, LEFT(CONVERT(nvarchar, SaleDate, 120), 7) AS SaleMonth
FROM Sale

```

	SaleID	SaleDate	SaleMonth
1	1002	2006-03-17 00:00:00.000	2006-03
2	1003	2006-06-25 00:00:00.000	2006-06
3	1004	2006-06-30 00:00:00.000	2006-06
4	1005	2006-04-26 00:00:00.000	2006-04
5	1006	2006-01-31 00:00:00.000	2006-01
6	1007	2006-02-19 00:00:00.000	2006-02
7	1008	2006-04-12 00:00:00.000	2006-04
8	1009	2006-03-09 00:00:00.000	2006-03
9	1010	2006-03-07 00:00:00.000	2006-03

7) AS SaleMonth. The value of 120 specifies the date format. Several standard formats are available and each has a given style number. The SQL Server Books Online provides a list of the various options. The Left command is used to throw away the day and leave the year and month. You can also use the Year and Month functions, but those return numeric values instead of characters.

Action

Create a new query.

Use only the Sale table.

SELECT SaleID and SaleDate.

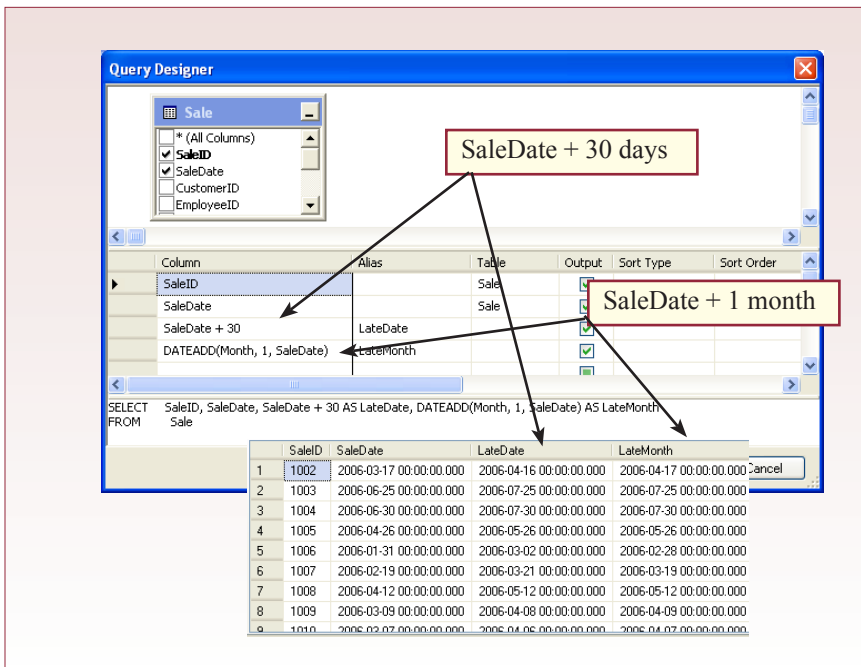
Add 30 days to the SaleDate to get LateDate.

Use DateAdd to add one month to the SaleDate to get SaleMonth.

Run the query.

SQL automatically performs data arithmetic with days. Adding or subtracting a number from a date results in a new date that is different by the specified number of days. Figure 4.17 shows how easy it is to add 30 days to a SaleDate to produce a common billing late date. Notice that the date arithmetic is correct in that it automatically handles months, years, and even leap years. If you want to add or subtract in increments other than days, you need to use SQL Server's DateAdd function. To subtract dates in terms of months, use the DateDiff function. Both the day and month arithmetic can use fractional values. For example, you could add 1.5 months to a date. You will often see fractional values if you subtract a date from today's date, which is given by GetDate. Since GetDate also includes the time of day, you will get noninteger results. If you only want the integer portion, you can use the Floor or Round functions. The Floor function truncates fractional values by throwing away all digits to the right of the decimal point. The Round function performs standard rounding to the specified decimal place.

Figure 4.17





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. Just remember that it simply counts the number of rows, while Sum adds up the numbers within a row. The challenge is to identify when you need to use Count instead of Sum.

Action

Create a new query.

Add the Sale table.

SELECT ShipState and SalesTax

WHERE ShipState = 'CA'.

Run the query.

Verify the correct states are displayed.

Uncheck ShipState as output.

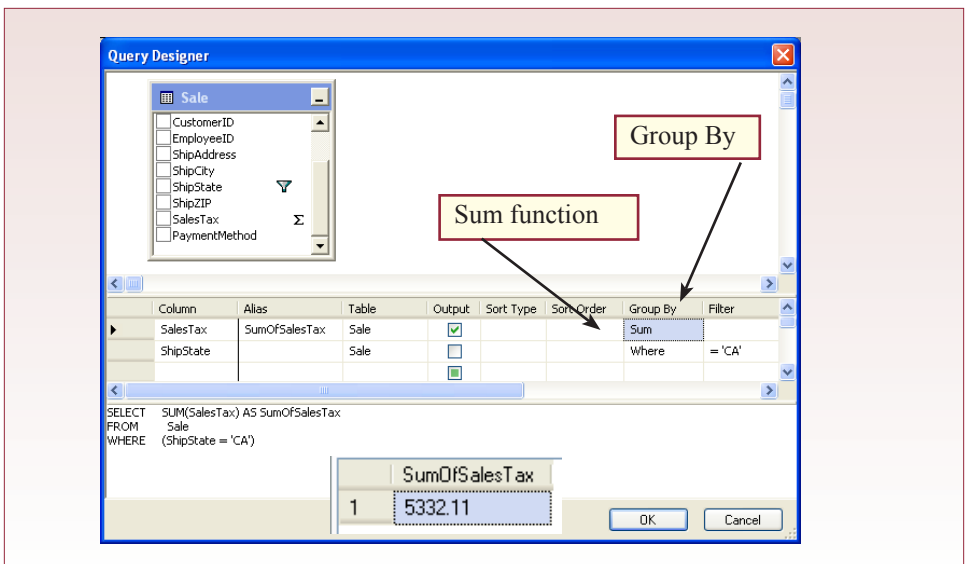
Set Sum option for SalesTax.

Run the query.

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. Ignoring totals for the moment, 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 SQL. Remove the SaleState from the SELECT statement and add the Sum function around the SalesTax: Sum(SalesTax) AS SumOfSalesTax. Figure 4.18 shows the total you should receive when you run the query. Why was it important to run the query first without the total? Because the total shows you only one number. How do you know the number is correct? You should always run a straight retrieval to ensure that the correct rows are being selected before you perform calculations on them. Of course, most aggregation queries will also use multiple tables—which makes it even more important that you check the detail rows first.

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 4.18



Question	Compute the sales tax total for each state.																																		
SQL	COLUMN ShipState Format A10 SELECT ShipState, Sum(SalesTax) AS SumOfSalesTax FROM Sale GROUP BY ShipState'																																		
<pre>SHIPSTATE SUMOFSALESTAX ----- -</pre> <table> <tr><td>AK</td><td>0</td></tr> <tr><td>AL</td><td>784.77</td></tr> <tr><td>AR</td><td>313.25</td></tr> <tr><td>AZ</td><td>510.93</td></tr> <tr><td>CA</td><td>5332.11</td></tr> <tr><td>CO</td><td>347.48</td></tr> <tr><td>CT</td><td>254.38</td></tr> <tr><td>DC</td><td>285.95</td></tr> <tr><td>DE</td><td>0</td></tr> <tr><td>FL</td><td>1404.34</td></tr> <tr><td>GA</td><td>470.61</td></tr> <tr><td>HI</td><td>175.49</td></tr> <tr><td>IA</td><td>164.01</td></tr> <tr><td>ID</td><td>330.12</td></tr> <tr><td>IL</td><td>1200.57</td></tr> <tr><td>IN</td><td>952.35</td></tr> <tr><td>KS</td><td>302.96</td></tr> </table> <p>(other states not shown)</p>		AK	0	AL	784.77	AR	313.25	AZ	510.93	CA	5332.11	CO	347.48	CT	254.38	DC	285.95	DE	0	FL	1404.34	GA	470.61	HI	175.49	IA	164.01	ID	330.12	IL	1200.57	IN	952.35	KS	302.96
AK	0																																		
AL	784.77																																		
AR	313.25																																		
AZ	510.93																																		
CA	5332.11																																		
CO	347.48																																		
CT	254.38																																		
DC	285.95																																		
DE	0																																		
FL	1404.34																																		
GA	470.61																																		
HI	175.49																																		
IA	164.01																																		
ID	330.12																																		
IL	1200.57																																		
IN	952.35																																		
KS	302.96																																		

Figure 4.19

replace it with each state, but there is an easier way. As shown in Figure 4.19, start a new query the same way as the last one. Use the Sale table and select the ShipState and SalesTax columns, but do not specify any limiting conditions. Click the Group By button and use the Sum function to total the SalesTax column and be sure to set the alias name. Set the Group

By option for the SaleState, which adds a GROUP BY clause at the end of the command. Tell it to compute the totals for each state with GROUP BY ShipState. When you run the query, you will get a list of all of the states with sales followed by the total sales tax collected for that state. 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

Action

Create a new query.

Use the Sale table.

Select columns: ShipState and Sum(SalesTax) AS SumOfSalesTax.

Add a row at the bottom: GROUP BY ShipState.

Run the query.

Figure 4.20

Question	Compute the total value of all sales to Colorado.
SQL	SELECT Sum(QuantitySold*SalePrice) As SaleTotal FROM Sale INNER JOIN SaleItem ON Sale.SaleID = SaleItem.SaleID WHERE Sale.ShipState='CO';
<pre>SALETOTAL ----- 4964</pre>	

```
CREATE VIEW ColoradoSales AS
SELECT Sum(QuantitySold*SalePrice) AS SaleTotal
FROM Sale INNER JOIN SaleItem ON Sale.SaleID = SaleItem.SaleID
WHERE ShipState='CO'
```

Figure 4.21

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. Create the formula on the SELECT row with the command: QuantitySold * SalePrice AS SaleTotal. To select the state, enter 'CO' as the criteria in the WHERE clause. 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 the SQL, you need to compute the total. As shown in Figure 4.20, simply add the Sum function and place the parentheses around the multiplied values.

There is one more trick you need to learn before finishing this lab. You need to be able to save a query so that you can use it in other queries or reports. In SQL Server, a saved query is called a View. You can save it directly from the View designer, or Figure 4.21 shows you the CREATE VIEW command you can use in Query Analyzer. You now have a view called ColoradoSales that performs the SELECT statement. To test it, clear the SQL window and create the simple query: SELECT * FROM ColoradoSales. Run this query and it will execute the stored query to compute and display the total sales in Colorado. You could use the enterprise manager to delete this new view, or simply issue the SQL Drop command: Drop View ColoradoSales.

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.

Advanced Queries

Chapter Outline

Advanced Database Queries, 68

Case: All Powder Board and Ski Shop, 69

Lab Exercise, 69

All Powder Board and Ski Data, 69

SQL Data Definition and Data Manipulation, 78

Exercises, 84

Final Project, 86

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.

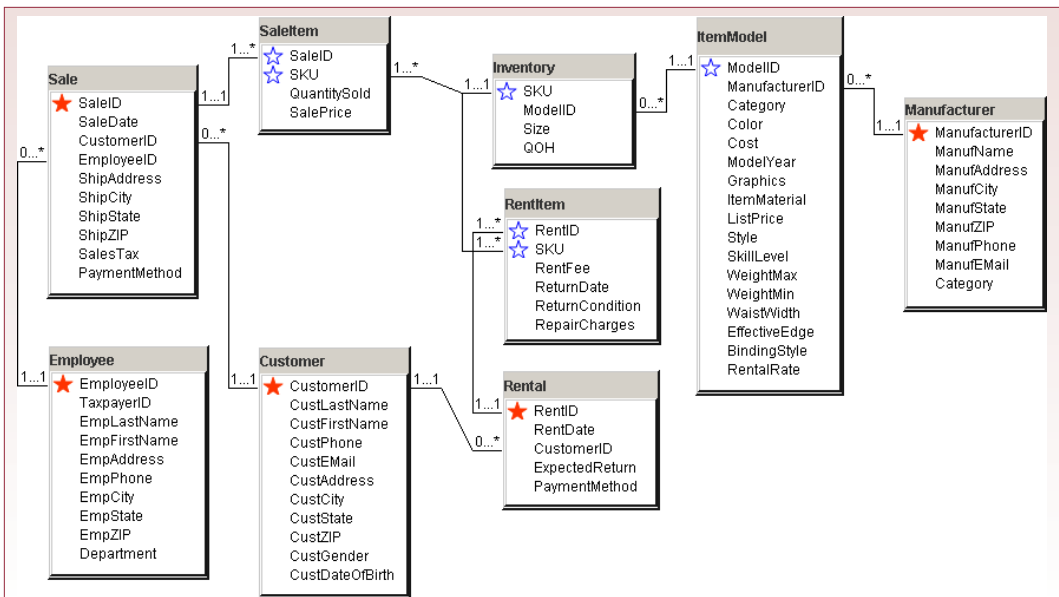
Subqueries are one of the more interesting features of SQL. A subquery is a query that calls a second query to obtain additional data. Instead of looking up a second set of numbers yourself, you can add a second query to do the work automatically.

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.

Advanced queries generally rely exclusively on text-based SQL. Even if you have a visual QBE system available, it is much safer to use straight text to create difficult queries. One of the most dangerous aspects of any query is that the system will almost always return some type of data. You need to make sure the system is returning the correct data by ensuring that the query is actually asking for exactly what you think it is asking. Most of the data definition statements (such as CREATE TABLE, INSERT, DELETE, and UPDATE) will often be stored in text files that can be run as separate batches later to accomplish some larger task. Just remember to test all of them first.

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. While the query is more important than the actual results, the results are useful to help you decide if you have constructed the query properly.

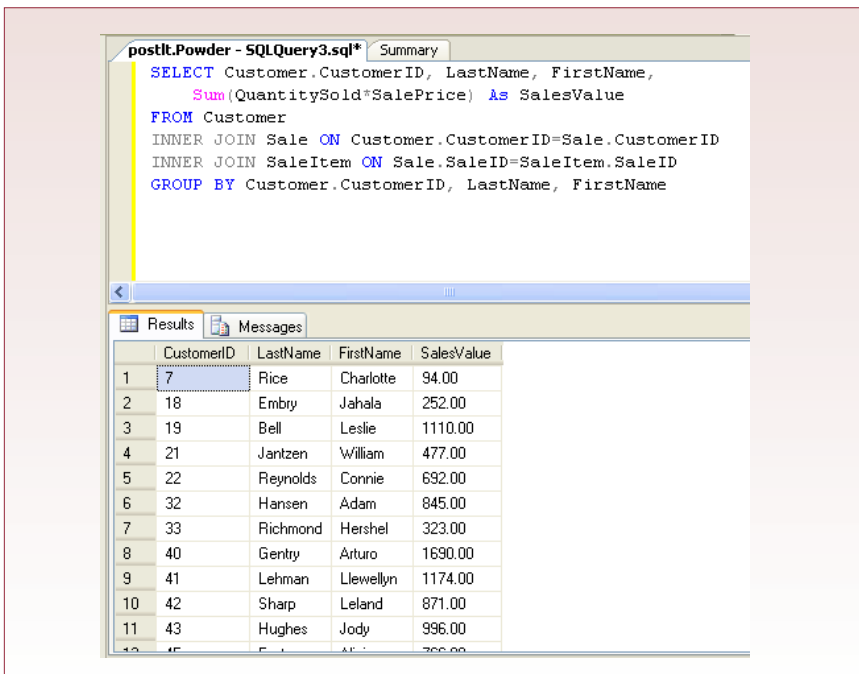
One of the greatest challenges with any database query is that most queries return values, but they might not be answers to the question you thought you were asking. You must learn to carefully build the queries and test each intermediate step so that you can be sure the final result is an accurate answer 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. However, eventually this list would be too long. Instead, she wants a list that displays the customers whose

Figure 5.2



The screenshot shows a SQL query window titled "postIt.Powder - SQLQuery3.sql" with a "Summary" tab. The query is as follows:

```
SELECT Customer.CustomerID, LastName, FirstName,
       Sum(QuantitySold*SalePrice) As SalesValue
FROM Customer
INNER JOIN Sale ON Customer.CustomerID=Sale.CustomerID
INNER JOIN SaleItem ON Sale.SaleID=SaleItem.SaleID
GROUP BY Customer.CustomerID, LastName, FirstName
```

Below the query, the "Results" tab displays a table with the following data:

	CustomerID	LastName	FirstName	SalesValue
1	7	Rice	Charlotte	94.00
2	18	Emby	Jahala	252.00
3	19	Bell	Leslie	1110.00
4	21	Jantzen	William	477.00
5	22	Reynolds	Connie	692.00
6	32	Hansen	Adam	845.00
7	33	Richmond	Hershel	323.00
8	40	Gentry	Arturo	1690.00
9	41	Lehman	Llewellyn	1174.00
10	42	Sharp	Leland	871.00
11	43	Hughes	Jody	996.00
12	45	F...	A...	300.00

total purchases are larger than the average number of purchases per customer. Although the business question is reasonable, this question is slightly tricky because you have to build the query in pieces.



Activity: Create a Subquery

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. Note that the examples in this chapter use the Query Analyzer instead of the View designer. Figure 5.2 shows the initial query that computes these subtotals. Of course, it lists the sales for every customer, and Katy only wants the sales of greater than average amount. But this query is an important step and needs to be saved as CustomerSales.

Action
 Create a new query.
 Tables: Customer, Sale, SaleItem.
 Columns: CustomerID, LastName, FirstName, Sum(QuantitySold*SalePrice) AS SalesValue
 Group By the other columns.
 Run the query.
 Save as a view CustomerSales.
 Create new query.
 Table: CustomerSales query.
 SELECT Avg(SalesValue) ...
 Run the query.

The next step is to use this first query to compute the average amount of sales for customers. This computation is straightforward. You 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 statement, 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 the customers with the largest total purchases are listed at the top. Also, always

Figure 5.3

Question	Compute average total purchases for customers.
SQL	SELECT Avg(SalesValue) As AvgOfSalesValue FROM CustomerSales;
<pre>AVGOFSALESVALUE ----- 942.1141</pre>	

Question	List customers who purchased more than the average customer.	
SQL	<pre>SELECT LastName, FirstName, SalesValue FROM CustomerSales WHERE SalesValue > (SELECT Avg(SalesValue) FROM CustomerSales) ORDER BY SalesValue DESC;</pre>	
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
McCartney	JoAnne	\$2,237.00
Crowe	Vicky	\$2,165.00
Ford	Katy	\$2,098.00
Cardwell	Christina	\$2,092.00
(only the top 17 are shown)		

Figure 5.4

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.



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 same. The effect of this join is that the results show the customers (ID

Action

Create a new query.

Table: CustomerSales query.

Columns: LastName, FirstName, SalesValue.

Criteria for SalesValue

>(SELECT Avg(SalesValue) FROM CustomerSales).

Action

Create a new query.

Tables: Rental and Sale.

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

Join the tables on CustomerID.

Run the query.

Add a join between the tables on Rental.
CustomerID=Sale.CustomerID

Run the query.

Question	List rental customers who also purchased items, at any time.		
SQL	SELECT RentDate, Rental.CustomerID, Sale.CustomerID, SaleDate FROM Rental INNER JOIN Sale ON Rental.CustomerID = Sale.CustomerID;		
RENTDATE	CUSTOMERID	CUSTOMERID	SALEDATE
-----	-----	-----	-----
2006-03-28	1535	1535	2006-01-25
2006-12-02	1455	1455	2006-09-17
2006-11-12	1642	1642	2006-12-25
2006-11-04	1186	1186	2006-04-30
2006-11-19	51	51	2006-10-28
2006-11-01	1602	1602	2006-11-01
2006-03-11	1452	1452	2006-08-30
2006-11-06	1455	1455	2006-09-17
2006-11-27	1645	1645	2006-03-12
2006-01-03	1992	1992	2006-06-28
2006-12-11	1861	1861	2006-06-19
504 rows			

Figure 5.5

only) who participated in a sale and a rental—at anytime.

If you want to know which customers made a purchase on the same day as the rental, 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 `CustomerID`s to be equal, which is not what Jim wants. Then connect `Rental` to `Sale` by `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 SQL Server 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.

Figure 5.7 also 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 participated in a rental. If you

Action

Create a new query.

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

LEFT JOIN Rental ON Sale.CustomerID
= Rental.CustomerID.

Columns: LastName, FirstName, and
CustomerID from Sale and Rental.

Run the query.

Question	List all customers who rented and did or did not make a purchase.		
SQL	<pre>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 ORDER BY LastName, FirstName;</pre>		
LASTNAME	FIRSTNAME	CUSTOMERID	CUSTOMERID
Abel	Marshall	1406	1406
Abel	Marshall	1406	1406
Abel	Melinda	1467	1467
Abrams	Marc	603	
Adkins	April	413	413
Adkins	Manuel	1499	1499
Aldrich	Jewell	142	142
Aldrich	Jewell	142	142
Aldrich	Jewell	142	142
Aldrich	Jewell	142	142
Alexander	Marvin	645	645
Allen	Laura	1085	
Allen	Orson	1928	1928
Allen	Orson	1928	1928
Baez	Agnes	302	
Bailey	Takao	879	879
Baker	Hazel	1664	1664

Figure 5.6

want to see only this list of people, 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

The screenshot shows a SQL query editor window titled "postIt.Powder - SQLQuery3.sql*" with a "Summary" tab. The query is as follows:

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

Below the query editor, the "Results" tab is active, displaying a table with the following data:

	RentDate	CustomerID	CustomerID	SaleDate
1	2006-01-07 00:00:00.000	1291	1291	2006-01-07 00:00:00.000
2	2006-11-09 00:00:00.000	930	930	2006-11-09 00:00:00.000
3	2006-11-10 00:00:00.000	1629	1629	2006-11-10 00:00:00.000

Question	List customers who bought items but never rented.	
SQL	<pre>SELECT LastName, FirstName, Customer.CustomerID FROM Customer INNER JOIN Sale ON Customer.CustomerID = Sale.CustomerID WHERE Customer.CustomerID NOT IN (SELECT CustomerID FROM Rental) ORDER BY LastName, FirstName;</pre>	
LASTNAME	FIRSTNAME	CUSTOMERID
-----	-----	-----
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
Cardwll	Christina	1377
Carrow	Keith	1403
Carter	Ruth	757
188 rows		

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

reason is because an inner join automatically leaves out the customers you are searching for. This question can also be answered with a subquery. Figure 5.8 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.

Action

Create a new query.

Tables: Customer and Sale.

Columns: LastName, FirstName, and CustomerID.

WHERE CustomerID Not In (SELECT CustomerID FROM Rental).

Run the query.



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

Question	Compute quantity on hand by ModelID.
SQL	<pre>SELECT ModelID, Sum(QuantityOnHand) As QOH FROM Inventory GROUP BY ModelID ORDER BY Sum(QuantityOnHand) DESC;</pre>
MODELID	QOH
-----	-----
QDV-720	70
YCG-584	70
YXY-385	70
LDK-181	60
YKU-321	60
MHQ-568	60
NTE-526	50
QDG-75	50
NCS-293	40
CCS-411	35

Figure 5.9

descending order by ModelID. Figure 5.9 shows the total quantity on hand for the various models. Save the query as ModelsOnHand.

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: SalesCategory(CategoryID, CategoryName, LowLimit, HighLimit). If the QOH for a model is greater than or equal to the LowLimit and less than the HighLimit,

Action

Create a new query.

Table: Inventory.

Columns: ModelID and Sum(QuantityOnHand).

Sort by the Sum descending.

Run the query.

Save it as ModelsOnHand.

Create a new table: SalesCategory.

Columns: CategoryID, CategoryName, LowLimit, HighLimit.

Enter data from Figure 5.10.

Figure 5.10

Question	Data for the new SalesCategory table.		
SQL	<pre>INSERT INTO SalesCategory VALUES (1, 'Hot', 0, 6); INSERT INTO SalesCategory VALUES (2, 'Good', 6, 10); INSERT INTO SalesCategory VALUES (3, 'OK', 10, 20); INSERT INTO SalesCategory VALUES (4, 'Weak', 20, 40); INSERT INTO SalesCategory VALUES (5, 'Slow', 40, 1000); SELECT * FROM SalesCategory;</pre>		
CATEGORYID	CATEGORYNAME	LOWLIMIT	HIGHLIMIT
-----	-----	-----	-----
1	Hot	0	6
2	Good	6	10
3	OK	10	20
4	Weak	20	40
5	Slow	40	1000

Question	Models categorized based on quantity on hand.		
SQL	<pre>SELECT ModelID, QOH, SalesCategory.CategoryID, CategoryName FROM ModelsOnHand INNER JOIN SalesCategory ON (ModelsOnHand.SumOfQuantityOnHand >= SalesCategory. LowLimit) AND (ModelsOnHand.SumOfQuantityOnHand < SalesCategory. HighLimit);</pre>		
MODELID	QOH	CATEGORYID	CATEGORYNAME
ENW-975	2	1	Hot
XUW-452	2	1	Hot
WER-904	2	1	Hot
PKT-115	2	1	Hot
KSB-825	2	1	Hot
EZX-852	2	1	Hot
IQE-600	2	1	Hot
LNH-128	2	1	Hot
RFL-870	2	1	Hot
SXT-833	2	1	Hot
BWE-236	3	1	Hot
CFO-752	3	1	Hot
ERK-571	3	1	Hot
HVD-690	3	1	Hot
<i>(many rows not shown)</i>			

Figure 5.11

it falls into the specified category. The CategoryID ensures a unique key and could be used to sort the rows if necessary. Figure 5.10 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 QOH along with the CategoryName. Note the SQL JOIN statement carefully, and be sure to match the Figure 5.11 inequality join statement. Figure 5.11 also shows the

Figure 5.12

The screenshot shows a SQL query editor window titled "postlt.Powder - SQLQuery3.sql*" with a "Summary" tab. The query is as follows:

```
SELECT SalesCategory.CategoryID, CategoryName,
       Count (ModelID) As CountModels
FROM ModelsOnHand
INNER JOIN SalesCategory
ON (ModelsOnHand.QOH >= SalesCategory.LowLimit)
AND (ModelsOnHand.QOH < SalesCategory.HighLimit)
GROUP BY SalesCategory.CategoryID, CategoryName
```

Below the query editor, the "Results" tab is active, displaying a table with the following data:

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

A callout box with a red border and yellow background points to the "CountModels" column in the results table, containing the text: "Find the number of models in each sales category".

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 5.12 shows the basic query. It is built using the results of the previous query. This query hides the complicated details and 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. However, the category definitions might not be accurate, but Jim can quickly alter the range numbers and rerun the query to see the results.

Action

Create a new query.

Columns: ModelID, QOH, CategoryID, and CategoryName.

Tables: ModelsOnHand and SalesCategory.

Add the inequality join.

Run the query.



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 sentence 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.

Action

Create a new query.

Columns: CustomerID, LastName, FirstName, and SaleDate.

Tables: Customer and Sale.

Set January sale date in WHERE.

Copy the entire statement.

Add the word Union.

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

Run the query.

As shown in Figure 5.13, create a new query using the Customer and Sale tables. Display the CustomerID, LastName, and FirstName columns. If you are using the view designer instead of the Query Analyzer, 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, copy the entire statement without the semicolon. 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 (January) SELECT statement, 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 the query, then you can build a second query based on the first and sort the columns as needed.

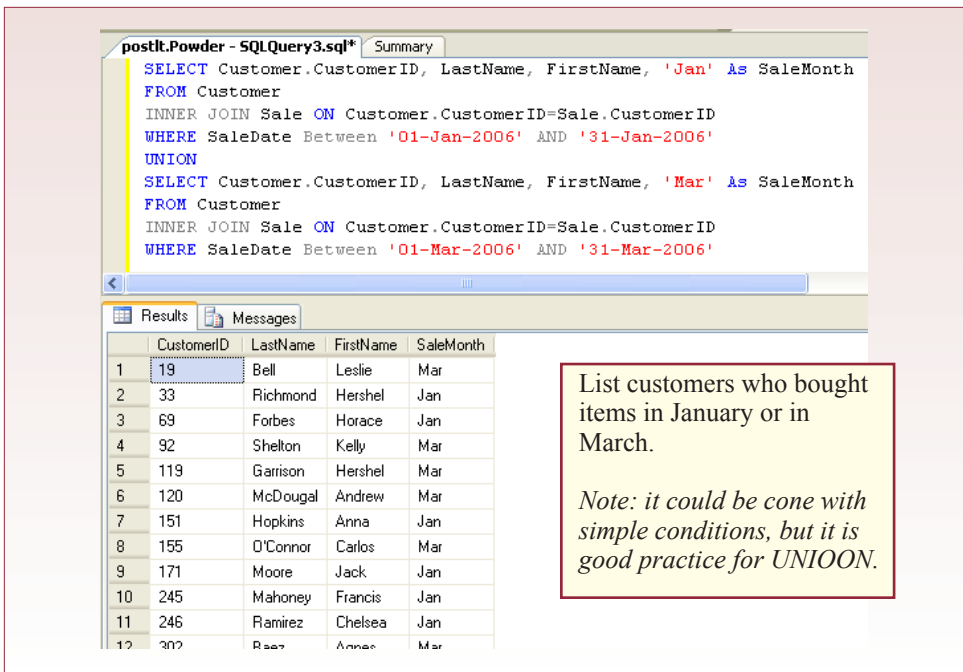


Figure 5.13

SQL Data Definition and Data Manipulation



Activity: Create Tables

Although it is possible to create and delete tables in SQL Server using the enterprise manager, you will often have to create a table using the data definition language (DDL) CREATE TABLE command. For example, after working with the database for a while, you realize that it would be useful 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 5.14 shows the CREATE TABLE command for the new Contacts table. Essentially, you list each desired column along with its data type. Note that by default SQL Server does not allow columns to contain null values. To enable a column to be empty, you need to add the “null” keyword to the column.

Enter the SQL code in the Query Analyzer. Run the query and you should receive the “Table created” message. If not, check your typing carefully. You should create the primary key constraint to indicate the ContactID is the sole primary key column. For other tables, 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. Be sure to specify the ON DELETE CASCADE option for the foreign key. If rows are deleted in the master table (Manufacturer), then any contacts in this table associated with that manufacturer will also be deleted automatically.

Action

Create a new query.

Enter the CREATE TABLE command.

Run the query.

```

CREATE TABLE Contacts
(
    ContactID          int,
    ManufacturerID    int,
    LastName           nvarchar(25),
    FirstName          nvarchar(25),
    Phone              nvarchar (15),
    Email              nvarchar (120),
    CONSTRAINT pk_Contacts PRIMARY KEY (ContactID),
    CONSTRAINT fk_ContactsManufacturer FOREIGN KEY (ManufacturerID)
        REFERENCES Manufacturer(ManufacturerID)
    ON DELETE CASCADE
);

```

Figure 5.14

Generally, with SQL Server it is easier to create tables with SQL. It is particularly useful to create a text file that contains several CREATE statements that will generate the database automatically. First, you want to test each statement individually and make sure it contains the correct statement. Then cut and paste the command into a separate text file. This file can be given to others to create the database on a different system. The CREATE TABLE command is also useful for creating temporary tables. Figure 5.15 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 if you work from a

Figure 5.15

```

CREATE TABLE MyTemp
(
    ID          int,
    LName      nvarchar(25),
    FName      nvarchar(25),
    CONSTRAINT pk_MyTemp PRIMARY KEY (ID)
);

```

good design. You can also use the Enterprise Manager console to add columns to a table—and it will show you the full syntax of the SQL command. For example, to add a TempCost column to the ItemModel table, the command would be ALTER TABLE ItemModel ADD (TempCost money).



Activity: Insert, Update, and Delete Data

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 5.16 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

Action

Create a new query.

Type the INSERT command: INSERT INTO Customer (CustomerID, LastName, FirstName, City, Gender) VALUES (4000, 'Jones', 'Jack', 'Nowhere', 'Male');

Run the query.

```
INSERT INTO Customer (CustomerID, LastName, FirstName, City, Gender)
VALUES (4000, 'Jones', 'Jack', 'Nowhere', 'Male');
```

Figure 5.16

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. Notice that you must include the CustomerID column at this point. Chapter 7 will explain how to create an identity number so this value can be generated automatically.

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 5.17 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. Be sure to issue a COMMIT command after any INSERT command to ensure changes are saved to the table.

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 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.

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 2005 snowboards. 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 data for the 2004 boards. You should run the query to ensure that it returns exactly the data that you want to update. Next, as shown in Figure 5.18, edit the query so that it uses

Action

Create a new query.

Type the INSERT command: INSERT INTO Customer (CustomerID, LastName, FirstName, City, Gender) VALUES (4000, 'Jones', 'Jack', 'Nowhere', 'Male');

Run the query.

Figure 5.17

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

Question	Increase the cost of snow boards by four percent for 2004.
SQL	<pre>SELECT Category, ModelYear, Round(Cost*1.04,2) FROM ItemModel WHERE Category='Board' AND ModelYear=2006 ; -- run the top query first, then edit it to make the actual changes UPDATE ItemModel SET Cost = Round(Cost*1.04,2) WHERE Category='Board' AND ModelYear=2006;</pre>
88 rows updated.	

Figure 5.18

the UPDATE command instead of SELECT. The Round function is used to ensure that the final Cost value is rounded off to cents instead of extended fractions. Be sure you run the SELECT query first to ensure the correct rows are selected by the WHERE clause. Then edit the query by adding the UPDATE statement. In practice, do not try to run both queries at the same time. They are shown here only so you can compare the two. After you run an UPDATE query, you should issue a COMMIT command to make sure the changes are recorded to the table.

Notice that the SQL statement is straightforward. It is also easy to change multiple columns at one time. Just separate the column assignments with commas. For example: SET Cost = Round(Cost * 1.04,2), ModelYear = 2005.

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 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

Action

Create a new query.

Columns: Category, ModelYear, and Round(Cost*1.04,2).

Table: ItemModel.

Criteria: Category='Board' And ModelYear=2006.

Run the query.

Change the first two lines to:

```
UPDATE ItemModel
```

```
SET Cost = Round(Cost*1.04,2).
```

Run the query.

Figure 5.19

Question	Delete sample row from the MyTemp table.
SQL	<pre>-- SELECT * DELETE FROM MyTemp WHERE ID > 100; Commit;</pre>
<pre>----- ID LNAME ----- FNAME ----- 1184 Cherry Louis 1 row deleted. Commit;</pre>	

```
DROP TABLE MyTemp;
```

Figure 5.20

you attempt major delete operations. You might want to use the Enterprise Manager/Management/Backup option to copy the data before making major deletions.

To be particularly safe, this example is just going to delete data from the temporary table that was created in the previous section. Create a new query using the MyTemp table. As shown in Figure 5.19, to see the rows you are going to delete, display the ID, LName, and FName columns and set a condition to show only rows with an ID > 100. Run the query to verify that it returns only one row. Now, edit the query and replace the entire SELECT row with the DELETE command. Run the query.

In practice, it is best to stick with simple WHERE clauses when possible. However, it can be complex and can 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 5.20, 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 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.



Activity: Create Parameter Queries

Parameter queries are useful when you need to create a complex 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 5.21 shows a query that displays the total rental income by Category for a specified range of dates. This query has fixed dates for the first quarter. The objective is to replace those fixed dates with parameters that can be entered quickly by the manager—preferably without having to see or edit the query.

Action

Create a new query: SELECT * FROM MyTemp WHERE ID > 100;

Test the query.

Change the SELECT row to DELETE.

Run the query.

Run a commit; command.

Action

Create a new query.

Columns: Category, Sum(RentFee).

Tables: Rental, RentItem, Inventory, and ItemModel.

GROUP BY Category.

Test the query.

Question	Show rental totals by category for a specified time period.
SQL	<pre>SELECT Category, Sum(RentFee) AS SumOfRentFee FROM Rental INNER JOIN RentItem ON Rental.RentID=RentItem.RentID INNER JOIN Inventory ON RentItem.SKU=Inventory.SKU INNER JOIN ItemModel ON Inventory.ModelID=ItemModel.ModelID WHERE RentDate Between '01-Jan-2006' And '31-Mar-2006' GROUP BY Category;</pre>
CATEGORY	SUMOFRENTFEE
-----	-----
Board	18660
Boots	14370
Electronic	1350
Poles	790
Rack	420
Ski	37900

Figure 5.21

In SQL Server, parameterized queries are straightforward, once you understand that they are created as stored procedures. Stored procedures can hold complex code, and are explained in more detail in Chapter 7. Fortunately, parameterized queries are easy enough to explain here.

Essentially, you just create a procedure, give it a name, and specify the parameters and the associated data types. Figure 5.22 shows that next you add the SELECT statement and insert the parameters. The commands are also stored as a text file on the student CD so you can cut and paste them to save some typing. The heart of the procedure is the query that you already created. The only difference is that the two dates are specified as parameters. The procedure only has to be created one time.

Once the procedure is defined, the manager only needs to issue one command to enter new dates and obtain the total rental fees by category. Figure 5.23 shows the command. Notice that you can specify the parameter values by giving them a name when you execute the procedure. Although this approach requires more

Action

- Create a new query.
- Create the stored procedure.
- Execute the new procedure.
- Change the dates to Oct-Dec.
- Run the query.

Figure 5.22

```
CREATE PROCEDURE GetCategoryFees
    @StartDate datetime,
    @EndDate datetime
AS
SELECT Category, Sum(RentFee) AS SumOfRentFee
FROM Rental INNER JOIN RentItem
    ON Rental.RentID=RentItem.RentID
INNER JOIN Inventory
    ON RentItem.SKU=Inventory.SKU
INNER JOIN ItemModel
    ON Inventory.ModelID=ItemModel.ModelID
WHERE RentDate Between @StartDate And @EndDate
GROUP BY Category
GO
```

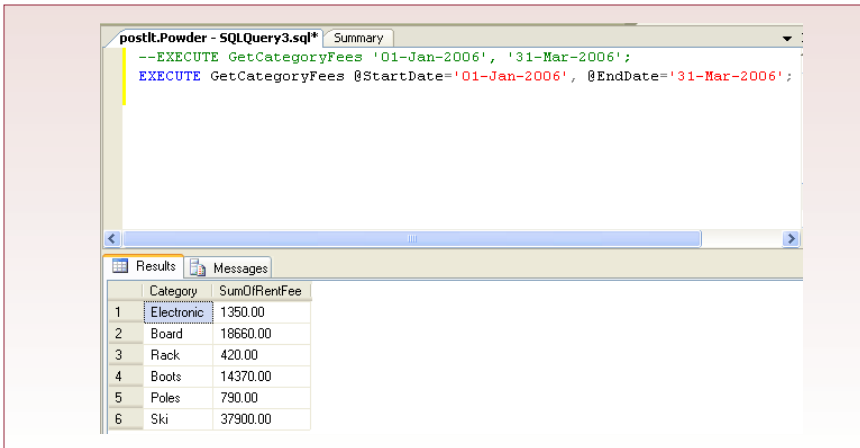


Figure 5.23

typing, it ensures that the values are assigned to the correct parameters. Passing parameters based on position is easier to type, but can lead to errors.

You can build complex queries and insert parameters to request specific data from the person running the query. Although it requires several steps, query parameters are a useful method to quickly build queries that users can control without having to alter the query.

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 of the 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 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, weak). 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 expensiveness of cleanups. For example, spills that cost more than \$1 million to clean up are expensive; splits that cost \$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-wise 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, perform the following tasks. 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.

Forms and Reports

Chapter Outline

Forms and Reports, 88

Case: All Powder Board and Ski Shop, 89

Lab Exercise, 90

All Powder Board and Ski Shop Forms, 90

All Powder Basic Reports, 109

Exercises, 116

Final Project, 117

Objectives

- Create forms that make it easy for users to enter data.
- Create three types of forms (main, grid, subform) to understand the purpose of each.
- Create reports to display and summarize data.

Forms and Reports

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. SQL Server does not have a forms-builder tool. Instead, you create forms and applications using Visual Studio (VS). Visual Studio is a development environment that supports several programming languages. The framework contains tools that make it relatively easy to connect to databases and build forms. You can also create forms within Visual Studio, or you can use the report server.

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 Sale 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.

A powerful capability of Visual Studio is the ability to create standalone executable programs. Many commercial applications are built with these tools. This power comes at a price—the system contains thousands of options and takes a while to learn. On the other hand, you can start with the simple wizards and let them do most of the work until you learn to use the detailed language and framework features. One of the first questions you will have to answer to use the system is whether you want the forms to run as a Windows application or an Internet application that requires a Web server. The Web versions of forms are more difficult to create, and the options are different from those in Windows forms. You will also have to choose a language—the most common choices are Visual Basic and C#. Fortunately, the underlying programming model is the same for all languages—the main differences are in syntax, so it is relatively easy to convert from one language to another. The examples in this chapter are created in Windows using Visual Basic.

Figure 6.1

Ski Board Style

Customer

Last Name

First Name

Phone

Address

City

Style	Style Description	Category

Grid

Main

Sale

Customer		Salesperson		
ItemID	Description	Price	Quantity	Value

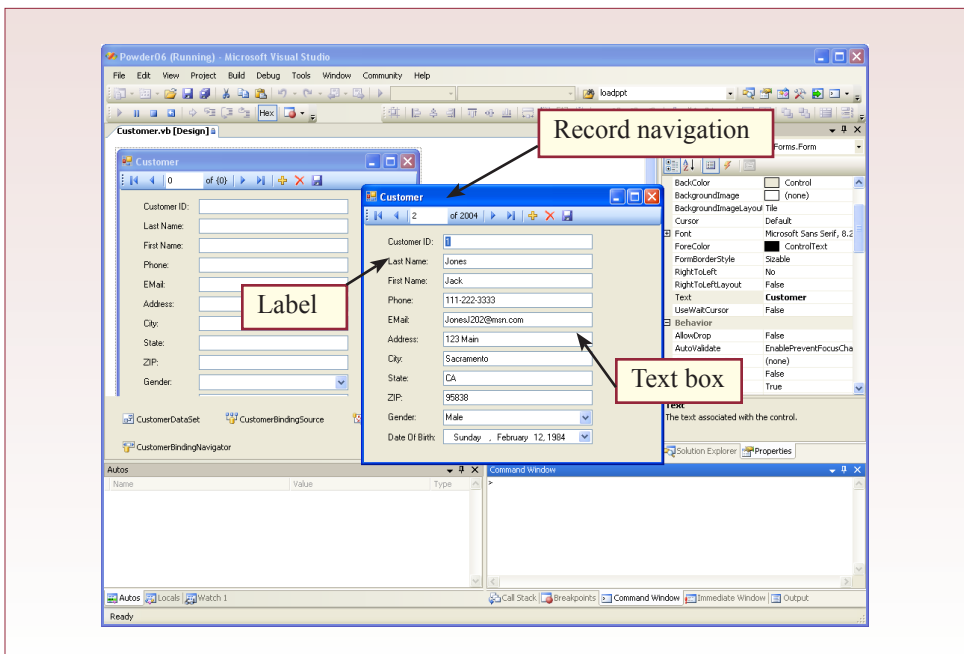
Main and Sub-form

If you are running Windows Vista or Server 2008, you need to follow a couple of additional steps when installing forms and reports. Several of the security provisions introduced with those operating systems cause problems. First, use Visual Studio 2008 instead of 2005. You can use VS 2005 for some things but it must be started with run as administrator. Second, you can build reports with Visual Studio, but to deploy them you need to install and run the SQL Server Reporting Services. This option is available even with the Express version. Again, use the 2008 version with Vista and Server 2008. Also, you will have to add reporting services administrator roles to your login account. To add these roles, start Internet Explorer using run as administrator. Connect to your reporting services, probably using <http://localhost/reports>. If asked, login as the administrator. Click the Properties tab, then New Role Assignment. Enter your login name and give yourself the Content Manager role. Click OK to return to the main screen. Click the Site Settings link on the top-right, then the Security link on the left. Again, click the New Role Assignment, enter your login name, and give yourself the System Administrator role. Click OK and exit. You will now be able to connect to the Reporting Services using your personal account.

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 grid form appears similar to the Table view in that it

Figure 6.2



shows several 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. Fortunately, the Visual Studio wizards make it relatively easy to create main forms.



Activity: Create Basic Main Forms

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

Action

Start Visual Studio and create a new Windows-based Project using Visual Basic.

Right-click the Form1.vb entry and rename it to Customer.vb.

Create a new Data Source.

If necessary, add a new data connection.

Save your login credentials.

Pick the Customer table and click the Finish button.

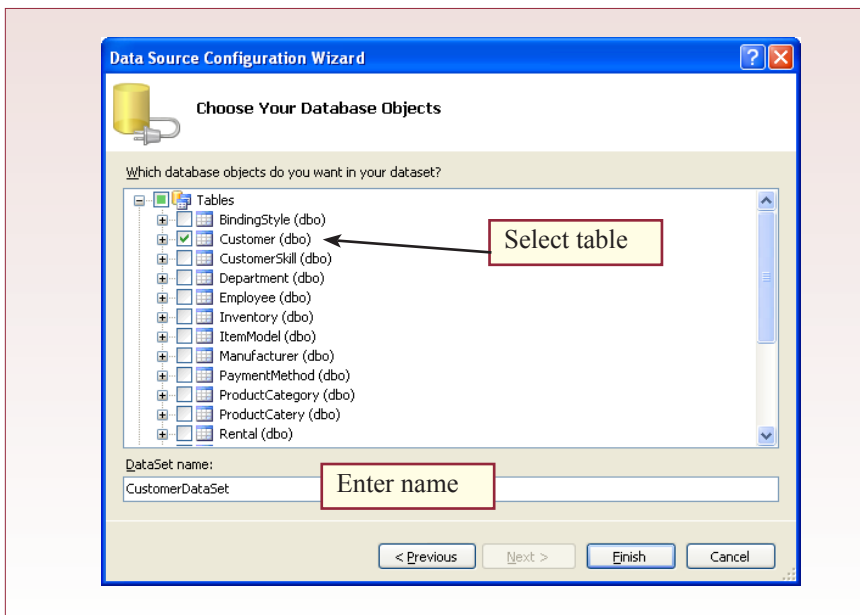
In Data Sources, change the Customer table type to Details.

Drag the Customer table onto the form.

Save everything.

Run the form to test it.

Figure 6.3



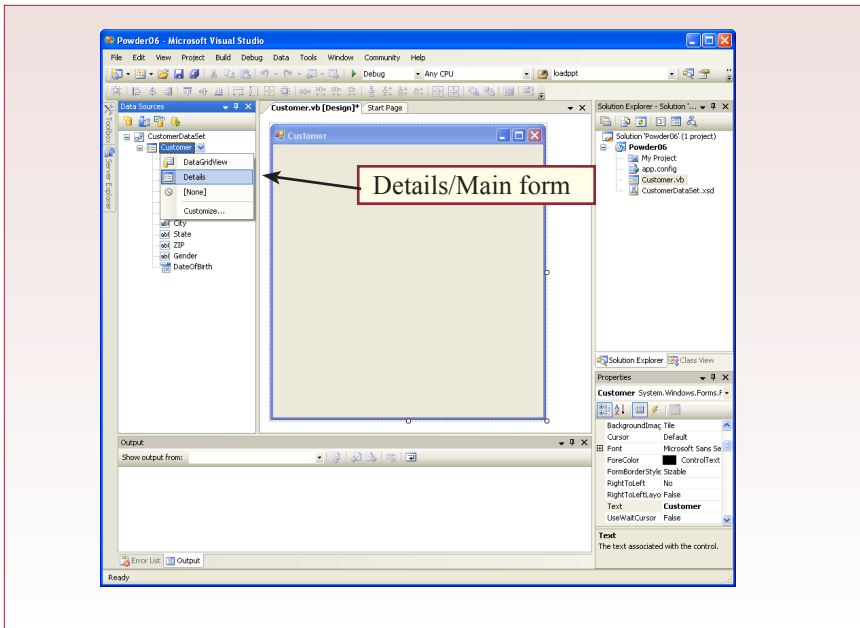


Figure 6.4

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 descriptors such as size, position, and color. You can add new controls to display images or include buttons to delete or find records.

Begin by creating a new project. Select the Windows and Visual Basic options. The system will automatically create a blank form (Form1). Rename the form by right-clicking on its name in the Solution Explorer and choosing the Rename option. Enter `Customer.vb` to help you remember the purpose of the form. You should also click on the form in the design window so you can change its properties. Find the Text property in the Properties window and change it from Form 1 to Customer. You should drag the bottom-right corner to enlarge the form so you have room to work.

In Visual Studio 2005, forms are connected to the database through a Data Source. The Data Source is a Microsoft object that contains SQL statements to retrieve and edit data. Any time you want a form to retrieve data, you need to create a new Data Source that specifies the exact table and columns you need. One of the most important elements of a Data Source is the Data Connection. The Data Connection is usually stored as a string that specifies exactly how the form will connect to your database. Typically, you will need to create one database connection for a project. This connection string is automatically saved with the project, and you can reuse it for all of your forms.

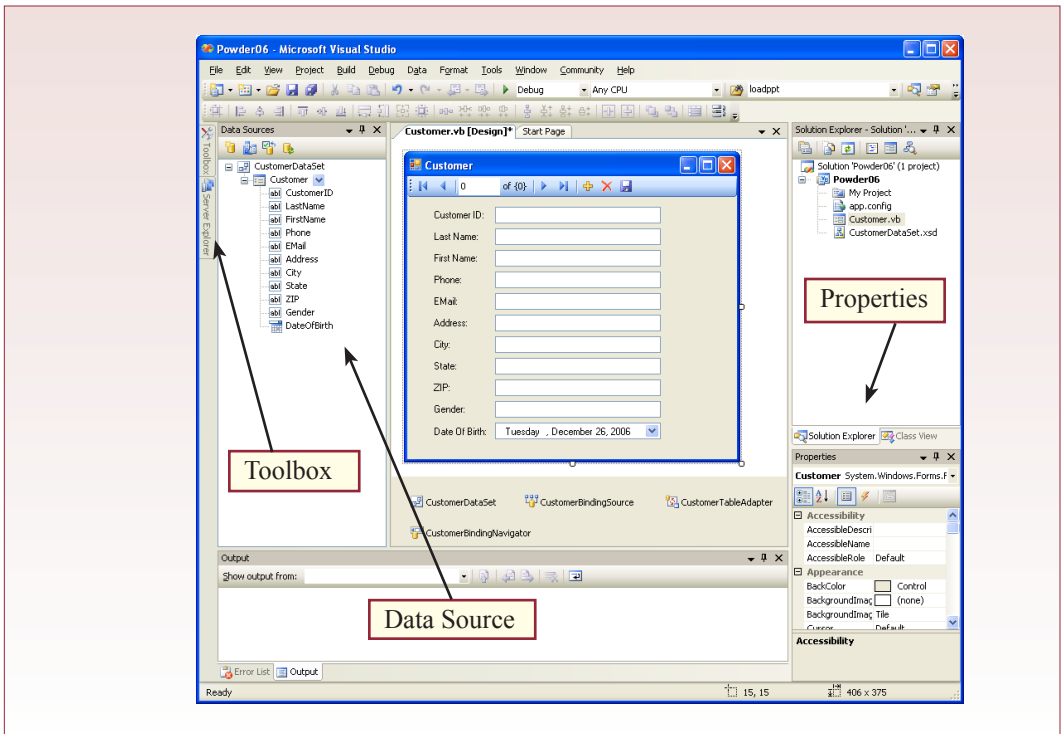
If necessary, open the Data Sources Window (Data>Show Data Sources). Click the link to Add New Data Source. The simplest approach is to connect directly to the database, so pick that option. Visual Studio supports more complex projects with a Web Service as the data source, or you can create a custom program object to write your own code to retrieve and manipulate data. Pick the database option.

You will probably have to create a new database connection, so click the New Connection button. If necessary, click the Change button to select Microsoft SQL Server as the type of database (instead of Access). At this point, you will see a version of the familiar login form. Enter or choose the name of your database server. You can use Windows Authentication or SQL Server Authentication. Typically, learning environments use SQL Server Authentication, but corporate applications might rely on Windows Authentication to handle all logins. For the SQL Server Authentication, enter your user name and password. Check the box to save your password. Select the database you have been assigned to use, and click the Test Connection button to verify that all data is entered correctly. This approach is the simplest for building sample forms and reports. However, your login information will be stored in the application. Any user that runs your application later will use these login credentials. For final projects, you might create a special SQL Server login for the application, or rely on Windows Authentication to provide separate access permissions to each user.

After closing the New Connection wizard, the login information is saved within the project. The Data Source Configuration Wizard asks if you want to save the password information in the connection string. For learning and development, it is much simpler to save this information in the connection string so you are not required to log in every time you test a form. You can display the full Connection String if you want to see what it looks like. This string is stored in the project's app.config file, which you can edit later if need to remove or change the password.

Finally, as shown in Figure 6.3, you can choose the Customer table as the source of the data for the form. In future forms, you will save time by selecting the existing data connection and jump right to choosing the table. Be sure to enter

Figure 6.5



a useful name, such as CustomerDataSet. When you have multiple sources of data on a form, you will need to be able to identify which one you need, so pick a descriptive name.

The CustomerDataSet with the Customer table is now displayed in the Data Sources window. Click the plus sign in front of the Customer table to see the list of columns in the table. Notice that the DateOfBirth column will automatically use the calendar date picker object. The others are set as simple text boxes. You can select any column and override the type of data control by selecting one from the drop-down-list. For these columns, stick with the default choices. Before creating the form, you need to do one more step: As shown in Figure 6.4, you have to tell Visual Studio that you want to use a main (details) form instead of a grid form. Select the Customer table in the list and open the drop-down-list. Pick the Details control instead of the grid.

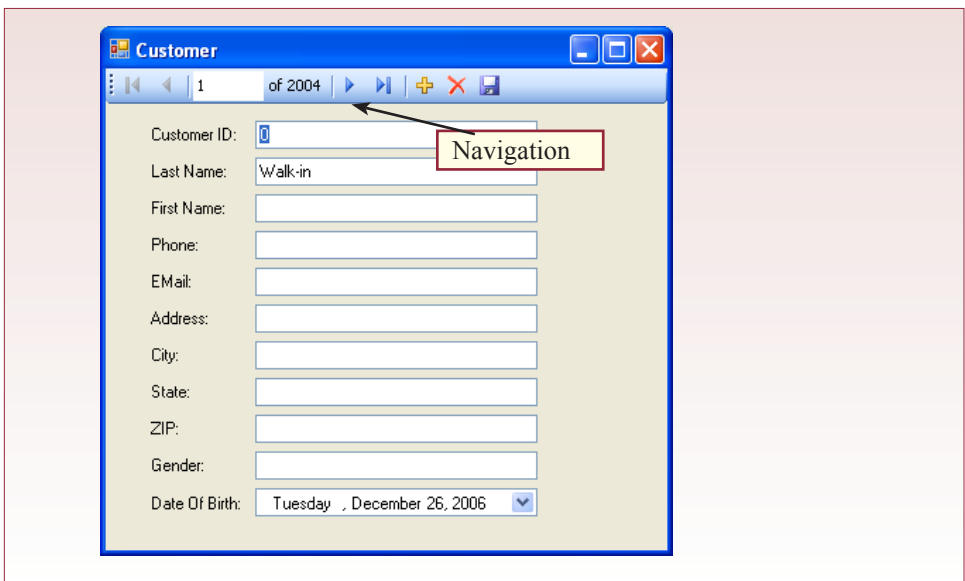
To create the Customer form, click the Customer table icon and drag-and-drop it onto the body of the form. Visual Studio will automatically add the data objects and controls to the form. Figure 6.5 shows the resulting design view of the form. You can drag the labels and text boxes as a group and resize the form if you need to create a better-balanced display.

The wizard will build the basic form and place you in Design view. The layout usually needs considerable work. Before you make too many changes, you should think about all of the forms you will need and develop a common design. You have total control over the form, including colors, font sizes, and layout. A profes-

Action

- Delete the text box for Gender.
- Drag a combo box from the ToolBox.
- Expand the Property window.
- Name it GenderComobBox.
- Open the Items property and enter Female, Male, and Unidentified.
- Expand the DataBindings section.
- For SelectedItem and SelectedValue properties select CustomerBindingSource – Gender.
- Run the form and test the combo box.

Figure 6.6



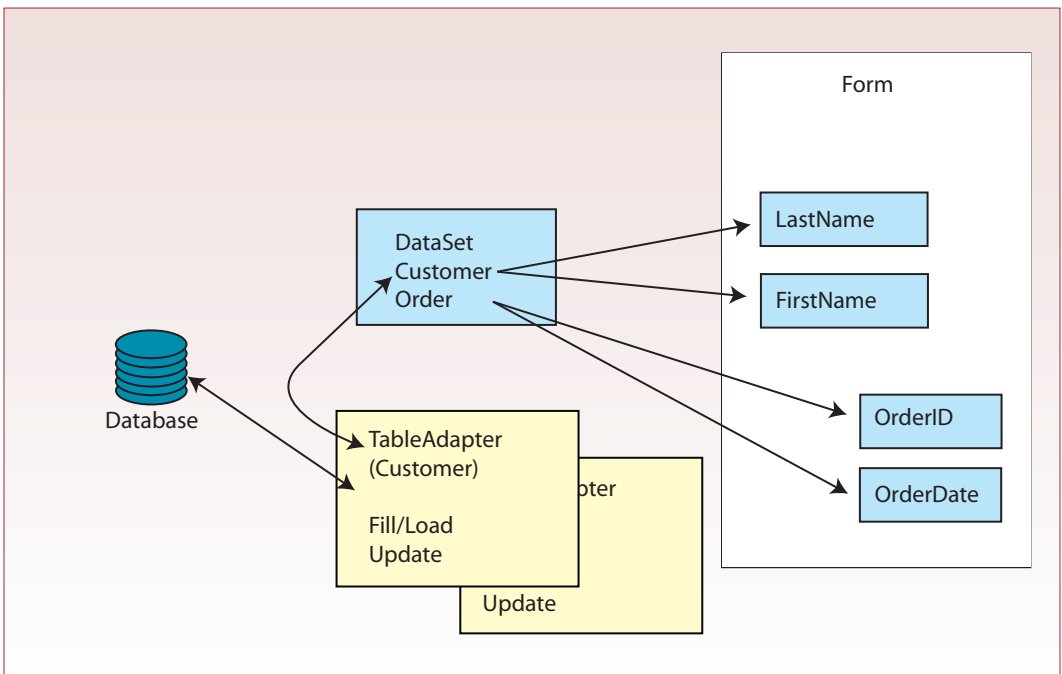
sional application will have a standard design so that all of the forms look alike and behave the same way.

The form is ready to run. Click the Start Debugging button (or F5) and the application will compile and run the form as a standalone application. Figure 6.6 shows the form in operation. Notice the navigation bar contains buttons to make it easy to scroll through records. It also includes buttons to insert or delete rows and save changes. These items were added automatically when you dragged the Customer table onto the form. Close the Customer form to stop the debugger.

The wizard manages to display all of the data and provides the navigation buttons to move between rows. However, you will want to modify the design of the form. Sometimes you simply need to change the layout, formatting, and colors. Other times you want to add buttons to open additional forms or reports or to add or delete data rows. As shown in Figure 6.5 you can use properties to set the details of the form and its controls. Click a control to select it and you can change its display 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. For example, you should add a label control and enter a title for each form.

The controls on the form are straightforward—the text boxes are used to display and edit the data. However, the process that Visual Studio uses to handle the data is somewhat complex. The main thing to understand is that data from the database is copied into an in-memory dataset. As shown in Figure 6.7, this DataSet contains tables (and relationships) similar to those in the DBMS. But, the data is held in RAM as a copy. A Table Adapter handles the transfer of data between the DBMS and the dataset. It contains SQL commands that can retrieve (SELECT), delete, or update the data. Be grateful, the wizard wrote all of those commands for you. You can right-click the CustomerTableAdapter on the form and

Figure 6.7



choose the Edit Queries option to see the SQL statements. One consequence of the DataSet approach is that your application stores the data temporarily in memory. When users insert data or make changes, these changes are applied to the in-memory copy. The data is only written back to the database

when the user clicks the Save button on the form. This step is slightly different from earlier database applications, so you might have to explain it to users. On the other hand, most users are familiar with this approach based on the way that word processors and spreadsheets handle data.

To see the cool standalone projects that are created, save and rebuild the project. Then, through Windows/My Computer, navigate to the project folder and open the bin directory. You can run the project executable (.exe) file directly and use the forms as a Windows application. Visual Studio also contains extensive debugging features. Back in the editor, you can place a breakpoint on one of the lines—try one in the initialization or update routines. Then run the project in debug mode. When the breakpoint is hit, execution stops and you can single-step through the code, look at variable values, and even test new lines of code.

One more option should be added to the form. Notice the gender text box. Remember that users should only be able to enter one of three values (Male, Female, or Unidentified). But, looking at the form, how does the user know those are the only three legitimate values? To improve usability and reduce errors, you need to make that control a combo box so users can simply pick from the list.

Action

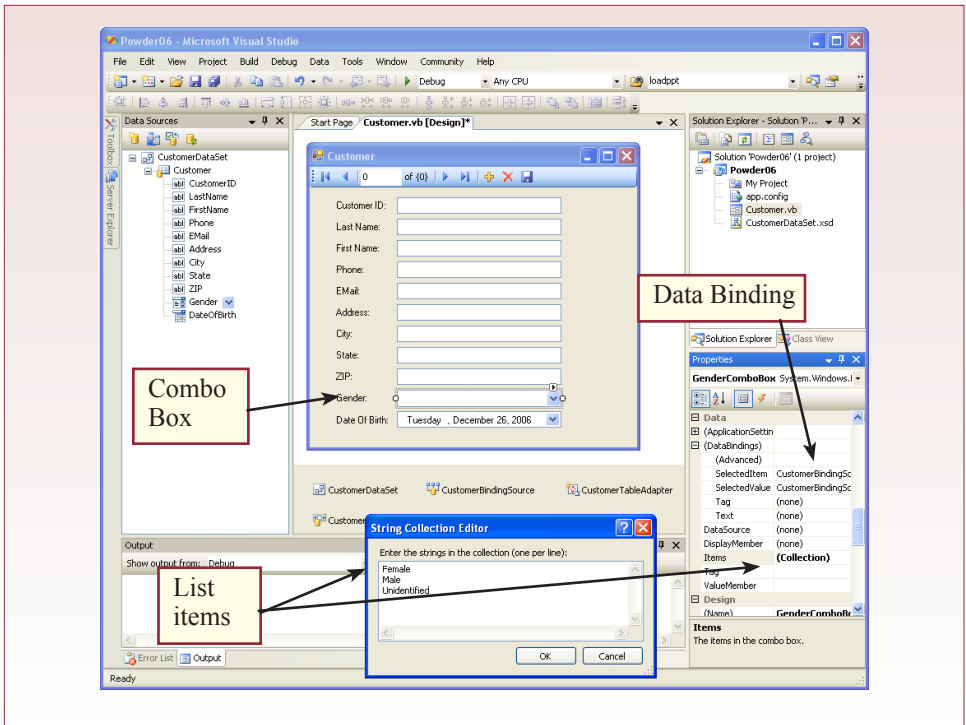
Add a new blank form with Add/New Item.

Name it Menu.vb.

Add a button to open the Customer form.

Test it and save it.

Figure 6.8



The first step is to simply delete the existing text box. For more complex cases, you might want to just drag it out of the way temporarily and delete it after you have built and tested the combo box. Second, drag a combo box from the ToolBox onto the form. If you had planned ahead, you could have converted the Gender column to a Combo Box before you dragged it onto the form, but you still have to perform the next steps.

As shown in Figure 6.8, the details are handled in the Property window for the new combo box. First, set the Name property to GenderComboBox so you can identify it later. Since you have a fixed list of values, you will type them in. In more complex cases, you will want to pull the list from a database table, and you can use the smart tag arrow for the Combo Box. Click the Items property and then the ellipses (...) button to open the Items window. Type in the three choices—one on each line—and close the window. This list will be displayed to the users, but at the moment, it is not connected to the dataset.

The big step in any database form is to ensure that the control is bound to the proper column in the dataset. Expand the Data Bindings property by clicking the small plus sign (+) in the property window. Since there is only one column (Gender), you want to set both the SelectedItem and SelectedValue to the same Gender column. Click the drop-down arrow for SelectedItem and expand the CustomerBindingSource, which is basically the DataSource. Select the Gender column as the binding source. Select the same column for the SelectedValue entry. Save everything, compile and test the form. See how much easier it is for the user. Whenever possible, you should use Combo boxes or lists to make it easy for users to enter the proper values.

Before finishing, you need to do one more important step. Try using the form and pressing the Tab key, or hit the Enter key in each control. Most likely, the cursor jumps all over the form page. It should follow a nice predictably logical pattern. You control this flow by assigning a sequence value to each control's Tab Index property. Use View/Tab Order and click each control in the proper sequence. Select View/Tab Order to accept the new values. You will have to renumber most of the controls. This step is easy to forget but critical to users. Add it to your checklist of things that must be done before a form is finished.

Before creating new forms, you need to build a menu form. When you run an application, one form is displayed at startup. Currently, you have only the one Customer form, so it is displayed. You will want to test new forms that you create, and ultimately, users will need a method to access those forms. The most common approach is to create a startup menu form. This form is explored in more detail in Chapter 8, which explores applications. However, creating a simple menu form now will make it easier to test all of the forms that you build.

Right-click the project name in the Solution Explorer and choose Add/New Item. Pick the Windows form type and change the name to Menu.vb. Drag a button from the Toolbox onto the form. Name it CustomerButton and set its text property to Customer. Double-click the new button on the design screen to open its code window. Enter the code to create a Customer form and display it:

```
Dim CustomerForm As Customer = New Customer()  
CustomerForm.Show( )
```

Save everything. You need to set this form as the one to be opened when the application starts. Right-click the project name in the Solution Explorer and choose Properties to open the property-edit window. Change the entry for Startup form

from Customer to Menu. Click the Save button and close the property-edit window. Run the form to ensure the Menu form starts. Click the Customer button to open the Customer form.



Activity: Create Grid Forms

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—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 6.9 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 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.

Creating a basic grid form follows the same basic steps as creating a main form, with one minor change. Begin by adding a new form. Right-click the project name, and choose Add/New Item. Be sure Windows form is selected and set the name to SkiBoardStyle.vb.

If necessary, open the Data Sources window. Click the Add New Data Source button. Pick the Database option. Stick with the existing connection string that you created when you built the Customer Data Source. Expand the Tables list and select the SkiBoardStyle table. Be sure to set the name to something that describes

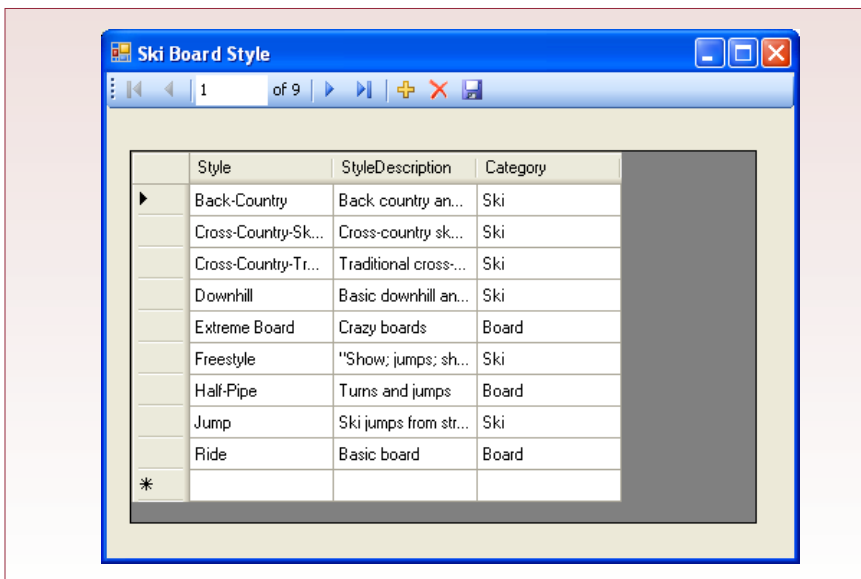
Action

Add a new form (SkiBoardStyle).

Add a new Data Source using the SkiBoardStyle table.

Drag the SkiBoardStyle table onto the form.

Figure 6.9



the new data set, such as `SkiBoardStyleDataSet`. Click the Finish button to save the new design. Note: If you make a mistake creating a `DataSet`, you can edit it later. It is also possible to delete a `DataSet` and try again, but it is a little tricky. You delete a `DataSet` by finding it in the Solution Explorer (such as `SkiBoardStyleDataSet.xsd`). Right-click the entry and choose the Delete option. Back in the Data Sources window, click the Refresh button to see the updated list.

Action
Open the Menu form.
Add a button for Ski Board Style.
Add the code to show the new form.
Test the Ski Board Style form.

Some developers prefer to include all tables within a single `DataSet`. This approach has the benefit of building all relationships within the `DataSet`. It might simplify development if you simply add all tables to the single `DataSet` and then choose the tables you need for each form. However, it also means that your application has to build the entire `DataSet` in memory for each form, even when only a single table is needed.

Once a table is defined in the Data Source, it is almost ready to be dragged onto the form. However, any time you build a form, you should check two things: (1) The type of form that you want (grid or details), and (2) The control type of each of the columns. Click the drop-down box for the `SkiBoardStyle` table and you will see that the `DataGridView` is selected by default. Because you want this form to use the grid, leave it as it stands. On the other hand, all of the columns will be created as text boxes, and it would be better to handle the Category column as a Combo Box. Open the drop-down-list for the Category column and select the Combo Box option. You can now drag the entire `SkiBoardStyle` table onto the form.

Although you are not finished, you might want to test the form. Open the Menu form and add a button. Set the button's Name and Text properties to `Ski Board Style` and `SkiBoardStyleButton`. Double-click the button and add the code to create and display the form:

```
Dim SkiBoardStyleForm As SkiBoardStyle = New
SkiBoardStyle()
SkiBoardStyleForm.Show()
```

Run the application and test the button and the new Ski Board Style Form. Notice that the Category column remains as a Text Box, even though you wanted a Combo Box. The data grid wizard is not yet smart enough to build the Combo Box for you. Close the running forms to stop debugging. Figure 6.10 shows the design-view of the new form along with the Data Sources window. You can resize the form and the `DataGridView` object to ensure that all three columns are displayed and multiple rows will be visible.

You will almost always want to customize the grid for the form. For example, you can set the widths of the various columns, change colors, or even lock certain columns so they cannot be edited. More importantly, you can set columns to use Combo Boxes to ensure data is entered consistently.

The Combo Box for the Category column will retrieve data that is already stored in a different table (`ProductCategory`). Recall that there is a foreign key relationship between the Category column in the `SkiBoardStyle` table and the `ProductCategory` table. When you build a form on a table that contains a foreign key relationship, you should almost always build a Combo Box to make it easier to

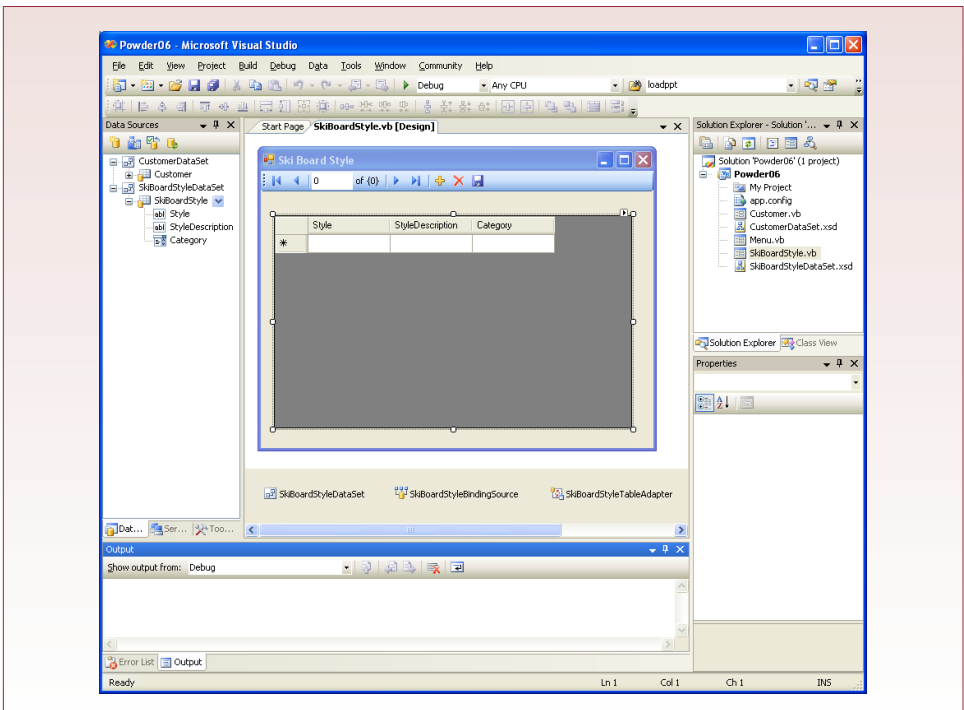
select the data. The Combo Box displays the potential data items by retrieving the list from the base table—ProductCategory in this case. When the user selects an entry, the matching value (or key) is transferred into the target table (SkiBoardStyle). Combo Boxes can handle two columns: One that is displayed to the user (typically descriptions), and one that is transferred to the target table (typically the key column). If you want to display a more complex row to the user that contains multiple columns (such as a customer's first name and last name), you first have to build and save a view that concatenates the desired columns. You would then add this view to the DataSet instead of the raw table.

To display any data on a form, you must include the data table (or view) in the DataSet. Open the Data Sources window and select the SkiBoardStyle DataSet. Click the button (or right-click) Configure DataSet with Wizard. Open the Tables list and check the box to add the ProductCategory table. This step adds the ProductCategory table to the SkiBoardStyle DataSet. If you look at the design view of the DataSet, you will see that it also adds a TableAdapter with a Fill command. However, adding a table to a DataSet does not make it immediately available to your form. You have to define an instance of the TableAdapter on the form itself.

For some reason, there is no easy method to add a single TableAdapter to the design view of the form. You could drag the entire ProductCategory table onto the form and then delete everything except the new TableAdapter, but that seems like overkill. Instead, it is relatively easy to add one to the code by hand. Use View/Code to switch to the code view of the form. On the line above the definition of the subroutine to load data (Private Sub SkiBoardStyle_Load) add a line to instantiate the ProductCategoryTableAdapter:

```
Dim ProductCategoryTableAdapter As _  
Powder06.SkiBoardStyleDataSetTableAdapters.
```

Figure 6.10



```

ProductCategoryTableAdapter _
= New Powder06.SkiBoardStyleDataSetTableAdapters.
ProductCategoryTableAdapter ()

```

Yes, it is a little long, but as you type the elements after the word “As,” the editor will display choices in a popup box, so you do not have to type every character. While you are in the code window, you should also add the command to tell the form to go to the database and get all of the rows in the ProductCategory table and put them into the DataSet. Inside the Load subroutine, right after the existing TableAdapter.Fill command, add the line:

```

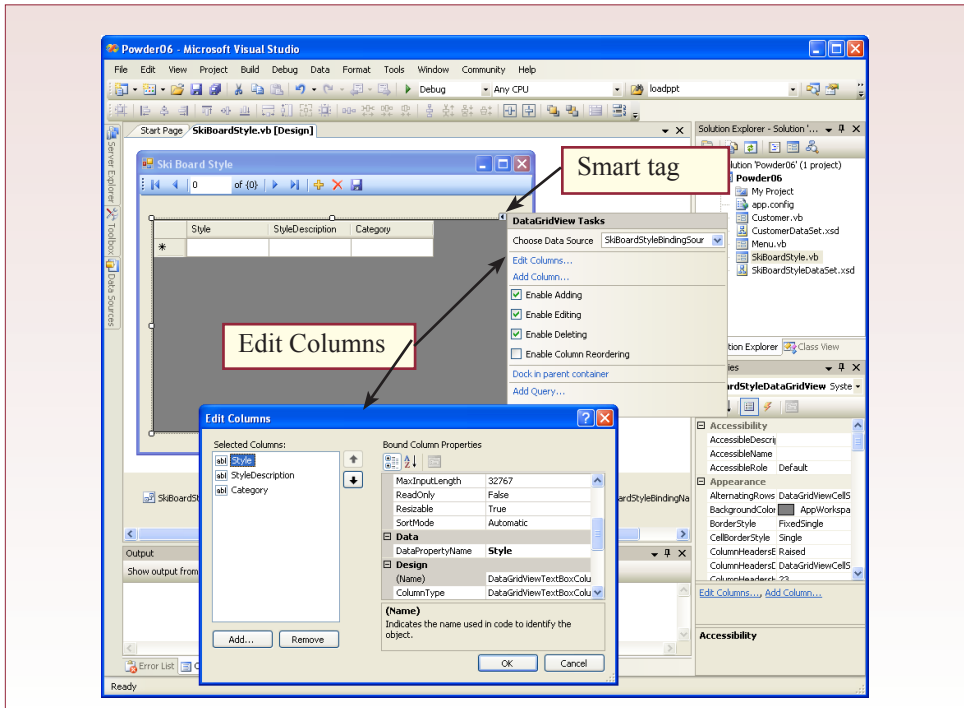
Me.ProductCategoryTableAdapter.Fill ( _
Me.SkiBoardStyleDataSet.ProductCategory)

```

You can close the Code window and return to the Design view. The next step is to convert the Category Text Box into a Combo Box and tell it to use the data from the new ProductCategory table. Click the DataGridView to select it. Click the smart tag (small arrow at the top-right corner of the grid object) to bring up the list of DataGridView tasks. In Figure 6.11, notice the checkbox options to control edit, insert, and delete. If you just need a grid to display data, or you want to prevent users from deleting items, you can quickly set these options. The Combo Box takes a few more steps.

Select the Edit Columns option to open the column-edit window. Select the Category column in the list of columns to display its properties. The first property to set is the ColumnType. Change it to a Combo Box (DataGridViewComboBoxColumn). Now you just have to tell it where to find the data rows. In the Data section, click the arrow in the DataSource property. Expand the Other Data Sources entry, and then the SkiBoardStyle List Instances item. Pick the SkiBoardStyle-

Figure 6.11



DataSet, remembering that you added the ProductCategory table to that DataSet instead of creating a separate one. Set the DisplayMember property to the Category column in the ProductCategory table. Do the same thing for the ValueMember column. Technically, the ProductCategory table has a CategoryDescription column and you could use this for the DisplayMember column. However, the category names (Ski, Board, and so on) are straightforward and easier to read. That's it. You now have a Combo Box that will let users pick values from the list of entries in the ProductCategory table. Save everything and run the form to test it.



Activity: Create Main Forms and Subforms

Now that you understand the main forms 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 including the SaleID and SaleDate. It also has a section of repeating data to hold the specific items being purchased by the customer. Keep in mind that each form can be associated directly with only one table. In this case, the Sale form will be based on the Sale table, and the subform will be based on the SaleItem table.

It is easiest to start with the data on the main form, so begin by creating a new form called Sale.vb. The next step is to add a new DataSet. You definitely want to include the Sale table. However, you need to include the relationship to the SaleItem table, so you must also include the SaleItem table. Be sure to give it a distinctive name: SaleFormDataSet. Since the Sale table represents the main form,

Action

Create a new form called Sale.vb.

Create a new DataSource using the Sale and SaleItem tables.

Set the form type to Details for the Sale table.

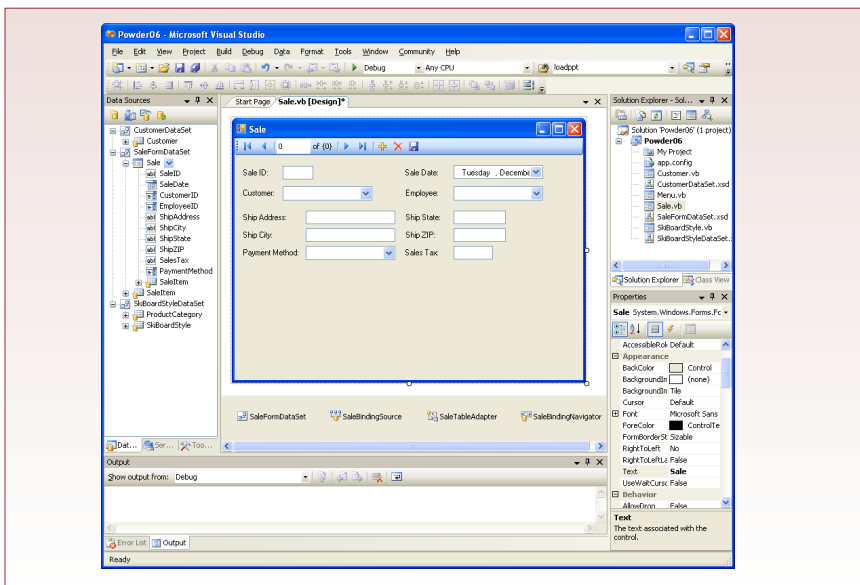
Set the column type to ComboBox for EmployeeID, CustomerID, and PaymentMethod.

Drag the Sale table onto the form.

Resize and rearrange the controls to create space at the bottom of the form.

Add a button to the Menu form to open the Sale form.

Figure 6.12



use the drop-down-list to change it to Details instead of grid. Look through the columns in the Sale table and change the foreign keys into Combo Boxes: CustomerID, EmployeeID, and PaymentMethod. Drag the Sale table from the DataSource onto the main form. You will have to resize

and rearrange the columns to move them to the top of the form, leaving space at the bottom for the grid you will add in a minute. Add a button to the Menu form with the code to open the new Sale form. Test the form. You should save your work at this point, in case you make a mistake later and want to return to this point. Notice that the Combo Boxes do not work yet—you will need to add the DataSets and TableAdapters for each of them, but you can do that after you add the grid subform. Figure 6.12 shows the basic elements of the initial design.

The next step is to add the SaleItem as the subform grid. It is similar to creating a tabular grid form, but there is one tricky part to the step. Look closely at the entries in the Data Sources window. You will see that the SaleItem table is entered twice: Once by itself and once beneath the Sale table entry. To enforce the relationship between the two tables, you must use the SaleItem table that is listed beneath (or inside) the Sale table. Drag that entry onto the lower-part of the Sale form. You will probably have to resize the grid and the overall form to improve the layout. You can run the form again to test it. Scroll through a few sales and the entries in the subform should change to match the SaleID in the main form. If they do not change, it means you dragged the wrong SaleItem table onto the form. Delete the grid and look more closely at the Data Sources window. Figure 6.13

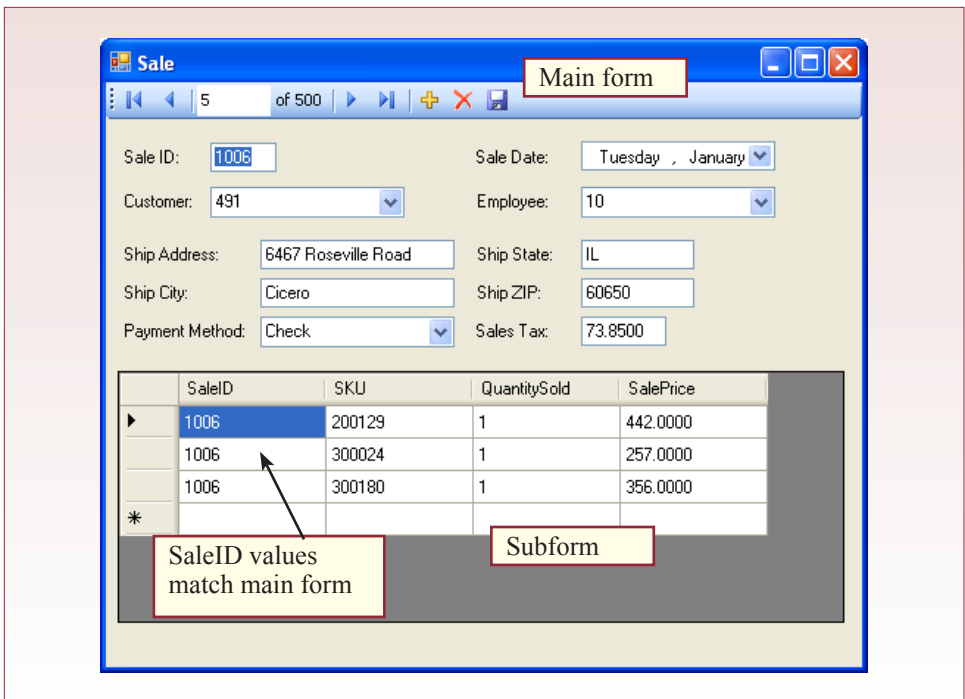
Action

Find the SaleItem table in the Data Sources window that is within the Sale table.

Drag it onto the Sale form.

Test the form to ensure the grid values change as new sales are selected.

Figure 6.13



shows the correct version of the subform where the SaleID values match those in the main form.

Now that you know the subform is properly linked, there is no point in displaying the SaleID column repeatedly in the subform. Also, you need a column that multiplies price times quantity to show the value of each line. Both of these tasks are handled by editing the columns for the grid. Use the smart tag, or right-click the grid, to edit the columns. The SaleID column is the easiest to handle. Select SaleID in the column list and click the Remove button.

The best way to compute price times quantity is to handle it in the DataSet, so that the value is updated automatically when entries change. Go back to the Data Source window and right-click the SaleItem table (VS 2008: right-click the DataSet at the bottom of the Sale form). Choose the option to Edit DataSet with Designer. As shown in Figure 6.14 you will see the Sale and SaleItem tables and the relationship that connects them. Right-click the SaleItem table and choose Add/Column. Change the name of the new column to Value. Use the Properties window to change the Data Type to System.Decimal. This column is not bound to the database, so you need to define the calculation to compute the value. For the Expression property, enter the computation: $QuantitySold * SalePrice$. Double-check

Action

Edit the grid's columns.

Remove the SaleID from the grid.

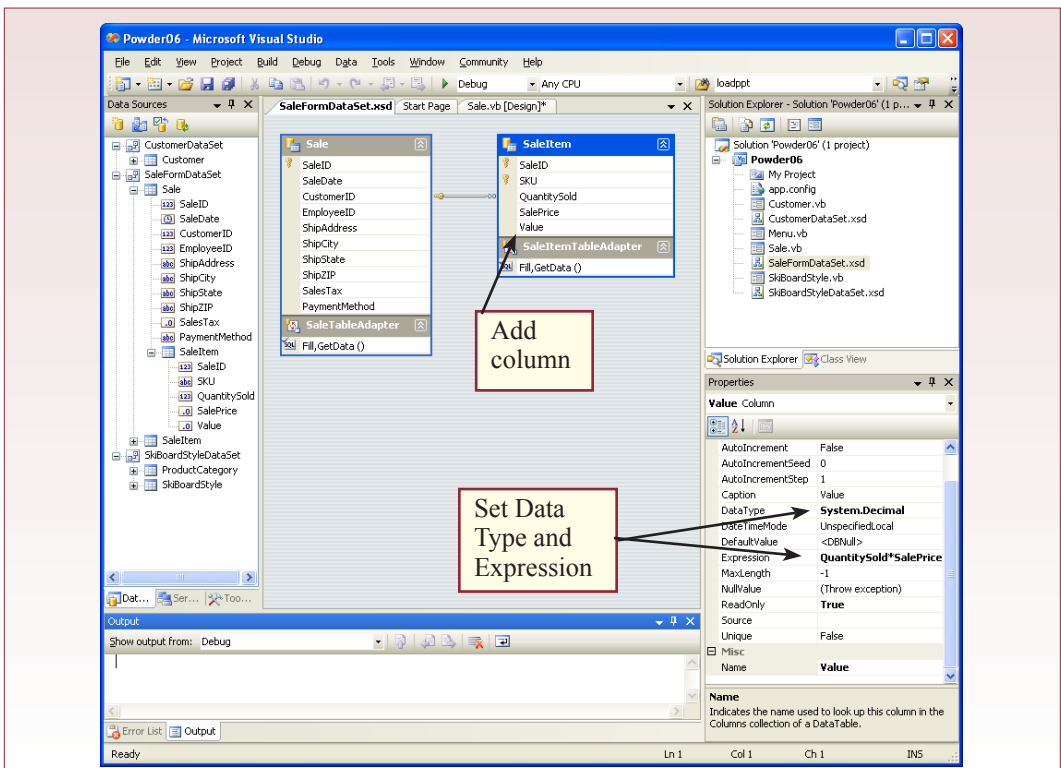
Open the DataSet Designer for the SaleItem table.

Add a new column in the SaleItem table named Value.

Set the DataType to System.Decimal.

Set the Expression to $QuantitySold * SalePrice$.

Figure 6.14



to ensure that the `ReadOnly` property is set to `True` so no one tries to change the value directly. Save and close the designer page.

You now have to add the new column to the subform grid. Note that if you had added the column to the `DataSet` before dragging it onto the form, it would automatically be included. You could delete the grid and

recreate it, but you should learn how to edit the grid. Edit the columns for the grid and select the `SalePrice` entry in the column list. Click the `Add` button to add a new column to the grid. Select the new `Value` entry in the `DataSource` list and check the `Read Only` option. Click the `Add` button to add the column. You can set the `DefaultCellStyle` property for the `SalePrice` and `Value` columns to improve the display of the data. For example, set the `Format` to a `Numeric` data type with 2 decimal places. You should also set the alignment to `Middle-Right`.

Figure 6.15 shows the running version of the `Sale` form. Try changing the `QuantitySold` or `SalePrice` and watch as the `Value` entry is updated. Remember that the `Value` column is only available in the `DataSet` and that the items you changed will not be updated unless you click the `Save` button on the navigation tool bar.

The form looks much better and is beginning to provide useful information. However, something important is missing. It should automatically total the `Value` column so clerks can provide subtotals to the customers. Unfortunately, there is no automatic mechanism to compute subtotals. However, you can use the form's event code to compute the totals whenever the data is changed.

Action

Edit the grid's columns.

Add the `Value` column as `Read Only`.

Edit the `DefaultCellStyle` for the `SalePrice` and `Value` columns.

Set the `Format` to `Numeric`.

Set the `Alignment` to `Middle-Right`.

Save everything and test the form.

Figure 6.15

The screenshot shows a Windows application window titled "Sale". At the top, there is a navigation bar with "1 of 500" and several icons. Below the navigation bar, there are several input fields and dropdown menus for form data:

- Sale ID: 1002
- Sale Date: Friday, March
- Customer: 1900
- Employee: 2
- Ship Address: 5516 Smallhouse Road
- Ship State: AL
- Ship City: Phenix City
- Ship ZIP: 36867
- Payment Method: Cash
- Sales Tax: 57.6800

Below these fields is a data grid with the following columns: SKU, QuantitySold, SalePrice, and Value. The grid contains four rows of data, and the 'Value' column is calculated based on 'QuantitySold' and 'SalePrice'.

SKU	QuantitySold	SalePrice	Value
600017	1	32.00	32.00
600046	1	15.00	15.00
800115	1	425.00	425.00
800126	1	352.00	352.00

At the bottom of the grid, there is a row with an asterisk (*) in the first column, indicating a subtotal or total row.

```

Public Sub ComputeSaleTotal()
    Dim sum As Decimal
    Dim dgvr As Windows.Forms.DataGridViewRow
    For Each dgvr In Me.SaleItemDataGridView.Rows
        sum += dgvr.Cells("Value").Value
    Next dgvr
    Me.SubtotalLabel.Text = sum.ToString("$ #,##0.00")
End Sub

```

Figure 6.16

Begin by placing a Label on the form to hold the computed subtotal. First add a Label and change the Text property to Subtotal. Add a second label that falls beneath the Value column. Set the Text property to 0.00 and the Name to SubtotalLabel. This label is the one that will hold the computed values. You can now write the code that will calculate the subtotal on demand.

Use View/Code to switch to the code editor. Just above the existing End Class line, add the code shown in Figure 6.16. If you want to handle taxes, you should also add a TotalLabel to the form and include a row immediately before the End Sub that adds the computed sum to the taxes:

```

TotalLabel.Text = sum
    + Decimal.Parse(Me.SalesTaxTextBox.Text)

```

You could also write a function to compute the value of the sales tax automatically, but this computation is relatively complex in most states.

The ComputeSaleTotal routine performs the calculation, but you still must decide when it should be called. If you call it too often, it will slow down the user interface. If you do not call it often enough, the user will see incorrect or missing totals. As shown in Figure 6.17, two event triggers can cover the cases efficiently: (1) When a new sale is selected, and (2) When the user finishes editing a cell in the grid. The easy way to create the subroutine structures is to use the Combo Boxes at the top of the code editor. Pick the object in the left box (SaleItemDa-

Action

Place a Label below the grid.
 Change the Text to Subtotal.
 Place a Label beneath the Value column.
 Name it SubtotalLabel and set its value to 0.00.
 Create the code subroutine to compute the total.
 Create the two events that need to call the ComputeSaleTotal routine.

Figure 6.17

```

Private Sub SaleItemDataGridView_CellEndEdit(
    ByVal sender As Object,
    ByVal e As System.Windows.Forms.DataGridViewCellEventArgs)
    Handles SaleItemDataGridView.CellEndEdit
        ComputeSaleTotal()
End Sub

Private Sub SaleBindingSource_PositionChanged(
    ByVal sender As Object,
    ByVal e As System.EventArgs)
    Handles SaleBindingSource.PositionChanged
        ComputeSaleTotal()
End Sub

```

taGridView in the first case) and the event in the right box (CellEndEdit). On picking the event, the editor will generate the Sub and End Sub lines. You simply add the ComputeSaleTotal() call in the middle.

If you decide to add a text box to display the total that includes the subtotal and the tax value, you might need to add another event trigger. If

you do not compute the taxes automatically, you should add an event trigger to the TextChanged event of the tax box that also calls the ComputeSaleTotal function. For now, you can ignore sales taxes.

You are not quite finished. Test the form and see if you can find the problem. When you make a change to the subform, click the Save button. You will see that only changes made to the main form are saved. You need to add two lines of code to ensure that changes made in the grid are also written to the database. Close the running forms if needed and switch to Design view for the form.

Double-click the Save button icon (small disk) on the navigation tool bar to open the code editor for the button. Figure 6.18 shows the existing three lines of code handled by the Save button, along with the two new lines that you need to add. The wizard built the first three lines when you initially added the Sale table to the form. They save the changes made to the Sale table. The last two lines mimic the others. First you end the edits on the SaleItem binding source. Second you call the SaleItem TableAdapter to update the SaleItem table.

At this point, everything should work. You should be able to change data and enter new rows. Scroll through the sales and you will see that the totals are updated automatically. Figure 6.19 shows the working version of the form.

Before you start thinking that you are finished, go ahead and try to create a new sale and enter new data. It will work, but what do you enter for a CustomerID, PaymentMethod, and EmployeeID? In practice, each of these should have a combo box or lookup list. You also might think about using a combo box for SKU, but those could probably be read from the item tag. You also might want to leave EmployeeID as a text box for security purposes—so that employees memorize their own ID numbers.

Adding a Combo Box for CustomerID, EmployeeID, or Payment Method is straightforward. For customers, the Combo Box gets its list from the Customer table in the database, so you have to create a new Data Source to retrieve that information. One important catch with combo boxes is that they can handle only two “columns:” (1) A value column which is usually the key column, and (2) A

Action

Test the form and recognize that it does not save changes in the grid.

Double-click the Save icon.

Add two lines of code to save the changes from the grid.

Test the form.

Save everything.

Figure 6.18

```
Private Sub SaleBindingNavigatorSaveItem_Click_1(...)
    Me.Validate()
    Me.SaleBindingSource.EndEdit()
    Me.SaleTableAdapter.Update(Me.SaleFormDataSet.Sale)

    Me.SaleItemBindingSource.EndEdit()
    Me.SaleItemTableAdapter.Update(Me.SaleFormDataSet.SaleItem)
End Sub
```


	SKU	QuantitySold	SalePrice	Value
▶	600017	1	32.00	32.00
	600046	1	15.00	15.00
	800115	1	425.00	425.00
	800126	1	352.00	352.00
*				
Subtotal				\$ 824.00

Figure 6.19

display column. The trick is that you generally create a view so that the display column can contain several actual database columns. In the case of the customer, you want to include the LastName, FirstName, and Phone number so clerks can readily identify the customer. It is easiest to create the View on the database.

Figure 6.20 shows the query that you run on the database to create the view. You can run this query from the Server Explorer in Visual Studio. Create a new Data Source that retrieves the rows from this new view. One useful trick: After the wizard creates the Data Source, edit it in the designer and modify the SELECT statement to include ORDER BY CustomerName so that the display list is sorted. SQL Server does not allow you to save ORDER BY clauses in server-based views.

Binding a Combo Box is slightly tricky, and it is safest to remove the existing binding first. Select the CustomerComboBox on the form design. Expand the DataBindings list in the Properties window. For the Text property, click the drop-down-list arrow and select None to remove the existing setting. To fill the Combo Box with data, use the drop-down-list arrow for the DataSource property to ex-

Action

Run the query to create the CustomerList view.

Add a Data Source that retrieves the rows from the view.

In Customer Combo Box, Data Bindings, remove the binding for Text.

For DataSource, select the CustomerListDataSet/CustomerList.

Set DisplayMember to CustomerName.

Set ValueMember to CustomerID.

Set SelectedValue to SaleBindingSource/CustomerID.

Test the form.

Figure 6.20

```
CREATE VIEW CustomerList AS
SELECT CustomerID, LastName + N', ' + FirstName
      + N' (' + Phone + N')' AS CustomerName
FROM Customer
```

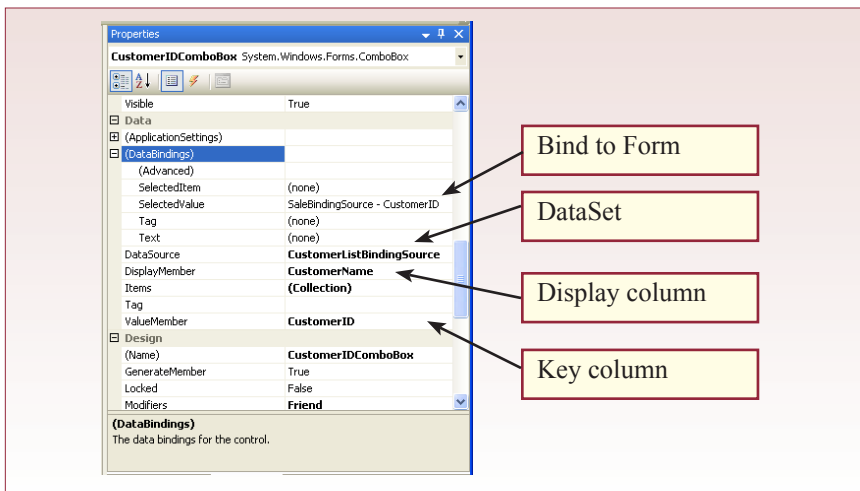
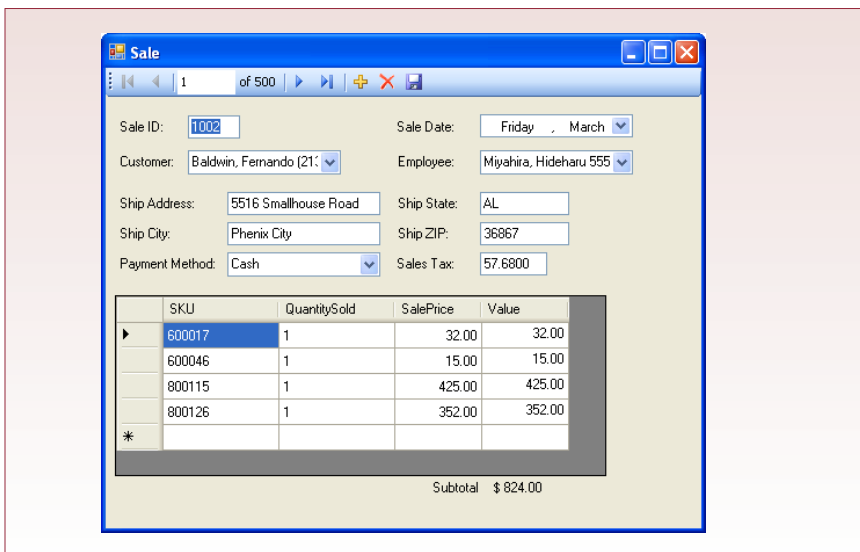


Figure 6.21

Expand Other Data Sources, then Project Data Sources, then CustomerListDataSet, and select the CustomerList view. Choose CustomerName as the DisplayMember and CustomerID as the ValueMember. Now you have to rebind the Combo Box to the underlying Sale table. For the SelectedValue property, select the SaleBindingSource-CustomerID. Figure 6.21 shows the DataBinding properties for the Customer Combo Box.

Save everything and test the form. If the Combo Box does not display the names correctly, remove all of the data binding properties and try again—being careful to enter them in the order described here. You can follow a similar process to set the Combo Boxes for Employee and Payment Method, but the Payment Method has a single column so it does not need a view. Figure 6.22 shows the final form.

Figure 6.22



All Powder Basic Reports

At the moment, Microsoft Visual Studio has two tools to create reports: Crystal Reports and the SQL Server Reporting Services. Crystal Reports is older and licensed from another company—which helps explain why Microsoft released the Reporting Services in 2004. Because the Microsoft tool is the most likely one to survive, it will be used in this chapter. The underlying concepts of any reporting tool are similar, so you can apply most of the same concepts if you do not have access to the SQL Server Reporting Services. The Reporting Services package is usually installed when you install SQL Server. However, you might not have set it for automatic start. If you want to avoid warning messages, you should check your system services now. For testing purposes, it is convenient to run the Reporting Services on your development computer—although you will eventually transfer them to a shared database server. Assuming you have Administrative tools active on your computer, use Start/Administrative Tools/Services to open the service manager. Look for the SQL Server Reporting Services and see if its status is set to “Started.” If not, select the service row and click the Start button. One other useful trick to know is that the Microsoft Reporting Service will import reports from a Microsoft Access database.

One other installation trick is that the Reporting Services editors work with Visual Studio. However, the standard Visual Studio installation does not include the reporting tools. Instead, you need to install the Client tools from the SQL Server database installation disk. Even if you installed the full SQL Server DBMS on your workstation, you should reinsert the SQL Server installation disk and install the Client tools. You can verify the installation from within Visual Studio. Use File/New Project and verify that the Business Intelligence Projects are available.



Activity: Create Reports with Subtotals

Most managers want reports so they can evaluate the progress of the 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 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 the 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.

Visual Studio treats Reporting Services as a new project. However, solutions in Visual Studio can have multiple projects. Your current solution has one project

Action

- Add a BI Report Server Project.
- Create a shared data source.
- Add a new report.
- Build a query using Customer, Sale, SaleItem.
- Select Tabular report.

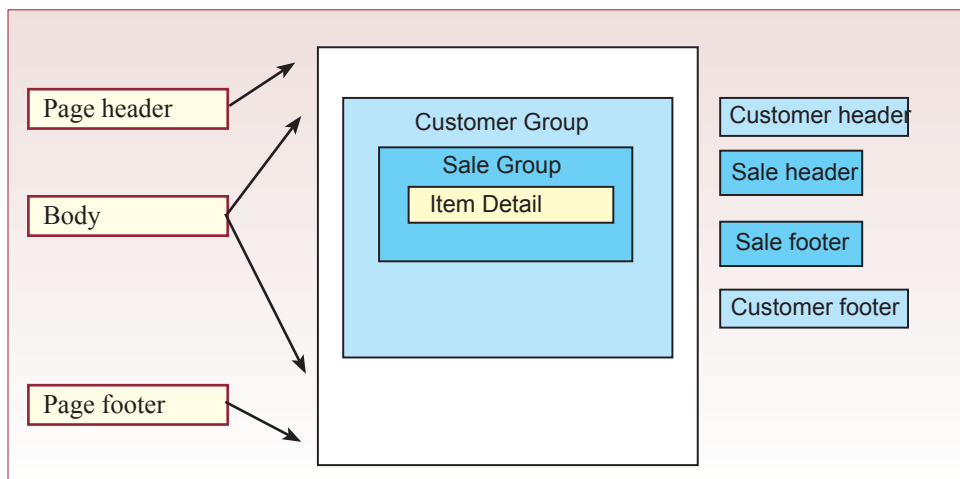
that consists of the forms you have built. You can add a new reporting project to this solution, or create a separate solution. To make it easier to integrate the reports with the forms, you should add the reporting project to your current solution.

Begin by adding a reporting project to your current solution with File/Add/Project, then pick Business Intelligence and Report server Project. Give it a recognizable name such as Powder06Reports. Do not use the report wizard. It only builds limited types of reports. Instead, it is easier to build a report from scratch where you have more control. Figure 6.23 shows the basic layout of the report you need to build. The page header and footer are optional, but you will probably want to print page numbers in the page footer. The grouping is the important part to understand. Each group represents a repeating section of data. For example, there are many customers, each customer can participate in many sales, and each sale can consist of many detail items. SQL Server provides three ways to display repeating groups of data: List, Table, or Matrix. The list is the most general approach and gives you complete control over layout. A table has a fixed number of columns and displays data in a grid. A matrix can have a variable number of columns and rows—it is used to handle cross-tabular designs. For this example, you will use list groups: (1) Customer, (2) Sale, and (3) Item detail.

Since you are going to create several reports, you should create a shared data source. A data source is essentially a connection string that tells the report service how to find the database. Right-click the Shared Data Sources folder in the Solution Explorer, and choose the option to Add New Data Source. The connection design screen is similar to the other ones in Visual Studio. Give it a name such as AllPowerDataSource. Click the Edit button to create the connection string. Enter the server name, login account information, and the name of the database (Powder). By sharing the data source, you only have to create it once for the entire project.

Now, add a new report to the project. The report wizard simplifies some of the details, but it is not much harder to build a report from scratch. For now, use the wizard. Right-click the Reports folder and choose Add New Report to start the wizard. Select the shared data source you created. The first step is to create a query to retrieve the data needed by the report. Report queries need to retrieve rows of data that match the desired detail level of the report. In this report, you need the

Figure 6.23

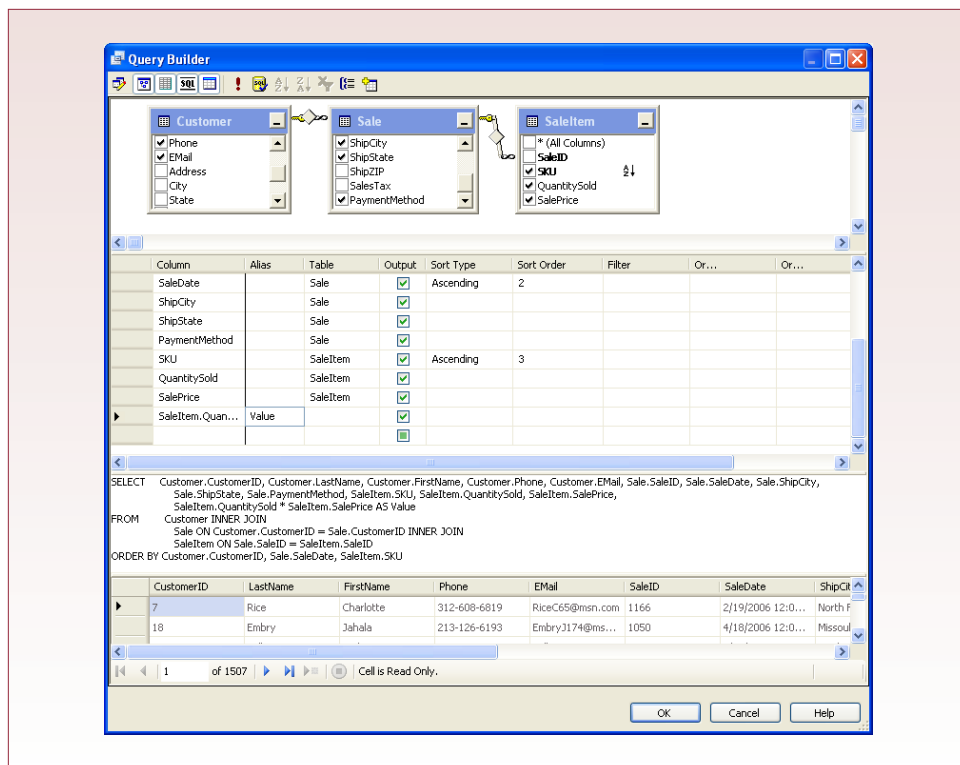


Customer, Sale, and SaleItem tables because you want the SaleItem detail. Click the Query Builder button to open the query designer screen. In 2005, click the strange Generic Query button to switch to the more familiar designer. Right-click or click the Add Table button and select the three tables. Because you want to control the column order and the sorting, click columns individually instead of selecting the All Columns (*) option. As shown in Figure 6.24, select basic columns from the Customer table (CustomerID, LastName, FirstName, Phone, and Email). From the Sale table, select (SaleID, SaleDate, ShipCity, ShipState, and PaymentMethod). From the SaleItem table, you need (SKU, QuantitySold, and SalePrice).

You also need to add a Value column that multiplies price by quantity for each row of data. In the last row of the Column section of the grid, enter $\text{QuantitySold} * \text{SalePrice}$ and set the alias to Value. You should sort the data by CustomerID, SaleDate, and SKU. Run the query to test it and ensure that you typed the calculation correctly for the Value column.

The Report wizard enables you to build two basic types of reports: Tabular and Matrix. The Matrix report is generally used when you need to compare two columns against each other (such as Employee sales over Year). Select the Tabular type for this report. Setting the report structure is the trickiest part of the wizard. You want each customer displayed on a separate page, so add the CustomerID to the Page level break. The trick is that you must use the CustomerID by itself. Similarly, you want to show sales separately, so move the SaleID into the Group break by itself. If you add multiple columns to the Group (or Page) break, the report will create multiple nested levels of breaks. On the other hand, you have to add all of the SaleItem columns to the Details level. Figure 6.25 shows the layout

Figure 6.24



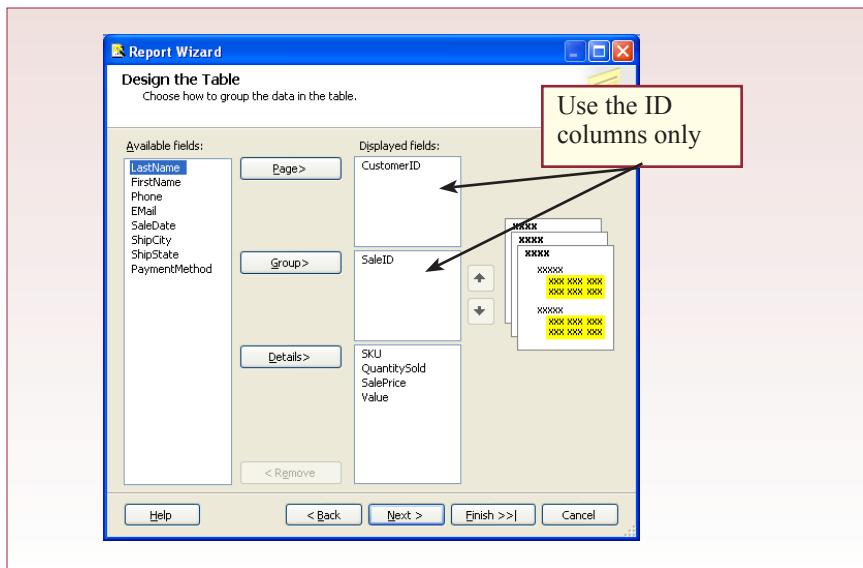


Figure 6.25

that you need for this report. The next step enables you to choose some options on how the report will be used. To demonstrate some of the capabilities of the Reporting Services, select the Stepped option, include subtotals, and enable drilldown. You can also select a color style template. Ultimately, you should pick one style and use it for all reports. You will probably have to build several reports with each style to see the differences and find one you like. Alternatively, pick the blank style and add your own custom touches later.

Action

Design the grouping structure.
 Put only CustomerID at the page level.
 Place only SaleID at the Group level.
 Place all SaleItem data at the Details level.
 Choose Stepped layout, include subtotals and enable drilldown.
 Set the name to Customer Sales.

Figure 6.26 shows a preview of one page of the report. This particular customer was chosen because there are multiple sales. The Drilldown/Rollup feature is one of the more interesting aspects of the Reporting Services. Instead of trying to squeeze all of the detail for every sale onto one report, the report displays summary totals for each Sale. The user can then click the drilldown icon (+) to see the item data for a particular Sale. This feature is particularly useful when you create a report with multiple nested group levels. But, if the users do not want this feature, you could have chosen to disable it when you created the report—and all detail rows would be displayed automatically.

One of the most glaring problems with this report is that it displays only the CustomerID and SaleID. Most organizations will want to see the other data related to customers and sales. It is relatively easy to add those items now that the structure has been built by the wizard. Click the Layout or Design tab to switch to design view. Note that the Data tab/window opens the query editor so you can modify the underlying query if you forgot to include a column or two. You could add text boxes manually to the report and assign the Value property to lookup the columns that you want to see. However, there is an easier method. First, make

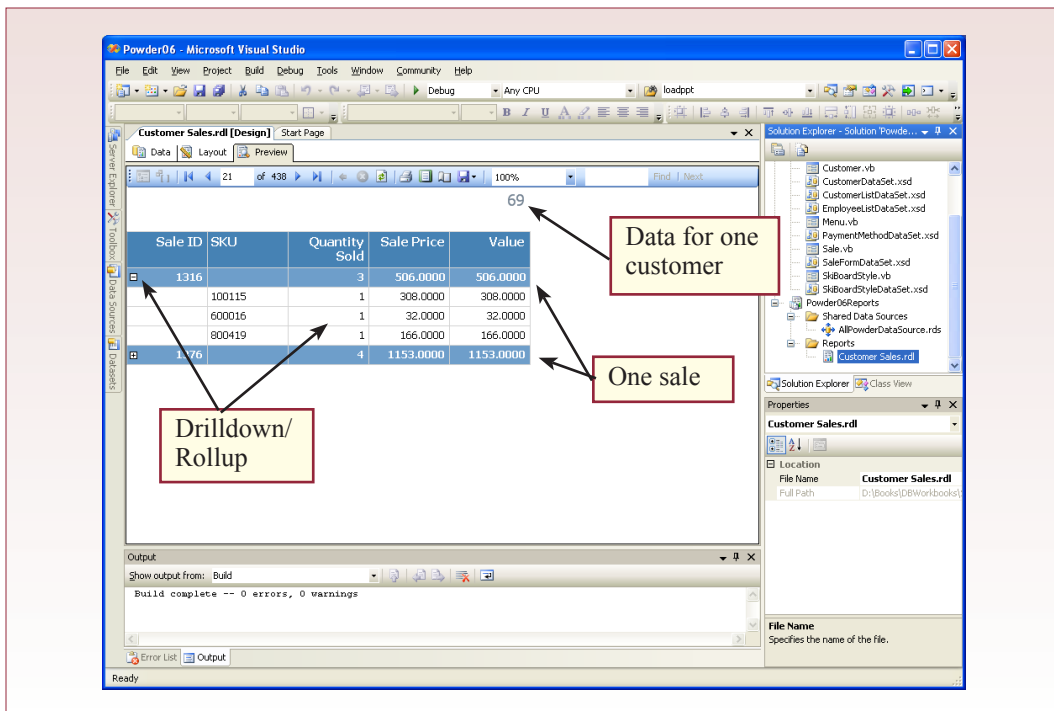


Figure 6.26

some space on the report by dragging the left edge of the CustomerID box to the right. The ID numbers take up minimal space, so free up the left side and center of the report.

Now you can add more fields to the report. Open the DataSet tab that is usually on the left side of the Visual Studio window (or use View/DataseTs on the main menu). If necessary, use the Window Position button to remove the Auto Hide option so that the window stays open. Expand the AllPowderData-Source that you created to see the list of columns. Drag the LastName column and carefully drop it onto the report at the left edge aligned with the CustomerID field. Follow the same process to drop the FirstName column between the LastName and the CustomerID. Resize the new text boxes and set their properties to match those used for the CustomerID box. Add the Phone and EMail columns just below the LastName and FirstName text boxes. Preview the report and scroll through a few pages to ensure that the new values change as you change customers. Figure 6.27 shows the current layout of the report. Notice that ListBox1 defines the Page level grouping. You must ensure that the new text boxes you added fall completely within ListBox1, and do not fall on the Sale-level grouping area.

It is a little trickier to add columns at the Group-level break. Currently SaleID is the only data displayed at the Group level, but the wizard also placed the detail headings at that level. You need to add more space to the layout. The Group and Details are managed by a matrix-type layout. You add space by inserting rows at the desired level. Figure 6.28 shows the basic process. First, select the SaleID text

Action

- Click the Design/Layout tab.
- Open the DataSet window.
- Adjust the CustomerID TextBox.
- Drag LastName, FirstName, Phone, and EMail columns next to the CustomerID.
- Preview the Report.

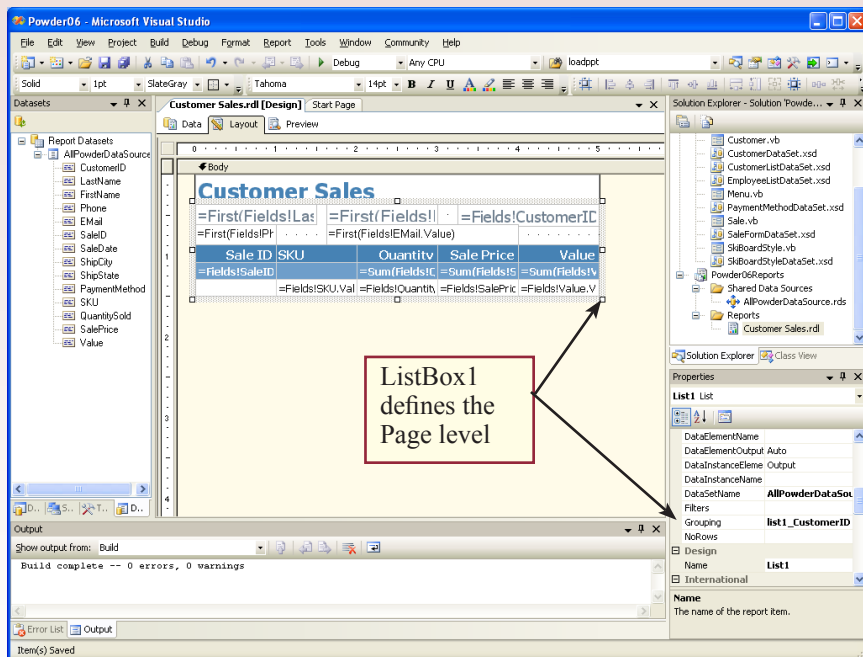


Figure 6.27

box because it is at the Group level. Be sure to pick the Text Box that displays the value, not the label that says “SaleID.” The grid icons will be displayed on the left side. Right-click the icon to the left of the Text Box and choose the option to Insert Row (Inside Group) Above. Now you have space to add the rest of the Sale data. In VS 2005, drag SaleDate into the first cell. with VS 2008, click the item list box inside the desired cell. Put PaymentMethod, ShipCity, and ShipState into the other cells. Preview the report and you will see the new data. You can return to Layout view and resize the grid’s columns to minimize the text wrapping within the new cells. Just keep in mind that if users are going to print the report, you should keep the total width under 6.5 inches (8.5 inch paper with 1 inch margins).

You are almost finished, but a professional job requires that you pay attention to details. One big detail is the data formats. Notice that the prices and sums use four decimal places by default; and the date includes time, making it too big to fit the field. Fortunately, formats are easy to fix—select each Text Box and set its Format property. For the case of the prices, set the Format value to 0.00. For the sum of the Value column, you might want to include the comma separator, so set the Format to #,##0.00. You can choose various date formats, but the medium date (such as 01-Mar-2007) is easily recognizable by almost any user in any nation. Select the SaleDate text box and set its format to dd-MMM-yyyy. Note that

Action

- Click the SaleID TextBox.
- Right-click the icon to the left of the SaleID TextBox.
- Choose Insert Row Above.
- Drag SaleDate, PaymentMethod, ShipCity, and ShipState onto the new row.
- Preview the Report.

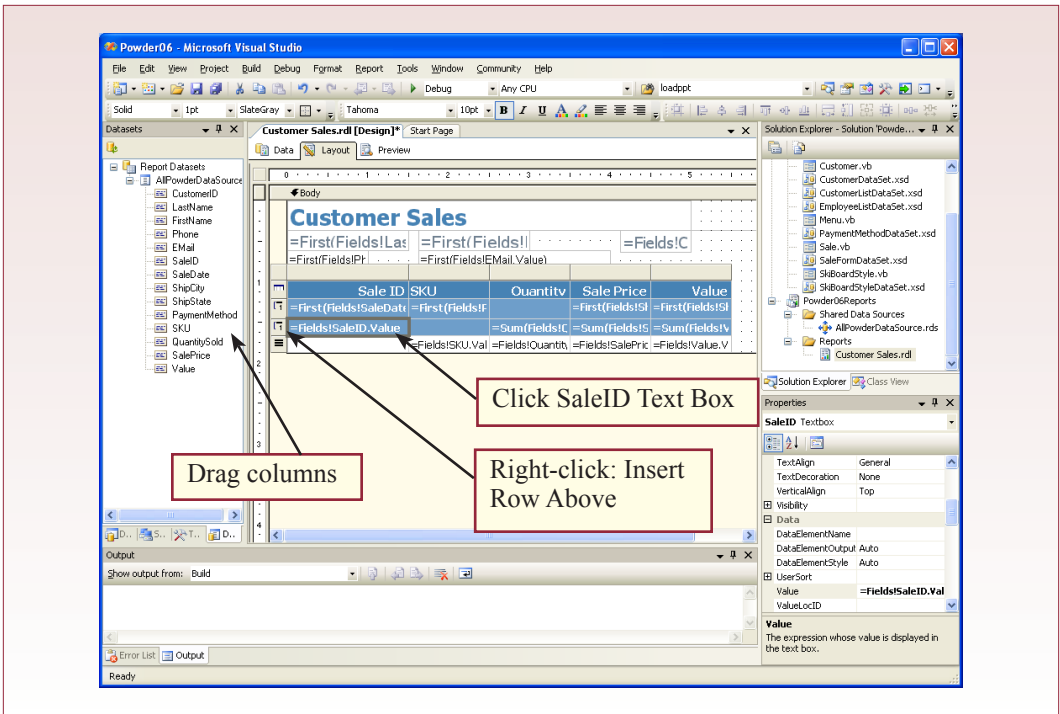


Figure 6.28

the month specifier (MMM) is case-sensitive and must be capitalized or it will display the month number instead of the abbreviation.

You need to make one more important change. When you allow the wizard to compute the sums, it automatically calculates and displays the total for every numeric column of data. In the Sale case, it makes no sense to total the SalePrice column. Select the Text Box that displays the sum of the SalePrice. Press the Delete key to remove the entire Text Box. Unfortunately, in VS 2005 this action also removes the background color. Select a cell next to the one you are in. Find the Background property and copy it. Return to the original cell and paste the copied color into the Background property. You might also consider removing the total for the QuantitySold column, but some users might want to see the total number of items purchased for each sale, so leave it in unless the users ask you to remove it. Figure 6.29 shows the final version of the report.

You can customize the report or even build one from scratch. The key to reports is that you need a ListBox whenever you want to display repeating rows of data. In the Customer Sales report, one large ListBox contains all of the controls; except for the report title. Tables are often used to show Groups and Details. Tables provide the support for Drilldown and Rollup interaction. If you do not like the layout of a report produced by the wizard, you can customize it by adding or removing ListBoxes and Tables. You should experiment with the report layout. Remember that the List boxes are flexible. You can draw them almost anywhere

Action

- Format price items to 0.00 or #,##0.00 for larger numbers.
- Format the date to dd-MMM-yyyy
- Remove the Value property for Sum of SalePrice text box.
- Preview the report.
- Save everything.

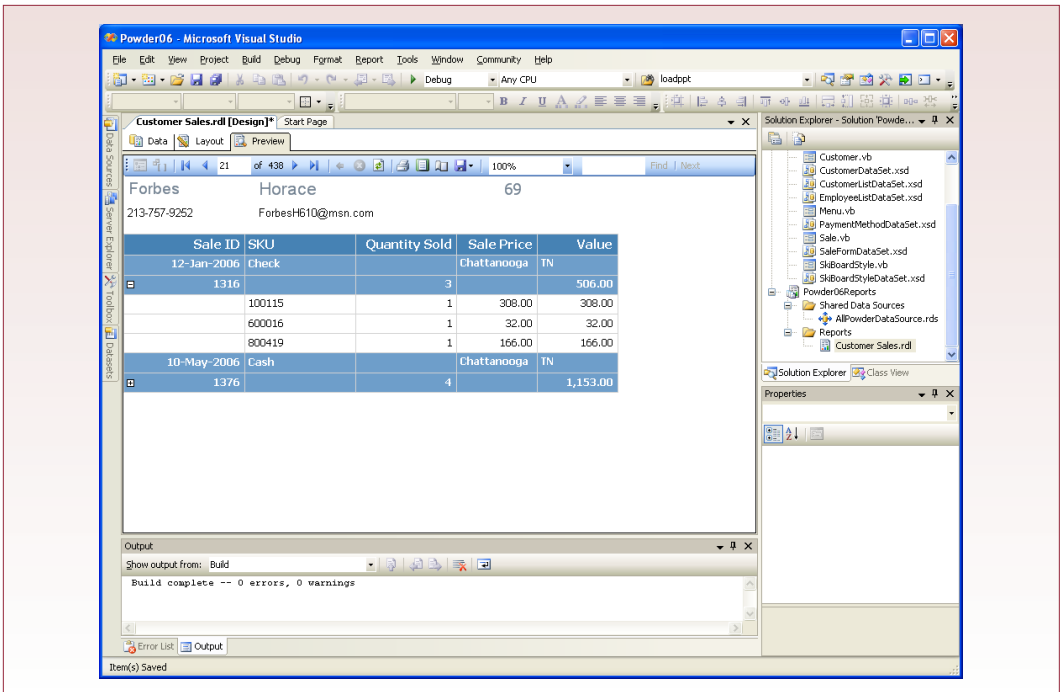


Figure 6.29

on the page. For example, you could draw two different list boxes side-by-side. Use one to show Sales for a customer. Use the other to show rentals. You will get two parallel lists. If you encounter problems building a report, try creating it in Microsoft Access first, and then importing it to SQL Server.

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 number of hours 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, their phone numbers and e-mail addresses. Sometimes, she also wants a similar list for 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 in 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, 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 are having trouble keeping some items in stock, and other items seem to sit on the shelves forever; but 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.

Database Integrity and Transactions

Chapter Outline

Program Code in SQL Server, 119

Case: All Powder Board and Ski Shop, 120

Lab Exercise, 121

All Powder Board and Ski Data, 121

Database Cursors, Keys, and Locks, 136

Exercises, 146

Final Project, 148

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 SQL Server

SQL Server supports code directly within the SQL as database triggers that are fired when some database event arises. For example, when a row of data is inserted into a table, code can be executed to validate the data. In addition, if you build forms in Visual Studio, you can write code behind the form that responds to events that arise within the form itself. For example, you commonly have to write code that executes when someone clicks a button on a form.

When building applications, one of the first questions you have to answer is whether you should write code as a database trigger or a form trigger. Unfortunately, there is no good single answer. As with all programming, you have to evaluate the competing benefits and costs and make the decision based on the situation. Sometimes there are overarching concerns. For instance, if you are concerned about database portability, because the organization is thinking about switching to a different backend DBMS, you would try to write most of the code within Visual Studio. On the other hand, if you know you will stick with SQL Server and you want to make it as easy as possible to create and modify forms, you will want to write most of the code as procedures within SQL Server.

Writing triggers within the DBMS has the advantage of centralizing most of the functions. You can write them once and they can be used by multiple developers. However, complex applications built within the DBMS can be difficult to debug, and place heavy loads on the database servers. Remember that code executed in Visual Studio forms runs on the client computers and the database server is primarily responsible for storing and retrieving data. When you build thousands of lines of code within the DBMS, this code runs on the server itself—placing additional demands on the server.

More complications arise when your database has hundreds or thousands of these triggers. A simple change to one table could cascade to dozens or hundreds of additional updates propagated through the trigger code. This cascade of events can be difficult to trace and understand—particularly when the code sections have been written by dozens of different programmers. On the other hand, database triggers are an important tool to provide additional security and ensure that certain tasks are performed correctly. The code is created in one location and it cannot be circumvented. Once the events and code are defined, it does not matter what users and application developers create—the trigger updates are processed behind the scenes without additional intervention.

Visual Studio uses form triggers to provide customized responses to user events. You can create simple or detailed code when a user presses a key, clicks a form button, or changes a piece of data. Dozens of events can accommodate your customized code, but generally you need to write only a couple of lines of code for one or two primary events. The other event triggers are available in case you need them for a special feature.

SQL Server stores code in procedures. A procedure contains a declaration section and a body. The declaration section lists the variables used within the procedure. With SQL Server, you write the code using Transact-SQL. With SQL Server 2005, it is also possible to write modules in Visual Basic or C# and compile them so they can be called as functions within a SQL query. This technique is useful for functions that are mathematically or processing-intensive, but is not covered in this chapter.

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 as a database trigger 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 Sale 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 look it up correctly in a table. Instead, you need to write a function that will compute the sales tax correctly and transfer it to the form and the database. Sales taxes can be highly complex. Some items might be taxable, while others are not. Since each state and local district is different (and there are several thousand tax districts in the United States alone), this presentation is simplified and assumes a single tax rate that is applied to all sales and to 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 Sale form, but since the code will also be useful for rentals, it makes more sense to generalize it and place it either in the database itself so that it is available to any form, que-

Figure 7.1

The screenshot shows a 'Sale' form with the following data:

SKU	Quantity Sold	Sale Price	Value
600017	1	32.00	32.00
600046	1	15.00	15.00
800115	1	425.00	425.00
800126	1	352.00	352.00
			Subtotal \$ 824.00

ry, or report within the application. Placing the code in the DBMS also makes it easier to find later. It is also possible to write the tax computation within a global function within Visual Basic. This function would be accessible from all forms within the application, but not from queries.

Lab Exercise

All Powder Board and Ski Data

In many cases, it is best to place functions and procedures inside of SQL. From this location, they are accessible by any query or form within the application. SQL Server stores user-defined functions directly in the database. You create them using SQL commands, but can find them using the Management Studio.



Activity: Create Sales Tax Function

Figure 7.2 shows the Transact-SQL code used to create a simple function to compute taxes. Note that you should include the Go command after most Transact-SQL Create statements to tell the system to store the new function.

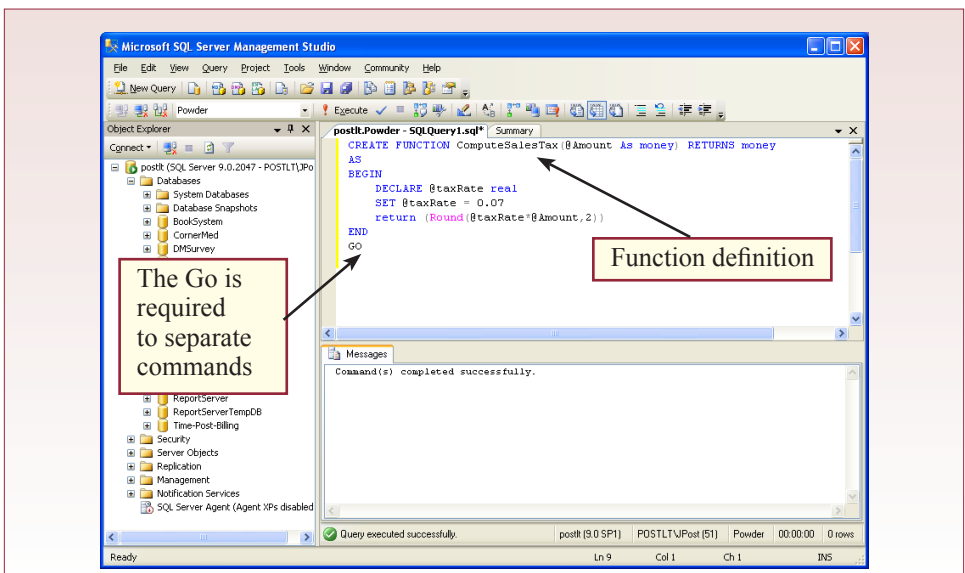
This command is particularly important when you write a batch file that includes several commands.

The tax calculation function is deliberately simple to highlight the process instead of the accounting rules. Be sure to use a variable for the tax rate, since it makes the code easier to understand, which reduces errors when someone tries to modify it later. Also, make sure you use the Round function to truncate the tax due at two decimal places. Run the commands to create the function. You can now use this function in queries and forms just as you would use any other function. A cool feature in SQL Server is that you can call a function without needing a FROM statement. However, you generally have to specify the full name of the function,

Action

Use SQL Server Management Studio to create the ComputeSalesTax function.
Test the function with an SQL statement.

Figure 7.2



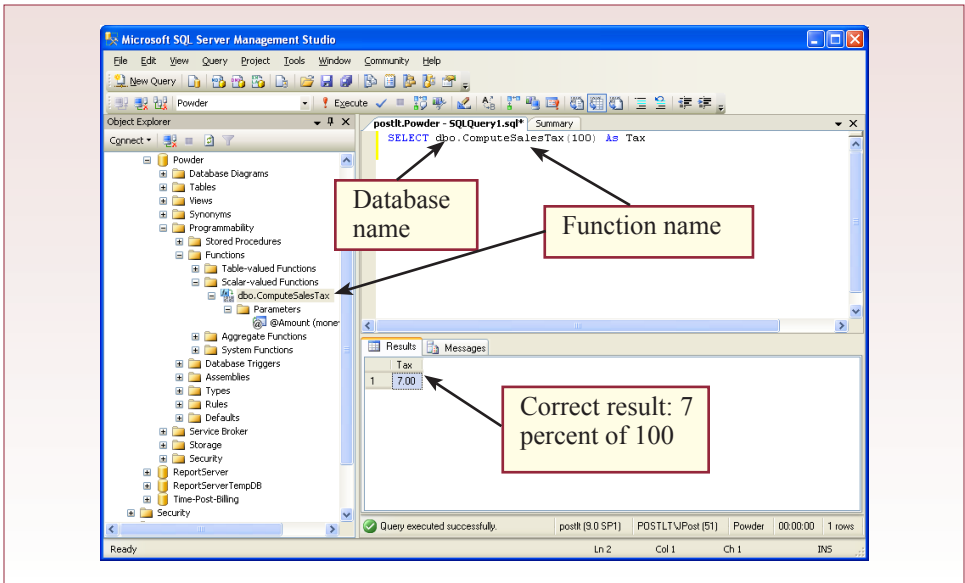


Figure 7.3

which includes the name of the database. In this case, you use `dbo.ComputeSalesTax` because `dbo` is the shortcut for database owner. Figure 7.3 shows the command and the correct result. You should be able to find the function in the object browser—that will give you the full name of the function if you have any questions. The function was tested directly using a fixed input value. Once you know the function works, you can use it in more complex queries and apply the function to data selected from tables.



Activity: Create Login Form and Connection Strings

The next step is to use the formula to automatically compute the sales tax on the Sale form. This process is a little more complicated since the form is in Visual Basic which is outside of SQL Server. Two basic methods can be used to access the function in the database: (1) compute the sales tax due for each item on the sale and then add the values, or (2) issue a separate function call to the database to compute the tax due. In some states, where some items are taxable and some are not, it might be best to use the first method. Basically, you would change the adapter for the subform so that it uses a query to return all of the columns from the `SaleItem` table along with a new column the computes the tax due for each item. You could then use a subtotal to compute the total tax due. The drawback to this approach is that you have to set the query and the data adapter properties carefully so that the dataset is updateable. You might even have to write a new update statement for the data adapter so that it does not try to update the sales tax column.

Action

Add a login form.

Edit the connection string in the app. config file to remove the values for Data Source, User ID, and Password.

Add code to the Menu form to open the Login form.

Add code to the Login button to modify the connection string and test the login.

The second method of issuing a new query to the database is a little easier and illustrates some techniques that are useful in other applications. The main step

Figure 7.4

will be to create a Visual Basic class that calls the SQL Server function to compute the taxes. This step causes a complication that needs to be addressed first. To access a database in .NET, you need to define a connection string that holds the username, password, and server name. In fact, you have probably already encountered problems with this requirement. By default, Visual Studio creates and stores this connection string in the app.config file. The good part is that the string is stored in only one location. The drawback is that every user will access the database using the same stored username and password. You can reduce this problem if you can use Windows authentication for your application, but you often need more flexibility.

You need a method for users to log in and enter their own username, password, and server. Then, enter these values into the global connection string that you can use throughout the application. Visual Studio 2005 makes it relatively easy to change the configuration settings, so all you really have to do is create a login form; and transfer the data to the connection string.

It is relatively easy to create a simple login form. As shown in Figure 7.4, just create a small windows form with three text boxes and a button. Be sure to include a label to display error messages and make sure you accurately name each text box (for example, UsernameTextBox). When users click the Login button, you want to make a quick test to see if the data entered is valid. If not, display an error message. Once a valid login has been achieved, you can close the form.

Now you need to put the Login form in your application someplace so that users are asked to login before using the other forms. Because most of the forms rely on the database, it makes sense to attach the form to the main menu form.

Figure 7.5

```
Private Sub Menu_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load

    Dim frmLogin As New Login
    frmLogin.ShowDialog()

End Sub
```

```

Private Sub btnLogin_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnLogin.Click
    ' Set the global connection string stored in the Settings.
    ' This code assumes you have modified the string so that it reads:
    ' connectionString="Data Source=;Initial Catalog=Powder;
    ' Persist Security Info=True;User ID=;Password="
    Dim sCn As String = My.Settings.AllPowderConnectionString.Replace _
        ("User ID=", "User ID=" & UsernameTextBox.Text)
    sCn = sCn.Replace("Password=", "Password=" & PasswordTextBox.Text)
    My.Settings.Item("AllPowderConnectionString") _
        = sCn.Replace("Data Source=", "Data Source=" & ServerTextBox.Text)
    ' Test the login with a simple query
    Try
        Dim cnn As New SqlClient.SqlConnection _
            (My.Settings.Item("AllPowderConnectionString"))
        cnn.Open()
        cnn.Close()
        ' MessageLabel.Text = "Successful login."
        Me.Close()
    Catch ex As Exception
        MessageLabel.Text = "Invalid login. Please try again."
        ' Note, add a counter and allow no more than three tries
    End Try
End Sub

```

Figure 7.6

Open the Menu form and switch to code view. Select the Menu Events in the object drop-down-list, and choose the Load event from the event list. Enter the two lines of code shown in Figure 7.5 to popup the Login form. Using the ShowDialog command will force users to deal with the login before proceeding.

The next step is to write the code for the login button that builds the connection string and tests the login data. Figure 7.6 shows the code for the button. You must first open the app.config file and edit the connection string that was built when you created the first Data Source. An easy approach is to simply delete the name of the Data Source, the Username, and the Password. Leave the tags with their equals signs (Data Source=). The code uses a simple string search and replace to add the three elements back into the connection object string using the values entered by the user. Note the use of the My.Settings object to retrieve the data stored in the app.config file.

Once the new string is created, and saved in the in-memory Settings object, the code calls attempts to open a connection to the database. If the connection fails, it displays an error message, and leaves the form displayed on the screen. Eventually, you will want to add more security options to the code, such as limiting the number of login attempts, recording login attempts to a security database, and closing the entire application on failure. You should save all of the new code and the forms and test the application. The login form should pop up when the application starts, and you should test correct data and incorrect data to ensure that the login works properly.



Activity: Add Tax Computation to the Sale Form

Finally, you are ready to compute the sales taxes from the Sale form. With the global connection string built, this process is relatively easy. You could perform the calculations inside the Sale form code, but it is better to put this code into its

own class to make it easier to find later and to be reusable across the application. Begin by creating a new Taxes.vb class. Right-click the project name/Add/New Item, choose Class and name it Taxes.vb. Create a function that will compute the taxes. The code in Figure 7.7 uses the SQL connection string. It creates a SQL SELECT statement to use the new tax function in the database. It then opens the connection and executes the command. The simple data reader retrieves the resulting value and returns it to the caller. Eventually, you should write a better error-handling routine. A user seeing that message would know that the tax value is wrong but would have no idea what to do next.

The logic of the code is straightforward: (1) Create a connection to the database. (2) Create a command to issue a SQL SELECT statement. (3) Add the total amount as a parameter in the query. (4) Open the connection and issue the command. (5) Retrieve the result using a data reader. (6) Close the reader and the connection. (7) Return the tax value to the caller.

Notice the use of the parameter (@Amount) in this example. In theory, you could have used string concatenation commands to build the SELECT statement without using parameters. However, for security reasons, you should always use formal parameters when creating queries with SQL Server. The reason is because SQL can be tricked with a SQL injection attack. A user could possibly enter a malicious command that would convert your simple SELECT command into something far more dangerous. You can find details on Microsoft's Web site, but the

Action

- Add a new Taxes.vb class.
- Add the parameterized code to compute the taxes.
- Edit the Sale form and switch to code.
- Select the editSalesTax box and Enter event.
- Add the code to compute the sales tax.
- Run the form to test the computation.

Figure 7.7

```
Public Class Taxes
    Public Shared Function ComputeSalesTax(ByVal Amount As Decimal) As Decimal
        Try
            Dim cnn As New _
                SqlConnection(My.Settings.AllPowderConnectionString)
            Dim cmd As New SqlCommand
            cmd.Connection = cnn
            cmd.CommandText = "SELECT dbo.ComputeSalesTax(@Amount) AS Tax"
            cmd.Parameters.Add(New SqlParameter("@Amount", Amount))
            Dim rdr As SqlDataReader
            cnn.Open()
            rdr = cmd.ExecuteReader(CommandBehavior.SingleRow)
            rdr.Read()
            Dim tax As Decimal = rdr.GetDecimal(0)
            rdr.Close()
            cnn.Close()
            Return tax
        Catch ex As Exception
            System.Windows.Forms.MessageBox.Show("Error computing taxes.")
            Return 0.0
        End Try
    End Function
End Class
```

```

Private Sub SalesTaxTextBox_Enter(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles SalesTaxTextBox.Enter
    Dim tax As Decimal = Taxes.ComputeSalesTax(Me.SubtotalLabel.Text)
    SalesTaxTextBox.Text = Format(tax, "0.00")
    Me.TotalLabel.Text = Format(CType(SubtotalLabel.Text, Decimal) _
        + tax, "0.00")
End Sub

```

Figure 7.8

problem applies to any DBMS. The key is to use parameters any time you need to place data into a SQL command.

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. However, the form has no good way to know when the sale is completed. Probably the easiest solution is to compute the sales tax due when the user clicks on the SalesTax box in the Sale form. For new orders, a simple click generates the correct value and the order total. You need to add a label (SaleTotalLabel) to display the order total.

You need to attach code to the form even when users enter into the tax box on the form. Open the Sale form and switch to code (View/Code). Choose the SalesTaxTextBox object in the drop-down-list and the Enter event in the other drop-down-list box. Using C# you should select the SalesTaxTextBox in Design view. Look at the properties window and click the Events button. Double-click the Enter event to create the desired function. Figure 7.8 shows the code that you need to execute for this event. Note that the code has been formatted to fit into the box. You need to type only the three lines of code.

Figure 7.9

The screenshot shows a Windows application window titled "Sale". It contains a form with various input fields and a table. The form fields include:

- Sale ID: 1002
- Sale Date: Friday, March
- Customer: Baldwin, Fernando (21)
- Employee: Miyahira, Hideharu 555
- Ship Address: 5516 Smallhouse Road
- Ship State: AL
- Ship City: Phenix City
- Ship ZIP: 36867

Below the form fields is a table with the following data:

	SKU	QuantitySold	SalePrice	Value
▶	600017	1	32.00	32.00
	600046	1	15.00	15.00
	800115	1	425.00	425.00
	800126	1	352.00	352.00
*				

At the bottom of the form, there are summary fields:

- Subtotal: \$ 824.00
- Sales Tax: 57.68
- Total: 881.68

The Payment Method is set to Cash.

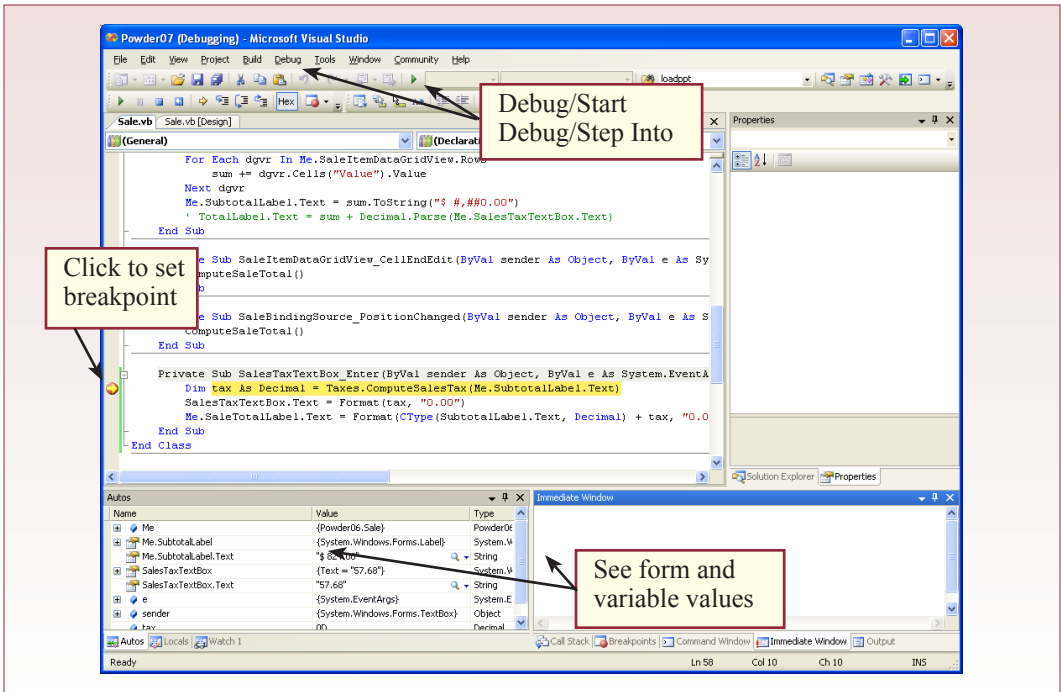


Figure 7.10

To test the code, run the form and create a new sale. Select an employee as the salesperson and choose the customer from the list. Check to ensure that the total also updates correctly—it should be the sum of the subtotal and the newly generated sales tax value. Think about the steps you performed to create this trigger, and consider why it is so important to use the global `ComputeSalesTax` function. If you had buried this calculation on the form, the next developer that had to change it could search for days trying to find the calculation. Of course, good documentation is also important. Figure 7.9 shows the Sale form that has been modified to place the payment data at the bottom of the form.

In real life, program code rarely runs correctly the first time. To find mistakes, you need to use the debugger. Open the Sale form and switch to code view. Click the gray column to the left of the line of code. Select `Debug/Start` from the main menu to start the application menu and open the Sale form. Click on the SalesTax box to trigger the code. The debugger will stop on the marked line. As shown in Figure 7.10, you can see the values stored on the form and any variables you might have created. You can single step through the code and evaluate the variables to see exactly how your code runs. Choose the Stop button on the toolbar to close the debugging session.



Activity: Update Inventory with Data Triggers

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 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. Yet, if you store this data separately, 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. With Visual Basic, you could handle this subtraction with form events. However, it is better to

use SQL Server and compute the subtraction using triggers on the data tables. These data triggers are simply code that is executed whenever a specified event occurs. The three events are DELETE, INSERT, and UPDATE. The main advantage to using SQL Server trigger code instead of Visual Basic form code is that with forms, someone might circumvent your code by writing new queries directly to the database.

The first step is to examine the tables and understand how they are related. You need to change the Inventory table whenever changes occur to the SaleItem table. The SaleItem table specifies the SKU value that matches exactly one row in the Inventory table. Also, when testing, remember that you need a matching entry in the Sale table to provide the SaleID key. You should look at some sample data in the three tables so you can enter consistent values to test. In this case, a new SaleID of 3000 to CustomerID 582 by EmployeeID 5 should work. SKU values of 500000 and 500010 both have an initial QOH of 10 units.

The next step is to think about the events that can occur and determine what they mean and how they will affect the QOH. It is easier to understand the process by considering one event at a time. Think about the first step in a sale. A row is entered into the Sale table: INSERT INTO Sale (SaleID, CustomerID, EmployeeID) VALUES (3000, 582, 5). You could enter the data for SaleDate and so on, but since this data is temporary, these three items are sufficient. Now, the next logical step that occurs in a sale is that the SKU for the item being purchased is entered into the SaleItem table: INSERT INTO SaleItem (SaleID, SKU, QuantitySold, SalePrice) VALUES (3000, 500000, 1, 100). At this point, the Quanti-

Action

Insert a new row into the Sale table with a SaleID of 3000, CustomerID of 582, and EmployeeID of 5.

Create the INSERT trigger for the SaleItem table.

Insert a new row into the SaleItem table (3000, 500000, 1, 100).

Check the value of QuantityOnHand in the Inventory table for SKU=500000 and ensure it was decreased from 10 to 9.

Figure 7.11

```
IF EXISTS (SELECT name FROM sysobjects
  WHERE name='NewSaleQOH' AND type='TR')
  DROP TRIGGER NewSaleQOH
GO
CREATE TRIGGER NewSaleQOH
ON SaleItem
FOR INSERT
AS
  UPDATE Inventory
  SET QuantityOnHand = QuantityOnHand - inserted.QuantitySold
  FROM Inventory INNER JOIN inserted
  ON Inventory.SKU = inserted.SKU
GO
```

tySold of one unit means that the system should subtract that value from the quantity on hand. To accomplish this task automatically, you need to establish an insert trigger on the SaleItem table. Figure 7.11 shows the SQL used to create this trigger. First, the code checks and deletes any existing trigger with that name to make it easier to fix a trigger. Second, the trigger is given a unique name. The next lines specify

Action

Delete the SaleItem row (SaleID=3000 And SKU=500000).

Check the quantity on hand.

Add the DELETE trigger.

Insert the SaleItem row again.

Check the quantity on hand.

Delete the SaleItem row.

Check the quantity on hand.

that the trigger should be fired when a row is inserted into the SaleItem table. The main body of the trigger is the UPDATE statement that subtracts the quantity sold from the quantity on hand. The UPDATE statement should look familiar, with a small twist. The twist is that it refers to the values being inserted into the SaleItem table using a reference to the inserted table. In triggers, SQL Server creates an inserted table to hold copies of the values that are being added. It also created a deleted table to hold values that are deleted—such as those being replaced with the UPDATE statement. The main UPDATE statement simply tells the database to subtract the new quantity sold from the existing quantity on hand for the SKU value just entered into the SaleItem table. When you have successfully created the trigger, issue the INSERT statement to add the row to the SaleItem table. Now verify that the QOH was modified with the query: SELECT SKU, QuantityOnHand FROM SaleItem WHERE SKU=500000.

You could continue to issue INSERT commands for different quantities, and the quantity on hand will decrease. Everything seems to be fine. However, what happens if there is a data entry error? Try deleting the row you inserted: DELETE FROM SaleItem WHERE SaleID=3000 And SKU=500000. Check the QOH in the Inventory table and you will see that it does not change. Why is that bad? Because the delete statement implies that the item was not actually sold, and since you have already subtracted the quantity, you need to add that value back to the QOH. In other words, you need another database trigger—one that fires when a row is deleted in the SaleItem table. Figure 7.12 shows the statement to create the trigger. This code is similar to the insert version. The only differences are that the quantity sold is added back to the quantity on hand, and the syntax uses a reference to the deleted. The deleted tables holds data the old version of data that is being

Figure 7.12

```
IF EXISTS (SELECT name FROM sysobjects
  WHERE name='DelSaleQOH' AND type='TR')
  DROP TRIGGER DelSaleQOH
GO
CREATE TRIGGER DelSaleQOH
ON SaleItem
FOR DELETE
AS
  UPDATE Inventory
  SET QuantityOnHand = QuantityOnHand + deleted.QuantitySold
  FROM Inventory INNER JOIN deleted
  ON Inventory.SKU = deleted.SKU
GO
```

deleted or replaced. In this case, there are no new or inserted values because the Delete command does not create anything. To test the new trigger, insert the SaleItem row again and check the quantity on hand. Now, delete the SaleItem row and check the quantity on hand again. It should be restored to its value before the latest INSERT command.

<p>Action</p> <p>Add the UPDATE trigger.</p> <p>Check the quantity on hand.</p> <p>Issue an update to change the QuantitySold in the SaleItem table.</p> <p>Check the quantity on hand.</p>

The two triggers you created are powerful tools. Once they have been defined, you never need to think about them. Anytime a process inserts or deletes a row, they are activated and inventory is changed immediately. You could test these actions using the Sale form, and you should see the same results. However, there is still something missing. One of the trickiest aspects to event programming is that you need to think hard about possible actions by users, and the consequences. In the inventory situation, what happens if a clerk goes back and changes a value? Originally, an SKU and quantity were entered, then the clerk sees an error or a customer changes his mind. Try it first with a change in quantity. Check the current value for QOH then insert the row to sell one unit. Check the QOH again to see that it was reduced by one, say from nine to eight units. Now, consider what if the customer actually purchased two units. Issue the statement to change the QuantitySold to two units: UPDATE SaleItem SET QuantitySold=2 WHERE SaleID=3000 And SKU=500000. Check the QOH and you will see that it still shows only one item was removed from inventory (eight units remaining instead of seven).

You need to add an UPDATE trigger to the SaleItem table to handle this problem. Figure 7.13 shows the code to create the trigger. Again, it uses a familiar UPDATE statement. However, check the use of the references to the deleted and inserted tables carefully. They contain the heart of the logic. The deleted values are the data that was stored in the SaleItem table before the update was initiated. The inserted values are the data in the row after it has been changed. In this case, the QuantitySold changed from one (deleted) to two (inserted). For the specified product SKU, this query adds the old value back and subtracts out the new value instead. Remember that a change in quantity means that the original subtraction

Figure 7.13

```

IF EXISTS (SELECT name FROM sysobjects
           WHERE name='ChangeSaleQOH' AND type='TR')
  DROP TRIGGER ChangeSaleQOH
GO
CREATE TRIGGER ChangeSaleQOH
ON SaleItem
FOR UPDATE
AS
  UPDATE Inventory
  SET QuantityOnHand = QuantityOnHand - inserted.QuantitySold + deleted.QuantitySold
  FROM Inventory INNER JOIN inserted
  ON Inventory.SKU = inserted.SKU
  INNER JOIN deleted
  ON Inventory.SKU = deleted.SKU
GO

```


was incorrect, so it is restored while the new value is subtracted. Again, you should test this trigger by checking the current QOH value, issuing an Update statement to the SaleItem table to change the quantity sold value, and then examine the new QOH to see that it holds the proper total.

If you look closely at the Update trigger code and think about the problem for a minute, you will see that one additional situation has to be handled. What happens if a clerk changes the SKU? In this case, you need to add the QuantitySold back to the original SKU item, then subtract the QuantitySold from the new SKU item. Of course, the QuantitySold might have been changed at the same time, so you need to be careful about which one you add and subtract.

Figure 7.14 shows a revised version of the update trigger. At this point, it is useful to point out the value of the existence query at the start of these statements. Since the trigger already exists, you cannot simply issue another CREATE statement with the same trigger name. This code checks to see if the trigger already exists and deletes it.

Notice the use of the IF statement to divide the trigger so that it handles the two cases separately. This code executes the appropriate set of commands depending on which column is being updated. The columns are identified by powers of two

Action

Create the full UPDATE trigger.
Check the quantity on hand.
Change the QuantitySold and SKU (to 500010) in the SaleItem row.
Check the quantity on hand for SKU 500000 and 500010.

Figure 7.14

```

IF EXISTS (SELECT name FROM sysobjects
  WHERE name='ChangeSaleQOH' AND type='TR')
  DROP TRIGGER ChangeSaleQOH
GO
CREATE TRIGGER ChangeSaleQOH
ON SaleItem
FOR UPDATE
AS
  IF (COLUMNS_UPDATED() & 4) > 0      /* 3rd column QuantitySold is changed */
  BEGIN
    UPDATE Inventory
    SET QuantityOnHand = QuantityOnHand - inserted.QuantitySold + deleted.QuantitySold
    FROM Inventory INNER JOIN inserted
    ON Inventory.SKU = inserted.SKU
    INNER JOIN deleted
    ON Inventory.SKU = deleted.SKU
  END
  IF (COLUMNS_UPDATED() & 2) > 0      /* 2nd column SKU is changed */
  BEGIN
    UPDATE Inventory          /* restore QOH for original SKU */
    SET QuantityOnHand = QuantityOnHand + deleted.QuantitySold
    FROM Inventory INNER JOIN deleted
    ON Inventory.SKU = deleted.SKU
    UPDATE Inventory          /* subtract for QOH for new SKU */
    SET QuantityOnHand = QuantityOnHand - inserted.QuantitySold
    FROM Inventory INNER JOIN inserted
    ON Inventory.SKU = inserted.SKU
  END
GO

```

(for example, 1, 2, 4, 8 for columns 1, 2, 3, 4). If you need the ninth or greater column, you need to use a different method that you can find in the SQL Server documentation. Notice that when the SKU is changed, the code has to issue two separate UPDATE commands: one to restore the QOH for the old SKU, and one to subtract to get the QOH for the new SKU. Finally, notice that the use of the COLUMNS_UPDATED command makes the entire trigger more efficient. If neither of these columns is being changed, the trigger falls through and does nothing. When possible, you should use similar conditions on most update triggers so they are only fired when absolutely necessary.

These three triggers should now handle all of the sales situations that affect the inventory quantity on hand. You should reset the QOH value and test all of the changes. In particular, in the SaleItem row change both the QuantitySold and the SKU.

Of course, if you have created purchase order and purchase item tables, you would have to add similar triggers to the purchase item table. The only difference is that a purchase adds quantity to the QOH instead of subtracting it, so you have to reverse the signs in the code.



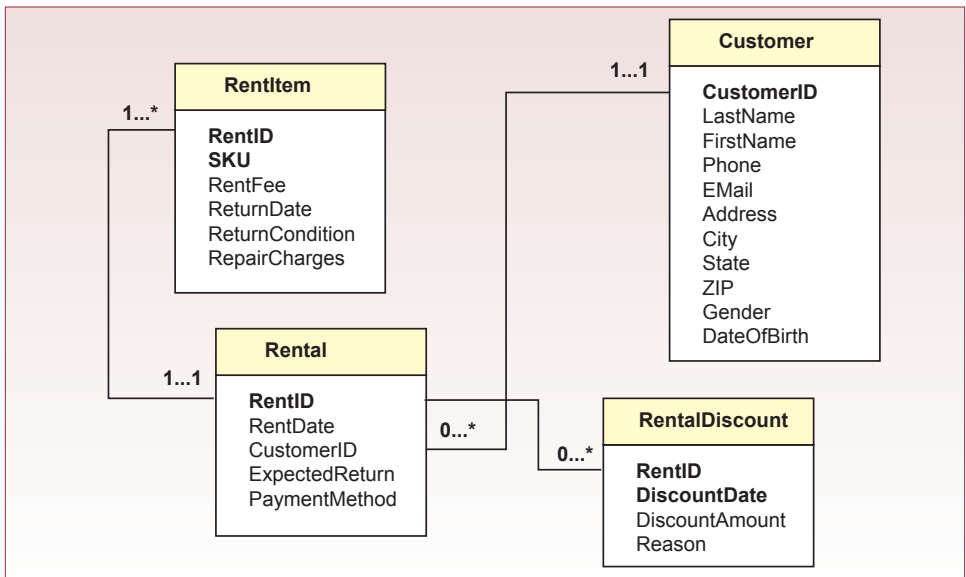
Activity: Define Transactions

Transactions consist of multiple changes that must succeed or fail together. One of SQL Server's strengths is its support to ensure that transactions are completed correctly. In particular, all changes are written to journal logs. If the system crashes in the middle of a transaction, the system can still recover the transactions that were interrupted or roll them back to the point where the

Action

- Create the Rental form.
- Create the RentalDiscount table.
- Create the Rental Discount form.
- Add the text boxes and button.
- Save the form.
- Add a button to the Rental form that opens the Discount form.

Figure 7.15



The screenshot shows a window titled "Rental" with a navigation bar at the top. The main form contains the following fields:

- Rent ID: 3862
- Rent Date: Friday, February 17, 2006
- Customer ID: Guthmiller, Asao (213-741-4453)
- Expected Return: Monday
- Subtotal: \$ 180.00
- Charges: \$ 16.00
- Total Due: \$ 196.00
- Payment Method: Cash

A "Give Discount" button is highlighted with a red box and an arrow pointing to it, with a callout box that says "Button to open discount form". Below the form is a table with the following data:

SKU	RentFee	ReturnDate	ReturnCondition	RepairCharges
100080	60.0000	2/18/2006	Good	0.0000
800375	60.0000	2/18/2006	Some damage	16.0000
800816	60.0000	2/19/2006	Good	0.0000
*				

Figure 7.16

changes began. The other important aspect of transactions is the ability to prevent or handle collisions of two processes altering the same data at the same time.

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 is simply wearing out and failing. She also notices 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 for employees who rent equipment. Since multiple discounts can be applied to a

Figure 7.17

The screenshot shows a window titled "RentalDiscount" with a form for recording a discount. The form contains the following fields:

- Rent ID: 3862
- Date: 12/7/2004
- Amount: 16.00
- Reason: Old equipment

A "Record Discount" button is visible. Three callout boxes provide additional information:

- "RentID and Amount are determined by the Rental form" (pointing to Rent ID and Amount fields)
- "Date defaults to today" (pointing to the Date field)
- "This is an unbound form built as a blank Window form" (pointing to the form area)

```

Private Sub DiscountButton_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles DiscountButton.Click
    Dim frmRentalDiscount As New RentalDiscount()
    frmRentalDiscount.Show()
    frmRentalDiscount.RentIDTextBox.Text _
        = Me.RentIDTextBox.Text
    frmRentalDiscount.AmountTextBox.Text _
        = Me.ChargesLabel.Text
End Sub

```

Figure 7.18

single rental, a new table is needed. Figure 7.15 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 Sale form. Figure 7.16 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 7.17 shows the RentalDiscount form. It is built from Design view and not tied to the database. Add the text boxes by hand. Set the default value on the date field to Today, so the current date and time are entered by default.

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 7.18 shows the code used on the Rental form button click event. The values are transferred to publicly shared variables, which will be loaded when the Discount form opens.

Figure 7.19 shows the code for the two trigger events in the discount form. The form trigger is fired when the form first opens, so it retrieves the values stored in the two global variables and places them onto the form as the default values. The second code is triggered when the button is clicked to save the changes. First, it zeros any charges for that rent, and second, it adds a row to the new discount table to record the rental, the date of this action, the amount of the discount, and the rea-

Figure 7.19

```

Button: Click
UPDATE RentItem SET RepairCharges=0
WHERE RentID = @RentID;

INSERT INTO RentalDiscount(RentID, DiscountDate, DiscountAmount,
Reason)
VALUES (@RentID, @DiscountDate, @DiscountAmount, @Reason);

MessageLabel.Text = "Changes recorded"

```

son for the discount. In terms of business policy, the first step might be overkill. It is possible that the rental manager would want to give only a partial discount to the customer. If so, he can first run this routine, then return to the Rental form and enter the remaining value of any charges. If this activity is common, you should change this form and code so that only the partial amount is subtracted from the charges.

Figure 7.20 shows the detailed code and one more important addition to the button code that handles the discount. What happens if something goes wrong between the two DML commands? The Try/Catch code traps all errors and rolls back any changes made. Without this code, it is possible for the UPDATE command to change the value to zero and then the INSERT command could fail and it would never record the reason for the change. With the exception handling and the transaction code, both changes will commit or fail together.

Figure 7.20

```

Private Sub RecordButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles RecordButton.Click
    Dim cnn As New SqlConnection(My.Settings.AllPowderConnectionString)
    Dim cmd As New SqlCommand
    Dim trn As SqlConnection.SqlTransaction
    Dim sSQL As String
    cnn.Open()
    trn = cnn.BeginTransaction()
    Try
        cmd.Connection = cnn
        cmd.Transaction = trn
        sSQL = "UPDATE RentItem Set RepairCharges=0"
        sSQL &= " WHERE RentID=@RentID"
        cmd.CommandText = sSQL
        cmd.Parameters.Add(New SqlParameter("@RentID", _
            Me.RentIDTextBox.Text))
        cmd.ExecuteNonQuery()
        sSQL = "INSERT INTO RentalDiscount(RentID, DiscountDate,
            DiscountAmount, Reason)"
        sSQL &= " VALUES(@RentID2, @DiscountDate, @DiscountAmount, @Reason)"
        cmd.CommandText = sSQL
        cmd.Parameters.Add(New SqlParameter("@RentID2", _
            CType(Me.RentIDTextBox.Text, Integer)))
        cmd.Parameters.Add(New SqlParameter("@DiscountDate", _
            CType(Me.DateTimeTextBox.Text, Date)))
        cmd.Parameters.Add(New SqlParameter("@DiscountAmount", _
            CType(Me.AmountTextBox.Text, Decimal)))
        cmd.Parameters.Add(New SqlParameter("@Reason", _
            Me.ReasonTextBox.Text))
        cmd.ExecuteNonQuery()
        cmd.Transaction.Commit()
        MessageLabel.Text = "Changes recorded."
    Catch ex As Exception
        cmd.Transaction.Rollback()
        MessageLabel.Text = "Error saving changes."
    Finally
        cnn.Close()
    End Try
End Sub

```

There is one more complication with the Rental form. If you test the Discount form, you should see the problem. Even when the Discount form successfully saves the changes, the new values are not displayed on the main Rental form. The reason is because the Rental form works with a partial copy of the data that is held in a dataset in memory.

The Discount form wrote the changes directly to the database. For the form to pick up the changes, you have to do one of two things: (1) have your code also update the in-memory dataset, or (2) have the Rental form re-query the database to get the current values. It is easier and somewhat safer to re-query the database. The only twist is that it is safer to ask the user to do the refresh manually. If your code does it automatically, it might throw away changes that the user wants to save. So, add a button to the Rental form and add two lines of code:

```
Me.RentItemTableAdapter.Fill(Me.RentalDataSet.RentItem)
ComputeRentalTotal()
```

Action

Edit the Rental Discount form.

Add the specified code to the Click event.

Add the Refresh button and code to the Rental form.

Test the forms.

Note that the Fill command was written by the wizard and it will discard any changes that were not saved earlier with the Save button.

Finally, note that it is also possible to write the transaction update code inside of SQL Server as a stored procedure. The Visual Basic code is then simplified because it simply calls that procedure and passes in the RentID, Date, Amount, and Reason as parameters. The transaction processing is handled at the database level. An example of this type of code is presented in the following sections.

Database Cursors, Keys, and Locks



Activity: Read Rows of Data

Direct SQL commands are useful for DML issues where you need to change or delete rows of data. When you need program code to examine several rows of data, database cursors are the answer. 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 7.21 shows the query. Note that you need to format the SaleDate using the DatePart function with a format of “ww” to get the number of the week. Make sure you compute the Sum of the price times quantity and that the total is computed for each week with

Action

Create a new query.

Tables: Sale and SaleItem.

Create column DatePart(“ww”, SaleDate)
As SaleWeek

Create column QuantitySold*SalePrice
AS Value.

Sum the Value column by week.

```

CREATE VIEW WeeklySales AS
SELECT DatePart("ww", SaleDate) AS SaleWeek, Sum(QuantitySold*SalePrice) As Value
FROM Sale INNER JOIN SaleItem
ON Sale.SaleID=SaleItem.SaleID
WHERE SaleDate Is Not Null
GROUP BY DatePart("ww", SaleDate)
go

```

Figure 7.21

the GROUP BY clause. A couple of entries have missing dates, so they can be removed from this query. Use the CREATE VIEW line at the top to save the query, but make sure you test the query before you add this line.

The next step is to compute the percentage change between the rows. The code for this step will be created within the AvgPercentWeeklyChange function stored as a Transact-SQL function in the database. Eventually, you can add a button and result box to a form to display the computation, but it is better to place the code in the database to make it easier to access from any form, query, or report.

Figure 7.22

```

IF EXISTS (SELECT name FROM sysobjects
  WHERE name = 'AvgPercentWeeklyChange' AND type = 'FN')
  DROP FUNCTION AvgPercentWeeklyChange
GO
CREATE Function AvgPercentWeeklyChange ( )
  RETURNS float
AS
BEGIN
  DECLARE c1 CURSOR FORWARD_ONLY READ_ONLY FOR
    SELECT SaleWeek, Value FROM WeeklySales ORDER BY SaleWeek
  DECLARE @Avg1 float
  DECLARE @N int
  DECLARE @PriorValue money
  DECLARE @cWeek int, @cValue money
  SET @Avg1 = 0
  SET @N = 0
  SET @PriorValue = -1.00
  OPEN c1
  FETCH NEXT FROM c1
    INTO @cWeek, @cValue
  WHILE @@FETCH_STATUS = 0
  BEGIN
    IF @PriorValue > 0
    BEGIN
      SET @Avg1 = @Avg1 + (@cValue - @PriorValue)/@PriorValue
      SET @N = @N + 1
    END
    SET @PriorValue = @cValue
    FETCH NEXT FROM c1
      INTO @cWeek, @cValue
  END
  CLOSE c1
  DEALLOCATE c1
  RETURN (@Avg1/@N)
END
go

```

Define the SELECT statement for the cursor to trace through

Create variable to hold the value from the previous row

Skip the first week because there is no prior value

Compute the percent change and keep a running total

Save the current row value and move to the next row

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 7.22 shows the main code. The SQL statement is opened as a cursor, which retrieves one row of data at a time using the loop. The Avg1 variable keeps the running total of the percentage increase, while N counts the number of operations. The role of the PriorValue variable is the most important. At the end of the 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 (PriorValue) value. This trick is useful for many cursor-based programs, so you should study the code until you understand it. Use a basic SELECT statement to test the function in the package. Depending on the actual values in your database, the result should be about 16 percent. Note that this routine does not quite provide the detail Katy wants, but it is straightforward to restrict the query using starting and ending week parameters and call the function twice.

Action

Create the AvgPercentWeeklyChange function.

Use SQL to call the function: SELECT AvgPercentWeeklyChange.



Activity: Generate and Use Keys

SQL Server uses an Identity to generate unique key values. For the most part, you have to specify the Identity property for a table at the time you create the table. You have the ability to specify a starting value and an increment for each identity. As shown in Figure 7.23, most people stick with the default values of 1 and 1.

Action

Create the new TestIdentity table.

Insert a row and display the generated key value.

Insert a row with an existing key value by temporarily turning off the identity.

Insert a row without a key value and display the generated value.

Once you have assigned the Identity property to a table, the DBMS will automatically generate new values for you. For many operations, such as when clerks enter data into a form to create a new customer, this process is invisible and painless. However, sometimes you will need to know what value was just created. For instance, you might use Transact-SQL code to create a new customer, then enter the newly-generated ID into a Sale table. Technically, Transact-SQL has three methods to return a newly-generated ID value. Note that in all three cases, the value is not generated until after the row is actually inserted into the table. The three methods are: (1) SCOPE_IDENTITY, (2) @@IDENTITY, and (3) IDENT_CURRENT. Many books and online articles recommend the second method (@@

Figure 7.23

```
CREATE TABLE TestIdentity
(
  CID          int Identity(1,1),
  LastName     nvarchar(50) null,
  FirstName    nvarchar(50) null,
  CONSTRAINT pk_TestIdentity PRIMARY KEY (CID)
)
```



```
INSERT INTO TestIdentity (LastName, FirstName)
VALUES ('Jones', 'Joe');
SELECT SCOPE_IDENTITY();
```

Figure 7.24

IDENTITY), but you should avoid this approach. As shown in Figure 7.24, the proper method is to use the SCOPE_IDENTITY. In this example, it should return the value of 1 as the first key. The difference between the three approaches is technical, but you should understand the main problem with @@IDENTITY so that you learn to avoid using it. The problem is that @@IDENTITY returns the last generated key value—regardless of the table or scope. It can cause serious problems when the database has several triggers that insert rows into multiple tables.

Figure 7.25

```
SET IDENTITY_INSERT TestIdentity ON;
INSERT INTO TestIdentity (CID, LastName, FirstName)
VALUES (10, 'Brown', 'Bobbie');
SELECT SCOPE_IDENTITY();
SET IDENTITY_INSERT TestIdentity OFF;
```

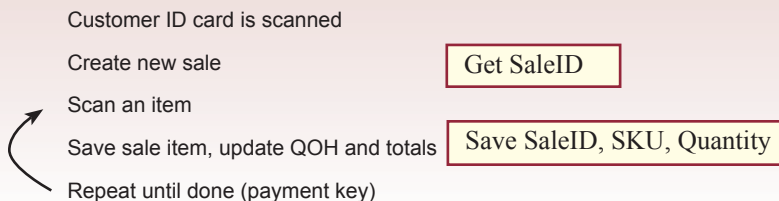
For example, you could add a trigger to the first table (TestIdentity) that causes a row to be inserted into a second table (say Customer). The @@IDENTITY variable will return the Identity value for the last table in the chain (Customer) instead of the one you expected (TestIdentity). The problem is particularly hard to spot when someone else writes the trigger code later and you do not know about it.

Action

- Create the SQL NewSale Procedure.
- Create a new form with no data.
- Add boxes for CustomerID, EmployeeID, SKU, and txtSaleID as the generated key.
- Create a command button and add the indicated code.
- Test the form.

Identities present another problem. You cannot alter a table later and add an identity value with SQL. However, in an emergency, you can change the table definition in the Enterprise Manager to add an Identity property to a table. But, the Enterprise Manager goes through some extreme steps to make this change, so you want to avoid it. Essentially, it creates an entirely new table, copies the data from the old table and rebuilds all relationships. Because of this complication, you should always try hard to add Identity properties when you design the tables. If you really have to go back and add them later, make sure you do it when no one else is using the system.

Figure 7.26



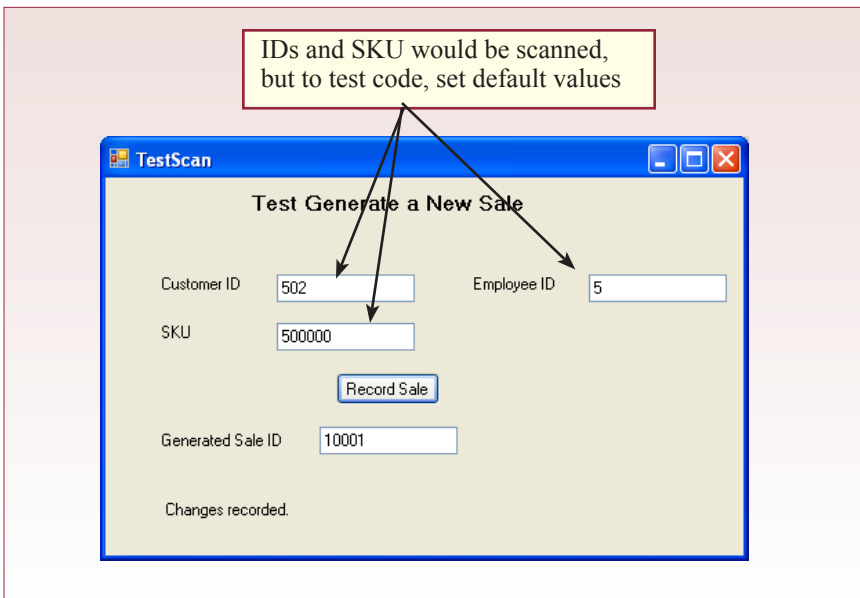


Figure 7.27

You are likely to face one more problem with Identities. What happens when you want to add existing data to a table with an Identity property? In most cases, you would like to keep the original key values in the existing data. Otherwise, your relationships might not be valid. If you are using SQL INSERT statements,

Figure 7.28

```

CREATE PROCEDURE NewSale(@CustomerID int, @EmployeeID int,
    @SKU nvarchar(50), @NewSaleID int OUTPUT)
AS
BEGIN
    INSERT INTO Sale(CustomerID, EmployeeID, SaleDate)
    VALUES (@CustomerID, @EmployeeID, GETDATE())

    DECLARE @tmpSaleID int
    SET @tmpSaleID = (SELECT SCOPE_IDENTITY())

    DECLARE @tmpListPrice money
    SET @tmpListPrice = (SELECT ListPrice
    FROM Inventory INNER JOIN ItemModel
    ON Inventory.ModelID=ItemModel.ModelID
    AND SKU=@SKU)

    INSERT INTO SaleItem(SaleID, SKU, SalePrice, QuantitySold)
    VALUES (@tmpSaleID, @SKU, @tmpListPrice, 1);

    SET @NewSaleID=@tmpSaleID
END
go
-- Use the following commands to test the function:
DECLARE @NewSaleID int
EXEC NewSale 502, 5, N'500000', @NewSaleID OUTPUT
SELECT @NewSaleID

```

you tell the DBMS to keep the original values with the `IDENTITY_INSERT` command. Of course, existing data presents problems when you are using identities. You must make sure that your generated Identity values can never duplicate an existing value. When you know that you will be importing data, make sure that you specify a starting value substantially higher than any existing key value.

Integrating the key generation with forms is an extension of this process. The trick is to write a Transact-SQL function to handle the insert and key generation. Consider a case where you need custom code to generate each sale and enter the sale items. For example, perhaps you have a bar-code scanner and want to automate as much of the checkout process as possible.

Figure 7.26 outlines the basic events that will occur. Notice that when the new Sale is created, the Identity value will be created automatically. The catch is that you need to get this value so that you can save it in the `SaleItem` table for each scanned item.

Now consider the issue of the bar-code scanner. 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 7.27 shows a sample form with default values that will work. Add a command button and a text box to display the `SaleID` that will be generated within the code.

The main trick in this example is to write the SQL code within a Transact-SQL procedure. Figure 7.28 shows the code that runs the process. First, the row is inserted into the `Sale` table using the current date. This insertion automatically

Figure 7.29

```

Private Sub RecordSaleButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles RecordSaleButton.Click
    Dim cnn As New _
        SqlConnection(My.Settings.AllPowderConnectionString)
    Dim cmd As New SqlCommand
    Try
        cnn.Open()
        cmd.Connection = cnn
        cmd.CommandType = CommandType.StoredProcedure
        cmd.CommandText = "NewSale"
        cmd.Parameters.Add(New SqlParameter("@CustomerID", _
            Me.CustomerIDTextBox.Text))
        cmd.Parameters.Add(New SqlParameter("@EmployeeID", _
            Me.EmployeeIDTextBox.Text))
        cmd.Parameters.Add(New SqlParameter("@SKU", _
            Me.SKUTextBox.Text))
        Dim prmSaleID As New SqlParameter("@NewSaleID", 0)
        prmSaleID.DbType = DbType.Int32
        prmSaleID.Direction = ParameterDirection.Output
        prmSaleID.SqlDbType = SqlDbType.Int
        cmd.Parameters.Add(prmSaleID)
        cmd.ExecuteNonQuery()
        Me.GeneratedIDTextBox.Text = prmSaleID.Value
        MessageLabel.Text = "Changes recorded."
    Catch ex As Exception
        MessageLabel.Text = "Error saving changes."
    Finally
        cnn.Close()
    End Try
End Sub

```

generates a new key value. Second, this generated value is retrieved and held in a temporary variable. Third, the list price of the item being scanned is retrieved and also placed into a temporary variable. Finally, a row is added to the SaleItem table using the generated SaleID value to link it to the Sale table and the retrieved list price so clerks do not have to memorize prices. You should add exception handling to the code in case anything goes wrong.

The last step is to display the newly generated SaleID on the form so you can see it. You should be able to use a SELECT command to retrieve the inserted Sale and SaleItem values. You could also use the Sale form and search for the new sale. Of course, you could modify the code to handle multiple items being scanned, along with a screen to add the payment data, but they are not needed at this point.

Visual Basic code can call a SQL procedure directly. Figure 7.29 shows that most of the steps are similar to those used earlier. Make sure you declare the CommandType as a stored procedure. Input parameters can be added directly by specifying the name and the value. The output parameter is used to return the generated key value. If you need this value in your Visual Basic code, you must declare the parameter as a separate variable. You can then use its Value property to get the returned value and use it in other sections or display it on the form.



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 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.

Action

Create a blank new form.

Add a text box for Customer ID.

Add a text box to enter a new ZIP Code.

Create a button and add the indicated code for it.

Test the form.

Use the data wizard to create a second form that displays CustomerID and ZIP Code in a tabular list.

Create a new form that is not bound to the database. As shown in Figure 7.30, add a box to select a customer. You should consider adding an Combo box for

Figure 7.30

The screenshot shows a Windows-style application window titled "LockTest". Inside the window, the text "Record Locking Tests" is centered at the top. Below this, there are two text boxes. The first is labeled "Customer ID" and contains the number "1". The second is labeled "New ZIP Code" and contains the number "95839". At the bottom center of the form area, there is a button with the text "Change ZIP Code". The window has standard Windows controls (minimize, maximize, close) in the top right corner.

```

CREATE Procedure UpdateZIP(
    @CustomerID int,
    @NewZIPCode nvarchar(20),
    @ErrorCode int OUTPUT)
AS
BEGIN
    UPDATE Customer
    SET ZIP = @NewZIPCode
    WHERE CustomerID = @CustomerID
    SET @ErrorCode = @@ERROR
END

```

Figure 7.31

practice, but it is not required since you will be able to find the ID in a second form. 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.

The actual code to change the zip code will be written as a Transact-SQL procedure. As shown in Figure 7.31, the procedure uses a straightforward UPDATE statement. The one catch is that the UPDATE command might fail. If it does, Transact-SQL records an error value in the @@ERROR variable. You should test this value after every SQL command, particularly data manipulation commands. In this example, the procedure does not know how to handle an error, so it simply returns the error value to the calling program. If the value is nonzero, an error has occurred. Test this new form to ensure it works.

The next step is to write the standard code to call the UpdateZIP procedure from Visual Basic. Figure 7.32 shows the code. Be sure to include the code to create the output parameter so the VB code can retrieve any error values. Since it is not clear what errors might arise, the code simply displays a message to the user. By now, this code should be familiar. The only tiresome part is creating the parameters, but you can copy the code that you created earlier and edit it to save time.

You need two processes changing 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 data grid Wizard to create a form based on the Customer table showing CustomerID, LastName, FirstName, ZIP, Phone, and EMail. Choose the tabular layout so you can see several rows at one time. Figure 7.33 shows the basic form. Keep the form small so you can display it on the screen along with the TestLock form. Test the form by resetting the zip code for the first customer.

The update code written by the Visual Studio data wizard supports optimistic locking. These forms will illustrate how the system works. Open both of the new forms. Begin with the List form, click on the ZIP code for the first customer and change the last digit. Do not click the Save button yet, so that the changes are only made to the internal dataset. Switch to the LockTest form, enter the Customer ID

Action

Open both forms so they are both visible on the screen.

In the list form, change the last ZIP digit but do not save the changes.

In the test form, enter the same Customer ID and a different ZIP code, then click the Save button.

Return to the list form and save the changes.

You should receive a concurrency error message.

```

Private Sub btnChangeZIP_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnChangeZIP.Click
    Dim cnn As New SqlConnection(My.Settings.AllPowderConnectionString)
    Dim cmd As New SqlCommand
    Try
        cnn.Open()
        cmd.Connection = cnn
        cmd.CommandType = CommandType.StoredProcedure
        cmd.CommandText = "UpdateZIP"
        cmd.Parameters.Add(New SqlParameter("@CustomerID", _
            Me.CustomerIDTextBox.Text))
        cmd.Parameters.Add(New SqlParameter("@NewZIPCode", _
            Me.NewZIPCodeTextBox.Text))
        Dim prmErrorID As New SqlParameter("@ErrorCode", 0)
        prmErrorID.DbType = DbType.Int32
        prmErrorID.SqlDbType = SqlDbType.Int
        prmErrorID.Direction = ParameterDirection.Output
        cmd.Parameters.Add(prmErrorID)
        cmd.ExecuteNonQuery()
        If (prmErrorID.Value <> 0) Then
            MessageLabel.Text = "Error in database."
        Else
            MessageLabel.Text = "Changes recorded."
        End If
    Catch ex As Exception
        MessageLabel.Text = "Error saving changes."
    Finally
        cnn.Close()
    End Try
End Sub

```

Figure 7.32

for the same customer you selected in the list form (the ID is probably 1). Enter a new ZIP in the text box, then click the button to submit the changes. You should not receive any error messages, and the change will be written to the database. Now, return to the CustomerList form and click the Save button. This action will attempt to write internal changes to the database, but it will generate a concurrency violation error message. If you look through the code written by the wizard, you will see that the update command includes a WHERE clause that compares the value original read from the database. Consequently, the form cannot update a column where the data has been changed by a different process.

The code written by the wizard does a good job of catching the concurrency error. However, it does nothing to help resolve the problem. But, finding a general solution to the problem is difficult. Think from the perspective of the user looking at a list of customer data. Remember that the list is held in memory until the Update button is clicked, so if there are several users, the chance of a conflict could be high. On clicking the Update button, you receive a notice that someone else has modified the data since you first retrieved it. (At a minimum, the error message has to be rewritten so it clearly explains what happened.) What do you want to do now? Ultimately, you have only two choices. (1) You could ignore your changes and reload the dataset to pick up the change made by the other person, or (2) you could force your new value to overwrite the other change. In general, it would be nice to see the other change before you make a decision.

CustID	LastName	FirstName	ZIP	Phone	EMail
0	Walk-in				
1	Jones	Jack	95838	111-222-3333	JonesJ202@msn.com
2	Sanchez	Paul	95838	111-444-9999	SanchezP844@msn.com
3	Garner	Chad	60601	213-080-4599	GarnerC73@msn.com
4	Reeves	Gil	35401	213-186-6502	ReevesG690@msn.com
5	Hicks	Evelyn	7003	213-959-5499	HicksE808@msn.com
6	Grimes	Ernest	55420	312-817-7845	GrimesE460@msn.com
7	Rice	Charlotte	94025	312-608-6819	RiceC65@msn.com
8	Marlow	Jerry	45202	213-606-0452	MarlowJ674@msn.com
9	Rogers	Robin	38138	213-149-9519	RogersR135@msn.com
10	Riggs	Harriet	23232	213-584-6864	RiggsH590@msn.com
11	Grimes	Greg	92077	213-288-6416	GrimesG964@msn.com
12	Ken	Wilkes	79701	213-451-2006	KenW245@msn.com

Figure 7.33

From a programmer's perspective, the simplest solution is to leave the existing changes in the database and retrieve the new values. Then, the user can decide whether to keep them, or re-enter new data. All you have to do is add a requery button on the form and use the code from the original Load button. This approach may not be perfect, because it throws away the user's change, but for small updates, this concern is not critical.

Visual Basic datasets have a feature that makes it possible to provide more sophisticated concurrency error handling. You can write code that automatically retrieves the new values entered into the database and holds them as alternate data in the dataset. Both the new value and the user's change can be held simultaneously. You could then include a button on the form so the user can switch back and forth

Action

Add a requery button to the CustomerList form.

Test the button when concurrency violations arise.

Add the concurrency test to the procedure code and test it.

Figure 7.34

```
CREATE Procedure UpdateZIP(
    @CustomerID int,
    @NewZIPCode nvarchar(20),
    @OldZIPCode nvarchar(20),
    @ErrorCode int OUTPUT)
AS
BEGIN
    UPDATE Customer
    SET ZIP = @NewZIPCode
    WHERE CustomerID = @CustomerID
    AND ZIP = @OldZIPCode
    IF (@@ERROR = 0) AND (@@ROWCOUNT < 1)
        SET @ErrorCode = -1
    ELSE
        SET @ErrorCode = @@ERROR
END
```

between the two values and choose which one to use. This approach is more user friendly, but the coding is a little tricky. Reading the data and holding both values in memory is straightforward. Your code examines an item in a data row within a dataset you can use the `DataRowVersion.Original` and `DataRowVersion.Current` indicators to specify which of the values you want to see. However, when you retrieve new values, you have to watch out for deleted rows and maintaining the primary key properties. Also, the code for creating a swap button is beyond the scope of this book. With some additional experience, you can create a more sophisticated error handling code. The goal is to automate as many steps as possible and make the job of the user easier.

Notice that the original test form with the `UPDATE` statement does not have any concurrency control. This code will ignore any other changes and simply write the new value to the database. Sometimes, this approach is useful, but you should understand how to add optimistic checking to the code in case you need it for a more complex project.

Optimistic concurrency is easy to implement in an SQL `UPDATE` statement. Simply add one more condition to the `WHERE` statement: `AND ZIP = @oldZIP`. Of course, you need to add the `@oldZIP` parameter and your form needs to keep an original value around so that it can be passed to the procedure. Since the zip code update form does not need to read or keep original values, there is no reason to implement optimistic concurrency testing. But, if you wanted to do so, you would have to examine the `@@ERROR` code and the `@@ROWCOUNT` variables after the `UPDATE` statement to ensure that no error arose and that at least one row was updated. As shown in Figure 7.34, failure of either of these conditions would generally indicate a concurrency violation, and you could return a special error code to the calling program.

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 continually watch the supplier bills to ensure that they are only billed 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 a product category and metal, 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 often ships orders to three states, each of which charge different sales tax rates. 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 source query using code.) Create a text box so Laura can enter an average rating as a cut-off 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 have 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 clean-up 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 the 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, 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. Mostly they should 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, assess 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.

Applications

Chapter Outline

Applications, 150

Case: All Powder Board and Ski Shop, 150

Lab Exercise, 151

All Powder Board and Skip Shop Application, 151

Exercises, 164

Final Project, 165

Objectives

- Build applications that connect forms and reports.
- Add toolbars and menus to forms.
- Add Help files to the database application.
- Deploy the 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 showed you how to build 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. Eventually, you will have forms that store data into each of the tables in the relationship diagram. 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

The figure illustrates the integrated forms and reports for the All Powder Board and Ski Shop application. It shows four main components:

- Menu:** A sidebar menu with options like Customer, Sale, and Customer Sales Totals.
- Startup form:** A form with a 'Customer' button and a 'Customer Sales Report' link.
- Help files:** A window titled 'Introduction to the All Powder Board and Ski Shop' providing user guidance.
- Sale form:** A form for entering sales data, including fields for Sale ID, Customer, Employee, Ship Address, and a table of items.
- Report Manager:** A window displaying a report titled 'Customer Sales' with a table of sales data.

The 'Sale' form contains the following data:

SKU	Quantity Sold	Sale Price	Value
600017	1	32.00	32.00
600046	1	15.00	15.00
800115	1	425.00	425.00
800126	1	352.00	352.00

The 'Report Manager' window displays the following report data:

Sale ID	SKU	Quantity Sold	Sale Price	Value
12-Jan-2006	Check		Chattanooga TN	
1316		3		506.00
	100115	1	308.00	308.00
	600016	1	32.00	32.00
	800419	1	166.00	166.00
10-May-2006	Cash		Chattanooga TN	
1376		4		1,153.00

Lab Exercise

All Powder Board and Skip Shop 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.

It is possible to create a template form for use in Visual Studio. However, you cannot use the data-form wizard with a custom template. Because experienced developers rarely use the form wizard, this limitation is rarely an issue. A template is basically a blank form that contains standard design elements. You simply create a new blank form, add the logo, title, menu bar, and any code that will apply to all forms. Save the form with a name that everyone will recognize. Unfortunately, you cannot apply a template to an existing form. Instead, you make a copy of the template form, and then build the form using that base. The goal of a template form is to make it easier to create a standard look-and-feel for all input forms (and reports). Templates are particularly helpful in a project with multiple developers.



Activity: Create a Template Form

Applications need a consistent look and feel. Forms should have common elements, such as a title, menu, a Close button, a message area, and a standard location for the data elements. Many applications also include a company logo and possibly text recommended by the legal department. All of these elements should appear in consistent colors and locations on the form. When

you built forms in Chapter 6, you probably spent most of your time just getting the data elements to work the way wanted. Now, you have to pay attention to colors and formats. Actually, you would normally create a standard template before you build the first forms. You cannot apply templates directly to an existing form. Instead, you have to create a new form based on the template and add the data elements to it. But, you had enough things to worry about in Chapter 6, so templates were deferred.

You create a template file by building a blank form and adding the elements that are common to all forms. Typically, these elements include a logo, a form title, a close button, and a label for messages to be displayed to the users. So, right-click the project name and add a new blank form. Rename it as `TemplateForm` to help you remember its purpose. Find an image logo file and place it on the new form. Add a label for the title. Set its name to `FormLabel` and enter `Form Name` as the text. Set it to a slightly larger font (such as 10 point and bold). Add a label at the bottom of the form and set its `Name` property to `MessageLabel`. Remove the

Action

Add a new form.

Name it `TemplateForm`.

Set the `BackColor` to `Window (white)`.

Add a logo file.

Add a form title.

Add a Close button.

Add a `MessageLabel` at the bottom.

text, which will make it invisible, but harder to select. Add a button to the form and set its name to CloseButton. Double-click the button and set the code to close the form:

```
Me.Close( )
```

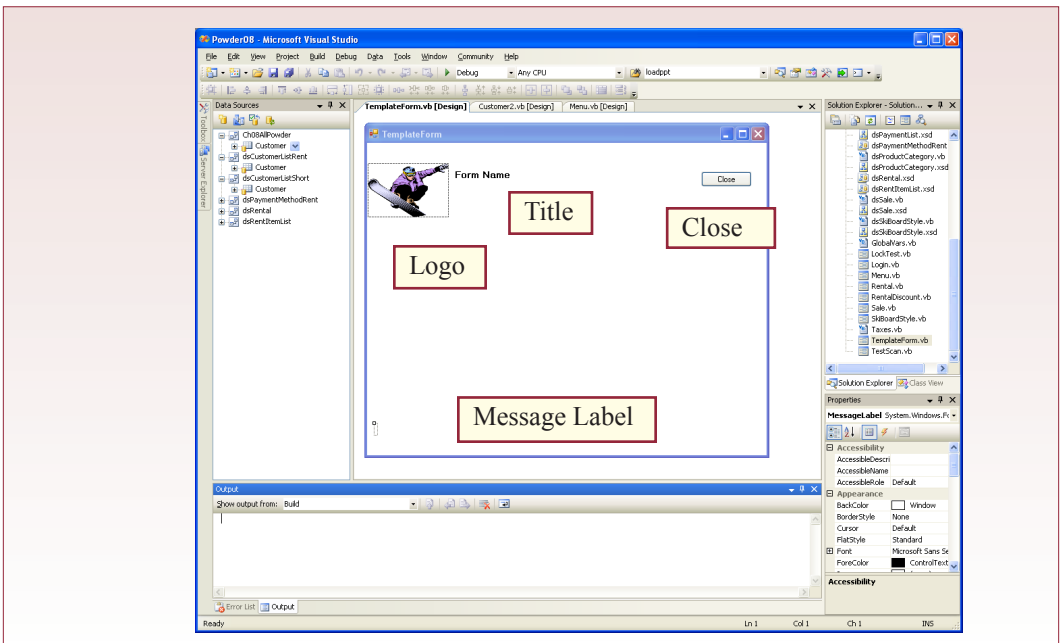
Save the new template form. If you add complex code to the template form, you should test it by itself. You can temporarily set TemplateForm as the startup form in the project's properties. In general, this form will not be opened directly, but is used to create new forms. Figure 8.2 shows the basic elements of the new template form. Of course, you can use a graphics designer to create a more complex form. You can include additional buttons, custom menus, or code that needs to be present on all forms. Notice that this design leaves a row of space at the top of the form. Visual Studio tends to put the record-selector menu in that location, so you can save some time by leaving that space blank.

To use the template form, you must first make a copy of it—to preserve the original. Right-click the TemplateForm object in the Solution Explorer and choose Copy. Right-click the project name in the Solution Explorer and choose the Paste option. A new form, usually named “Copy of TemplateForm” will be created. Right-click it to rename it as Customer2. Double-click the new form to open. You will probably see an error page—because renaming a form does not rename the underlying class. Choose View/Code from the main menu. To remove the duplication, change the class name from TemplateForm to Customer2:

```
Public Class Customer2
```

Save everything and close the new Customer2 form to clear the error page. Reopen the Customer2 page and you should see a working version of the template. Now you can build the form just as you would any other data form. You can follow the steps in Chapter 6 by dragging the Customer table from the Data Source window onto the main form. You could also open the original customer form and

Figure 8.2



copy every object onto the new Customer2 form. Just be sure to copy all of the objects, including the dataset and table adapter. You will also have to copy the code for the Save button and Load event.

To test the form, open the Main (startup) form and edit the code for the Customer button. Currently, it opens the Customer.vb form. Double-click the Customer button to open the code window. Change the two references from Customer to Customer2. Save everything, rebuild the project and run it in debug mode. Figure 8.3 shows the new version of the customer form that is opened when you click the Customer button.

By building all forms from a common template, they will have the same look and feel. As users work with the application, each form will become familiar and they know that actions they take on one form will work the same on other forms. The template makes it easier to maintain visual consistency; you will still have to be careful to ensure that all forms behave the same way.

One of the drawbacks to this type of template is that it is not dynamic. If you want to make changes after forms have been built, you will have to go back to every single form and make the same changes. Otherwise, you would have to rebuild each form again. It is possible to copy-and-paste elements from one form onto a new template, but it still takes considerable time and patience. Consequently, you really want to be sure the template is complete and correct before you use it to build forms. In a large project, you might spend several weeks working with a graphics designer to create the template form.

Action

Copy the TemplateForm.

Paste a new form and name it Customer2.

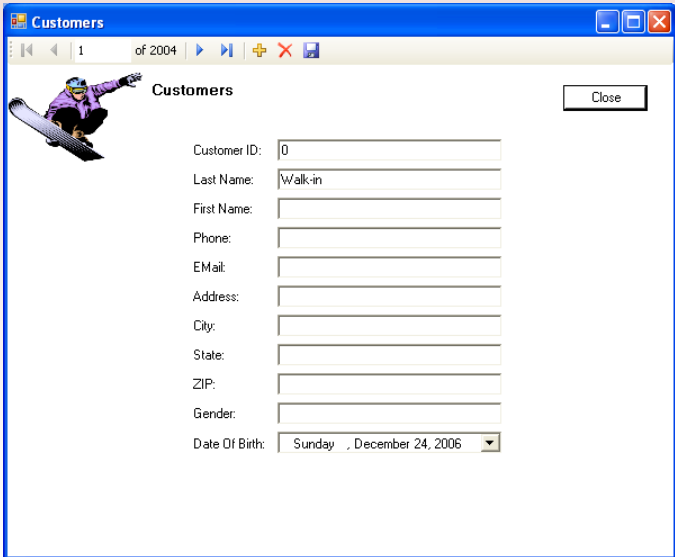
If necessary, create a new Customer dataset in the Data Source.

Add the Customer table to the form.

Change the Menu to open Customer2.

Test the form.

Figure 8.3



The screenshot shows a Windows application window titled "Customers". The window has a blue title bar and standard Windows window controls (minimize, maximize, close). Below the title bar is a toolbar with navigation icons (back, forward, home, search, print) and a status bar showing "1 of 2004". The main content area features a snowboarder icon on the left and a "Close" button on the right. The form contains the following fields:

- Customer ID:
- Last Name:
- First Name:
- Phone:
- EMail:
- Address:
- City:
- State:
- ZIP:
- Gender:
- Date Of Birth:



Activity: Create the Startup Form

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 only see the application through your forms. They will almost never want to open forms directly from the database. You have to create a structure, beginning with the switchboard 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.

Remember that Visual Studio automatically creates a blank form when you start a new project. You have been using that form as the main menu when you created your other forms. So, you already have a start at creating the application. However, as your application grows in size, you will have to spend some time to think about the overall structure of the menus. You cannot put dozens of button links on the main menu and expect people to find the right one. Instead, you might have to add submenus—splitting the buttons across several new menu forms. You will have to talk with the users to understand how they work and how the links should be grouped for associated tasks.

Figure 8.4 shows a start of the main menu form. An image or logo is often used to add some color and personalize the menu. The form includes a link to display one of the reports. You can use the LinkLabel or a command button. In either case,

Action

Add an image or logo to the menu form (try clip art).

Arrange the form buttons.

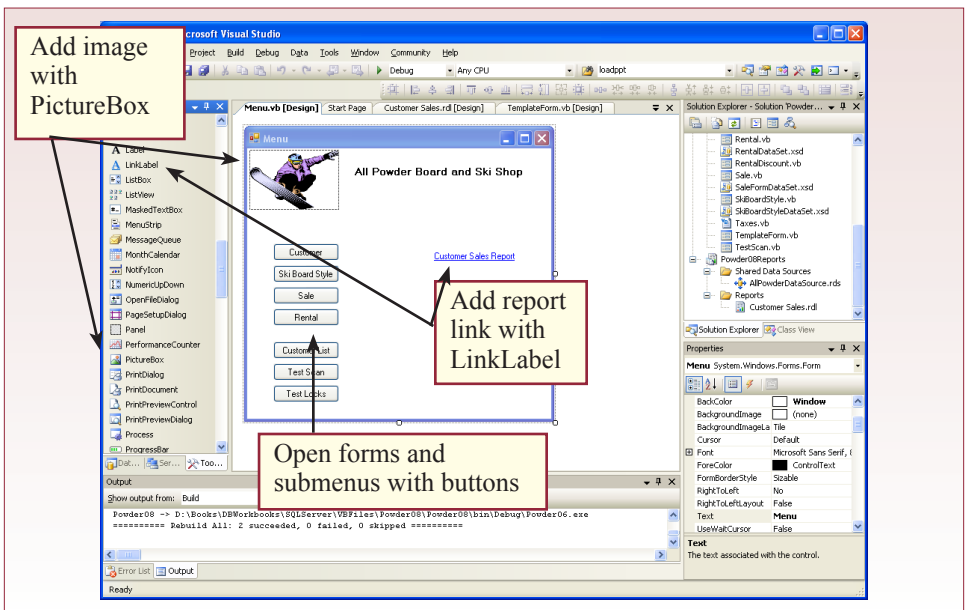
Add a LinkLabel to open a report.

Add the VisitLink subroutine to your code.

Enter the code to call the VisitLink with the appropriate URL to open the desired form.

Compile and test the links.

Figure 8.4



you will have to add a couple of lines of code to activate the report—it cannot be done by setting a property. The SQL Server reporting service is powerful—all reports are delivered through a Web browser. Actually, users could go directly to the report server and run all of the reports in the folder you created (Reports08). In the application, you will want a link to bring up only one form. You accomplish this task by providing the URL of the specific form.

The first step in displaying a report form is to make sure you know where the reports are stored. Begin by realizing that they are not stored in your project. The actual report definition is held on a Web server. So where is this Web server? The answer is that it depends on how you set up your system. If you just installed the reports service for your computer, then it uses the localhost Web server. In an actual project, you would install the reporting service on a specific company Web server—just be sure to record the name of the server and the name of the default service. Use `http://<server name>/Reports` in your Web browser to ensure that the Reporting Services are running correctly. Right-click the name of your Reports project in the Visual Studio Solution Explorer window and choose Properties. Set the `TargetReportFolder` (such as `Powder08Reports`) and the `TargetServerURL`. If you are testing everything on a single computer, you can use `http://localhost/ReportServer`. Now you need to deploy your reports to that server, using the `Build/Deploy` the reports project.

Once you know where the reports service is running, and the name of the folder, and the name of the report, you can create a URL to open a specific report. For example: `http://localhost/Reports/Pages/Report.aspx?ItemPath=%2fPowder08Reports%2fCustomer+Sales`. The first part (`localhost`) is the Web server machine name. The next (`Reports`) is the name of the reporting service that you created when you installed it. The other parts are required and defined by Microsoft, until you get to the actual report. You specified the folder (`/Reports08`) when you added the project to Visual Studio. The last part (`CustomerSales`) is the name of the report within that folder. You should create this link and test it using your browser.

The last step is to tell Visual Studio to open the URL and display the report in the default browser. Figure 8.5 shows the code needed to open a URL within Visual Basic. You have to enter the `VisitLink` subroutine only one time. It does the

Figure 8.5

```
Sub VisitLink(ByVal Ink As LinkLabel, ByVal sURL As String)
    ' Change the color of the link text
    Ink.LinkVisited = True
    ' Call the Process.Start method to open the default browser
    System.Diagnostics.Process.Start(sURL)
End Sub

Private Sub InkCustomerSalesReport_LinkClicked( _
    ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.LinkLabelLinkClickedEventArgs) _
    Handles InkCustomerSalesReport.LinkClicked
    Try
        VisitLink(sender, _
            "http://localhost/Reports/Pages/Report.aspx?ItemPath=%2fPowder08Reports%2fCustomer+Sales")
    Catch ex As Exception
        MessageBox.Show("Unable to open the report.")
    End Try
End Sub
```

actual work of opening the specified URL. To call this subroutine, double-click the link on the Design page and Visual Studio will create the LinkClicked subroutine structure. Type the word “Try” and when you press then Enter key, Visual Studio will add the Catch and End Try elements. All you have to do is enter the VisitLink(sender, http...) line, and type in an error message. Save and compile the project and test everything.

The report approach shown here opens a separate Web browser and delivers the report to the user as if the person had browsed to the server. This approach gives the user complete access to all of the Web Reporting Services. And users could bookmark the site and return to the report without needing to run your application. If you want to reduce these options, you can embed a Web browser control into a new blank form in your project and set its URL to the same Report. Visual Studio also provides a ReportViewer object that you can place on your form. In both cases, the report is displayed as a form within your application—which provides a more integrated appearance, and reduces the user-control over the report. If you need to ship the application as a standalone system, it is better to use these methods. Otherwise, users have more control if you simply link them to the online Reporting Services.



Activity: Build Menus and Toolbars

Startup switchboard forms and command button links help users navigate from one form to another, but in complex applications, users might need additional support. Menus and toolbars are another method of displaying available actions to the users. Menus are usually displayed at the top of the application to provide 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. You can also create secondary toolbars that are customized for each form. The Visual Studio toolbox includes the MainMenu, Toolbar, and ContextMenu objects to help you create menus. The hardest part is determining what items should be on the menu and how they should be organized. Again, you have to work hard to match the user tasks.

Figure 8.6 shows the main steps to create a menu for the main form. Drag a Menu Strip from the Toolbox onto the form, and you will see a set of boxes at the top of the page. Simply type your menu choices into the boxes. You can expand both horizontally and vertically. You can right-click an item to add a separator. In the example, the customer-oriented reports are separated from the forms.

Once you have created the menu items, you have to write the code that is executed when an item is clicked. You will have to double-click each item in design view and enter the appropriate code. In most cases, you will already have similar code—because you already wrote it for each button or link. For example, double-click the entry for the Customer form and copy the code from behind the Customer button. Follow the same process for each entry and you will have a func-

Action

- Drag a Menu Strip onto the menu form.
- Add top-level links for at least Customers, Close, and Help.
- Under Customers, add the Customer and Sale forms, and the two reports.
- Rename each menu item.
- Double-click the Customer and Sale entries and add the code to open the forms.
- Add the code to open the two reports.
- Save, compile, and test everything.

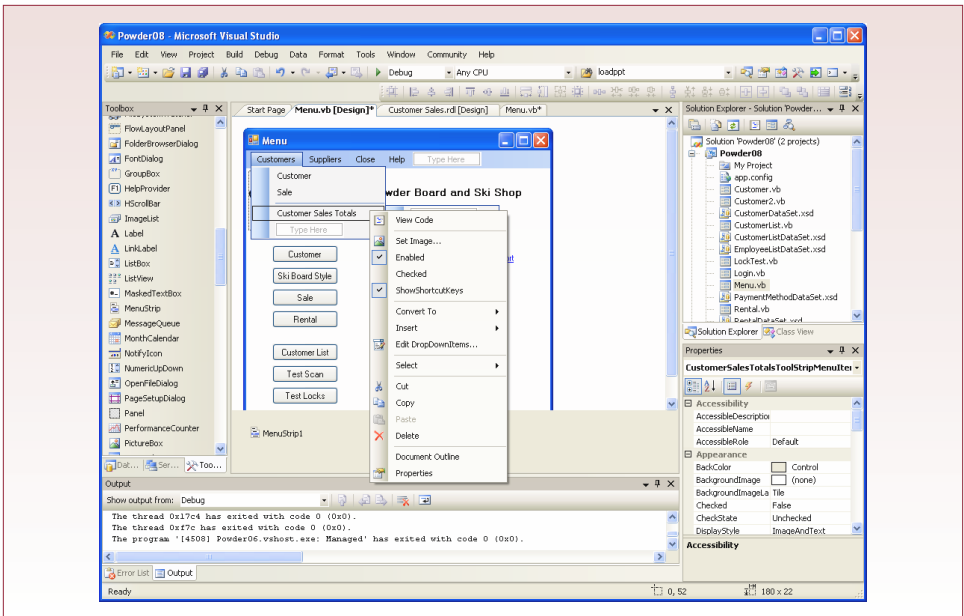
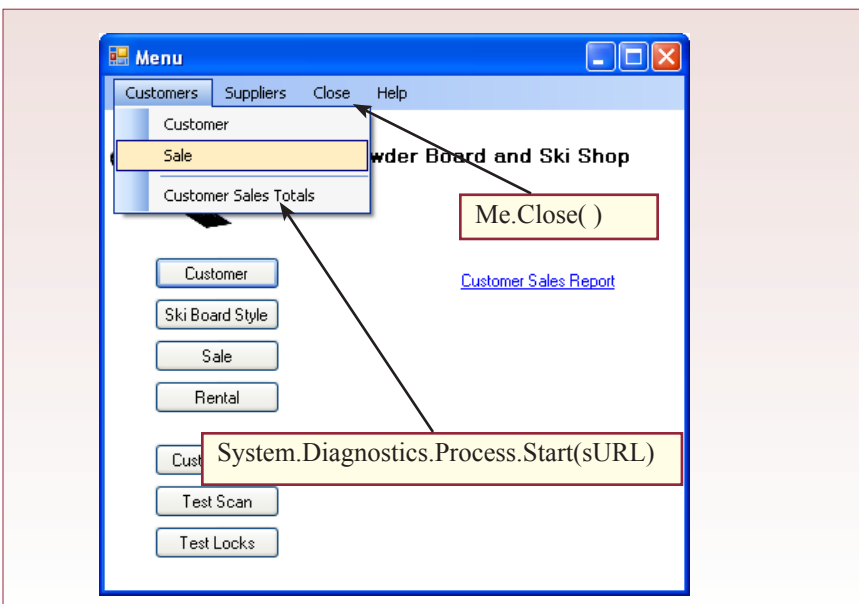


Figure 8.6

tioning menu. The one catch is that you have to be careful with the reports. The `VisitLink` changes the display of the specified link. From the menu, you cannot call `VisitLink(sender, ...)`. Instead, you could pass the name of the actual link—if it is on the same form. Otherwise, instead of calling `VisitLink`, just open the report URL directly with the `System.Diagnostics.Process.Start(sURL)` procedure. To close a form, use the simple line: `Me.Close`. Figure 8.7 shows the sample menu.

Figure 8.7





Activity: Write 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 8.8 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 8.9 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

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.

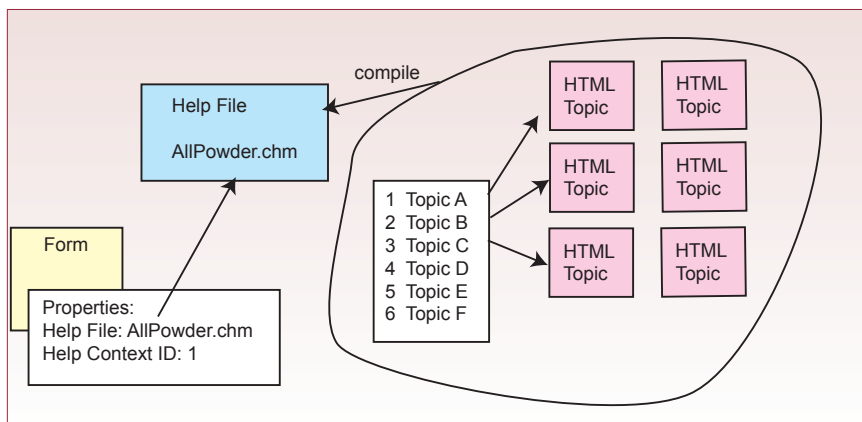
Edit the HTML files to add keywords.

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.

Figure 8.8



```

<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 8.9

Help editor, or you can copy, paste, and edit the keyword information. At this 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 8.10, 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. You might be able to skip the topics file for SQL Server, but if you include it, the resulting help file can be used by other applications.

Once the pages have been created, you compile the files into a single chm file that is distributed with the executable file. The Help compiler does most of the

Action

Add a HelpProvider object to the main form.

Set its HelpNamespace property to AllPowder.chm.

Select the Form object, set the HelpNavigator property to TableOfContents

For the Customer button, set the HelpKeyword to Customer and HelpNavigator to KeywordIndex.

Write the code for the Help menu item.

Compile and test everything.

Figure 8.10

```

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

```

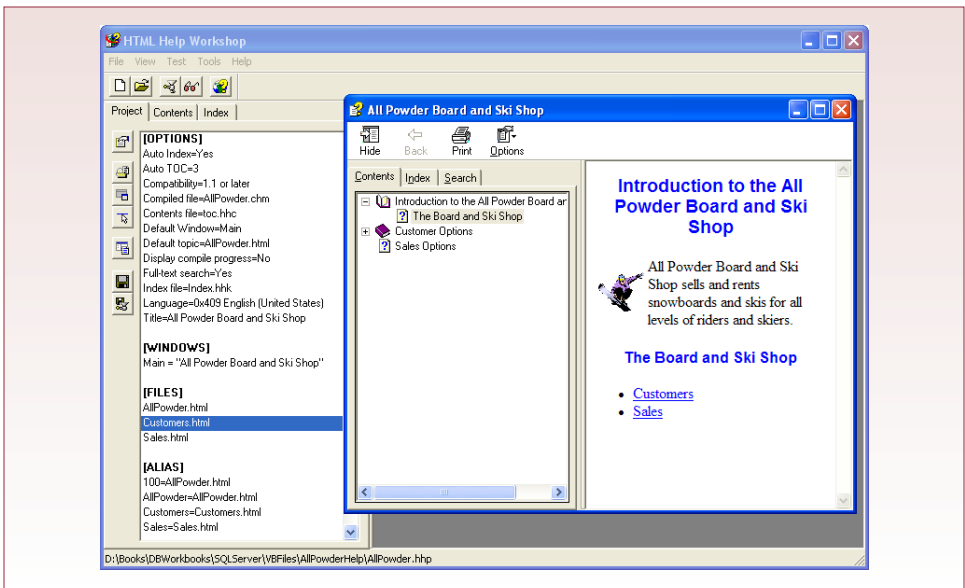


Figure 8.11

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 8.11 shows the basic elements of the HTML Help compiler and the resulting compiled Help file.

The final step is to set up the Visual Basic forms so that they open the correct Help file to the correct page. Keep your list of keywords handy, you will have to enter them into the form properties. The first step is to drag a HelpProvider object from the Toolbox onto the form. Set the HelpNamespace property to the name of the help file (AllPowder.chm). Do not enter other path or drive location information since you cannot control the location the user will choose. By using only the file name, you simply have to make sure that file is stored in the same folder as the executable file. Use Add/Existing and add AllPowder.chm to the project, then set its “Copy to Output” property to Copy if newer.

Figure 8.12 shows the properties that have to be set for each control. Enter the HelpKeyword (Customer), and set the HelpNavigator action to KeywordIndex. This action performs a search for the keyword and displays the matching topic. To be safe, make sure the ShowHelp property is set to True. In some cases, you might want only a single topic for the entire form. In that case, select the Form object and set the Help properties, then every control that does not override those settings will display the same Help form when the user presses the F1 key. It is usually easier to set one topic for a form and then change entries for only one or two controls; instead of setting the same properties for every control.

Remember that you created a Help entry on the main menu. Menu options do not use the HelpProvider, so you need to add program code to open the help file. You can call the help system directly, so you really only need one line of code:

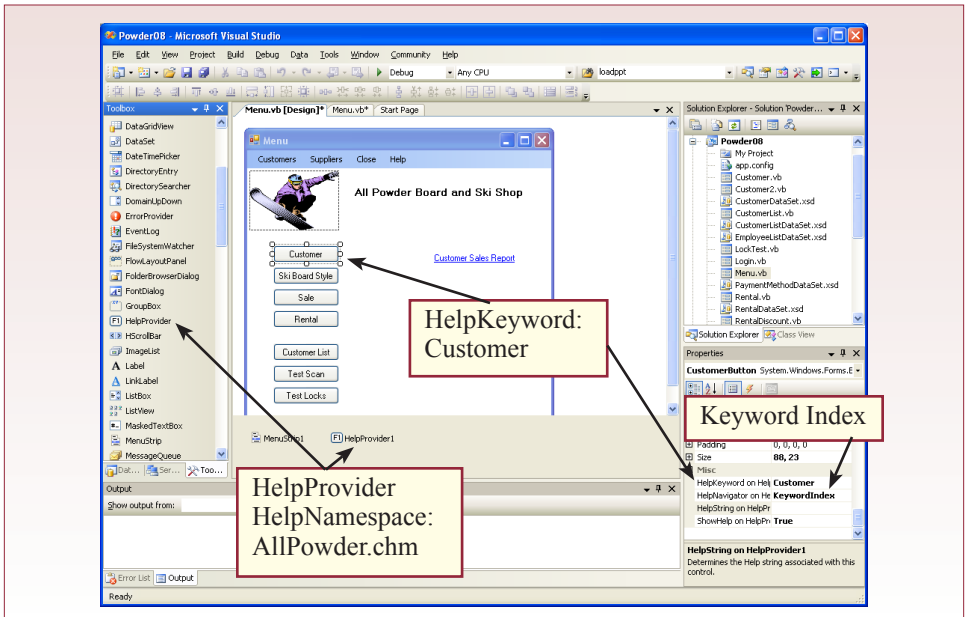
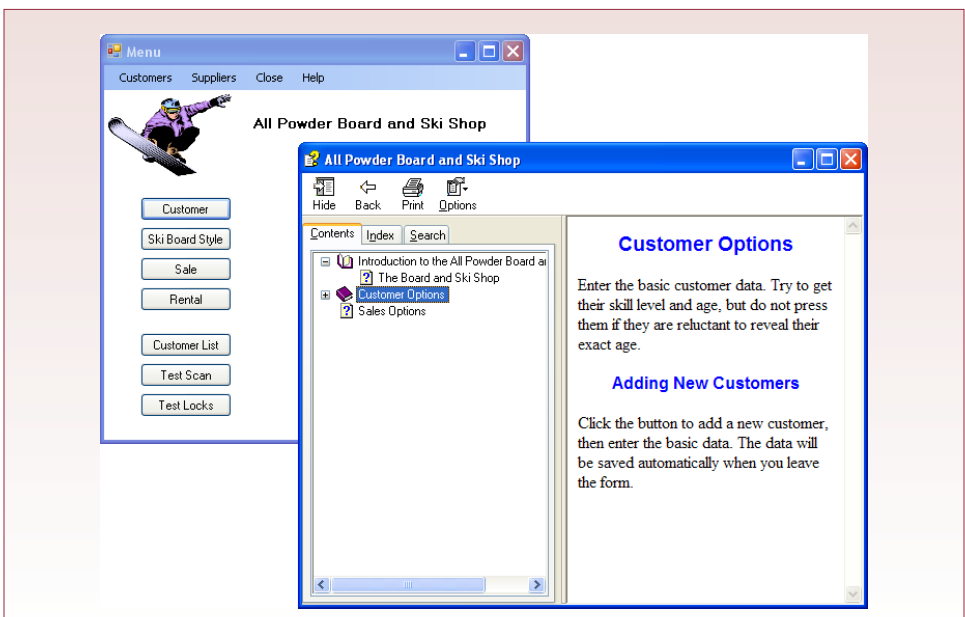


Figure 8.12

Help.ShowHelp(Me.btnCustomer, "AllPowder.chm", HelpNavigator.TableOfContents). The ShowHelp command needs to be associated with a control and not the menu, so pick the first button (btnCustomer). You have to enter the name of the help file since it does not use the value from the provider. The next parameters tell the help system to display the table of contents page. You might change those to use the KeywordIndex and enter a search word to open a specific page. But, often, people who select Help from a menu are looking for a more general starting point. Compile everything and test it by moving to various controls and pressing

Figure 8.13



```

Private Sub HelpToolStripMenuItem_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles HelpToolStripMenuItem.Click
    ' Open Table of Contents in Help
    Help.ShowHelp(Me.CustomerButton, "AllPowder.chm", _
        HelpNavigator.TableOfContents)
End Sub

```

Figure 8.14

the F1 key. Also test the Help menu item. Figure 8.13 shows the simple help file for the Customer button. In a real case, you will spend considerably more time writing detailed help instructions. Someone should also proofread all of the topics to make sure they contain no errors. On large projects, a separate team is often created just to write and maintain the help system. You can also purchase more advanced tools to help create the topic files.

Most applications include a general Help entry on the main menu. You need to handle the Click event on this button so that the general help file opens to the Table of Contents when users click on the Help button or icon. Visual Basic has a built-in ShowHelp function that makes this step relatively painless. Figure 8.14 shows the basic code that you need.



Activity: Deploy an Application

Creating and building an application can be a complex and time-consuming task. However, you have one more important step to complete before you are finished. You need to package the application so it is easy to deploy and install on the client computers. If you are

building the application as a Web-based system, it is relatively easy to deploy—you basically install everything on the server and send the URL of the startup page to users. You will probably have to give users their login credentials, and probably provide some training, but deploying the application is relatively easy because it is in one location. Remember that the reports will be deployed to the Reporting Services folder, which might be on a separate server. If you are deploying a more traditional PC-based system, you need to work a little harder.

Fortunately, Visual Studio has the ability to create an installation file that will copy files and install your application on client computers with minimal hassle. You can also buy third-party installation systems. Many of these systems have complex options that enable you to test the target machine, install updates, and process complex scripts to handle multiple configurations. They also have automatic tools to uninstall your application. Visual Studio 2005 has a relatively powerful publishing system that quickly builds an installation file that can be installed from a variety of locations, including downloaded from a Web site. One of its strengths (particularly compared to the 2003 version) is the ability to automatically check for updates and download them from a specified Web server. Even if your application is installed via CD, the deployment system can periodically check your site for updates and install them automatically.

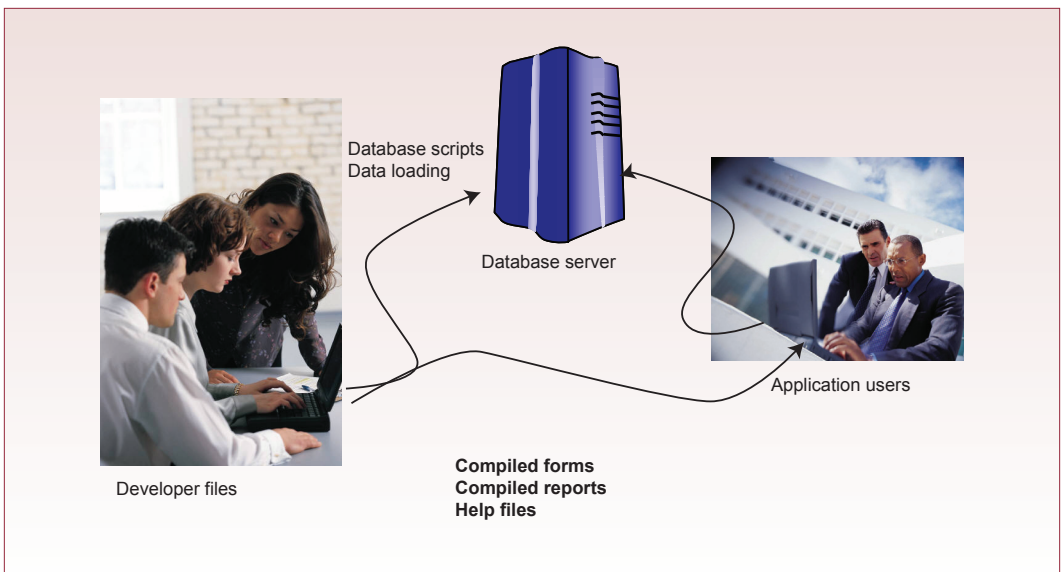
Action

- Choose the Build/Publish option to create an installation file for your application.
- Copy it to a new computer and test it.
- Test the uninstall option.

SQL Server complicates the installation process because the client computers need to be able to locate and reach the database server. Figure 8.15 shows the basic concepts when the database is stored on a central server that is accessed by multiple client computers. As a developer, you create SQL scripts that define the database tables, relationships, queries, and stored procedures. These are run on the server and the initial data is loaded into the tables. The client application is compiled in Visual Studio and published as an application. Within that application, you include database connection strings that specify the location of the database server. These strings are usually automatically created with in the application properties and stored in the app.config file. In the case of a single database server, you can usually preset these variables and the application will find the database when it is run. In other cases, you might have to hand-edit the app.config file after the application is installed on a client computer. Or, you might have to create a separate installation script or startup routine that asks the user to select the appropriate server, and then write the proper connection string to the file.

The installation process is slightly different if you are building a standalone application where the database is stored on the client computers. Today, this approach is relatively rare, because it is difficult to share the data with other users. If you choose this approach, you would generally use the SQL Server option to create independent files that are stored within your application directory. That way, you always know where the data will be stored. Otherwise, you need a startup script to ask the user to browse and locate the database needed for your project. Few users would understand the question, and would be unlikely to perform the task correctly, so try to stick with simple solutions. Otherwise, you will need to provide additional installation help when the application is deployed.

Figure 8.15



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 number of hours 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, their phone numbers and e-mail addresses. Sometimes, she also wants a similar list for 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, and remember that the users have limited computer experience.



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 in 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.



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, 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 are having trouble keeping some items in stock, and other items seem to sit on the shelves forever; but 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.

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. 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 menu toolbar that makes the application easier to use.
4. Create help files for the system.

Data Warehouses and Data Mining

Chapter Outline

Data Warehouse, 167

Case: All Powder Board and Ski Shop, 168

Lab Exercise, 168

All Powder Board and Ski Shop, 168

Introductory Data Analysis, 184

Exercises, 196

Final Project, 198

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

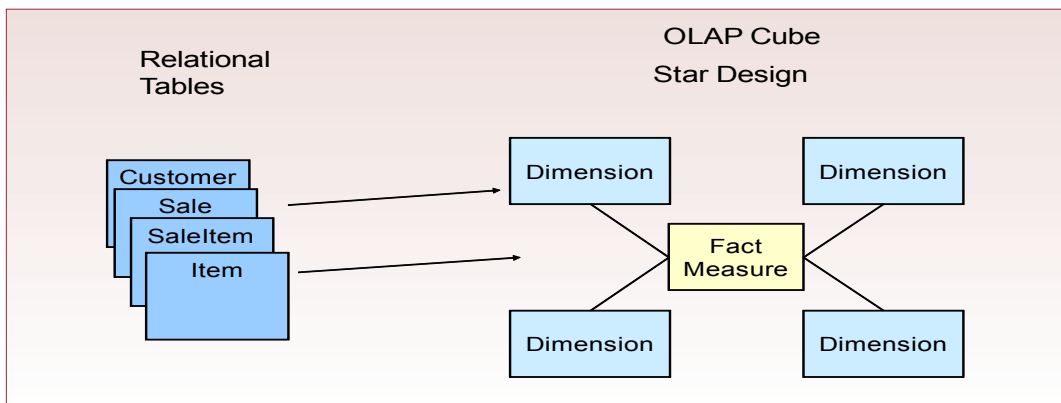
Data warehouses have evolved because of the need for online analytical processing (OLAP) and its conflicts with online transaction processing (OLTP). The goal of a data warehouse is to hold consistent data, possibly obtained from several sources, which can be quickly searched and analyzed. Microsoft handles data warehouses and OLAP capabilities through the Business Intelligence components. Note that these components are not provided in the Express version of SQL Server. You need at least the Standard edition to handle the tasks in this chapter. Another option is to use tools in Microsoft Office. The Microsoft Access Workbook explains how to use those tools.

Microsoft has continually enhanced its data warehouse and business intelligence tools. Some of the tools can be integrated into the reporting systems, but the majority of the tools are based on first creating an OLAP cube. These tools are integrated with the Visual Studio development editor. Most of them are designed for programmers and developers who can use the advanced tools to extract data, clean it up, and place it into a cube. Once the data is in a cube, it can be turned over to business analysts who use the tools to search for useful patterns.

A large portion of many projects (often 60-80 percent of the development time) is spent on finding and cleaning up data. To be useful, a data warehouse needs to have an automated system to update data as it changes within the transaction systems. Consequently, developers often have to write routines to extract and clean the data automatically. These routines run at night in the off-hours so that managers and analysts have updated data the next day. Figure 9.1 illustrates the basic process. Along the way, the data is often stored in an OLAP cube format that is indexed and optimized for fast searches.

Microsoft's data warehouse tools focus on simplifying this process of extracting, cleaning, and transferring data into the cubes. The Integration Services tools contain predefined objects designed to handle the most common tasks. These objects can also be programmed with specific features to handle complex tasks involving data from multiple sources, and even timing issues. The examples presented in this workbook show the basic steps, but the data is simplified to keep the exercise relatively short. In actuality, the tool is powerful and can be programmed to handle highly complex data transfers.

Figure 9.1



Case: All Powder Board and Ski Shop

Like most businesses, the managers of All Powder need to analyze data to spot trends and 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 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 large selection of snowboards and skis. However, managers constantly need to 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 perhaps the Microsoft PivotTable that shows sales split by several features and categories. Figure 9.2 lists some of the main dimensions that managers want to examine in terms of sales. They are not certain about the validity of the last three, so they are displayed with question marks.

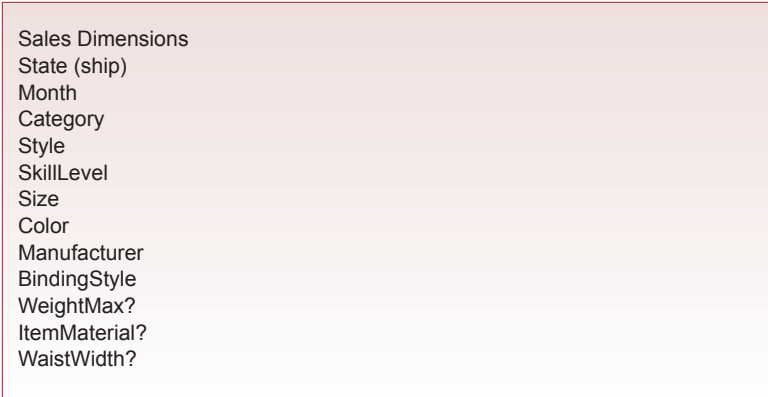
Managers also occasionally raise some more challenging statistical questions, such as whether customers who rent equipment are likely to buy that equipment, and whether skiers buy certain types of poles or boots with their skis. They also need to forecast sales by categories. In particular, they often 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 last like to see some rough analyses.

Lab Exercise

All Powder Board and Ski Shop

As organizations grow over time, the internal processes evolve, data changes, systems improve, and product identifiers 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. For the All Powder shop, before the database was created, 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, they are focused primarily on the equipment and do not have data on customers. From our more modern database perspective, the records

Figure 9.2



Sales Dimensions
State (ship)
Month
Category
Style
SkillLevel
Size
Color
Manufacturer
BindingStyle
WeightMax?
ItemMaterial?
WaistWidth?

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
RentDate	RentID	ExpectedRtn	Payment/SKU	RentFee	ReturnDate	ModelID	Size	Manufacture	Category	Color	ModelYear															
1/7/2003	6263	1/4/2003	Mastercard	151330	\$45.00	1/3/2003	BVG-290	6	44	Boots	White	2003														
1/1/2003	6263	1/4/2003	Mastercard	751214	\$60.00	1/4/2003	OVVO-707	210	5	Ski	Purple	2003														
4/1/2003	6263	1/4/2003	Mastercard	752736	\$60.00	1/2/2003	TZH-666	180	24	Ski	Green	2003	Ar													
6/1/2003	6491	1/2/2003	American I	151226	\$15.00	1/2/2003	AEF-37	6	41	Boots	Black	2003														
6/1/2003	6491	1/2/2003	American I	151226	\$15.00	1/4/2003	VCO-360	6	42	Boots	Magenta	2003														
7/1/2003	6491	1/2/2003	American I	51535	\$20.00	1/3/2003	EKR-463	600	26	Board	Yellow	2003	G													
12/2/2003	6028	1/3/2003	Other cred	650073	\$5.00	1/6/2003	MBC-999	900	87	Poles	Green	2003														
9/1/2003	6028	1/3/2003	Other cred	752503	\$20.00	1/3/2003	OVVY-990	165	10	Ski	Red	2003	G													
10/1/2003	6028	1/3/2003	Other cred	752509	\$20.00	1/4/2003	DMR-560	165	18	Ski	Turquoise	2003	S													
11/2/2003	6072	1/4/2003	Debit Card	151202	\$30.00	1/3/2003	AEF-37	6	41	Boots	Black	2003	S													
12/2/2003	6080	1/6/2003	Debit Card	751696	\$60.00	1/6/2003	CWVG-771	165	9	Ski	Blue	2003	Ph													
13/2/2003	6159	1/3/2003	Other cred	751714	\$20.00	1/3/2003	FISA-778	165	1	Ski	Black	2003	S													
14/2/2003	6159	1/3/2003	Other cred	752593	\$20.00	1/2/2003	YTN-679	180	16	Ski	Turquoise	2003	S													
15/2/2003	6180	1/5/2003																								
16/2/2003	6180	1/5/2003																								
17/2/2003	6180	1/5/2003																								
18/2/2003	6181	1/4/2003	1	1566	1/1/2005	CA	95014	Check	50993	1	\$289.00	MRY-77	1200	37	Board	Turquoise										
19/2/2003	6181	1/4/2003	2	1566	1/1/2005	CA	95014	Check	51591	1	\$399.00	HGC-505	750	39	Board	White										
20/2/2003	6415	1/4/2003	5	1587	1/3/2005	VA	23232	Debit Card	920182	1	\$5.00	QTN-6	2	84	Wax	Turquoise										
21/2/2003	6415	1/4/2003	4	1566	1/1/2005	CA	95014	Check	752776	1	\$379.00	LRY-336	160	15	Ski	White										
22/2/2003	6415	1/4/2003	6	1655	1/4/2005	FL	32858	American I	151154	1	\$102.00	YTI-104	6	40	Boots	Orange										
23/2/2003	6742	1/4/2003	7	1655	1/4/2005	FL	32858	American I	50920	1	\$544.00	RPY-837	750	33	Board	Orange										
24/2/2003	6742	1/4/2003	8	1655	1/4/2005	FL	32858	American I	752722	1	\$223.00	HCU-118	150	24	Ski	Purple										
25/2/2003	6742	1/4/2003	9	1658	1/9/2005	IL	60601	Mastercard	251871	1	\$105.00	CSQ-908	38	78	Clothes	Turquoise										
26/2/2003	6830	1/4/2003	10	1658	1/9/2005	IL	60601	Mastercard	751988	1	\$307.00	XCY-442	180	3	Ski	Orange										
27/2/2003	6830	1/4/2003	11	1543	1/8/2005	IA	50501	Debit Card	751596	1	\$82.00	QQQ-655	195	3	Ski	Purple										
28/2/2003	6830	1/4/2003	12	1521	1/19/2005	NV	89501	Check	151297	1	\$237.00	YEH-710	6	43	Boots	Blue										
29/2/2003	6857	1/4/2003	13	1521	1/19/2005	NV	89501	Check	251671	1	\$187.00	JVC-849	38	49	Clothes	White										
30/2/2003	6857	1/4/2003	14	1521	1/19/2005	NV	89501	Check	251839	1	\$65.00	JWT-760	34	74	Clothes	Blue										
31/2/2003	6857	1/4/2003	15	1521	1/19/2005	NV	89501	Check	752917	1	\$379.00	DMR-686	180	7	Ski	Blue										
32/3/2003	6276	1/6/2003	16	1561	1/21/2005	IL	62703	Visa	151262	1	\$237.00	YEH-710	6	43	Boots	Blue										
33/3/2003	6387	1/6/2003	17	1519	1/21/2005	WA	98520	American I	251802	1	\$153.00	MMX-405	38	70	Clothes	Black										
34/3/2003	6459	1/5/2003	18	1519	1/21/2005	WA	98520	American I	508827	1	\$466.00	MBS-240	750	29	Board	Purple										
35/3/2003	6459	1/5/2003	19	1561	1/21/2005	IL	62703	Visa	751676	1	\$71.00	FSA-779	195	1	Ski	Black										
36/3/2003	6459	1/5/2003	20	1690	1/22/2005	KY	40507	Mastercard	151207	1	\$62.00	AEF-37	6	41	Boots	Black										
37/3/2003	6536	1/5/2003	21	1690	1/22/2005	KY	40507	Mastercard	251816	1	\$31.00	XPT-418	36	71	Clothes	Orange										
38/3/2003	6536	1/5/2003	22	1690	1/22/2005	KY	40507	Mastercard	251893	1	\$142.00	QFA-671	34	77	Clothes	Turquoise										
39/3/2003	6536	1/5/2003	23	1690	1/22/2005	KY	40507	Mastercard	752711	1	\$262.00	JLI-393	180	21	Ski	Yellow										
40/3/2003	6536	1/5/2003	24	1622	1/24/2005	OH	44691	Visa	251854	1	\$43.00	SEE-739	38	77	Clothes	Turquoise										
25/1622	1/24/2005	OH	44691	Visa	251856	1	\$43.00	SEE-739	38	77	Clothes	Turquoise														
26/1622	1/24/2005	OH	44691	Visa	50994	1	\$300.00	PKR-373	1200	34	Board	Red														
27/1616	1/26/2005	ID	83702	Check	251931	1	\$181.00	OVVO-304	38	64	Clothes	Purple														
28/1616	1/26/2005	ID	83702	Check	251669	1	\$21.00	APZ-41	36	57	Clothes	Blue														
29/1616	1/26/2005	ID	83702	Check	751990	1	\$307.00	XCY-442	210	3	Ski	Orange														
30/1616	1/26/2005	ID	83702	Check	752744	1	\$438.00	MEL-771	150	9	Ski	Blue														
31/1703	1/26/2005	CA	90052	Other cred	752744	1	\$125.00	CWVG-931	195	15	Ski	Blue														
32/1703	1/26/2005	CA	90052	Other cred	752966	1	\$422.00	FINH-350	150	16	Ski	Orange														
33/1703	1/26/2005	CA	90052	Other cred	752966	1	\$219.00	ZCZ-652	38	60	Clothes	White														
34/1554	1/27/2005	WI	53201	Other cred	752586	1	\$395.00	HCM-85	165	16	Ski	Orange														
35/1554	1/27/2005	WI	53201	Other cred	752586	1	\$395.00	HCM-85	195	16	Ski	Orange														
36/1554	1/27/2005	WI	53201	Other cred	752586	1	\$395.00	HCM-85	165	16	Ski	Orange														
37/1554	1/27/2005	WI	53201	Other cred	752586	1	\$435.00	DMR-560	165	18	Ski	Turquoise														
38/1611	1/30/2005	MA	1930	Debit Card	151130	1	\$232.00	WYG-684	6	40	Boots	Green														

Figure 9.3

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. Figure 9.2 shows the layout of the data in the two worksheets. Again, notice the lack of normalization. Each row represents an item that is sold or rented. Fortunately, the worksheets repeat the SalesID and RentalID so you can still recover which items are grouped onto a single sale or rental. Likewise, they repeat the descriptive item data for each time the model was sold. To ensure your information is really accurate, you should eventually check to see that the managers were consistent in recording this data. For example, ModelID BVG-290 might have been given a different description 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.



Activity: Extract and Transform Data

Note: If you are working on this lab in a class, you might want to skip this particular activity because it is time consuming. You can download and install the full data set that already contains the adjustments made in this section. On the other hand, most of the time spent on data analysis and warehouse projects lies in building the systems described in this section to extract, clean, and load the data into a data ware-

Action

Delete all sales with SaleID>2000.

Copy the two CSV files to your server.

Create two SQL tables to hold the data.

Modify and run the SQL BULK INSERT command to import the early rental data.

Insert CustomerID 0 and EmployeeID 0.

house. So, it is worthwhile to at least read this section. And, if you plan to build a data warehouse project, this section provides some ideas on how to organize and clean the data.

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. SQL has at least three related methods of importing data. The simplest command is to use BULK INSERT within a SQL Query window. A variation of this command is used by the data files that accompany this book to create and load the sample databases. The main drawback is that you need to put the data files on the SQL Server computer. It is also relatively difficult to automate—so it can be a problem if you need to build an automated system to run every night and extract data from one computer, transfer it, and clean it. A second, older method is to use the command-line `sqlcmd` utility. With some effort, you can integrate this utility within a Windows script file and issue commands directly to the database. Along the same lines, you could write a custom program in Visual Studio to use ADO .NET to connect to data sources and transfer the data to the SQL Server database. This approach is powerful, but tedious. Microsoft created the SQL Server Integration Services (SSIS) tool that handles many common tasks for you automatically, yet still provides detailed control over the data. SSIS is a useful tool in a production environment, where you have multiple data sources and need to schedule transfers, trap errors, and make changes over time. But, its power means that it is relatively complex, so this section will show only a simple application.

The BULK INSERT command is relatively easy to use, and it is a convenient approach for manually loading large amounts of data stored in text or comma-separated values (CSV) files—which can be generated from Excel spreadsheets or other database systems. To illustrate the process, download the two CSV files that contain early Sales and Rentals data. Note that these files have already been cleaned to be acceptable for BULK INSERT. They contain simple data, with no header columns, no dollar signs, and no quotation marks. Put the files in a folder on your SQL Server computer and write down the location.

You need a table to hold the new data, so use the CREATE TABLE command shown in Figure 9.4, or use the table-builder in the Management Studio. Be careful to ensure that the columns in the new table exactly match the columns in the CSV data file. The column order and data types must match. You can delete columns or rows later, but the bulk insert command imports all rows and columns. Write the BULK INSERT command shown at the bottom of Figure 9.4. Enter the correct path for the file (without the angle brackets) and run the command. If your data files has errors, or you did not match the columns correctly, you will receive errors and the data will not be loaded. It is hard to track down which row caused problems, so sometimes you want to import data in small batches; or test it in Excel to ensure data is in the correct columns with no extra commas. BULK INSERT works on all versions of SQL Server, but your account needs the BULK INSERT role, or the DBO role, so do not assume casual users can run the command.

SQL Server Integration Services was created to handle vastly more complex tasks, such as scheduling data transfers, processing and sequencing transfers from multiple computers, and running unattended with a full error-reporting facility. However, the free (Express) version of SQL Server does not include this tool, and for complex situations, you will probably need the Enterprise version. The simple import command can be handled with the Standard edition.


```
CREATE TABLE OldRental
(
    RentDate      datetime,
    RentID int,
    ExpectedReturn datetime,
    PaymentMethod nvarchar (50),
    SKU          nvarchar (50),
    RentFee money,
    ReturnDate   datetime,
    ModelID nvarchar (250),
    ItemSize float,
    ManufacturerID int,
    Category nvarchar (50),
    Color nvarchar (50),
    ModelYear int,
    Graphics nvarchar (50),
    ItemMaterial nvarchar (50),
    ListPrice money,
    Style nvarchar (50),
    SkillLevel int,
    WeightMax float,
    WaistWidth float,
    BindingStyle nvarchar (50)
);

BULK INSERT OldRental
FROM '<folder name>\Lab 09-01 Early Rentals.csv'
WITH
(
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n'
);
Go
```

Figure 9.4

As of the initial release of SQL Server 2008, SSIS still has some “issues.” You can find notes on some of them on Internet forums. If you stray from the instructions in this section, you are likely to encounter some of those issues—even in this relatively simple example.

SSIS can retrieve data from Excel spreadsheets, flat files, and several other database systems. Although, a 64-bit operating systems seems to have compatibility issues with Excel, so this example will use a CSV file. SSIS runs as a project module within Visual Studio, which uses a graphical approach and object properties to define data sources, destinations, and connections. Ultimately, it generates packages that can be installed on various computers and run when data needs to be transferred. The system is designed to handle package scheduling and interactions, but this example is much simpler.

Begin by creating a CSV file from the Excel spreadsheet. You can save some time by including the column headings in the CSV file. It is also clearer if you create your own data file in the database. Figure 9.5 shows the columns needed in the OldSale table. These columns match the Excel/CSV columns, but later, you will have the ability to change the mapping of the Excel columns to the database table, so they do not have to be exact.

Now start SQL Server Business Intelligence from the Windows Start command. When it is ready, create a new project using the menu commands: File/

New/Project, and choose the Integration Services Project. You should see an empty graphics screen. Importing data is handled by defining a data flow and then specifying the source and destination. From the Toolbox, drag a Data Flow Task object onto the Control Flow page. In larger projects, you will want to name it and specify additional properties. Double-click the new object box to open its corresponding Data Flow page.

From the Toolbox, drag a Flat File Source onto the Data Flow page. Double-click it to open its property editor. Click the New button to add a new connection manager. Name it: Old Sales CSV. Browse to find the CSV file you created. Set the checkbox for “Column names in first data row.” This step saves you from having to retype all of the column names and reduces errors when matching the CSV columns to the database columns. Click the Columns entry in the menu list to see the column names.

One drawback to CSV files is that the system assigns all columns a simple text data type. The good news is that it is relatively easy to change these types. (It is

Action

Open the Excel spreadsheet and save the Sales data as a CSV file.

Create the OldSale table in SQL Server.

Start SSIS, use File/New/Project to create a new Integration Services Project.

Drag a Data Flow Task object onto the Control Flow pane.

Double click this object to open the Data Flow page.

Drag a Flat File Source onto this pane.

Double-click it and attach it to the CSV file.

Set the data types for the columns.

Drag an ADO .NET destination object onto the Data Flow page.

Drag the arrow from the Flat File source object to the ADO destination object.

Double-click the destination object and assign it to the OldSale table in your database.

Save and run the project to transfer the data.

Figure 9.5

```
CREATE TABLE OldSale
( SaleID          int,
  SaleDate        datetime,
  ShipState       nvarchar(50),
  ShipZIP         nvarchar(50),
  PaymentMethod  nvarchar (50),
  SKU            nvarchar (50),
  QuantitySold   int,
  SalePrice       money,
  ModelID        nvarchar (250),
  ItemSize       float,
  ManufacturerID int,
  Category       nvarchar (50),
  Color          nvarchar (50),
  ModelYear      int,
  Graphics       nvarchar (50),
  ItemMaterial   nvarchar (50),
  ListPrice      money,
  Style          nvarchar (50),
  SkillLevel     int,
  WeightMax      float,
  WaistWidth     float,
  BindingStyle   nvarchar (50) )
```

SaleID	long (DT_I4)
SaleDate	database timestamp (DT_TIMESTAMP)
ShipState	unicode string (DT_WSTR)
ShipZIP	unicode string (DT_WSTR)
PaymentMethod	unicode string (DT_WSTR)
SKU	unicode string (DT_WSTR)
QuantitySold	long (DT_I4)
SalePrice	currency (DT_CY)
ModelID	unicode string 250 (DT_WSTR)
Size	double (DT_R8)
ManufacturerID	long (DT_I4)
Category	unicode string (DT_WSTR)
Color	unicode string (DT_WSTR)
ModelYear	long (DT_I4)
Graphics	unicode string (DT_WSTR)
ItemMaterial	unicode string (DT_WSTR)
ListPrice	currency (DT_CY)
Style	unicode string (DT_WSTR)
SkillLevel	long (DT_I4)
WeightMax	double (DT_R8)
WaistWidth	double (DT_R8)
BindingStyle	unicode string (DT_WSTR)

Figure 9.6

more challenging with Excel files.) Click the Advanced option in the menu list and change the data types for every one of the columns. Figure 9.6 shows the values needed to match the database types.

With the data source defined, you simply need to pick a destination. The trick here is that you need to avoid using the SQL Server destination object. Although it has the most efficient insert method, Microsoft notes that the SQL object only works if the data source is located on the same machine as SQL Server. Other users report issues with security conditions, so it is better to use the ADO .NET object instead. Note that this object was introduced with SQL Server 2008. For SQL Server 2005, use an OLE DB object. Simply drag the ADO .NET destination object from the Toolbox onto the Data Flow page. Before configuring the destination, connect it to the source. Select the Flat File source object. Drag its arrow (on the bottom left) and drop it onto the new destination object.

Double-click the destination object to begin its configuration. Click the New button to create a connection manager. Click the second New button to create a new ADO DB connection. Pick the server hosting your database. If necessary, select SQL Server Authentication and enter your login credentials. Choose the database from the drop-down list (probably AllPowder). Click the OK buttons to accept your choices. With the connection defined, select the destination table that you created (OldSale). Click the Mappings menu item. Verify that all source columns match the destination columns. You will probably have to choose Size to match ItemSize. Click OK to return to the Data Flow page.

You have now completed all of the hard work of finding the source and the destination and mapping the columns. Figure 9.7 shows the source, destination, and two connection manager objects. Save the project and run it—using the small green (debug) arrow in the top menu. The package will run and transfer the data. Note that this process can be repeated, and the package can be deployed to a different computer and scheduled to run whenever you need to update the data. You

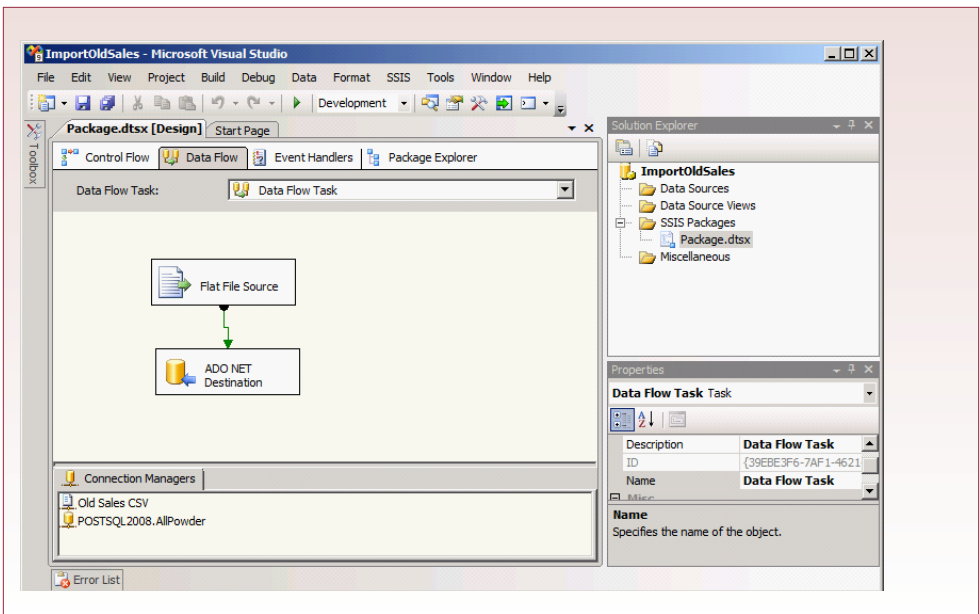


Figure 9.7

can add multiple controls and many sources and destinations. The system also tracks errors and you can write custom error controls.

In any case, you should now have two new tables (OldSale and OldRental) that contain earlier sales and rental data. However, remember that these tables are not normalized, and various columns need to be extracted with the data transferred to other tables. But, now that the data is in SQL Server tables, you can use SQL commands to do the hard work.

Looking through the temporary Sale 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 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. So your first task is to create these new entries in the respective tables. Figure 9.8 shows the basic SQL commands needed to create these two entries.

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

Figure 9.8

```
INSERT INTO Customer (CustomerID, LastName)
Values (0,'Walk-in')
INSERT INTO Employee (EmployeeID, LastName)
Values (0,'Staff')
```

```
SELECT DISTINCT OldSale.ModelID, OldSale.ManufacturerID, OldSale.
Category,
OldSale.Color, OldSale.ModelYear, OldSale.Graphics, OldSale.ItemMaterial,
OldSale.ListPrice, OldSale.Style, OldSale.SkillLevel, OldSale.WeightMax,
OldSale.WaistWidth, OldSale.BindingStyle
FROM OldSale;
```

Figure 9.9

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 model 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 9.9 shows the basic query to retrieve the model attributes from the OldSale table. Move this query to the side and 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 9.10 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.

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 9.11, 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 and they must be 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 data 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 items around, the quantity can be entered by hand later. Figure 9.12 shows the final step that inserts the data into the Inventory table. Remember that you have to create the UNION query first. Notice the use of the column alias to force a zero value into the QuantityOnHand column for each row.

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 Sale, Rental, SalesItem, and RentalItems tables.

Figure 9.10

```
SELECT DISTINCT ModelID, ManufacturerID, Category, ...
FROM OldSale
UNION
SELECT DISTINCT ModelID, ManufacturerID, Category, ...
FROM OldRental
```

```

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 9.11

The Sale and Rental data is considerably easier because they are separate and you will not need the UNION command to merge the two sets of data. In fact, you can copy the Sale (or Rental) data with one SQL command. First, build a query to retrieve the distinct sales data from the OldSale 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 9.13 shows an additional trick that is often helpful. If you added new rows of data to your Sale table, the system 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.

Figure 9.12

```

INSERT INTO Inventory (ModelID, SKU, Size, QuantityOnHand)
SELECT DISTINCT qryOldInventory.ModelID, qryOldInventory.SKU,
qryOldInventory.ItemSize, 0 As QuantityOnHand
FROM qryOldInventory;

```

Figure 9.14 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 the identical transformation for the SaleItem table. Otherwise, the data will never match and cannot be joined. If you forget, you will usually 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. The Rental table uses columns RentID, RentDate, ExpectedReturn, and PaymentMethod. The columns for the old rental table do not include repair charges and are limited to RentID, SKU, RentFee, and ReturnDate. At this point, you have successfully imported the old data and cleaned it up so it can be used within your database. At this point, you should also drop the two imported tables because they are no longer needed. You can use a simple DROP TABLE OldSale command.

Figure 9.13

```

INSERT INTO Sale (SaleID, SaleDate, ShipState, ShipZIP, PaymentMethod)
SELECT DISTINCT OldSale.SaleID+5000, OldSale.SaleDate, OldSale.
ShipState, OldSale.ShipZIP, OldSale.PaymentMethod
FROM OldSale;

```

```
INSERT INTO SaleItem (SaleID, SKU, QuantitySold, SalePrice)
SELECT DISTINCT OldSale.SaleID+5000, OldSale.SKU, OldSale.
QuantitySold, OldSale.SalePrice
FROM OldSale;
```

Figure 9.14



Activity: Create an OLAP Cube

If you skipped the prior section, you should download the “Full” version of the database files and rebuild your database. This copy contains the historical data that has been cleaned. You need the additional data to provide more meaningful reports and charts with the data mining tools.

Once the data has been consolidated in the primary database, you need to create an OLAP cube so that it can be browsed and analyzed. Microsoft uses a special view to hold the data for cubes and analyses. It basically pre-builds all table joins so that data is immediately available for browsing and analysis without having to retrieve it through queries. The first objective is to use the BI tools to create a Data Source View. Overall, the process is similar to that used when creating any query. Connect to the database, select the tables, and choose the columns. However, it helps to think in terms of the cube by focusing on the facts you want to observe relative to various data dimensions.

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 times quantity) as the measure fact, along with the various dimensions. Using Business Intelligence tools, the cube can be manipulated to see subtotals and sort or filter the dimensions. The cube can also be used as the source of data for statistical analyses.

When building OLAP cubes, you should think in terms of two major steps. In the first step, developers create a Data Source View that collects all of the data needed (facts and dimensions). Developers also define the initial cube structure. You can create multiple cubes for different purposes or managers. These cubes are then deployed to a database server where managers interact with the cube to perform their searches and analyses. In most cases, managers work with predefined cubes and do not need access to the underlying developer tools. Hence, it is critical to interview managers to determine what types of data they need to see and how it will be analyzed. For this example, you will create a simple cube that enables managers to analyze sales by various dimensions.

Begin the process by running the SQL Server Business Intelligence Development Studio from the main Windows menu. Create a new project from the menu by choosing File/New/Project, and selecting the Analysis Services Project tem-

Action

Use Business Intelligence Studio to create a new Analysis Services project. Create a new Data Source (server connection). Create a Data Source View using: Sale, SaleItem, Inventory, ItemModel, Manufacturer tables. For lookups, include BindingStyle, Customer, PaymentMethod, ProductCategory, SkiBoardStyle, and SkillLevel.

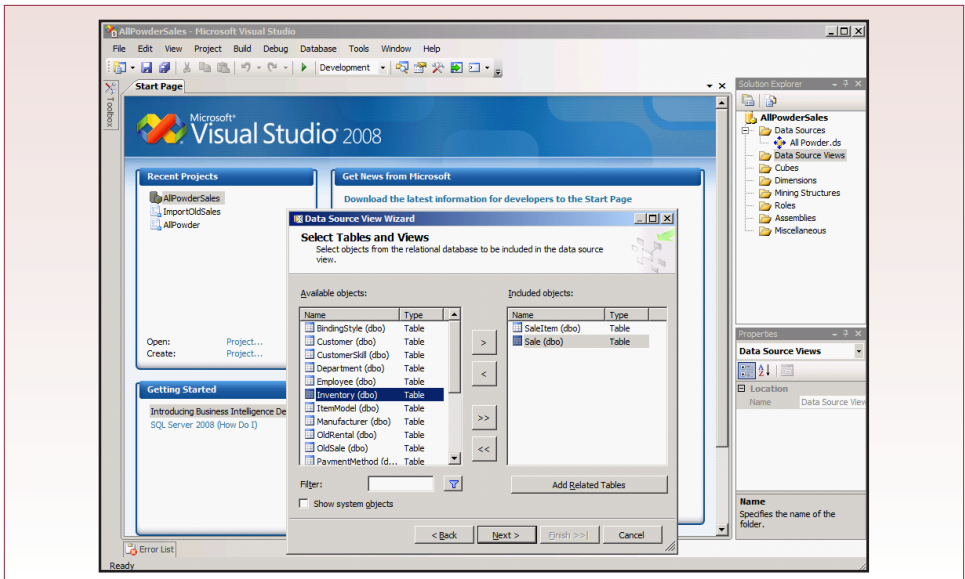


Figure 9.15

plate from within the Business Intelligence Project list. Enter a name that you will recognize, such as AllPowderSales. You can generally stick with the default folders. Remember that ultimately the project cube will be deployed to a database server. The folders hold the development templates and source code.

As with any database project, the first step is to create a database connection (Data Source) that defines the connection to the database server. In the Solution Explorer, right-click Data Sources and choose the New Data Source option. Click the New button to create a new connection. You will see the typical login screen. You should be able to create a standard connection based on your network. However, eventually, you will have to consider how users will connect to the data cube and how you want to handle security. Some companies emphasize the use of Windows Authentication and handle security rights through Active Directory. For simpler projects, it is often easier to create separate accounts within SQL Server and rely on permissions within SQL Server to handle security rights. For now, it is probably easier to use SQL Server Authentication, but the database administrator will have to set up this account ahead of time. In any case, be sure to test the connection to verify that you have the basic permissions to access the database. If you are asked to select how Windows should connect to the Data Source, choose the option to Use the service account. Note that it is possible to pull data from multiple servers, but this exercise will stick with this one connection.

Look in the Solution Explorer window and you will see that creating a Data Source View is the next step. Right click that entry and choose the New Data Source View option. Pick the AllPowder connection you created. Now you need to choose the tables that hold the data you will need. Figure 9.15 shows the basic process: Select a table on the left and click the > button to select it by moving it to the right-side window. To provides plenty of options for the cube, you will use the SaleItem table as the main facts (price and quantity). Dimensions will come from the Sale, Inventory, ItemModel, and Manufacturer tables. Additional dimensions and lookup lists are found in the BindingStyle, Customer, PaymentMethod, ProductCategory, SkiBoardStyle, and SkillLevel tables, so include all of those tables.

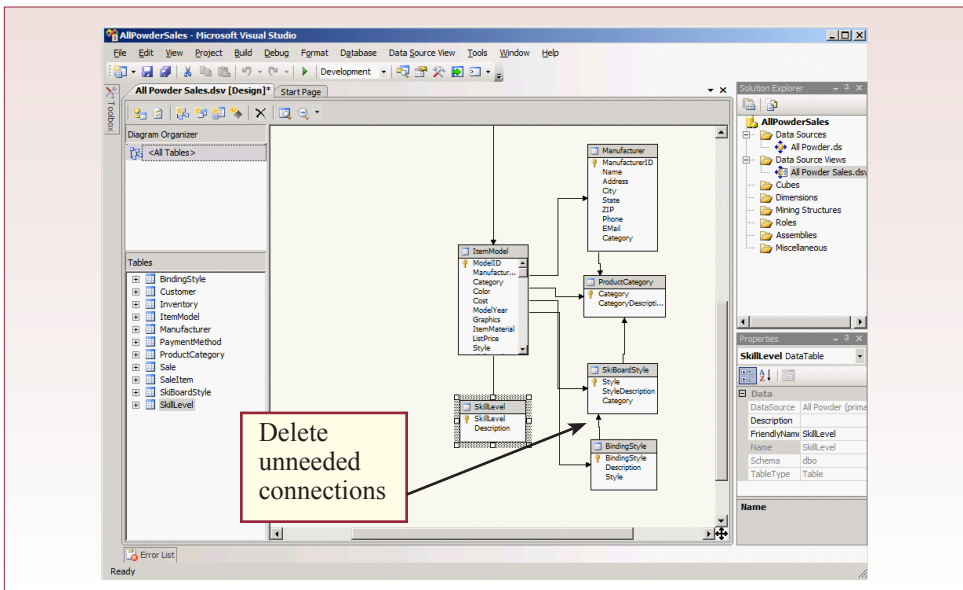


Figure 9.16

Double-check to verify that you have included 11 tables. Click Next and change the name to All Powder Sales.

When you finish your selections, the wizard will build the design of the new Data Source View. It places all of the tables in a diagram and uses arrows to show how they are connected. For the most part, the connections will follow the relationships specified in the original database design. However, some relationships or connections can cause problems. Notably, they might cause cycles in the design, which must be removed. In this example, as shown in Figure 9.16, scroll the design towards the bottom and notice the unwanted connections: BindingStyle-SkiBoardStyle, SkiBoardStyle-ProductCategory, and Manufacturer-ProductCategory. If you try to leave these in the design, you will eventually receive error messages. You can see the problem by tracing multiple paths from the ItemModel table to the ProductCategory table. All three of these connections need to be deleted to avoid cycles in the data. Simply click on each arrow and press the Delete key (or right-click and choose Delete). Click the Save All button once in a while to ensure that your project is saved as you make changes.

You need to make several changes within the Data Source to make it easy to use for the OLAP cube. The first change is tricky but seems to be required. The SaleItem table is important because it contains the primary facts to be evaluated. However, it has a composite primary key (SaleID, SKU). The system works best if you create a new column that concatenates the two keys to form a single column to represent the primary key. Right-click the edge of the SaleItem table and choose the New Named Calculation option. Enter SaleID_SKU as the name and Concatenated key as the description. For the expression, enter: Convert(nvarchar, SaleID) + '-' + SKU. You need to use the Convert function because SaleID is

Action

Remove extra (cycle) connections in the Data Source View.

In the SaleItem table, add a concatenated key: Convert(nvarchar, SaleID) + '-' + SKU

Add Value=SalePrice*QuantitySold to SaleItem.

Add SaleYear and SaleMonth to the Sale table.

numeric. It is helpful to test functions by opening a simple database query and building and testing the expression within the query editor. From within the BI studio, you can right-click the table and choose Explore Data to ensure you entered the expression correctly, but the database query window is more interactive. You also need to add a Value calculation that multiplies SalePrice by QuantitySold. To create a date hierarchy, you also need to add SaleYear and SaleMonth to the Sale table. Create new calculations and use the Year() and Month() functions to define these items.

Action

Create a new Dimension based on the Sale table using only SaleDate, SaleYear, and SaleMonth.

Set the Year, Month, and Date data types.

Create a new hierarchy starting with SaleYear, then SaleMonth, and SaleDate. Save the dimension as SaleDateH and test it.

Create a new dimension with at least Color, ItemMaterial, Manufacturer Name, ShipState, and Gender.

Another useful trick is that you can add a Friendly Name to any column or table within the Data Source View. Simply right-click a table or column and choose the Properties option. Scroll through the list to find the Friendly Name entry and type in a new name. This new name will be displayed to the managers, but the queries will still extract data based on the underlying real name. Friendly names are important when your tables and columns came from transaction databases that used abbreviations and hard-to-understand names. This is your opportunity to provide names that are clear to managers.

The third major step is to select basic dimensions and create any hierarchies that will make it easier to browse the data. Hierarchies are important to cube browsers because they enable managers to see overall totals and still drill-down to see the detail. The most common hierarchies are based on dates. Managers often want to see totals by Year-Month-Date, but you also often have to include quarters and weeks. Other natural hierarchies exist for geographic regions (continent-nation-state-city). Businesses also define hierarchies in terms of accounts in the general ledger, customers, plants, and employee job. Plus, companies often create custom hierarchies to meet their individual requirements. In SQL Server, the process of creating hierarchies is similar regardless of the type of data. However, many of the common hierarchies are predefined within the Dimension tool to make them easier to create.

To illustrate the process, create the simple Year-Month-Date hierarchy for the SaleDate. Save your work and close the Data Source View if you want to keep it out of the way. In the Solution Explorer, right-click the Dimensions entry and choose the New Dimension option. Keep the option to use an existing table (later, you can experiment with the time table option). Select the Sale table, leaving the key and name columns at their defaults. Uncheck the PaymentMethod and Customer entries in the Related Tables screen. Uncheck PaymentMethod and CustomerID in the Select Dimension Attributes screen.

Now select the SaleDate, SaleYear, and SaleMonth attributes, but pause for a second. At this point, the values are simply unrelated data. Because dates are commonly used in hierarchies, you need to tell the Dimension wizard the role of these three items. For each item, click the “Regular” entry next to it and select the drop-down arrow to open up your choices. Expand the Date and Calendar entries. Scroll through the list to assign the appropriate role to each value: SaleDate=Date, SaleYear=Year, SaleMonth=Month. Figure 9.17 shows the initial choices for the

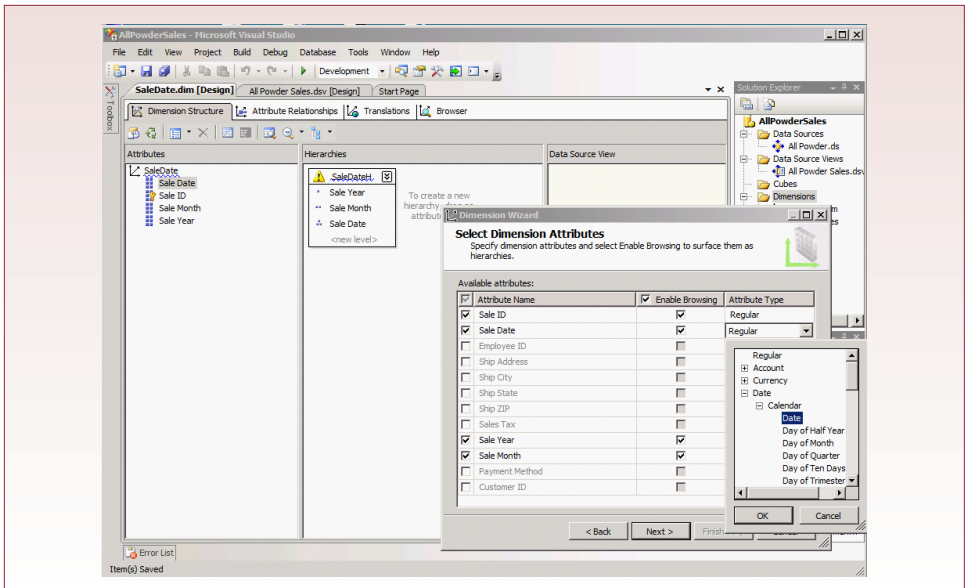


Figure 9.17

SaleDate attribute. You might want to look through the list to see the many other options that you can use for other types of data. Be happy that this lab is only using the three easy ones. But do not get too happy, you still have another step to complete.

Once you finish the wizard, you will see the design screen shown in the background of Figure 9.17 with the Attributes and (empty) Hierarchies panels. You create the date hierarchy by dragging the SaleYear attribute onto the Hierarchies panel. Then drag the SaleMonth attribute and drop it onto the <new level> spot. Do the same for the SaleDate. Rename the Hierarchy to SaleDateH so you (and the managers) remember what it represents. To test the dimension, right-click its name in the Solution Explorer and choose the Process option. When the main form appears, click the Run button. When processing completes, close the processing forms and click the Browser tab for the dimension. You should be able to expand the All entry to see the years, which can be expanded to see the month numbers. Note that it is possible to display month names instead of numbers, but it takes several additional steps, beginning with adding a MonthName column to the Data Source View using the DateName function.

Save your work and close this dimension. You now need to add the simple attributes into one large dimension group. Right-click the Dimensions row in the Solution Explorer and add a new dimension. Stick with Existing tables, and choose the SaleItem table. For the Name column, select the SaleID_SKU attribute that contains the concatenated keys. Keep all of the related tables. Now select the desired attributes, including Value and QuantitySold as facts. You can leave the attribute type as “Regular.” Your list should include at least the entries selected by default, plus: Color, ItemMaterial, Manufacturer Name, ShipState, and Gender. If you have time, you might want to add hierarchies for State, City, and possibly ZIP Code—which means you have to select those items now and build hierarchies later. These hierarchies can always be added later, because all of the objects you are creating are easy to edit. So, save your work and close the new dimension.

You are now ready to create an OLAP cube. Right-click the Cubes entry in the Solution Explorer and choose the New Cube option. Stick with the existing tables and pick the SaleItem table as the measure group (fact) table. You might as well use all of the entries (Quantity Sold, Sale Price, Value, and Sale Item Count) unless you need to limit the managers to just the Value entry. Keep both dimensions that you created and click Next and Finish to create the new cube. That is all it takes to create the cube, but it has to be processed before it can be used. Processing basically copies all of the data from the database and builds special OLAP tables to hold the final data to improve browsing speed. Whenever the underlying data gets reloaded, you will need to reprocess the cube; but the steps are relatively automatic.

Right-click the new cube name in the Solution Explorer and choose the Process option. Click the Yes and then the Run buttons to begin processing. Close the processing form when it is finished. You can right-click the cube name and choose the Browse option to run the cube, but this step is covered in more detail in the next section. .



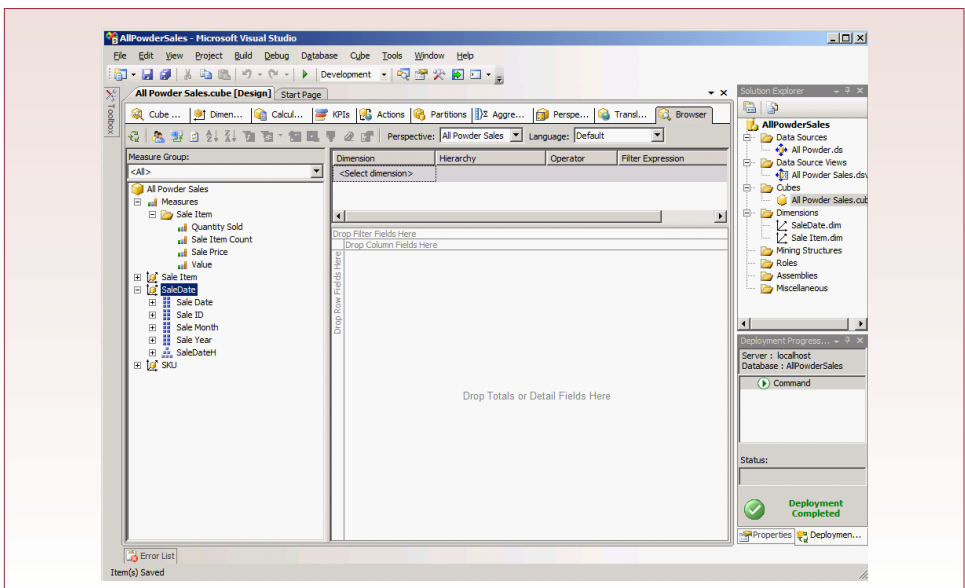
Activity: Browse the Sales Data OLAP Cube

Once you have created the cube and processed or deployed it, it can be used to search the data for various relationships. The easiest way to see how the cube works is to run the Browser built into the BI Developer Studio. If necessary, restart the Business Intelligence Studio and open the project. To start the browsing process, simply right-click the name within the Cubes section of the Solution Explorer and choose the Browse option.

Action

- Right-click the new Cube name and choose Browse.
- Set SaleDateH as the column field.
- Set Product Category as the row field.
- Place Value in the total section.
- Add Color, Manufacturer Name, and Ship State to the filter fields.
- Play with the cube to test the options.

Figure 9.18



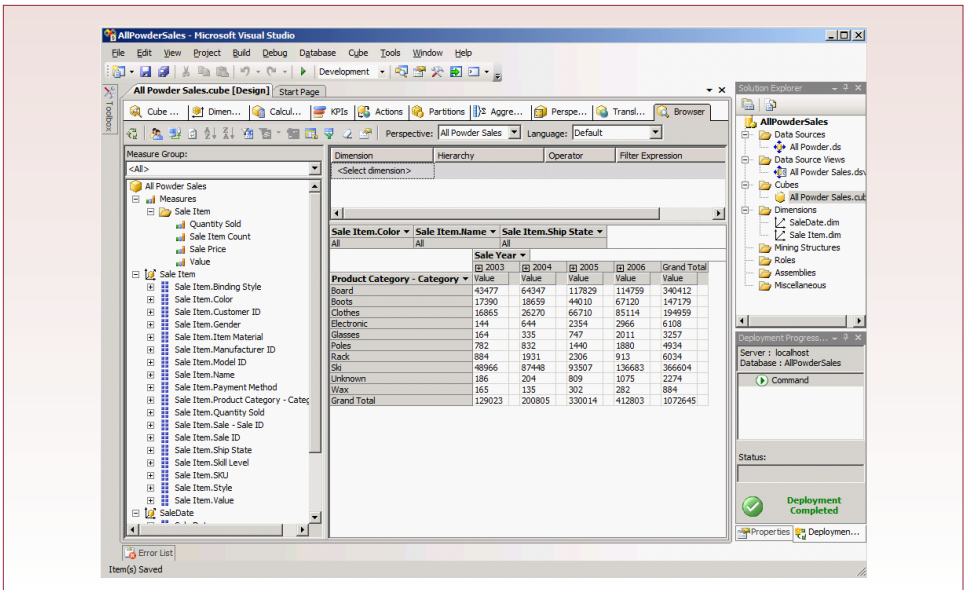


Figure 9.19

Figure 9.18 shows the basic outline of the empty cube. The left panel contains the list of fact and dimension attributes. Notice that they are organized into folder groups. You can create your own groups if you have a large collection of attributes, but it is not required. The cube structure is the most important element. Notice that it contains four major areas: Row Fields, Column Fields, Filter Fields, and Totals or Detail Fields. Remember that the purpose of the cube is to display subtotals (and details) for a given fact, based on various dimensions. Hence, you place a fact measure in the middle of the cube (Totals) and then position dimensions as either row or column fields. There is no difference between using rows or columns, but typically you place smaller dimensions as columns because a page can display more rows than columns. But the choice is not critical because managers can simply drag the fields around to wherever they want them. Filter fields are a little trickier. Anything placed in a Filter field can be used to restrict the data shown in the entire cube.

It is easiest to see the differences with an example. Expand the SaleDate dimension and drag the SaleDateH hierarchy to the Column Fields area. Expand the SaleItem dimension and drag the Product Category attribute to the Row Fields area. Expand the Measures, then Sale Item entry and drag Value onto the Totals area in the middle of the cube. For comparison, drag Color, Manufacturer Name, and Ship State from the Sale Item dimension onto the Filter Fields area. Initially, these three are set to “All” so they have no effect on the data.

Figure 9.19 shows the initial cube. To understand the value of hierarchies, click the plus sign (+) to expand a year. Instantly, you get subtotals for each month for each category. You can further drill down within a month to see sales on a specific date.

Another useful trick is to remove some items from the list. Click the drop-down arrow on the top-right of the Product Category list. Uncheck everything except the Board and Ski entries. When you click OK, you can focus on just the sales of the snowboards and skis.

Similarly, managers can open the Filter fields and choose which values should be displayed in the table. Try it by opening Color and deselecting some of the colors. Now, all values in the cube represent just the colors that remain. Reset the options by opening the Color field and clicking the All entry. Do the same for the Product Category. Now open the State field and click the All entry to deselect all items, and click just your state to see sales within that state.

If you, or managers, want to transpose the cube, you can drag the column date hierarchy entry over to the rows and drag the product category attribute to the column heading area. You can also drag attributes from the top-of-the-page filter area onto the cube itself. The cube is a flexible approach to viewing data from almost any perspective. However, it is important that the cube contain all of the fields that managers want to evaluate. It is possible to add new fields to the lists, but you have to go back and revise the Data Source View and dimensions, and then reprocess and redeploy the cube.

To set the format of the Value item—for example as currency—you have to go back to the cube definition. Click the Cube tab on the left of the designer. Select the Value attribute and examine its properties. You can set the DataType to Currency, or even specify a detailed format string. But, you will have to rerun the Process and Browse commands to see the changes.

For one more useful trick (there are many others), right-click a Value heading in the cube and choose the Show As option. Then select Percent of Grand Total to get percentages instead of totals.

Introductory Data Analysis



Activity: Analyze Time-Series Data

SQL Server provides some relatively powerful data mining tools—particularly in the 2008 edition. The good news is that they are relatively easy to use. However, ultimately, managers need to be familiar with the underlying statistical theories. This point is illustrated even in a simple time series forecast.

As a developer, one of the most important points to remember about Microsoft's Time Series tool is that it requires a data column that contains a time variable. This column must be numeric and continuous and it must be a single column. In other words, you somehow need to combine year and month into a single value. In most cases, the easiest solution is to create a query that basically counts the number of months. If you have limited data and some months are missing, you can use an outer join with a calendar table to fill in the missing months.

To analyze time series data, start with a new BI Analysis Services project. Sometimes you can simply embed the data mining tools into the existing data cube project, but for the lab it is better to create a new project to reduce errors by keeping the processing separate. You will need to add a new Data Source (right-click in the Solution Explorer). However, you should be able to reuse the existing connection—simply select the value shown. If you do not have an existing connection, click the New button and enter the basic login credentials.

More importantly, you need to create a new Data Source View. Even if you are reusing an existing project, it is important to build a new view. Right-click Data

Action

Start a new BI Analysis Services project.

Add a data source, but you can reuse the existing connection.

Create a new Data Source View, but do not add any tables.

Create a Named Query that totals $\text{SalePrice} * \text{QuantitySold}$ and computes a date sequence: $(\text{Year}(\text{SaleDate}) - 2003) * 12 + \text{Month}(\text{SaleDate})$.

Source View in the Solution Explorer and choose the option to build a new one. The trick here is to not pick any tables. Simply leave that screen blank and click Next. When you get to the View designer, right-click the main screen and pick the option to Create Named Query. (You also could have created this query within the database and then imported it as a table.) Use the query builder or simply type the SQL:

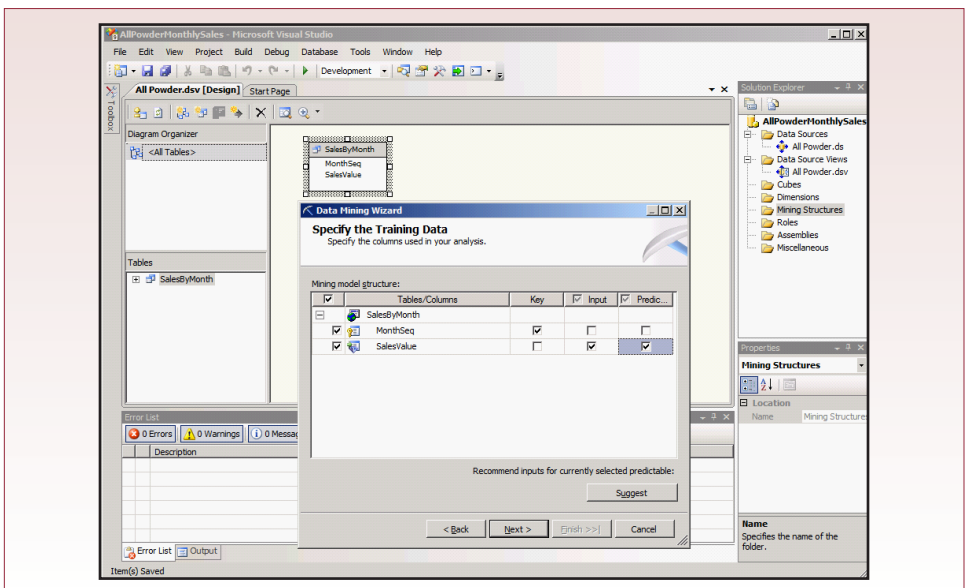
```
SELECT (Year(SaleDate)-2003)*12+Month(SaleDate) As
MonthSeq,
       Sum(SaleItem.QuantitySold*SaleItem.SalePrice) As
SalesValue
FROM SaleItem
INNER JOIN Sale ON SaleItem.SaleID = Sale.SaleID
GROUP BY (Year(SaleDate)-2003)*12+Month(SaleDate)
```

This query computes the total sales (price times quantity) for each month. It also computes a month sequence number that numbers each month consecutively. This sequence number is required by the Time Series tool. Notice that you need to know the starting year (2003). If necessary, you could use a subquery to compute this value, but the query is faster if you simply plug in the known value. Click the Save All button to save your work. Right-click the query and choose Explore Data to ensure the query works. Note that the data is not sorted—because SQL Server does not allow you to save an ORDER BY clause in a stored query. Do not worry, the Time Series tool will handle the sorting automatically.

Create a new data mining structure by right-clicking Mining Structures in the Solution Explorer and choosing New Mining Structure. Pick the option to use existing tables and choose the Microsoft Time Series technique from the drop-down list. Choose the Monthly Sales data source view that you just created. Accept the default choice of the single table.

The next steps are critical. As shown in Figure 9.20, set SaleYearMonth as the Key and set the Input and Predictable options for the SalesValue attribute. The Key

Figure 9.20



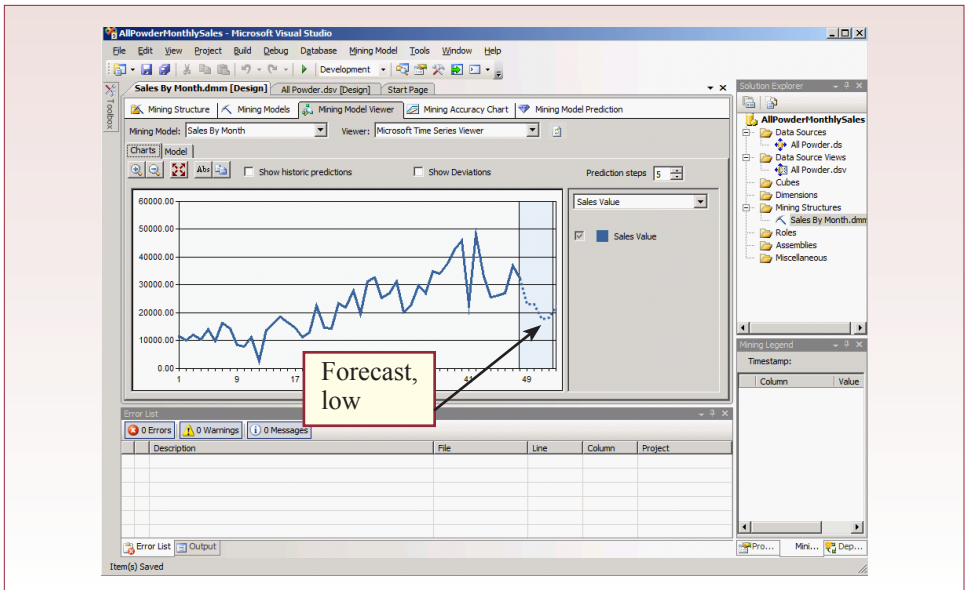


Figure 9.21

column must always be the time series sequence value. The Input and Predictable indicators are used for the primary data that you wish to forecast—often sales. The SQL Server method also allows you to add other attributes that might affect the time series. For example, you might want to know how the location or age of a customer affect the prediction. If these values are in the Data Source View, you can select them on this screen by checking the Input (but not the Predictable) option. They are handled using a Decision Tree approach. For now, stick with the two variables and follow the prompts to close out the wizard with the default values.

To see the results, you need to process the model and then switch to the viewer. In the Solution Explorer, right-click the new data mining structure name and choose the Process option. Follow the prompts to run the processing. When it is finished, close any open windows. Return to the Solution Explorer and right-click the model again, this time choosing the Browse option. To see the results, click the Mining Model Viewer tab on the main screen.

Notice in the resulting chart shown in Figure 9.21 that the forecast (in the shaded region) seems low. The Microsoft Time Series tool uses two approaches to forecast trends: (1) A decision tree that tries to factor the data into relevant groups, and (2) ARIMA—a standard statistical time-series tool. ARIMA (autoregressive, independent moving average) relies only on the underlying data series to estimate basic patterns in time series data. In this situation, it will be more accurate than the decision-tree approach.

Action

- Create a new mining structure choosing the Time Series technique.
- Use the Monthly Sales data source.
- Set SaleYearMonth as the Key.
- For SalesValue, check the options for Input and Predictable.
- Process the new structure and Browse it.
- Set the Algorithm Parameters to use only the ARIMA model and suggest a 12-month period.
- Reprocess and browse the model to evaluate the forecast.

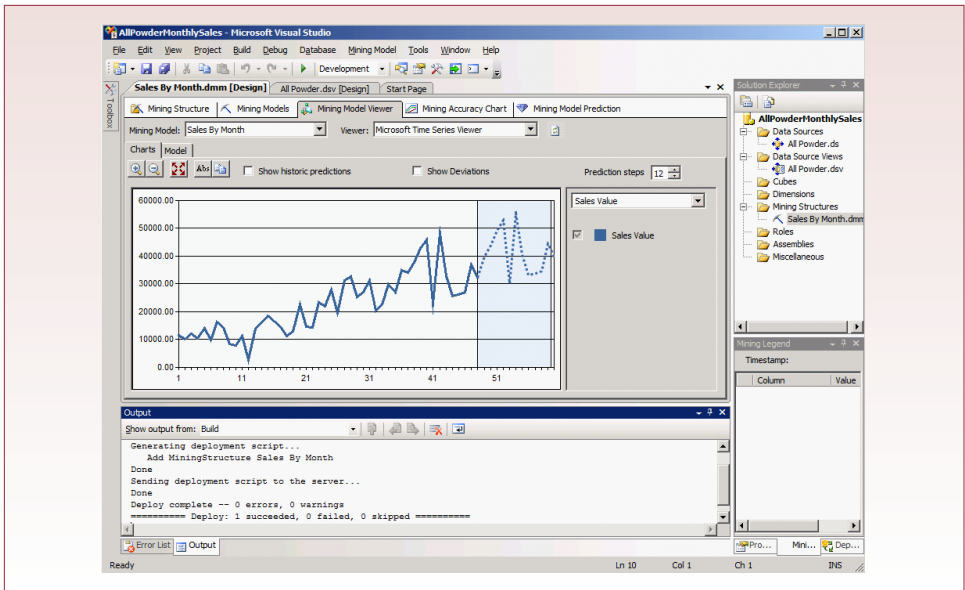


Figure 9.22

To alter the way the BI tool computes the forecast, you need to change a couple of parameters. (These steps are slightly different if you are using BI 2005 instead of 2008.) Click the Mining Models tab to return to the model. Right-click the Sales By Month header (or anywhere in that column). Choose the Set Algorithm Parameters option. Find the Forecast Method row and click the cell in the Value column. Enter ARIMA to force the model to use only the ARIMA approach. Since sales data probably also has an annual pattern, find the Periodicity Hint row and click the cell in the Value column. Enter {12} or perhaps {1, 12} to suggest that there is a 12-month cycle. Click OK to accept the changes.

Whenever you change the model or data structures, you have to reprocess the model. Right-click the Sales By Month heading and choose Process Mining Structure, followed by All Models. Click the Run button. When processing is complete, close the open windows. Switch to the Mining Model Viewer tab and click Yes to refresh the view. As shown in Figure 9.22, increase the Prediction Steps to 12 (months). The resulting forecast looks much more reasonable than the initial forecast. Notice that the ARIMA tool does a good job at picking up seasonal patterns.

Overall, the time series forecasting tool is relatively easy to use. However, you cannot just blindly follow the default suggestions. Developers have to work with managers to recognize when initial forecasts are weak, and then understand the tools (ARIMA) to know how to improve the forecast.



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. SQL Server 2008 BI tools has a version of linear regression, but it is based

Action

- Use the database studio to import the demographic data from Excel.
- Create a view to compute sales by state for 2006.
- Create a view to list the state, sales, population, and income.
- Start a new BI project and build a Data Source View from the database view.
- Create a new Mining Structure to apply Linear Regression to the Sales data.

on the Decision Tree algorithm so the results are presented somewhat differently. 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 or fit. In the classic multidimensional case, regression is used to determine how various dimensions (independent variables) statistically affect the fact (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 income level or population of the state. In this case, the income and population 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.

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 for an enormous amount of data. For this exercise, the 2006 population and 2006 per-capital personal income by state have been saved in an Excel spreadsheet. The worksheet is relatively easy to import into SQL Server as a one-time import. Start the database Management Studio. Right-click the All Powder database entry, select Tasks and Import. Select Excel from the drop-down list and browse to the location of the Excel file (which can be downloaded from the book's Web site). Import the data from the StateDemographics worksheet which has column names in the first row. This action will create a new table.

You next need to create a query that computes the sales by state for 2006. You could use all of the years, but it makes more sense economically to stick with a single year. It is straightforward to build the view within the main database:

```
CREATE VIEW StateSales2006 AS
SELECT Sale.ShipState, SUM(SaleItem.SalePrice*SaleItem.
QuantitySold) As Value
FROM Sale
INNER JOIN SaleItem ON Sale.SaleID=SaleItem.SaleID
WHERE Sale.SaleDate between '1/1/2006' and '12/31/2006'
GROUP BY Sale.ShipState
```

Now you need to match the state demographic data with the state sales data, which is relatively easy with a JOIN query:

```
CREATE VIEW StateSalesFactors AS
SELECT StateDemographics$.State, StateDemographics$.
Pop2006, StateDemographics$.Income2006, StateSales2006.
Value
FROM StateDemographics$
INNER JOIN StateSales2006 ON StateDemographics$.
State=StateSales2006.ShipState
```

This view now contains a simple list of the state, population, income, and sales. It would be easy to copy the data and paste it into Excel for further analysis. However, you can also use the BI tools to analyze the data—which makes it easier to share the results; or to apply additional tools.

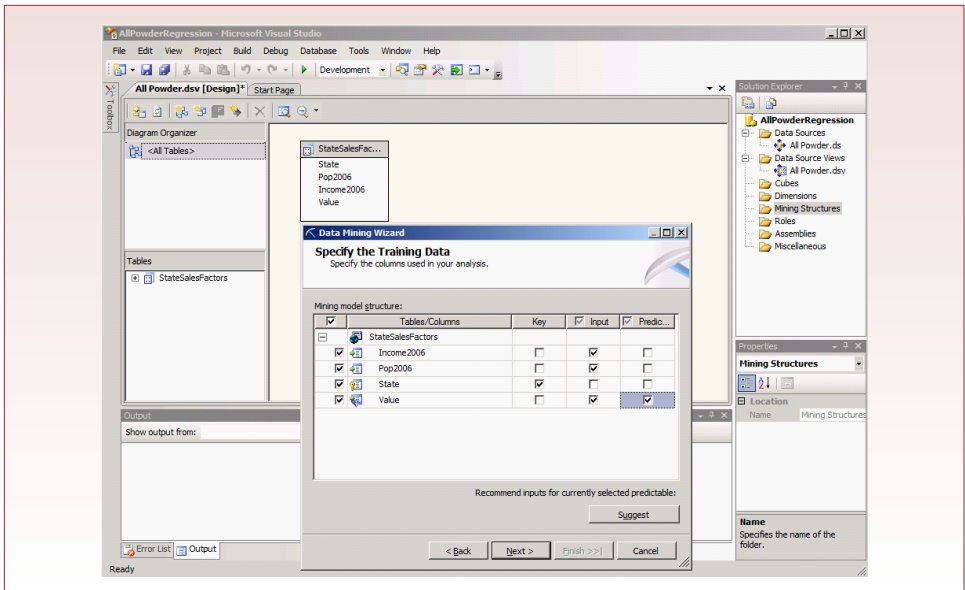


Figure 9.23

Start a new BI Analysis Services project (AllPowderRegression). Use the Solution Explorer to add a new Data Source. You should be able to reuse the connection created for the other labs, otherwise you will have to re-enter your login credentials. Create a new Data Source View and pick the the StateSalesFactors view you just created.

You are now ready to right-click the Mining Structures entry in the Solution Explorer and create a New Mining Structure. Choose the option to use existing tables. Select the Microsoft Linear Regression technique from the drop-down list and accept the default values until you get to the Training Data screen. As shown in Figure 9.23, select the State as the Key attribute, check the Input option for the Income and Population attributes, and select both Input and Predictable for the Value (sales) attribute. Click the Next button. The system automatically reserves data for testing (versus estimation or training). With a limited number of observations, you want to use most of them for training, so reduce the percentage for testing down to 10. Finish the wizard and use Save All to save your work.

Right-click the new mining structure in the Solution Explorer and choose the Process option. Click the Run button. When processing finishes, close the open pop-up windows. Right-click the mining structure and choose the Browse option, then switch to the Mining Model Viewer tab to see the outcome.

Figure 9.24 shows the basic results. Users with experience in statistical systems will find it hard to read. The best answer is to expand the Mining Legend

Action

- Create a new Data Mining Structure using the Linear Regression technique.
- Set State as the Key, Income2006 and Pop2006 as Input types, and Value needs the Input and Predictable options.
- Reduce the percentage of testing data to 10.
- Process the model and Browse it.
- Expand the Data Mining Legend to see the results.
- In the Model Viewer, set the Algorithm Parameters to Force_Regressor={Income2006}
- Reprocess and Browse the results.

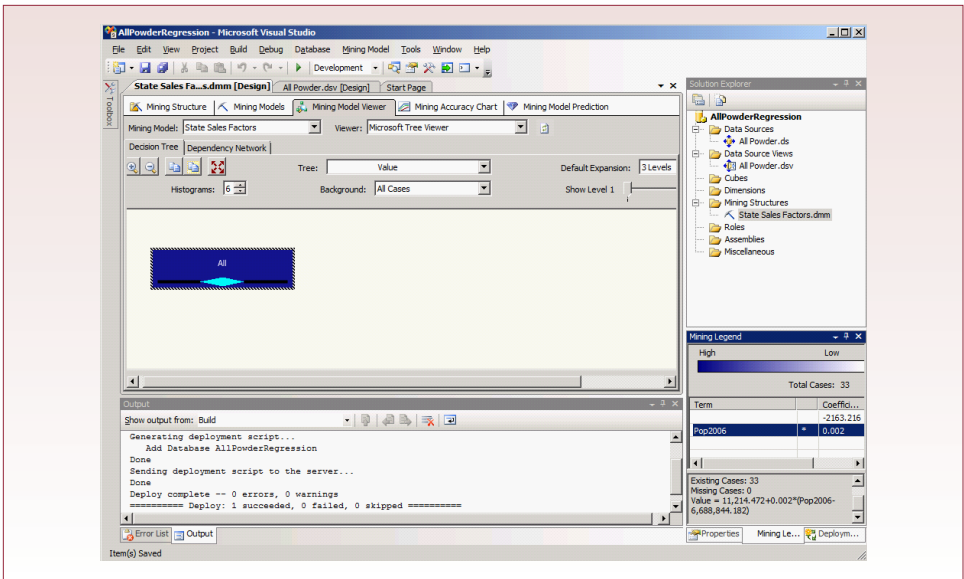
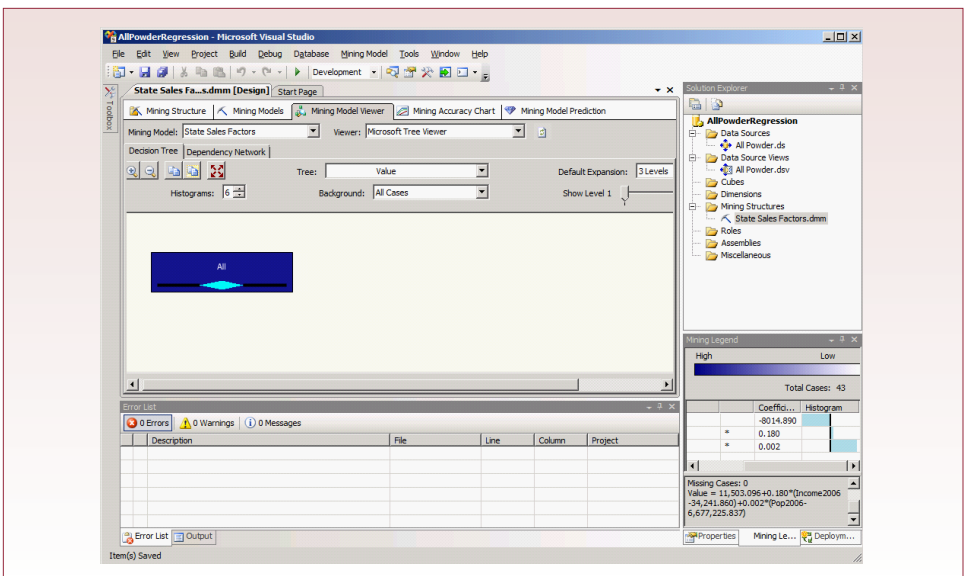


Figure 9.24

tab in the lower-right corner. The coefficients and basic regression equation are shown in the grid and the bottom section of the legend. First, notice that the system subtracts the mean from the input variables (e.g., Pop2006 – 6,688,844) so the constant coefficient is slightly different than in traditional regression. More importantly, the system does not report significance or coefficient estimates. However, the system simply does not report coefficients that lack significance—hence Pop2006 is included because it affects sales, but Income2006 is not displayed so it is insignificant.

It is possible to force variables into the regression computation. Click on the Mining Models tab to make some minor changes. Right-click in the State

Figure 9.25



Sales Factors column and choose the Set Algorithm Parameters option. Find the Force_Regressor row and click the cell in the Value column. Enter the value: {Income2006} with the braces. You can add other columns by separating the names with commas. Click the OK button to close the editor.

Right-click the State Sales Factors heading and pick Process Model option. When the reprocessing finishes, you can choose the option to Browse the model and return to the Mining Model Viewer tab to see the updates. Figure 9.25 shows the new results. Again, focus in the Mining Legend area and you will see that the Income2006 attribute is now included. Without significance values, it is difficult to evaluate the coefficients. However, scroll the Legend area to the far right and check out the histogram. Notice the relatively small value attributed to the Income2006 attribute versus the other two attributes (constant and Pop2006).

These results are accurate—the sample data was generated based on population without income. However, actual data from a real store would be more interesting and should indicate responses to more variables. You can use a similar process to use more complex tools such as non-linear regression, neural network, and Bayesian analysis..



Activity: Analyze Association Rules for Market Baskets

Although it requires some effort to use, the Oracle data mining package can provide some useful results. In particular, the association rule or market basket analysis is a classic data mining tool. The goal is to search for items that are likely to be purchased together. For instance, a person who purchases cross country skis might commonly purchase gloves. Knowledge of these patterns could provide insight into your customers and give you tips on how to train your sales staff to increase sales. On the other hand, the correlations could be purely arbitrary. Managers must critically evaluate all results to see if they make sense and determine if they can be applied to the individual store.

Microsoft's Association Rules tool is relatively easy to use and has some nice graphical features to make it easy for managers to search for interesting rules. The key to any market basket approach is to understand that you need a dataset that contains an identifier for the sale (SaleID), and a list of items purchased at each sale (SaleItem table). However, remember that the Inventory table (SKU) identifies every product variation carried by the company. For instance, skis of different lengths have different SKUs even though they represent the same model. Partly to reduce the number of dimensions, but mostly to make the results clearer, you want to examine sales baskets in terms of the broader ItemModel definition (a downhill ski is a downhill ski regardless of length or manufacturer). This reduction will be handled by creating a new Model attribute that concatenates the Category with the Style. For example, someone could purchase a Ski_Downhill.

Begin by creating a new Analysis Services project (AllPowderBasket). In a real project, you would probably build most of these tools into a single project, but for now it is safer to keep them apart so errors in one lab do not affect another one. In the Solution Explorer, right-click the Data Source entry and add a new one. You should be able to base it on the existing connection. If it is not displayed, click the

Action

- Create a new Analysis Services project.
- Add a new Data Source, reusing the existing connection.
- Create a new Data Source View with just the Sale table.
- Add a Named Query to the View using SaleItem, Inventory, and ItemModel tables.
- Set the primary keys and connect the NamedQuery to the Sale table.

New button and enter the connection and login information. Pick the Use service account option.

Right-click the Data Source View entry to add a new data source view. Pick only the Sale table and finish the wizard. Instead of the SaleItem table, you will build a query to compute the simplified Model definition. Right-click an open space in the View designer window and pick the option to create a Named Query. Name it SaleModel. Pick the SaleItem, Inventory, and ItemModel tables. Use the query builder to create the query, or copy the SQL:

```
SELECT SaleItem.SaleID, SaleItem.SKU,
ItemModel.Category + '_' + Coalesce(ItemModel.Style, '') As
Model
FROM SaleItem
INNER JOIN Inventory ON SaleItem.SKU=Inventory.SKU
INNER JOIN ItemModel ON Inventory.ModelID=ItemModel.
ModelID
```

The Coalesce function is important because some of the model Style entries are Null, and the analysis system does not allow Null values. Coalesce automatically picks the first value (Style), unless it is Null, in which case it switches to the second value (empty string). Test the query and accept it.

In the designer, you need to assign the primary keys to the new query, so select both the SaleID and SKU columns. Right-click them and pick the option to assign the Logical Primary Key. Now drag just the SaleID from the new SaleModel table and drop it on the SaleID column in the Sale table. Ensure that the columns are correct in the rela-

Action

Create a new Mining Structure and choose the Microsoft Association Rules.

Select the Sales table and mark the Case option.

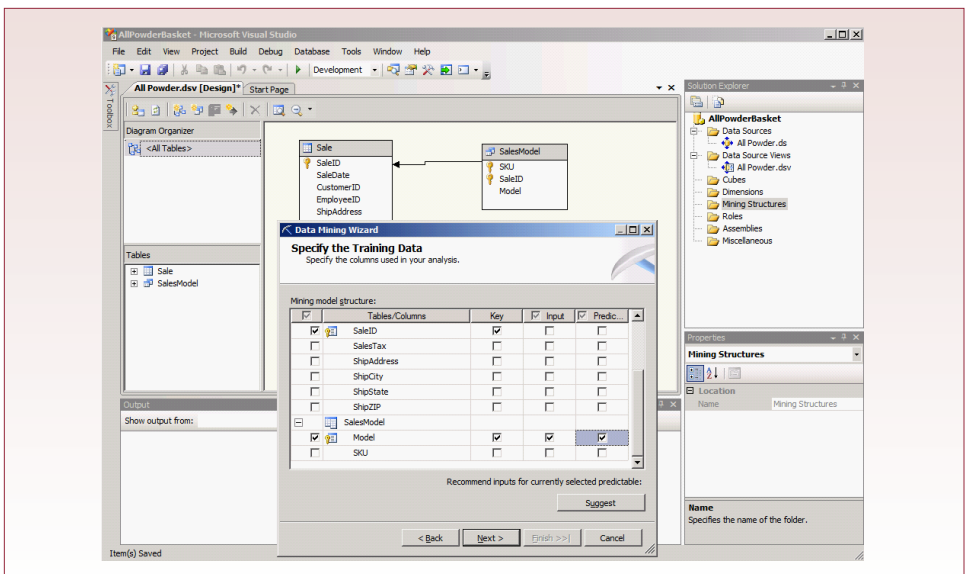
Select SaleModel and select the Nested option.

Choose the SaleID attribute and set the Key option.

Choose the Model attribute and set the Key, Input, and Predictable options.

Select the Allow drill through option at the end.

Figure 9.26



relationship box and accept it. Click the Save All button to protect your work and run the Explore Data option to check the result.

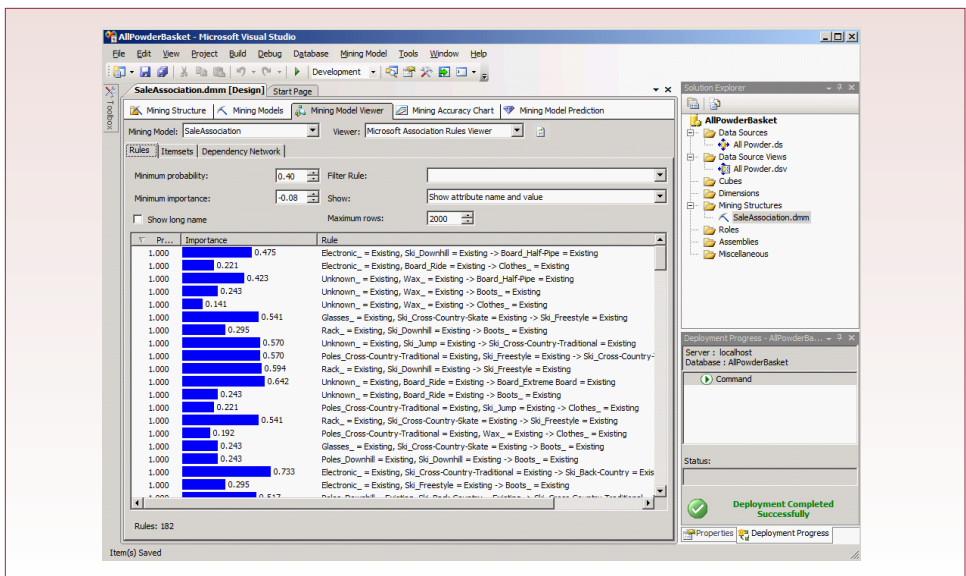
You can now create a new Mining Structure by right-clicking that option in the Solution Explorer. Use the existing data option and choose the Microsoft Association Rules entry in the technique drop-down list. Select table Sales and set the Case option checkmark next to it. Select the SaleModel table and check the Nested option to indicate there can be many models for each sale. Click the Next button.

You need to choose only two columns on the column selection screen: SaleID and SaleModel. But, Figure 9.26 shows that you have to be careful in setting the options. The SaleID should have only the Key option set. The Model attribute needs to have all three options (Key, Input, and Predictable) checked. You do not need any of the other columns. Click the Next option to continue. On the last screen, check the Allow drill through option, and click the Finish button to close the wizard. Click the Save All button.

You can now Process and Browse the results. In the Solution Explorer, right-click the new data mining entry and choose the Process option. When the control screen pops up, click the Run button. Processing could be time consuming in large projects with many sales and too many models. Close the popup windows when the processing is complete. Right-click the data mining entry and choose the Browse option.

Figure 9.27 shows the initial results. You should close extraneous windows to provide more space to see the list. Understanding probability (first column) and importance (second column) are critical to understanding the table. Probability is more commonly called “confidence” and it is the proportion of baskets containing X (the first item) that also contain Y (the second item). If Y is always purchased whenever X is purchased, the confidence is 1. Importance is typically called “support” and is the percentage of baskets in the database that contain both (all) items in the list. If two items are rarely purchased, the probability (confidence) could be high, but the importance (support) will be low because the event rarely occurs.

Figure 9.27



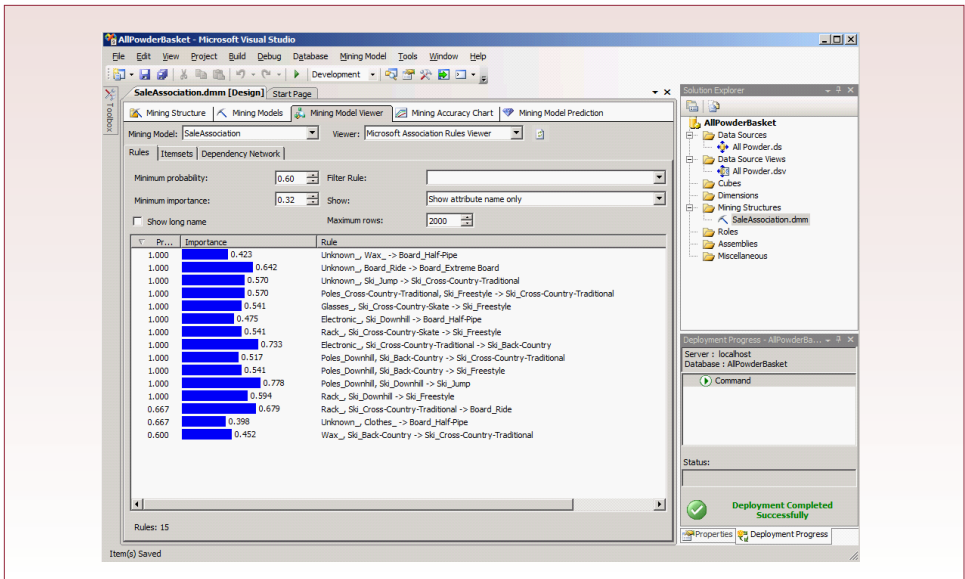


Figure 9.28

Figure 9.28 shows that you (or the manager-analyst) can clean up the results by changing the minimum values for probability and importance to hide weak rules. In the example, probability was set to 0.6 and importance to 0.32. Set the values and wait a couple of seconds for the display to refresh. The resulting rules are more likely to be valid. However, managers must still examine the rules to see if they make sense and can be applied correctly to the business. Also note that you can use the drop-down list to display just the attribute name—which makes the rules easier to read.

The data viewer contains two other tabs that are useful. The Itemsets tab displays a sorted list of item sets—or things that were purchased together. Essentially, it contains a list of the left-side of the rules. Figure 9.29 shows that the list is sorted by frequency of occurrence. In the example, Clothes appears by itself (Size = 1) 473 times. Similarly Board_Half+Pipe and Clothes appear together in 112 baskets. This list can help managers determine whether rules are important, or help them focus on specific combinations that have a large impact on sales.

The third tab (Dependency Network) is more fun, and more useful. Select the tab to get a network diagram of how the various items are related to each other. Again, you will want to select the option to show the attribute name only. Initially, the diagram shows all major itemsets and connections. Figure 9.30 shows that you can click one of the nodes to highlight it. By moving the slider down, you can filter out the weaker connections. The result in this case shows the five major items that directly affect the purchase of Boots. In this case, two of them—electronics and glasses—are somewhat unexpected. This result is probably due to a quirk in

Action

Process the rules and browse them.

Set the minimum probability and importance to reduce the number of rules displayed.

Check out the list of items in the itemset tab.

Switch to the Dependence Network tab.

Select a node and move the slider to remove the weaker rules.

Think about what the results would mean in terms of sales and marketing.

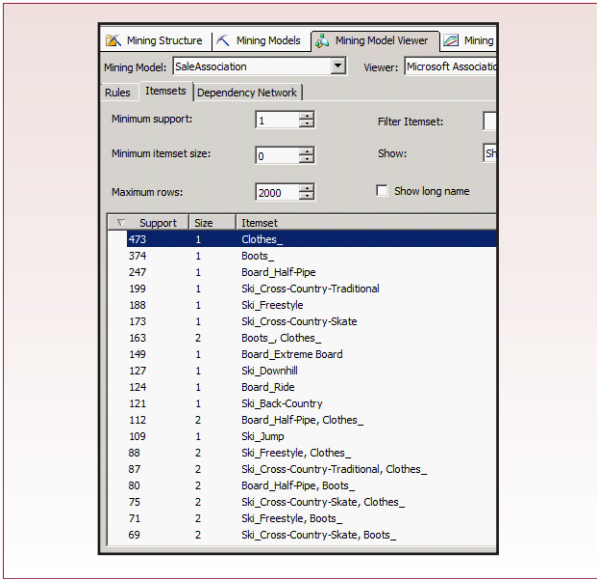
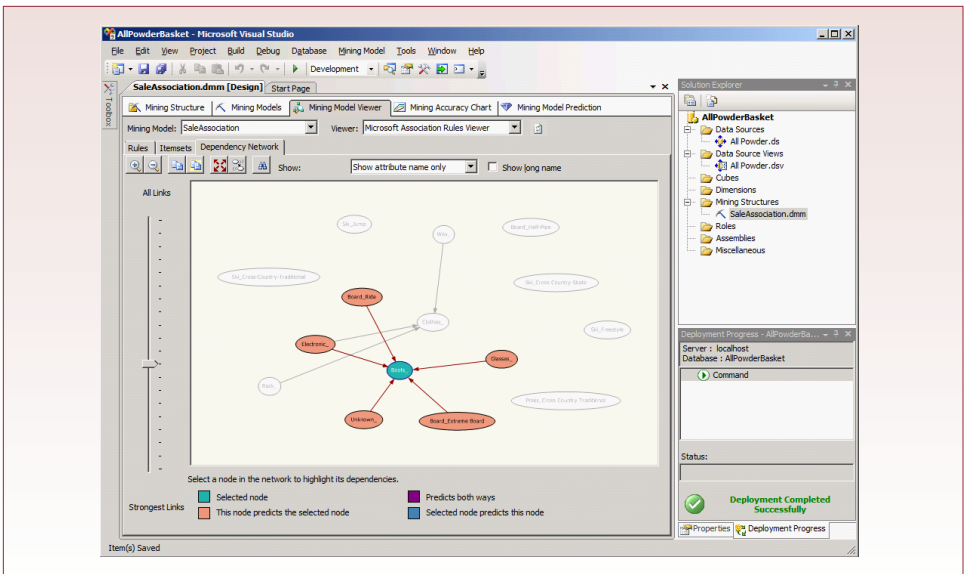


Figure 9.29

the generator used to create the data. However, if the data were real, the result could be used to target sales of new boots to customers who come in for electronics and new glasses/goggles. The slider makes it easy to explore relationships with any of the nodes.

Figure 9.30



Exercises



Crystal Tigers

The Crystal Tigers club does not have a huge amount of data to analyze within the organization. However, the club members 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 SQL Server 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 an OLAP cube that can be used by managers to analyze the data.
3. Create an OLAP cube 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 levels. Note that all employee tasks are supposed to be recorded in the system based on the client, job, and task involved. The firm has considerable information on clients, including a size classification (tiny, small, medium, and large), and type of company (such as printing 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 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 Oracle.
2. Create an OLAP cube 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 an OLAP cube 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 = printing shop, 2 = marketing, and so on). Perform a regression to see if the client size or industry influence the amount of sales revenue Capitol Artists generates.
5. Analyze the data with the association rules to see if there are relationships between the items purchased.



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 an OLAP cube 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 an OLAP cube chart 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 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 an OLAP cube to analyze the data.
2. Identify any data that could be analyzed over time, and create an OLAP cube chart and 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.

Database Administration

Chapter Outline

Database Administration Tasks, 200

Case: All Powder Board and Ski Shop, 202

Lab Exercise, 202

All Powder Board and Ski Shop, 202

Security and Privacy, 210

Exercises, 218

Final Project, 220

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 data with user-level security controls.
- Encrypt data in the database.

Database Administration Tasks

One of the powerful features of a major DBMS like SQL Server is its performance on large databases under a heavy load of users. However, obtaining this performance often requires detailed work by the database administrator. SQL Server provides several tools to monitor performance and tune the database. Advances in hardware and storage systems also play a role in handling huge databases. Although hardware details are not explored in this chapter, the monitoring tools can help tell you when you need to investigate new tools. Ultimately, large databases need to run the Enterprise version of SQL Server. Its primary contribution is the ability to spread workloads across multiple computers, but it also provides even more monitoring options.

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. You have seen that the Management Studio makes it easy to list databases, tables, and columns. However, as an administrator, you will often use the internal tables and views to search for information about tables. You can issue queries against this meta data, much the same as any other query.

SQL Server follows at least some of the SQL Standard defining some of the standard INFORMATION_SCHEMA views. More detailed meta data is provided in the proprietary sys views. Figure 10.1 lists some of the commonly-used elements of both views. You can find the complete list in the master table. Use the Management Studio to open the master table and expand the entries under the Views heading. Or you can get the list from a query: `SELECT * FROM sys.all_views ORDER BY name`. The beauty of the views is that you can retrieve the data with a simple query. You can also access the data programmatically, so you can write code to search for things, build backup structures, or transfer data automatically.

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, performance improvements simply

Figure 10.1

INFORMATION_SCHEMA	Description
CHECK_CONSTRAINTS	Table constraints and keys
COLUMNS	Table columns
REFERENTIAL_CONSTRAINTS	Foreign key constraints
TABLES	Tables
VIEWS	Views (saved queries)
SELECT Table_Name FROM INFORMATION_SCHEMA.Tables	
sys	Description
database_files	Storage files allocated
servers	List of servers available
synonyms	Synonyms/shortcuts
sysusers	User list
traces	Trace logs for performance
triggers	List of triggers
types	User-defined data types
SELECT * FROM sys.sysusers	

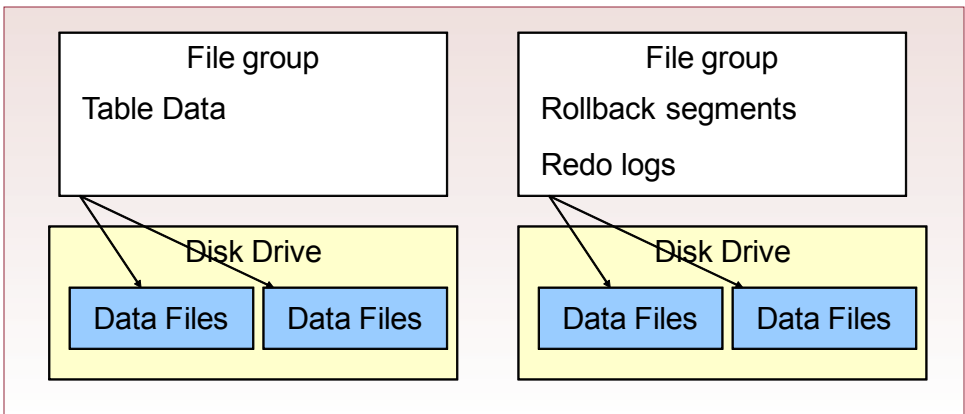


Figure 10.2

come down to “buy more processors and disk drives.” However, since one of SQL Server’s strengths is its ability to handle huge amounts of data, you will encounter some databases that will need changes to improve performance. To understand some of the performance controls, you need to be aware of how SQL Server stores data on the file system. At the base level, the DBA allocates data files on disk drives. If you are not using a RAID system to automatically store data on multiple drives, you can accomplish a similar effect by creating separate storage files on different disk drives.

To simplify installation and management, SQL Server handles most data-storage tasks automatically. However, tools exist to help you override the defaults and create custom configurations for more complex tasks. First, realize that all data is stored in files defined in the operating system. These files can be specified when you create a new database, or you can edit them later in the Management Studio by right-clicking a database, selecting Properties / Files option.

File groups are logical folders that can utilize multiple data files. Tables are assigned a specific file group to store the data. By default, SQL Server creates a primary file group with one (.mdf) file to hold the data. SQL Server also uses a special sequential log (.ldf) file to hold transaction and rollback data. As indicated in Figure 10.2, you get a substantial gain in performance if you store the table data and rollback segments in separate file groups on different drives. Two drives spinning independently means (1) the computer can write the data simultaneously, and (2) there is less chance of a loss in the event of a hardware failure.

Backup and recovery are critical aspects to a database designed to handle thousands of users and processes running at once. In many cases, the database must run 24-7, so you cannot stop it to make a backup copy. Consequently, even while you are backing up data, new rows are being added and data is changing. SQL Server has systems to protect all of this data, but if there is a hardware crash, you need to be careful about putting everything back together.

In some ways, security in SQL Server is straightforward. Security and user identification are an integral component of the DBMS. You also have the option of relying on Windows to identify users—this option is most useful in large systems where you are using Microsoft’s Active Directory to authenticate users across the network.

By default, users have minimal access to any data in the database. Consequently, the major security efforts consist of identifying the access that people need and

then enabling it with an SQL command. Of course, the security team will want to monitor system and database activity for potential breaches. Database triggers can be used to provide additional security controls by logging changes to sensitive tables.

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 does often hire 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 start changing form layouts or accidentally delete items.

The managers are also somewhat concerned about performance, particularly at the checkout machines. Sometimes the checkout lines get hectic, and the application has to be fast. Some of the issues can be handled by installing more computers, that way the salesperson can enter the basic customer data immediately, and the checkout clerk simply selects the customer and enters the product numbers. Of course, more computers mean that the company will need a network, and it means that more people will be simultaneously accessing the data, so the risk of collisions and locks increases.

Lab Exercise

All Powder Board and Ski Shop

DBMS developers learned early that indexes can significantly improve the performance of a relational DBMS. Primary key 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. SQL Server automatically builds indexes on primary keys. However, you also need to think about building indexes on foreign keys to provide performance gains for joining tables.



Activity: Monitor the Application Performance

Databases continually evolve over time. New data is added, new forms and reports are built and run, and users find new ways to explore the data. All of these changes alter the load on the DBMS, so performance changes over time. Every organization experiences periods of heavy use versus lighter usage. As tasks change, the effect on the DBMS can be huge. As DBA, you need to monitor the performance, forecast potential problems, and decide if performance can be improved with parameter and index changes or if it is time to implement new solutions.

Action

- Start the Database Management Studio.
- Right-click the server name and open the Activity Monitor.
- Run some queries to see if the monitor changes.
- Start the Windows Reliability and Performance Monitor and add counters for memory, drive, and some SQL statistics.
- Run some forms, reports, and queries to see if they influence the monitors.

SQL Server has three main tools to help you monitor performance: (1) monitoring reports within the Management Studio, (2) performance indicators for the Windows performance monitor, and (3) custom alerts e-mailed to you by the SQL Server Agent. SQL Server also has the SQL Server Profiler and Database engine

Performance Object	Counter
Processor	% Processor Time
Memory	Bytes Available
SQLServer: Access Methods	Full Scans/sec
SQLServer: GeneralStatistics	User Connections
SQLServer: Locks	Average Wait Time (ms)

Figure 10.3

Tuning Advisor to help you analyze and improve performance. Basic items can be monitored within the Management Studio. Right-click the server name in the Object Explorer and pick the Activity Monitor. Open a query window for All Powder and run a couple of queries. When you return to the Activity Monitor, you should see some basic activity in the processor, resource waits, and data file I/O charts. You can also expand the list of Recent Expensive Queries. If any of these basic items get out of control, your users will be unhappy with the performance. Of course, by the time these items indicate problems, it might take days or weeks to fix the problems. So, you need to keep an eye on trends within these indicator charts so that you can plan and implement improvements before they get out of control.

A drawback to the Management Studio reports is that you need to leave the Management Studio running. Also, you are limited to these basic items. Hence, the Windows Performance Monitor is a better choice for day-to-day monitoring. From Windows, run Start/All Programs/Administrative Tools/Reliability and Performance Monitor. The default view shows current usage of the CPU, Disk, Network, and Memory. But, the monitor is a powerful tool with several options. Expand the Monitoring Tools folder and select the Performance Monitor. The default chart shows CPU usage. Click the plus sign button on the main toolbar to add counters. Scroll down the list of categories to get to the large group of SQLServer entries. You have many choices, and DBAs eventually choose their favorites. Figure 10.3 shows a couple of choices. Typically, you want to monitor the processor, memory, and SQL server potential problems. You also might keep an eye on disk usage.

Many of the variables will change slowly over time, which helps you in long-run planning. But, if one suddenly spikes, you know you have a problem. DBAs often keep a performance monitor window open on the desktop to watch it during the day. Even though it rarely needs immediate attention, it provides a constant monitor on the status. Figure 10.4 shows a sample screen, but you can resize it can focus on the variables that are most limiting in your situation. The CPU and memory indicators can help the DBA identify hardware bottlenecks. If the system is constantly low on memory or the processors maxed, you need to either improve the database performance or purchase additional hardware. The graphs on wait times indicate if some process is blocking others. The main point of the chart is that you should watch it over time to monitor trends and catch problems before they become huge. In a production environment, you will also want to use the SQL Agent to monitor some critical items and send you e-mail alerts so you can stop impending disasters.

In terms of a development machine, these numbers typically will remain low. You can run a few queries or fire up some reports to see if the charts move, but the information will not be very useful. On the other hand, you can obtain stress-tester software that will enable you to set up a test environment and throw thousands of

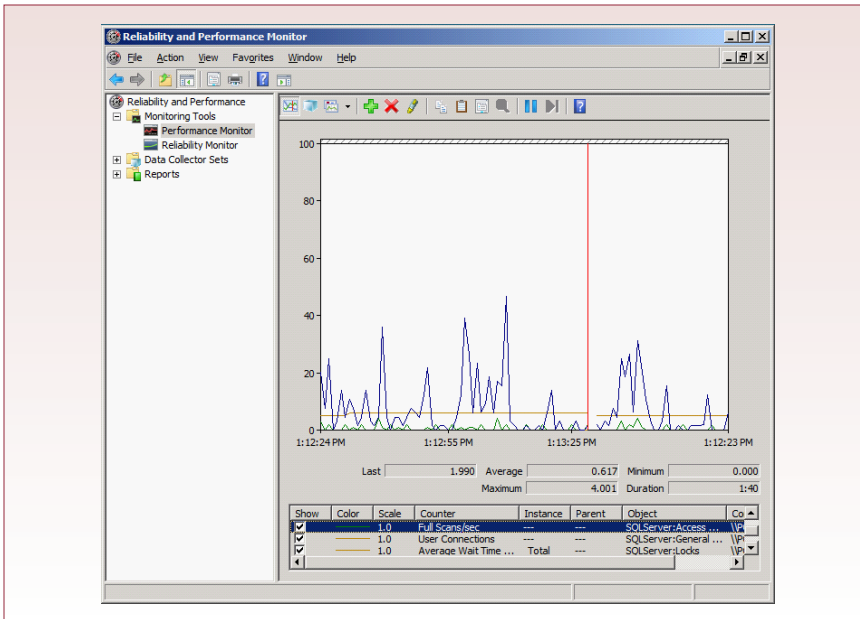


Figure 10.4

transactions at the database. Then you can watch the counters and statistics and figure out where the bottlenecks are likely to arise. You can also capture ongoing production data using the SQL Server Profiler and have the test environment scale it up to see what happens with realistic data scenarios. In both cases, once you encounter limits and problems, you can use the other performance tools to figure out how to improve the database system.



Activity: Expert Index Recommendations

Because of the complexity of managing large databases, SQL Server provides several tools for tuning the database performance. Although it takes years of experience (and reading) to become an expert in tuning databases, you can practice with a couple of tools to learn some of the main concepts. One of the easiest tools to use is the Database Engine Tuning Advisor. You can start it from the main Windows Start menu or from the Tools menu within the SQL

Server Management Studio. However, the Tuning Advisor needs a sample of commands to work with. The best approach is to create a workload sample captured by the SQL Server Profiler. Basically, you start the Profiler, run through typical operations with forms and reports (or capture actual transactions in a production environment), and save the workload into either a file or a database table. The other option is to write a SQL script that performs several tasks that you want to analyze as a group. This script could be as simple as a group of SELECT queries that execute within a Query window. If you use this approach, save the SQL script in a file with a .sql suffix.

Action

- Start the SQL Server Profiler.
- Work through typical applications in the application from Chapter 8.
- Stop the Profiler
- Open the Database Engine Tuning Advisor.
- Browse to the Profiler file data and run the Advisor.
- Evaluate the recommendations.

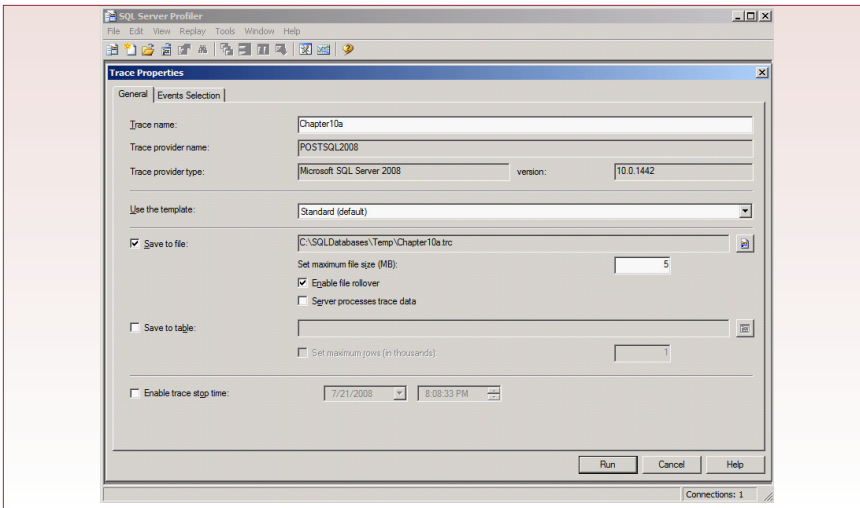
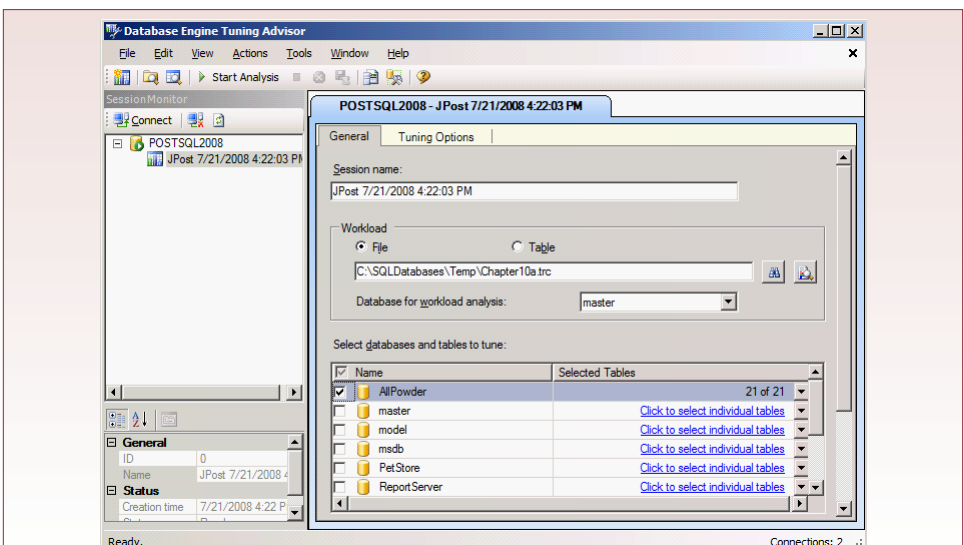


Figure 10.5

For now, the fastest approach to getting a workload sample is to simply run through the sample forms and reports created earlier. You might want to test them first to ensure that they connect to the database and work reasonably well. To create a workload, open the SQL Server Management Studio if necessary. Use Tools/SQL Server Profiler and log in with administrator privileges. Enter a name for the trace (Chapter10a). Accept the standard (default) template. As shown in Figure 10.5, check the box to save to a file. Use the browse button to find a location to store the file. Remember this location. Increase the maximum file size to at least 50 MB. When you are ready to proceed, and ready to run the All Powder application, click the Run button to start recording the trace file. Start up the All Powder application and work through whatever forms and reports are available. Try to mimic a typical workload—add a Sale and a Rental. When you are finished, close the All Powder application and return to the Profiler. Click the red Stop button to stop the trace recording. Close the Profiler.

Figure 10.6



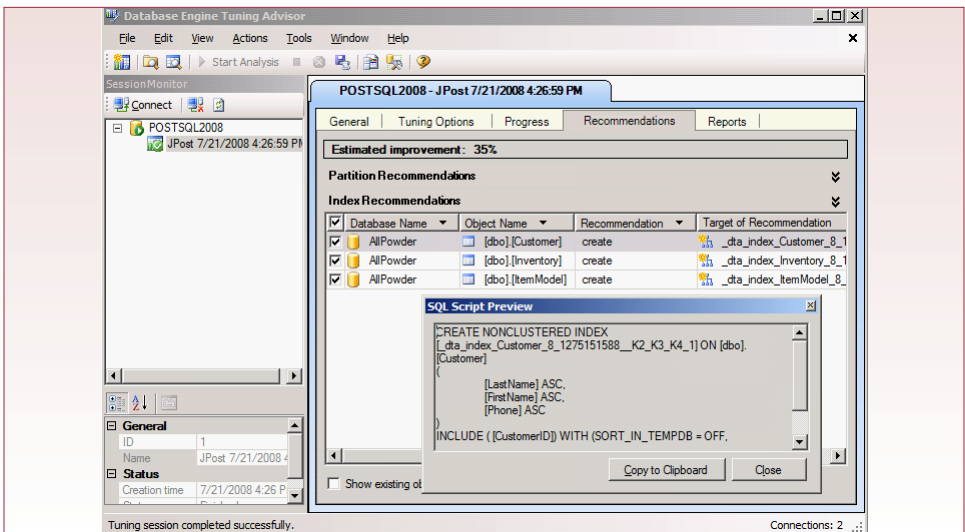
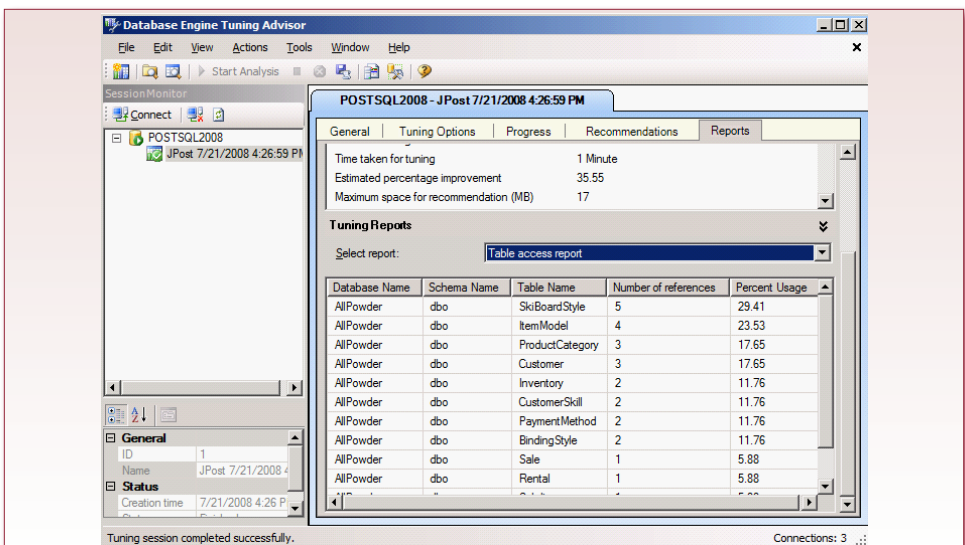


Figure 10.7

In the Management Studio, use Tools/Database Engine Tuning Advisor to start the Advisor. You will probably need the administrator role. As shown in Figure 10.6, in the Workload option, choose File and use the Browse button to find the workload file you just created (Chapter10a.trc). Choose your AllPowder database in the workload drop-down list. In the list of databases to tune, check the AllPowder database to select all of the tables. Click the green Start Analysis button in the toolbar and wait a couple of minutes. The system provides progress reports, so wait until it finishes.

For the most part, the Tuning Advisor recommends the addition of indexes. It generally checks that primary key columns are indexed and then examines the usage patterns to determine how additional indexes might improve performance. Figure 10.8 shows recommendations for adding three indexes. Your results may vary—depending on the forms and reports you tested. You can scroll to the right

Figure 10.8



and click on the entry in the Definitions column to generate the SQL syntax for creating the suggested index. It is straightforward to copy the script and run it to create the suggested index.

The Tuning Advisor also creates several reports that help you evaluate performance and understand the many components of your database. Figure 10.8 shows one of the reports. To choose a report, simply click the Reports tab and select the desired report from the drop-down list. This particular report shows how often each table was referenced during the sampling period. This knowledge will help you decide where to focus your efforts to improve performance. Obviously, the more realistic the data sample that you provide, the better the insights and recommendations from the system.

You might have noticed that the Tuning Advisor has an additional option. You can configure this option with the Advanced properties button. For small databases, it will not matter, but for large systems, you will want to allow the system to suggest partitions to the database. Partitions essentially split tables into pieces. Sections of data that are heavily used are stored on high-speed disks, while portions that are rarely used are stored on slower, less-expensive drives. Of course, you need multiple drives for partitions to make sense, and given the declining prices of even high-speed drives, partitions only make sense if the database contains huge amounts of data.



Activity: Analyze Query Performance

Many times as a DBA and as a developer, you will find that a few queries present the greatest performance issues. If a query is run once or twice a year, performance might not matter. If the query is an integral part of an application, or inside of a loop and executed thousands of times, it is worth the time to optimize the query. SQL Server provides a query analyzer to help you find ways to speed up queries. You can use the SQL Profiler to identify queries that are heavily executed making them likely candidates for improvements.

For specific queries, you can look at the Estimated Execution Plan to see how SQL Server analyzes the query and will process the results. It shows the estimated time required for each major step. Since SQL Server uses a cost-based optimizer, you cannot change the steps, but you can use the information to see where the most effort is spent and then decide if there is a way to avoid that step. Although

Action

- Open SQL Server Management Studio.
- Create a new Query with a subquery.
- Choose Query/Display Estimated Execution Plan.
- Choose Query/Analyze Query in Database Engine Tuning Advisor.
- Run the Advisor and check the recommendations.
- Choose Query/Include Client Statistics.
- Run the Query.
- Click the Client Statistics tab.

Figure 10.9

```
SELECT Lastname, Firstname, Customer.CustomerID
FROM Customer
INNER JOIN Sale
    ON Customer.CustomerID = Sale.CustomerID
WHERE Customer.CustomerID NOT IN
    (SELECT CustomerID FROM Rental)
ORDER BY Lastname, Firstname;
```

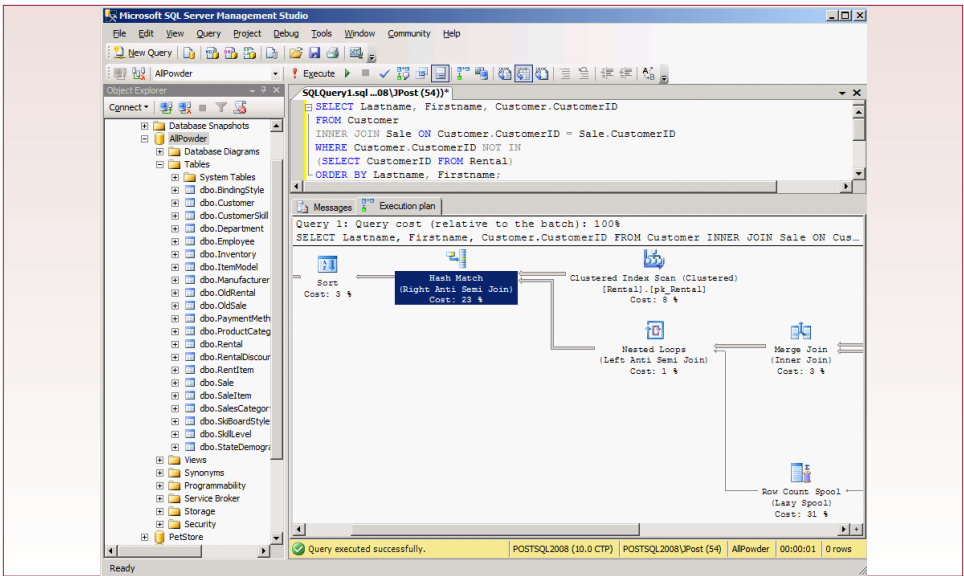
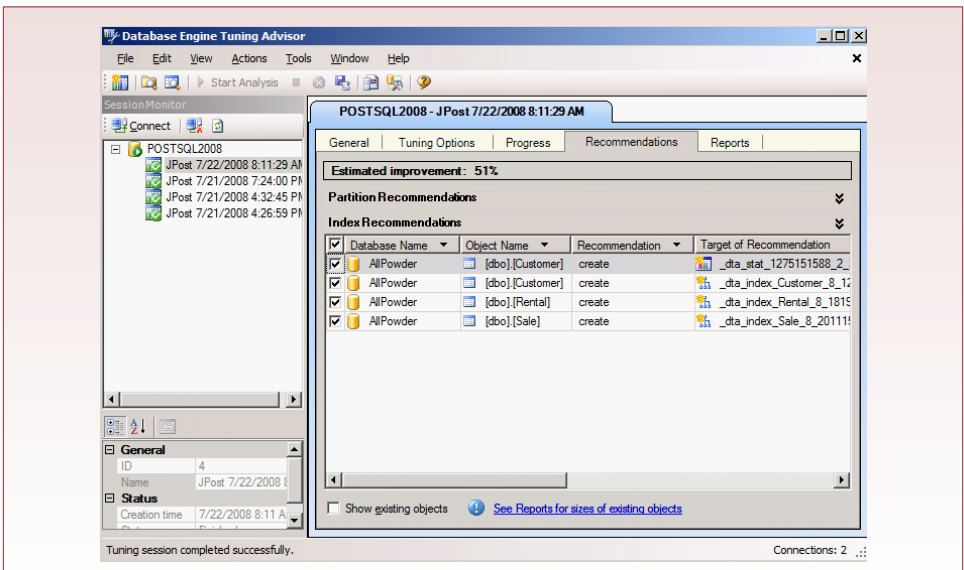


Figure 10.10

the optimizer is good, sometimes you can gain even more speed by rewriting the query—perhaps even breaking it into several pieces. You can manually review a query and the plan to look for potential gains. Figure 10.9 shows a query from Chapter 5 that returns a list of customers who have purchased items but never rented anything. As a challenge to the optimizer, it uses a subquery instead of a left join.

Open the SQL Server Management Studio and create a new query for the All Powder database. Enter this query and run it to ensure the syntax is correct. Use Query/Display Estimated Execution Plan on the main menu to see the plan. Figure 10.10 shows part of the graphical plan. As you select each major step, the chart

Figure 10.11



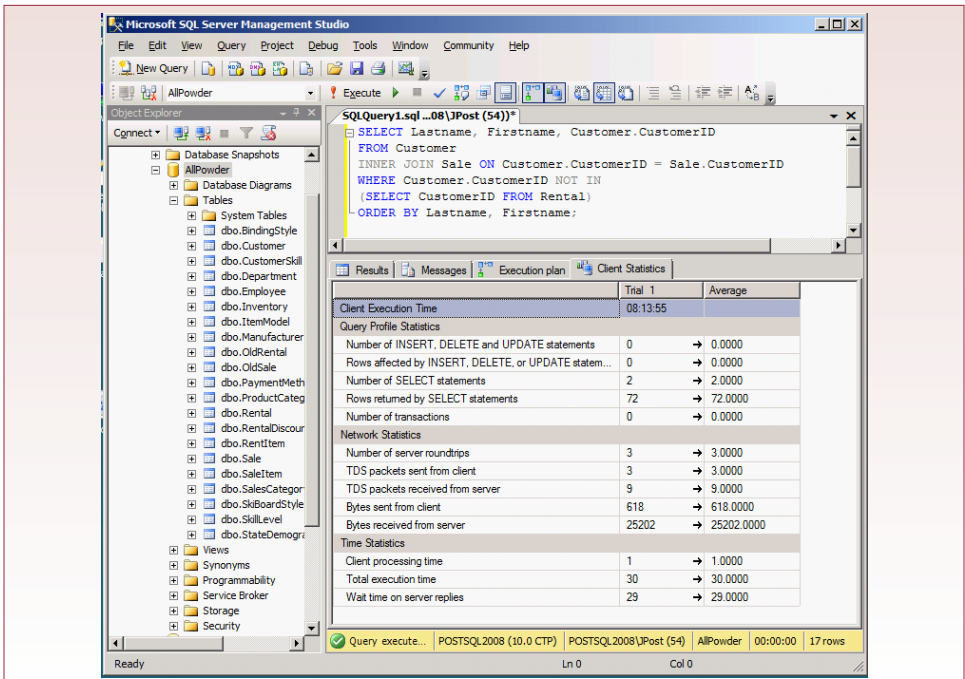


Figure 10.12

will display a tooltip with details about the operation and cost of each step. The Execution Plan provides information, but you would have to figure out how to improve execution time yourself. It is an interesting tool, and it can show you trouble spots, and for some queries, the only way to speed them up is to completely rethink the problem. But it is hard to do.

Another approach is to see if you can improve the query by adding more indexes. Again, you ultimately face the possibility of having too many indexes, but they do offer significant performance gains in many cases. The quick approach here is to let the Tuning Advisor examine the query and suggest new indexes. From within the Management Studio, select Query/Analyze Query in Database Engine Tuning Advisor and click the Start Analysis button after the tool loads. Figure 10.11 shows the results for this query. The Tuning Advisor suggests adding four indexes. You can scroll to the right to see more details and click the link to generate the SQL command that will create an index for you. The primary suggestion in this example is to index CustomerID in the Sale and Rental tables—because it is a foreign key. The system also suggests an index for customer name to reduce the time to sort the results. The Customer table probably does not change too often, so the name index makes sense. The additional indexes on the Sale and Rental tables should be tested before being implemented. It likely will have a net positive effect, but those two tables change often during the day, and the added overhead of more indexes might cause problems for time-critical operations. Since the original query is not likely to be run very often, it probably makes more sense to ignore the indexes and just let it run slower—to avoid interfering with day-to-day operations.

The Query menu contains another useful tool (Include Client Statistics). You can select that option and run the query. As shown in Figure 10.12, you will find a tab for Client Statistics next to the Results tab. These statistics provide information on the amount of data transferred and execution time. It also counts the trans-

actions and the number of SQL statements executed. These results are useful for complex queries—particularly triggers and other code blocks.

Improving performance of very large databases and complex queries can be a difficult process. Analyzing the tables is critically important for the query optimizer to function correctly. Even on small tables, the statistics can make an enormous difference in the performance. More detailed tuning is accomplished by using the index and tuning wizards. Just remember that adding too many indexes can cause problems with tables that have a high rate of change due to data entry or updates. Also, remember that it is important to monitor the daily performance aspects of the database. Keeping benchmarks on at least a weekly basis will enable you to spot long-term trends.

Security and Privacy



Activity: Backup and Recovery

Backup and recovery of a SQL Server database can be straightforward, or it can be complex. If you are able to shut down the entire database, you could simply use the operating system utilities to copy the underlying data files. More realistically, the business will want to run the database without interruption. SQL Server uses its redo logs to write all changes to a log file first. As DBA, you can set one of three levels of logging: full recovery mode which is the most intensive, bulk-logged recovery, and simple recovery mode. Full recovery is the default, which is useful, but can generate large redo log files—even on your development machine. Periodically, you have to issue a backup command to clear the logs or they get out of control. Generally, you want to leave the database in full recovery mode. If you are running the database largely for data warehousing, you could consider switching to bulk or simple recovery mode—since you could recover the database fairly quickly by reloading it from other systems.

Action

- Start SQL Server Management Studio.
- Right-click the All Powder database and choose Tasks/Back up.
- Set a location for the backup file.
- Run the backup process.
- Copy the backup file to a safe location.

Overall, SQL Server makes it relatively painless to back up the database files. Open SQL Server Management Studio, find the database in the list, right-click the name, choose Tasks/Back up. Figure 10.13 shows the basic choices. For smaller databases, choose the Full backup option. For huge databases, this choice takes considerable time and disk space, so organizations tend to make full backups perhaps once a week, and then differential backups in between—perhaps daily. Differential backups only store data that has been changed. This process takes less storage space but makes it slightly more complicated to restore the database if something goes wrong. The other main choice you have to make is where to store the backup file. You can accept the default or remove it and create your own. Ideally, it should be on a disk drive separate from the main database and log files. Either way, write down the location of the file and copy it to a safe place once the backup is completed. For example, you could write the file to a tape drive and move the tape to an off-site storage facility. Never leave the backup file on the same drive as the main database files.

The Options tab makes it easy to create a script file that can run the backup process for you at regular intervals—such as 3:00 AM when the load is prob-

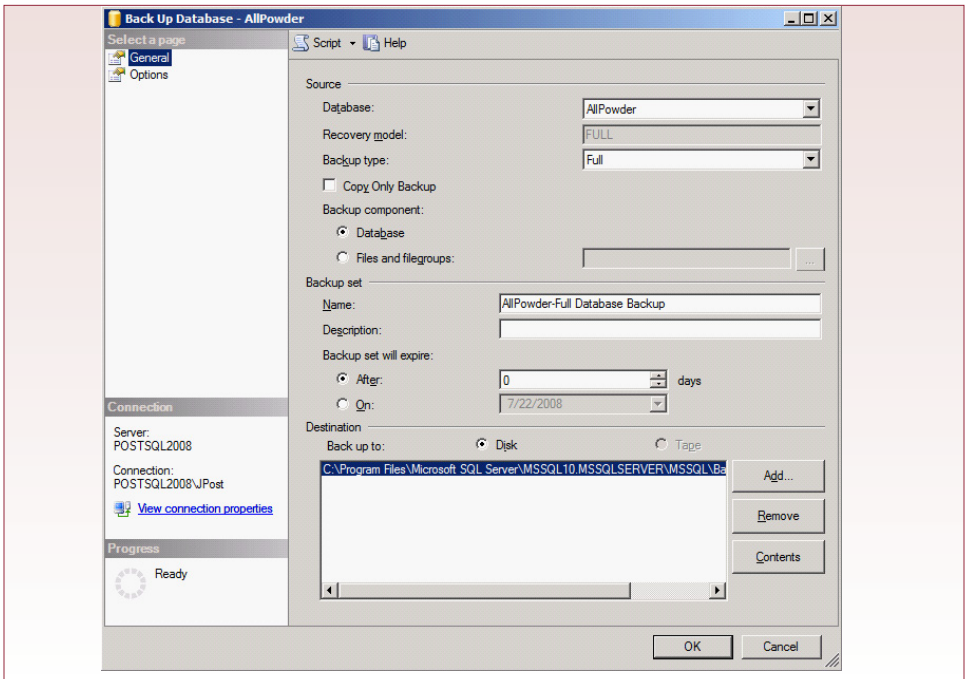


Figure 10.13

ably lighter. The system can send you an e-mail message when the job starts and finishes.

Recovering from a crash is also relatively painless. You simply load your backup tape and copy the backup file to an accessible drive. Inside the Management Studio, right-click the database name, choose Tasks/Restore, and fill in a similar form.

SQL Server (Enterprise version) also has the ability to mirror the database. You have many options, including server clusters that share processing tasks, to a single duplicate mirror server. Each server needs a copy of SQL Server, and for simple mirroring, your primary server simply sends all SQL commands across the network to the secondary server—providing an exact copy of the data. The SQL Server documentation explains the details. Keep in mind that you should also use RAID drives for most database files. These drives dramatically improve performance and provide immediate hardware-level backups. If one drive fails, you can replace it and the system can continue with no loss of data. High-end systems even support hot-swappable drives that can be replaced without stopping the database. With multiple servers in a cluster and RAID storage, the entire system can continue running even if an entire machine fails. With these techniques, it is possible to maintain very high reliability systems that can run without interruption for extended periods of time.



Activity: Setting User-Level Security Controls

The SQL Server database system is built and distributed with a complete security system. Users must log in to the system to see any of the data. The DBA can create new users and assign rights to the users or to groups of users. Initially, users have no permissions. Users and security rights can be created through SQL commands or by using the enterprise manager. The Management Studio provides a relatively

easy-to-use graphical interface, and is useful when you need to make simple changes or check on a particular item. However, if you need to set several security permissions at one time, it is often easier to write the SQL commands into a text file and execute the file as a query. If you do not happen to remember the exact SQL syntax, it is sometimes helpful to set up a test example using the Management Studio interface, and copy the SQL command that it writes.

The first issue to face is that SQL Server needs to be able to identify the individual users. Figure 10.14 outlines the basic process. The main database application contains forms, reports, and tables. As the 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.

Identifying a user is an important step in securing a database or a computer system. SQL Server has two primary means of identifying users: (1) Individual accounts can be created within the database, where users are assigned a unique username and a password, or (2) User accounts can be created on Windows, using the server or more likely, Active Directory. Each organization must balance the costs and benefits of the two methods. It is relatively easy to set up a new user account within SQL Server. The main drawback to this approach is that users need to remember yet another username and password. Firms are increasingly looking for single sign-on systems where users log into a central directory and all computers and applications pull the user identity from this central server.

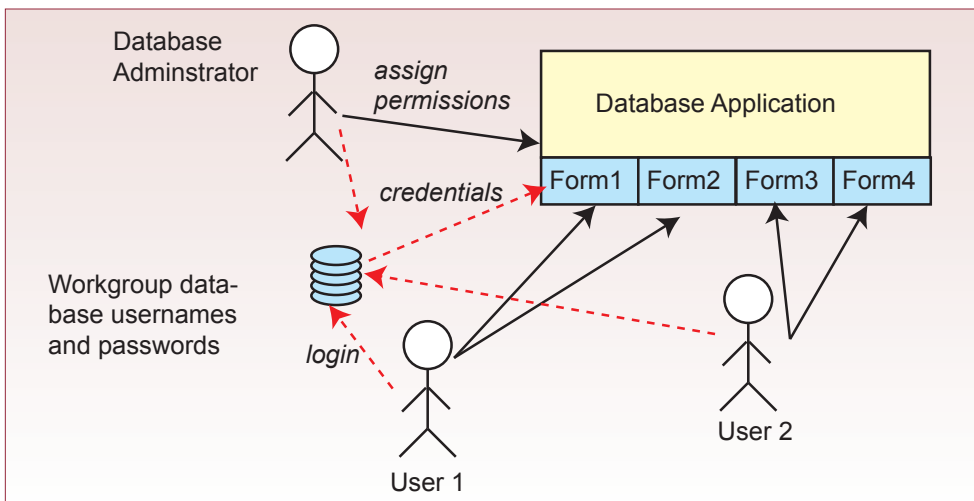
Before attempting to create users and assign security, you should write down a list of usernames and initial passwords that will be asked to enter into the workgroup database. While you are identifying users, you should also classify them in terms of tasks or groups. You almost never want to assign permissions to in-

Action

Identify the SalesClerk and SalesManagers roles and determine what permissions are needed on the basic Sale, SaleItem, Customer, and Inventory tables.

Create three new users and assign them simple passwords.

Figure 10.14



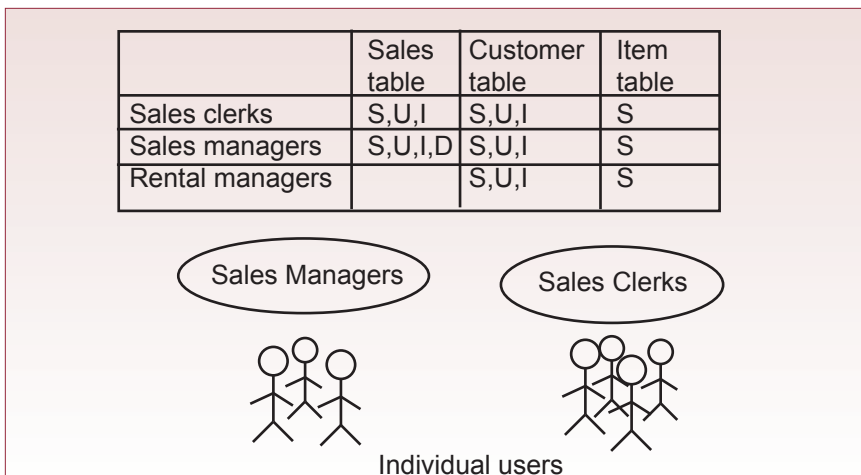


Figure 10.15

dividual users. Instead, you place users into groups and assign database permissions to the roles of these groups. Figure 10.15 illustrates the main concept. By assigning permissions to the role, you should only have to set permissions once. As individual roles are added to or removed from users, their permissions automatically change.

Creating a new user requires two basic steps: (1) Define the user identity—either in Windows or in the main Security section of SQL Server Management

Action

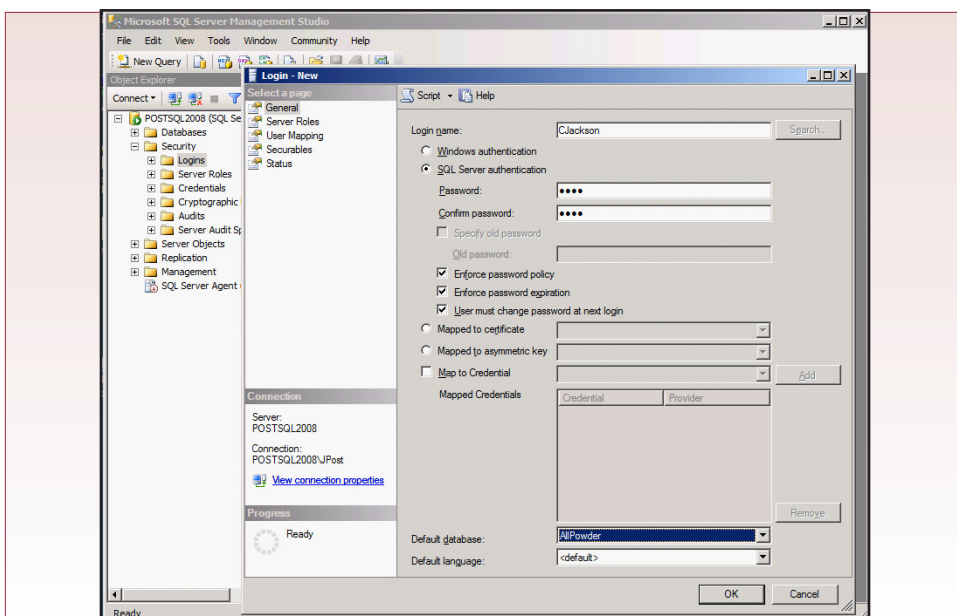
Create the SalesClerk and SalesManager roles.

Assign appropriate table permissions to the new roles.

Assign one of the roles to each of the new users.

Use the Sales form to test the accounts and roles. Test the roles by using SQL statements.

Figure 10.16



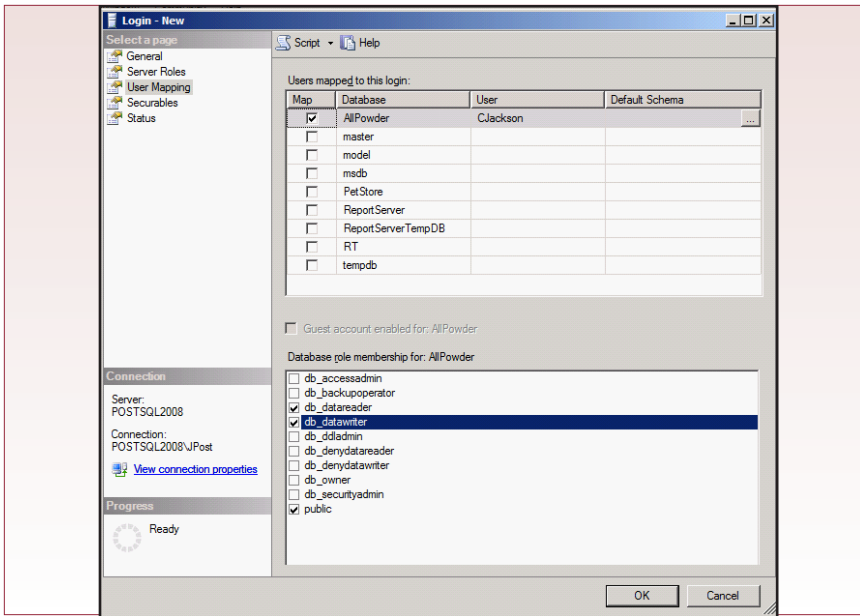


Figure 10.17

Studio; (2) Assign the user to a particular database. The example here will use SQL Server to define the user, but the process for using Windows is similar.

Figure 10.16 shows the basic process for defining a new user. The main trick in the Management Studio is that you must open the Security section that is not located within a database. Users or logins are defined within the entire SQL Server system, not a specific database. After a login is created, you can assign it to a specific database. Simply expand the main Security node, right-click the Logins entry and choose New Login. Enter a username that will be displayed within SQL Server. Choose the SQL Server authentication option, enter a password and verify it. Be sure you remember the password for later use. There is one important detail. When you installed SQL Server, you needed to select the option to support both authentication methods. By default, the installation tends to stick with just Windows. If you the SQL Server option is not available, you can use the Windows authentication until you get around to installing the SQL Server authentication. If you choose the Windows authentication, you pick a username from your Windows security directory and do not need a password. You should select the default database for the user (All Powder).

The second step you need to complete is to assign this login to a database. You can perform this step when you create the user, or you can do it later. The steps are similar in both cases. If necessary, open the user account by double-clicking the login name. As shown in Figure 10.17, you need to select the User Mapping option and assign the user to the All Powder database. You could also give the person basic read and write permissions at this point so you can test the login. Ultimately, you will define business roles, give the permissions to the roles, and assign the roles to each user.

If you need to create several users at the same time, it is often easier to create a script file with SQL statements. You can combine the login creation and database mapping with two basic commands:

```
CREATE LOGIN CJackson WITH PASSWORD = N'password' ;
```

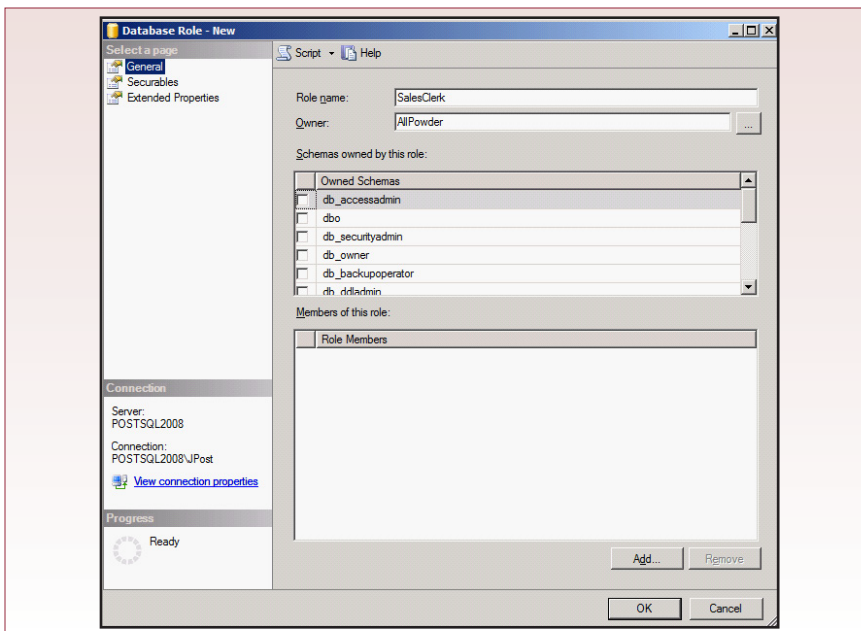


Figure 10.18

```
USE AllPowder;
CREATE USER CJackson FOR LOGIN CJackson;
```

You simply edit the name and password for each person and then execute the script to create all of the accounts at one time. You can also write a SQL program that would read the list of names and passwords from a file or table and execute the statement to create each account automatically.

The next step is to create the roles of SalesClerk and SalesManager. Again, you could use SQL (`CREATE ROLE SalesClerk AUTHORIZATION 'AllPowder'`), or you can use the graphical tools in the Management Studio. Figure 10.18 shows the basic steps for creating a role. Within the All Powder database, expand the Security node and the Roles node. Right-click the Database Roles entry and pick the option to add a new role. Enter the name of the role (SalesClerk).

The main task shown in Figure 10.19 is to click the Securables option and begin assigning permissions for the role. You should rely on your notes, such as those in Figure 10.15 to guide you in setting the permissions. Once you know what access rights are needed, the process is straightforward. Click the Search button to list items that can be secured. Often, the easiest starting point is to choose a list of all tables. Then select each table and place check marks in the lower grid to assign the permissions needed by this role. The two main choices are GRANT and DENY which are fairly clear. Generally, you assume all permissions are denied and you explicitly grant access to the items needed by each role. If you ever do select a DENY option, that choice always wins and blocks the role's access—even if some other permission granted a similar access elsewhere. Use it sparingly because it can be hard to track down later. The With Grant option should also be avoided in most cases. If you select that option, the role (and user) not only gains the ability to perform the task listed, but can pass that permission onto someone else. If you are using a decentralized security model, you might grant this option to department chairs and ask them to assign final permissions to everyone in the

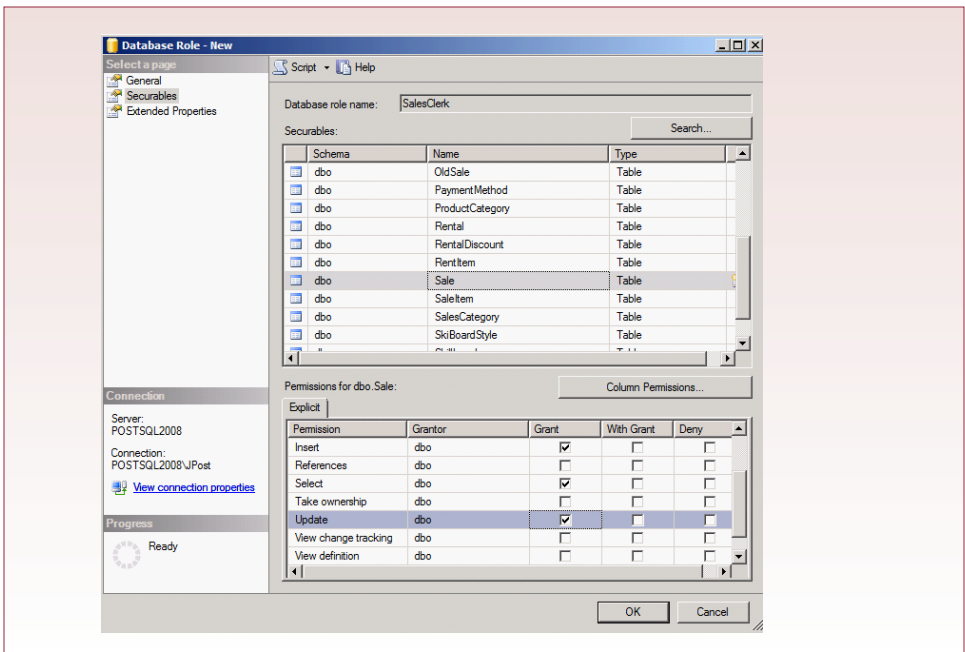


Figure 10.19

department. If you need to maintain central control over permissions, avoid giving the WITH GRANT permissions to any other role.

Again, it is actually easier to assign permissions via SQL:

```
GRANT SELECT, INSERT, UPDATE ON Sale TO SalesClerk;
```

When you write the SQL statements, you can collect them in a log file. Later, the auditors and you can refer to the file to see what permissions are assigned to each role. And, if you ever need to re-establish permissions, you can simply execute the script file.

The final step is to assign the SalesClerk role to individual users (who are sales clerks). You can make this assignment from within the role or within the security properties editor for each user. As shown in Figure 10.20, it is easier to work from the Role screen and simply add members—enabling you to see all of the users and which ones have already been assigned to the role. If necessary, open the role and click the General option. Click the Add button at the bottom of the Members box and select the users who should be assigned that role. It is also possible to assign members to roles using SQL, but there is no standard SQL command. Instead, you need to call the internal security function: `sp_addrolemember`.

When defining roles and assigning them to users, it is important to remember that users are often assigned multiple roles. Security is more effective when the roles are assigned with relatively small granularity. That is, instead of creating two or three all-encompassing roles and assigning one to a person, it is better to break roles into smaller pieces and assign multiple roles to each person. In the All Powder case, you should consider separate roles for Sales, Rentals, Receiving, Adding Customers, and so on. Then sales clerks would be granted the roles for sales, adding customers, and perhaps one or two other tasks. If a person is promoted or moved to a different position, you simply have to change the role assignment to

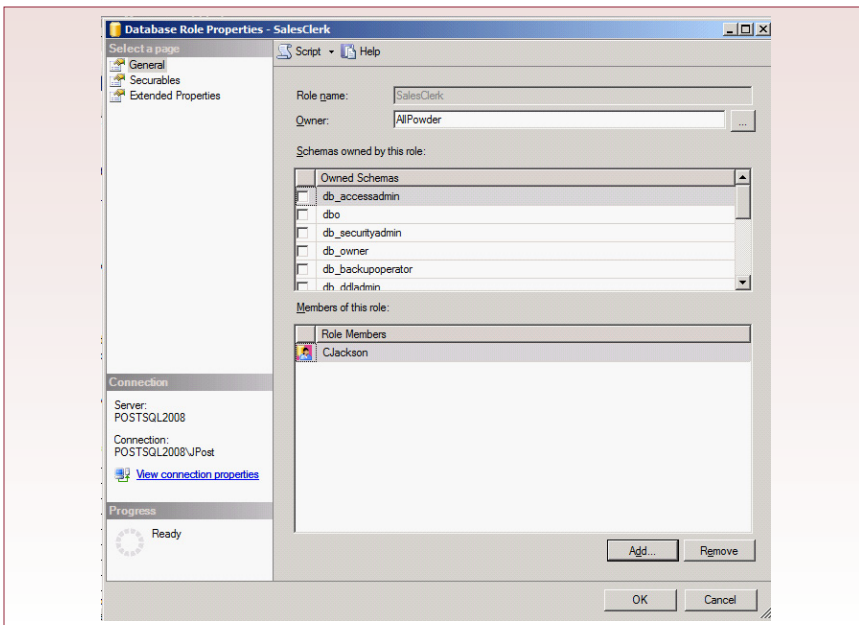


Figure 10.20

match the new job. It is important that the roles and their names closely match the business jobs.

Of course, you need to test the security assignments. Try the test first using the forms—which is how the sales clerks will generally use the application. Notice that the forms themselves are stored outside the database, so they are not directly subject to the security conditions. However, as soon as the form tries to retrieve data, the security conditions are imposed, so unauthorized users will not be able to see any data. In fact, the first time you try to run a form as a sales clerk, you will probably receive an error message.



Activity: Encrypt Data

Many situations arise where you need to encrypt data stored in tables. In particular, any personal data collected from customers or employees should be encrypted. The privacy laws are written so that if data is encrypted, even if it is stolen, you do not have to notify customers of the data theft (always consult an attorney for details). SQL Server provides a couple of interesting methods for encrypting data. The most useful is to use a security certificate.

At heart, encryption today is straightforward to implement. The biggest problem involves the keys—where do you store them to keep them secure and still make it relatively easy for your application to find the keys to encrypt and decrypt the desired data. The simplest approach in SQL Server is to create a security certificate and use it for encryption and decryption. Security certificates are created and assigned to the

Action

- Start the Database Management Studio.
- Right-click the server name and open the Activity Monitor.
- Run some queries to see if the monitor changes.
- Start the Windows Reliability and Performance Monitor and add counters for memory, drive, and some SQL statistics.
- Run some forms, reports, and queries to see if they influence the monitors.

specific database server and stored securely by the operating system. If an attacker manages to steal the database, it will still not be possible to decrypt the data. However, if the attacker acquires administrative rights on the server, anything is possible—so you still need to implement other security measures.

The first step is to create a certificate. SQL Server provides a couple of methods to create certificates, but the easiest to create (particularly for a lab exercise) is to generate one using a password. In SQL Server Management Studio, open a new query window for the All Powder database. You need to create the certificate by running this code once:

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'complex
password';
CREATE CERTIFICATE MyCert
    AUTHORIZATION dbo WITH subject='Main encryption
certificate';
```

Be sure you enter a random complex password—something that can never be guessed or figured out through brute force. You should also give the certificate a meaningful name (other than MyCert), because you will need to use the name whenever you encrypt or decrypt data. Finally, after you create the certificate, you should export the certificate as part of your backup routine in case you have to recover from a crash. You can also create symmetric and asymmetric keys, but you do not need them for this exercise.

Once the certificate has been created, you can use it for encryption and decryption by using the EncryptByCert and DecryptByCert functions. The following SQL commands illustrate the process. Create a new query window and test them in your database:

```
DECLARE @UserPassword nvarchar(250);
DECLARE @EPW varbinary(500);
DECLARE @DPW nvarchar(250);
SET @UserPassword='mypassword';

SET @EPW=CONVERT(varbinary(500), EncryptByCert(Cert_
ID('MyCert'), @UserPassword))
SET @DPW=CONVERT(nvarchar(250), DecryptByCert(Cert_
ID('MyCert'), @EPW))

SELECT @EPW As EncryptedPW, @DPW As DecryptedPW
```

To store encrypted data in your tables, simply define a column as varbinary—as a general rule, make it twice as large as the unencrypted data because encryption expands the length—and use the EncryptByCert function whenever you store data into the column. This function is probably not readily available for bulk data loads. However, you can bulk load unencrypted data and then write a short query that encrypts the data and writes it into a separate encrypted column. Then delete the unencrypted data with a second query.

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 ex-

cept the two of them. On the other hand, they do need a system on which it is easy to create backup copies. And, 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 arriving at one time, they do want to examine some lengthy reports to evaluate sales trends.

1. Run the performance analyzer to improve the performance of the database and identify indexes needed. Also check the performance for the report queries.
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 celebrity 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 approved managers and then give the managers access to the data through that query.

1. Run the performance analyzer to improve the performance of the database and identify indexes needed. Also, check the performance for the report queries.
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.

1. Run the performance analyzer to improve the performance of the database and identify indexes needed. Also, check the performance for the report queries.

2. Create a backup option and a written set of procedures to follow to protect the database.
3. Create the security provisions needed. Concentrate on protecting the data from external attacks.

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. Run the performance analyzer to improve the performance of the database and identify the indexes needed. Also check the performance for the report queries. 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.

Distributed Databases

Chapter Outline

Location, Location, Location, 222

Case: All Powder Board and Ski Shop, 223

Lab Exercise, 223

All Powder Board and Ski Shop, 223

The Internet, 231

Exercises, 239

Final Project, 241

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.

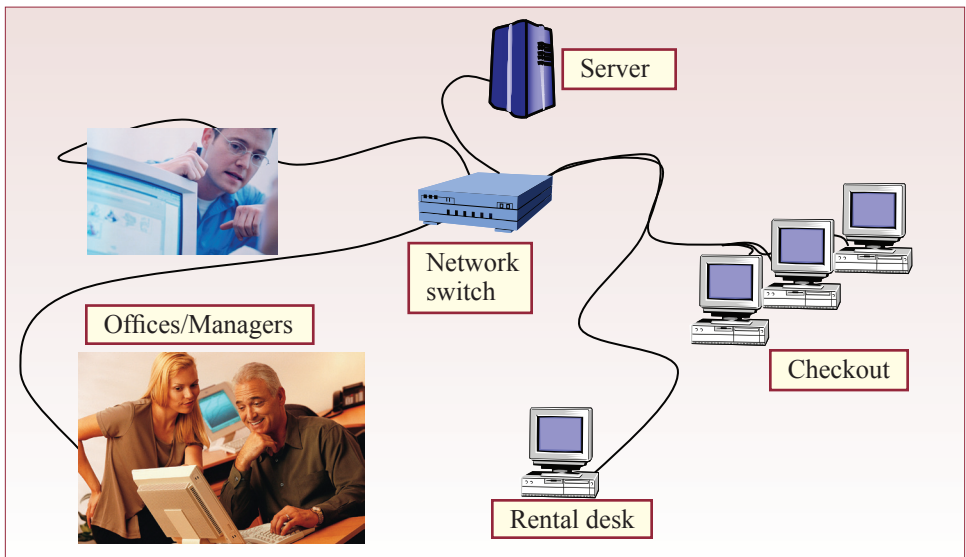
Location, Location, Location

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.

SQL Server provides several tools to support distributed access to data, but you most likely need the Enterprise version to use all three. The three primary approaches are (1) Internet access, (2) linked databases, and (3) data replication. You could also make the argument that the cluster system is a distributed system on a local scale. The primary purpose of the cluster system is to improve performance and reliability. You can use storage area networks to separate the data files from the processors. Clustering enables multiple processors to work on the same data at the same time. At some level, SQL Server treats all of this hardware as a single (really fast) system. On the physical side, you gain flexibility by being able to move, change, and add hardware without altering the database design.

In terms of distributed access, the use of Web-based forms and reports provides considerable flexibility in terms of client access. The database itself has become more centralized, which makes it easier to manage. Yet, managers can access the information from any place with an Internet connection. As wireless devices, including cell phones, gain more Internet features, managers will have almost continuous access to the data regardless of location. And this power comes almost automatically when you build Web-based forms and reports. Since the other labs cover these tools, this chapter will focus on database links and replication. Just remember that whenever you encounter the need for a distributed system, you should first ask whether the problem can be solved using the Internet.

Figure 11.1



Case: All Powder Board and Ski Shop

Initially, you might think that All Powder with 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 there are some simplified distributed issues to address. The distributed aspect arises because there will be several locations within the store that need access to the database—the check-out 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.

Distributed questions within a single building are much easier to solve than those spread across wide geographic areas. The reason is because of the speed of local area networks. 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. You have already built the system so that all of the data files, forms, and reports run on the server. To provide access from multiple locations, all you need to do is ensure that each station has a machine with a network connection back to the server. You might even consider portable wireless devices for some of the employees so they can help customers throughout the store. The key point is that the database will run without any changes. The forms application is installed on each Windows computer and it connects to the central database server over the network.

Lab Exercise

All Powder Board and Ski Shop

The existing single server with network access will work well as long as most of the operations occur in one location. How well would this system work if the company acquires an inventory warehouse or opens another store? The answer depends on how fast of a connection the company is willing to lease between the other locations and the database server. With a relatively high-speed connection, the Web-based approach will work fine. There might be slight delays if everyone opens major reports at exactly the same time, but most of the time, the connection is simply transferring small amounts of transaction data. With only a few users, even a fractional T1 line or frame relay might be sufficient to handle the typical loads. You could also use a cable-modem connection, but you will have to run a virtual private network (VPN) connection over it to emulate a local network with reasonable security; unless you completely switch to a Web application. In a real-life situation, you could monitor the amount of traffic and network usage within the existing store to get a better idea of how much bandwidth would be needed to connect to a second store.

On the other hand, you could eventually reach a situation where you need faster response times at each location. In this case, you might split the database and run two or more servers. The servers would support local operations, but some reports would need to retrieve data from both databases. As long as you have a network connection, you can create a database link that enables forms, reports, and SQL to access data from any connected database. The process is relatively easy, but you will need to think about security issues.



Activity: Create Database Links

The first step is to find or create a second database. It is even better if you happen to have two machines running as SQL servers. The easiest solution is to work with a colleague and share databases. If that is not possible, you will have to start a second instance of SQL Server on a second computer. The shortest approach is to create a virtual machine and run it as a second server.

To be safe, you can create a new TempCustomer table. Figure 11.2 shows the standard SQL commands along with some sample data. If you are connected to a database server run by a trusted friend, you could share your existing tables (but you might want to make a backup copy first). To share databases, you first need security permissions to access the database server. So you need to assign a login and access rights to the user who wants to access the system. Since you probably do not have control over a Windows Active Directory, it is probably easiest to create a new SQL Server login and give it at least SELECT rights to the All Powder Customer table and the new TempCustomer table. Share the login credentials with a friend if you are working together.

You can connect to several other types of databases, including SQL Server, Oracle, IBM DB2, Access, and even Excel spreadsheets. In all cases, you need a network connection, the name of the server, and the name of the database. Test the network connections (try a Web browser or the ping command) before proceeding.

For SQL Server connections, the simplest approach is to add the other server(s) to the explorer list in the Management Studio. You can click the Connect button if it is visible, or right-click in the Object Explorer and select the Connect option. You will be asked to enter the same type of information you used to connect to the initial database. Be sure to specify the server name and the proper login credentials. The new server will appear in the Object list and you can perform management tasks on the server—if you have administrative permissions.

Many things can go wrong in trying to establish a network connection—particularly with the security-enhanced Windows 2008 and SQL Server 2008. You can try several of the following steps, but there is not enough room to describe

Action

If necessary, create a second database, preferably on a different machine.

Create a small Customer table and load it with four or five rows of data.

Return to your main database and create a database link to the target.

Run an SQL statement that retrieves data across the link.

Figure 11.2

```
CREATE TABLE TempCustomer
( CustomerID    int,
  LastName      nvarchar(50),
  FirstName     nvarchar(50),
  Constraint pk_Customer Primary Key (CustomerID)
);
INSERT INTO Customer (CustomerID, LastName, FirstName)
Values (1,'Smith', 'Adam');
INSERT INTO Customer (CustomerID, LastName, FirstName)
Values (2,'Keynes', 'John');
INSERT INTO Customer (CustomerID, LastName, FirstName)
Values (3,'Samuelson', 'Paul');
INSERT INTO Customer (CustomerID, LastName, FirstName)
Values (4,'Robinson', 'Joan');
```

```
EXEC sp_addlinkedserver 'PostSQL2008c', N'SQL Server'  
GO  
  
SELECT *  
FROM PostSQL2008c.AllPowder.dbo.TempCustomer
```

Figure 11.3

them in detail. (1) Use ping to ensure the network connection works and your computer can find the other server. (2) Test your login credentials locally on the remote server to ensure they are correct. (3) Check the firewall connections on the target server. Microsoft has a list of ports you need to open, but try turning off the firewall completely first. If the connection then works, you can turn the firewall back on and dig into the details of configuring it to support SQL Server. (4) Use the SQL Server Configuration Manager to ensure that at least the TCP network services are enabled. You might also need named pipes.

When you have connected to the other SQL Server database, you should register the connection. Registration essentially stores the login credentials making it easier to establish connections and run queries in the future. The easy registration method is to right-click the server name and choose the Register option. You can also choose View/Registered Servers from the main menu to see a list of servers, to add new ones, or to create new groups. The next step is to add the server name as a linked server using a stored procedure. As shown in Figure 11.3, open a new query window in the All Powder database and run the EXEC command. Replace the server name with the name of your server.

The first parameter is the name of your server in quotes. The second parameter is always the same for SQL Server databases. The N (national/Unicode text) is required. If the database is something other than a SQL Server database, you have to enter the appropriate name; and you have to enter additional login information. See the SQL Server documentation for details and examples. The SELECT command tests the connection by retrieving the data from the new table you created on the remote server. You could also use the AllPowder customer table if you are connecting to a friend's server and did not create the TempCustomer table. Notice that you specify the entire table name with the syntax: Server.Database.Schema.Table. If you get tired of typing the long syntax repeatedly, you can define a synonym for it. For example:

```
CREATE SYNONYM tempC  
FOR PostSQL2008c.AllPowder.dbo.TempCustomer  
SELECT * FROM tempC;
```

The point is that once the link has been made, you can access any of the linked tables just as you would any other table. Just bear in mind that the data has to travel across the network connection, and if the connection is slow, the query might take a long time to run. Of course, you can also use UPDATE, INSERT, and DELETE commands as well—so linked servers are a useful method for transferring bulk data from one server to another.

Now it is time to learn a cool trick introduced in SQL Server 2008. Most DBMSs support some version of linked servers. Few (if any) support multi-server queries. A multi-server query runs on all of the servers in a group—at the same time.

The screenshot shows the Microsoft SQL Server Management Studio interface. The 'Registered Servers' pane on the left shows a local server group 'Local Server Groups' containing 'POSTSQL2008' and 'POSTSQL2008c', and a 'Central Management Servers' group. The 'SQL Query5.sql' window contains the query: `SELECT TOP 10 * FROM AllPowder.dbo.Customer`. The 'Results' pane displays the following data:

Server Name	CustomerID	LastName	FirstName	Phone	EMail	Address
POSTSQL2008c	0	Walk-in	NULL	NULL	NULL	NULL
POSTSQL2008c	1	Jones	Jack	111-222-3333	JonesJ202@msn.com	123 Main
POSTSQL2008c	2	Sanchez	Paul	111-444-9999	SanchezP844@msn.com	777 Oak
POSTSQL2008c	3	Gamer	Chad	213-080-4599	GamerC73@msn.com	555 Trident Pla
POSTSQL2008c	4	Reeves	Gil	213-186-6502	ReevesG690@msn.com	2914 Summers
POSTSQL2008c	5	Hicks	Evelyn	213-959-5499	HicksE808@msn.com	2815 Church St
POSTSQL2008c	6	Grimes	Ernest	312-917-7845	GrimesE460@msn.com	7118 Green Ric
POSTSQL2008c	7	Rice	Charlotte	312-608-6819	RiceC65@msn.com	411 Owens Stre
POSTSQL2008c	8	Marlow	Jerry	213-606-0452	MarlowJ674@msn.com	2250 Cave Spon
POSTSQL2008c	9	Rogers	Robin	213-149-9519	RogersR135@msn.com	2519 Pleasant
POSTSQL2008	0	Walk-in	NULL	NULL	NULL	NULL
POSTSQL2008	1	Jones	Jack	111-222-3333	JonesJ202@msn.com	123 Main
POSTSQL2008	2	Sanchez	Paul	111-444-9999	SanchezP844@msn.com	777 Oak
POSTSQL2008	3	Gamer	Chad	213-080-4599	GamerC73@msn.com	555 Trident Pla
POSTSQL2008	4	Reeves	Gil	213-186-6502	ReevesG690@msn.com	2914 Summers
POSTSQL2008	5	Hicks	Evelyn	213-959-5499	HicksE808@msn.com	2815 Church St
POSTSQL2008	6	Grimes	Ernest	312-917-7845	GrimesE460@msn.com	7118 Green Ric
POSTSQL2008	7	Rice	Charlotte	312-608-6819	RiceC65@msn.com	411 Owens Stre
POSTSQL2008	8	Marlow	Jerry	213-606-0452	MarlowJ674@msn.com	2250 Cave Spon
POSTSQL2008	9	Rogers	Robin	213-149-9519	RogersR135@msn.com	2519 Pleasant

Figure 11.4

Close any existing queries and choose View/Registered Servers from the main menu. Verify that your local server and at least one remote server are registered in the Local Server Groups. Right-click the group name (Local Server Groups), not an individual server, and choose the New Query option. Execute a simple query:

```
SELECT TOP 10 * FROM AllPowder.dbo.Customer;
```

Figure 11.4 shows the results. Check them carefully. Because both servers have a Customer table, the query returns results from both servers! In the example, both tables have the same data, but you can see the duplication, and the system returns the server name as part of the results (which can be controlled through properties). This trick could be useful for combining lists of customers or employees from multiple locations; or even for helping search for an item when you cannot remember which server holds the answer.

By default, the query merges the results into a single list. You can also right-click and choose Query Options and select the Multiserver line under the Results node. This screen enables you to set Merge results to False. When you rerun the query, the results from each server will be presented in a separate grid. In general, the merged option seems most useful for a SELECT command, but sometimes you will want to separate the results.

Whenever you link servers and run SELECT queries, always think about the amount of data that has to be returned across the network. In a large organization, servers could be scattered around the world, and you could clog expensive networks by issuing open-ended SELECT statements. At a minimum, you should use the TOP n command to reduce the amount of data returned from any single server.



Activity: Replicate and Synchronize a Database

Internet pages and linked tables are an efficient and easy solution when all of the computers are connected by high-speed networks, or when you only need to

transfer a limited amount of data at one time. They will not work as well when you have multiple locations that are connected by slower links. In particular, when most of the traffic is local, you should consider replication. For example, if the company had stores in two different countries, it is unlikely that workers in the first store would need to share data on a daily basis with the other store. In this configuration, it does not make sense to have one central database and transfer everything back to it. Instead, you would want to install separate database servers in each location that would handle the operations within that area. Yet, on a regular basis, you need to synchronize the data so that managers can still retrieve all of the information to make decisions.

Building replicated databases requires some decisions about how you want to link the databases. The main decision is whether you want the databases to communicate all of the time, or just once in a while on a schedule or on demand. The second approach works best if you have limited bandwidth, when the servers do not need to share data very often, or when the databases are mobile and not always connected. The SQL Server replication process is similar in both cases, so this exercise will use on-demand merges.

The replication and synchronization process in SQL Server has changed somewhat with each of the latest versions. SQL Server 2008 is used in this example. The steps are similar in SQL Server 2005, but the location of some of the commands has changed. Security is the most complicated problem with SQL Server 2008 and Windows Server 2008. You will probably need administrator rights on two servers and two databases. It might be best to work with a colleague—otherwise, server virtualization comes in handy. In the following discussion, one database will be the publisher and the other will be the subscriber. Most of the initial work is done on the publisher machine. A table on that computer will be replicated to the subscriber computer. Changes made on either machine will be synchronized with the other copy.

1. Configure Security on the Publisher and Subscriber computers.

Begin in Windows by starting the SQL Server Agent account on both computers. This account was installed by default but it was set to Manual start. Use Windows/Computer Management and expand Services. Find the SQL Server Agent service and start it (arrow in the menu or right-click). If you are building this connection for a real project, you probably want to change the startup property to Automatic instead of Manual. Remember to set it on both computers.

While you are in the Computer Management snap-in, you need to create new security accounts that will run many of the jobs behind the scenes. You need four local accounts on the publishing server. You can use any name and password that you want, but Figure 11.5 shows the suggested names that are easy to identify. Open the Local Users and Groups node in the Computer Management Console. Right-click the Users line and pick New User. Enter the names and passwords for all four entries on the Publisher server and for the last two on the Subscriber server. Be sure you remember the passwords.

Action

Configure security on the publisher and subscriber computers.

Configure distribution on the publisher.

Set database permissions on the publishing server.

Define publication data to be shared.

Role	Server	Suggested name
Snapshot Agent	Publisher	<server>\repl_snapshot
Log Reader	Publisher	<server>\repl_logreader
Distribution Agent	Publisher and Subscriber	<server>\repl_distribution
Merge Agent	Publisher and Subscriber	<server>\repl_merge

Figure 11.5

SQL Server replication works by storing temporary data in a shared folder on the Publisher server. You need to assign permissions to this folder, so you should create it by hand. Use Windows Computer explorer to create a new folder. The default location is usually: C:\Program Files\Microsoft SQL Server\MSSQL10.MS-SQLSERVER\MSSQL\repldata, but you could put it anywhere. When the folder is created, right-click it and choose the Sharing option. Click the Advanced button if necessary and share the folder. Be sure the Share name is repldata. Click the Permissions button to assign rights to the Share. If necessary, remove the Everyone entry from the list. Add the snapshot, distribution, and merge accounts you created. Set Full Control permissions to the snapshot agent and just Read for the others. You might also want to give yourself Full Control. Close the Sharing Window and select the Security tab. Click the Edit Permissions button and repeat the process to give permissions to the folder itself. Again, the snapshot user gets Full Control, the others need only Read permission. You can close the Management Console.

If your network uses Active Directory, you can create the accounts on that system. If not, you should also create a repl_merge account login on the Subscriber computer. Log in to the Subscriber computer and create a repl_merge user on that machine.

2. Configure Distribution on the Publisher

Back on the Publisher machine, start the SQL Server Management Studio and create a new table to share. You could share an existing table, but replication causes some internal changes to the table, so it is best to practice on a simple table until you get familiar with the process. If you have not done it already, create the Temp-Customer table shown in Figure 11.2 within your AllPowder database. Be sure to add at least a couple rows of sample data.

To enable distribution, right-click the Replication folder and choose Configure Distribution. Accept the default option to act as your own distributor. You need to specify the path location of the snapshot files that you created. Initially, the wizard sets the location using a full path/file name. You need to change this value to a network format so that it can be accessed by external computers (subscribers). Enter \\myserver\repldata, where you enter the name of your server for myserver. Accept the default options for the remaining screens and click the Finish button.

3. Set Database Permissions on the Publishing Server

The user accounts that you created need to be able to access the database. On the publishing server, in SQL Server Management Studio, open the main Security node, right-click the Logins entry and choose New Login. Enter login data for myserver\repl_snapshot—use the Search and Check Names buttons to ensure you get the username exactly correct. Select the User Mapping screen and add the snapshot user to both your AllPowder and distribution tables. The distribution

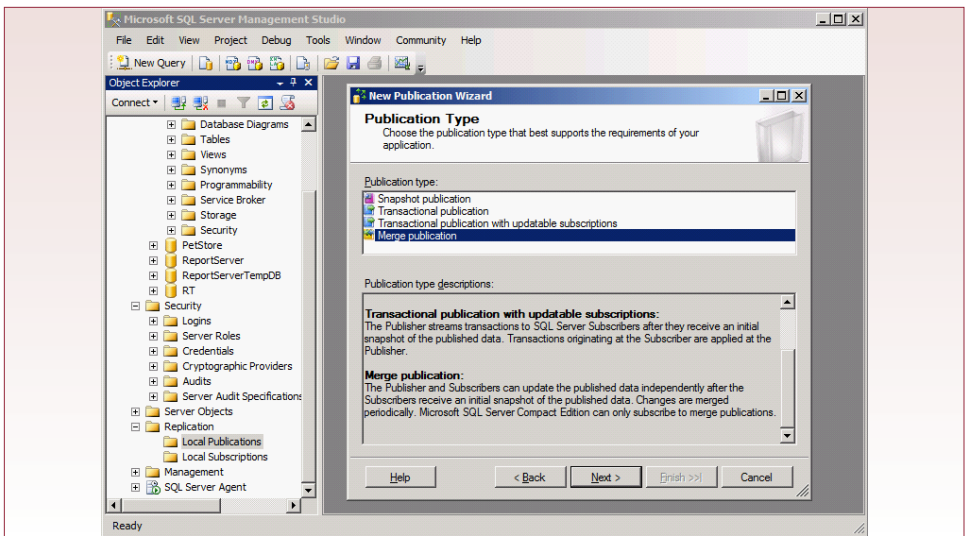


Figure 11.6

table was automatically created in the previous step. In both databases, assign the `db_owner` role to the snapshot user. Repeat the process for all three of the other logins: `repl_logreader`, `repl_distribution`, and `repl_merge`. Remember to assign the `db_owner` role in both databases to each of these login accounts.

4. Define Publication Data to be Shared

You need to define a new publication. A publication is a table or a view, but you can also control which columns and which rows are distributed. Read the Microsoft SQL Server replication tutorial and it will show you how to filter rows that are sent to each subscriber based on the individual machines. To keep the process simple, this example will share the entire `TempCustomer` table.

On publishing server, inside the SQL Server Management Studio, expand the `Replication` folder if necessary. Right-click the `Local Publications` node and choose `New Publication`. Choose your `AllPowder` database. As shown in Figure 11.6, select `Merge publication` as the type. If possible, stick with the SQL Server 2008 option. As shown in Figure 11.7, expand the table list and pick the `TempCustomer` entry. Ignore the column and row filters. Select `Create a snapshot immediately`, and clear the `Schedule the snapshot...` checkbox.

Agent security is an important screen and it is the reason you set up the accounts in the first step. Click the `Security Settings` button. For the `Process Account`, enter the `myserver\repl_snapshot` user and enter the password you assigned to that account. When asked for a name, enter something descriptive such as `AllPowderMerge1`. Finish the wizard.

You now need to provide read permissions to the `repl_merge` user. If needed, expand the `Replication` and `Local Publications` folders. Right-click the `AllPowderMerge1` entry and pick the `Properties` option. Select the `Publication Access List` page and click the `Add` button. Enter the `myserver\repl_merge` user and click `OK` a couple of times to accept and close the windows.

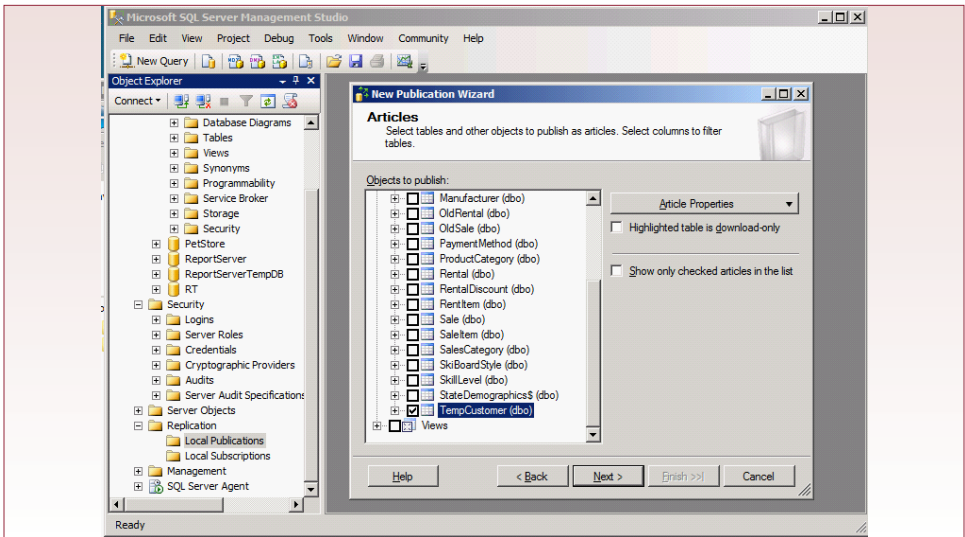


Figure 11.9



5. Create a New Subscription

The prior step created a publication. Now you need to switch to the subscriber computer and create a subscription that will retrieve the publication. Technically, you can connect to the subscriber computer using the same SQL Server Management Studio. But be extremely careful to ensure you know which database you are working with at all times.

Run SQL Server Management Studio on the subscriber computer. Expand the Replication folder, right-click the Local Subscriptions folder and choose the New Subscriptions option. The publisher is on a different computer, so pick Find SQL Server Publisher in the drop-down list. Enter the server name for the publisher computer as well as any login information requested.

Select the newly created AllPowderMerge1 entry and click the Next button. For the Agent location, pick the option to Run each agent at its Subscriber. This action specifies a “pull” subscription where each subscriber client will request synchronization to the server—as opposed to a push connection where the server sends changes out to each client. The choice depends on how the servers will be used and how much control and work you want to handle centrally.

To avoid creating problems with any existing databases on the subscriber, on the Subscribers page, on the row listing your subscriber server, pick the option to create a new database. Enter Customers Replica as the name and click OK and Next to accept defaults.

On the Merge Agent Security page, click the ellipses (...) button. This agent is running on your subscriber computer, so enter myserver2repl_merge as the user account. (If you are running Active Directory you might have a single repl_merge agent that you can use.) Enter the password you assigned and click the Next but-

Action

- Create a new subscription.
- Set security on the new subscriber database.
- Synchronize data changes.
- Test the synchronization.

ton. Accept the defaults for the Synchronization Schedule (on demand only). On the Initialize Subscriptions screen, select At first synchronization in the drop-down list. Accept the defaults for Subscription type and finish the wizard.

6. Set Security on the new Subscriber Database

You just created a new database, so you have to assign permissions to it. Within the SQL Server Management Studio connected to the subscriber database, expand the main Security node. Right-click Logins/New Login. Add myserver2\repl_merge to the list of logins. Under User Mapping, select the Customers Replica database and assign the db_owner role to the repl_merge user.

7. Synchronizing Data Changes

When you set up the subscription, the instructions specified that you would manually synchronize the databases. This recommendation works best when you are learning the process, or if you have a mobile subscriber that is only connected at random times. In other situations, you can set up schedules that will automatically synchronize the databases at specified intervals. In the manual case, you need to perform the following steps to synchronize the databases. First, ensure that the SQL Server Agent is running on the subscriber computer. If you truly have a mobile database, you might leave this Agent set to Manual start and only turn it on when you need to synchronize.

Connect to the subscriber database using the SQL Server Management Studio. Expand the Replication and Local Subscriptions folders. Right-click the Customers Replica database and choose the View Synchronization Status option. Click the Start button. If the process fails, you will have to check the log files to see what went wrong. The most likely errors are problems with setting security permissions.

8. Test the Synchronization

Once the publication and subscriptions are established, the day-to-day process is relatively easy. In the publisher database, make a change to one of the Last Name entries. Be sure the changes are committed. If you use the graphical editor, switch to a different row. In the subscriber database, run the Synchronize step again. Run a select command on the TempCustomer table and verify that the changes made in the publisher database appear in the subscriber database. You can also make changes in the subscriber database and they will be synchronized to the main publisher database.

The Internet



Activity: Public Web Pages with SQL Server

Sometimes you need to build Web pages that can be accessed by customers or suppliers—people outside of your organization. Generally, you cannot ask them to install an application, but you can provide a Web site that retrieves data from the database to answer basic questions. Microsoft has several powerful tools to help you provide (and collect) data using a Web interface. The Web is a good way to build interactive database applications. The data and the applications are stored centrally and fully under your control, yet users can reach the system from almost anywhere in the world from many different devices. You should consider building all of your applications using a Web-based approach. This short exercise only

touches on the abilities available, but it should be enough to get you started working on larger projects.

ASP .NET is Microsoft's primary technology for building applications and Web sites. It has progressed through several variations, has hundreds (or thousands) of options, is relatively easy to use, includes security features, and is amazingly fast and scalable. At heart, it is a programming system, but it supports a variety of languages, particularly C# (c-sharp) and VB (Visual Basic). If you need to avoid

detailed programming, and still want to implement complex Web sites, you should consider implementing Microsoft SharePoint—a Web-based technology for sharing files and applications. Many of the reporting and business-intelligence features of SQL Server 2008 interface fairly quickly with SharePoint and Excel.

This exercise presents a simple example of a customer who wants to look up some information about a Sales order. The customer has a SaleID and a CustomerID. You could build the system with just one of those numbers, but requiring both improves security by making it more difficult for people to randomly guess the values. You can build this application with a single Web-based form in .NET. You can test and debug the form in Visual Studio, but if you want to deploy it on the Internet, you need to have a server that runs Microsoft's Internet Information Server (IIS). Any Windows-based server (and Vista) can run IIS, but you need to install it. And, if you want to access the data from the Internet, you need to configure your computer and network—tasks that are straightforward but beyond this book.

To start, you need new application. Start Visual Studio and choose File/New Web Site. Accept the default ASP.NET Web Site type, choose a programming language (e.g., Visual Basic), and check the path location for the new site. You should probably rename the application to something like AllPowder11 instead of WebSite1. When the system completes the startup, you will have a new project with a new Default.aspx page. An aspx page generally contains HTML along with .NET objects. You could embed programming code on the page, but it is better to let the system store programming functions on a separate page.

The aspx page contains a basic shell with some HTML and setup information. In the text, change the title to All Powder and switch to Design view. Currently, the page contains no display data. Roll your cursor over to the Toolbox and drag a Label and drop it on the top of the form. Look at the Properties window and change the Text entry to: All Powder Board and Ski Shop Sales Check. Use the main menu (Table) to add a 2x2 table to the page to control layout. From the Toolbox, drag a label into the first row and column. Set its text property to Sale ID. Drag a TextBox to the second column and set its ID property to SaleIDTextBox. Repeat the process for the second row using CustomerID instead of SaleID. Finally, drag a button onto the bottom of the form and set its text to Retrieve. Figure 11.8 shows what your form should look like, but feel free to add styles and improve the appearance. All of the formatting and control methods in HTML and CSS style sheets can be used within an aspx page. You can switch between Design and HTML (Source) views to work on the page.

Action

Create a new Visual Studio Web site project.

Add SaleID and CustomerID text boxes to the main page.

Create a stored procedure in SQL Server to retrieve SaleItem data based on SaleID and CustomerID parameters.

Add a SqlDataSource to the aspx page that connects to the stored procedure.

Add a GridView object to the page to display the data.

Test the form.

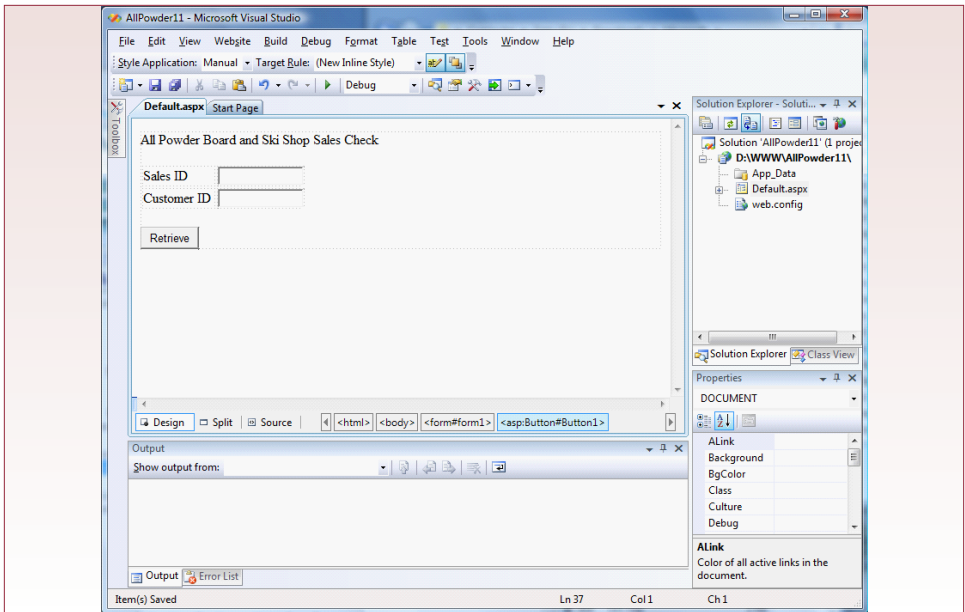


Figure 11.8

Currently, this page simply has text boxes so the customer can enter the SaleID and CustomerID values. The main step now is to go to the database and retrieve the SaleItem information for the specified sale. The real trick is to perform this step in SQL with a simple query—avoiding the use of any code.

So, you need to write a query. Actually, you need to put a query inside of a stored procedure. Start SQL Server Management Studio and connect to your database. Right-click the AllPowder database and select New Query. First write a simple query that retrieves data the customer would want to see—ignoring the SaleID and CustomerID data. Select the SKU, Price, Quantity, Size, Category, Style, and Color from the Sale, SaleItem, Inventory, and ItemModel tables. Run the query to test it. Now define the @CustomerID and @SaleID parameters and add a WHERE clause to the query. If you want to test the query at this stage, use CustomerID=871 and SaleID=1004. Figure 11.9 shows how to turn the query into a stored procedure that requires the two parameters. Run this query to create the

Figure 11.9

```
CREATE PROCEDURE GetOneSaleItems
  (@CustomerID int, @SaleID int)
AS
BEGIN
    SELECT SaleItem.SKU, QuantitySold, SalePrice,
    QuantitySold*SalePrice As Value, ItemSize, Category, Style, Color
    FROM Sale
    INNER JOIN SaleItem ON Sale.SaleID=SaleItem.SaleID
    INNER JOIN Inventory ON SaleItem.SKU=Inventory.SKU
    INNER JOIN ItemModel ON Inventory.ModelID=ItemModel.ModelID
    WHERE (CustomerID=@CustomerID) And (Sale.SaleID=@SaleID)
END
Go
```

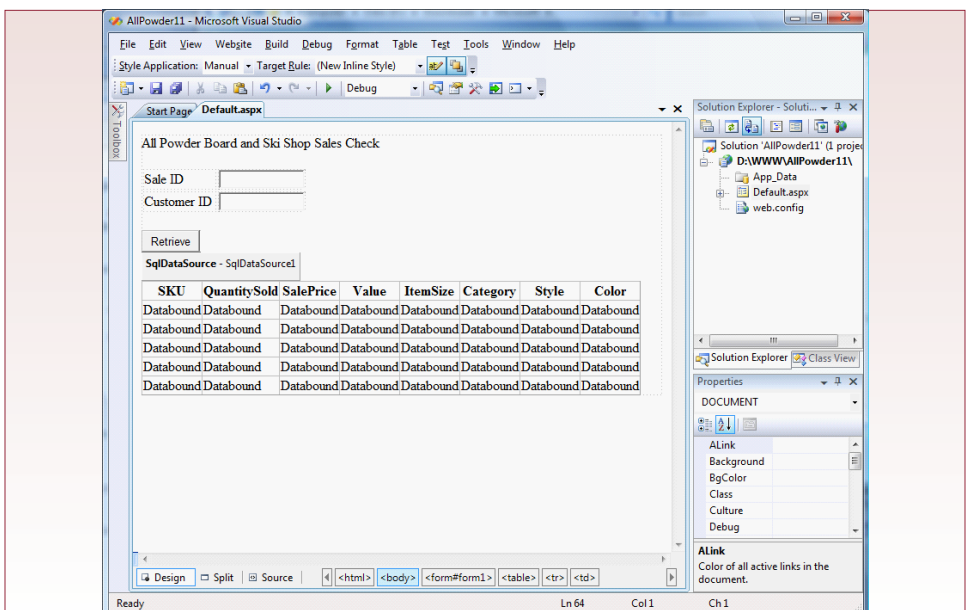
stored procedure (GetOneSaleItems). Your Web page will call this procedure and pass in the values for CustomerID and SaleID.

Before you close the Management Studio, you should add a new SQL Server login that can access this database. You could use the existing dbo account, or you can create a special user that can only retrieve the data you want to display. Create a new login if necessary and remember the username and password.

Return to your Visual Studio project. You are now ready to retrieve the data from the stored procedure and display it on the page. Open the Toolbox, expand the Data section if necessary, and drag a SqlDataSource onto the page. Click the button to Configure Data Source. Choose Microsoft SQL Server as the data type (read the choices carefully). Enter the name of your database server. It is often best to choose SQL Server Authentication instead of Windows, giving you greater control over exactly what data can be accessed via the Web pages. Be sure to check the box to “Save my password.” Select your AllPowder database and click the button to Test Connection. Click OK to close the window and Next to accept the connection. Also click Next to save the connection string in the application’s configuration file. Now you build a statement in ASP .NET that will be passed to the server. In almost all cases, you will want to use stored procedures. Click the option button for a stored procedure and go to the Next screen. Click the option button for a stored procedure at the bottom of the screen and choose the one you created (GetOneSaleItems). You will not need to change or delete any data so click the Next button.

Your ASP .NET statement has an option to take data from the HTML page and pass it to the stored procedure parameter. Your stored procedure has two parameters. Select the CustomerID parameter. In the Parameter Source drop-down list, choose the Form option. Enter CustomerIDTextBox into the FormField. Repeat the process for the SaleID parameter, entering SaleIDTextBox. When the page runs, it will pick up the values returned by the form, assign them to the parameters, call the procedure on the server and return the results to an internal dataset.

Figure 11.10



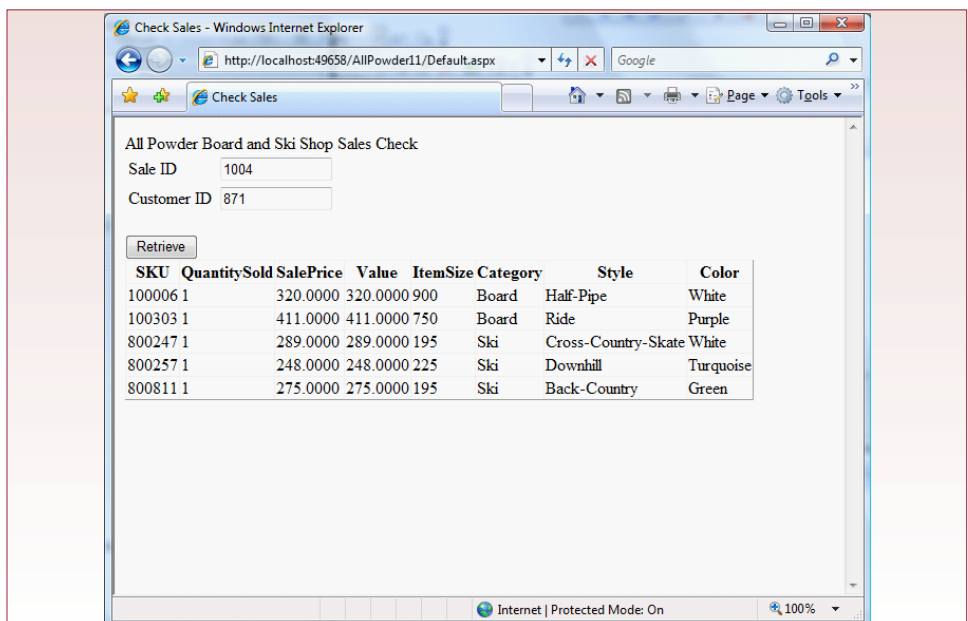
The only thing left to do is display the results on the page. Because it is likely that multiple rows will be returned, you need an object that can handle repeating data. Drag a GridView object from the Toolbox and drop it on the form. Set the data source to `SqlDataSource1`. (Note that in a large project, you should give all of these objects better names so that when you have multiple data sources you remember what each one does.) Figure 11.10 shows that the grid automatically picks up the columns from the data source.

You can edit the columns to set widths, styles, and formats, but you really want to see if the form works. Click the Save All button to be safe. Run the form by clicking the Start/Debug button on the main toolbar. You will probably be prompted to click an OK and a Yes button to accept debugging options. The basic form should appear with only the two boxes and the Retrieve button. Enter 1004 as the SaleID and 871 as the CustomerID, and click the Retrieve button. Figure 11.11 shows the resulting page that retrieves the matching items for that Sale. If you publish the project to a Web server, users simply access the `Default.aspx` page on the server and the form works the same way.

Notice that the Retrieve button does not actually do anything—it has no code. It is only there to provide a means to submit the page back to the server. You should also test the form with different values for SaleID and CustomerID. However, you must pick values that match a particular Sale to get results. Requiring the user to enter both numbers is a small security precaution. If you used only the SaleID, anyone could guess that the IDs are sequential, randomly enter numbers, and see results for almost any sale. The owners of All Powder Board and Ski Shop probably do not want that much information available to everyone on the Internet.

Visual Studio, SQL Server, and ASP .NET provide the ability to create vastly more complex applications—limited only by your imagination and your programming ability. But notice that you were able to create a fairly useful page with no coding.

Figure 11.11





Activity: Transferring Data with XML

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 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.

Action

Use SQL Management Studio to create an XML list of Employees using FOR XML.

Save the output as a file.

Edit the file and add <Employees> to the top and </Employees> to the bottom.

Open the file in Internet Explorer.

```
SELECT *
FROM Employee
FOR XML AUTO, ELEMENTS, XMLSCHEMA('Employees')
```

Figure 11.12

Exporting data in XML format is relatively easy with SQL Server, but there can be some complications. Outputting tagged data is straightforward using the FOR XML option of the SELECT statement. However, XML files should really include a schema definition (XSD) that describes the allowable content of the file. Systems that use an XML file can use the XSD file to verify that the data file is correct. Figure 11.12 shows the basic SQL command to create both the schema and output the rows as XML-tagged data.

Figure 11.13 shows part of the resulting XML file with some editing. Before distributing the file, you will have to open the file in a text editor to make some minor changes. First, if you generated it, save the xsd information in a separate file. Second, you have to add the starting <Employees> and ending </Employees> tag around the entire set of data. Note that each <Employee> data line will be displayed on one row.

Figure 11.13

```
<Employees>
<Employee xmlns="Employees">
  <LastName>Staff</LastName>
  <FirstName></FirstName>
</Employee>
<Employee xmlns="Employees">
  <LastName>Killy</LastName>
  <FirstName>Jean-Claude</FirstName>
</Employee>
...
</Employees>
```


Create a new query in SQL Server Management Studio. Read in (or paste) the entire `Employees.xml` file that you created earlier. At the top of this query/file, you need to declare the data as a new variable. The basic lines at the top are:

```
DECLARE @docHandle int
DECLARE @xmlDocument nvarchar(max)
SET @xmlDocument = N'
<Employees>
<Employee>
  <EmployeeID>0</EmployeeID>
```

...

You need to add only the first three lines. The rest represent your XML data. You also have to add a few lines at the bottom of the file. One of the most important is to place a single quotation mark at the end of the XML data. Then you add an EXEC line to a system stored procedure that prepares the text as an XML document. At that point, you can use a SELECT statement that calls the OPENXML function to retrieve rows directly from the XML data.

```
</Employees>'
EXEC sp_xml_preparedocument @docHandle OUTPUT, @
xmlDocument
SELECT *
FROM OPENXML(@docHandle, N'/Employees/Employee', 2)
  WITH (EmployeeID int,
        TaxpayerID nvarchar(50),
        LastName nvarchar(50),
        FirstName nvarchar(50),
        Address nvarchar(50),
        Phone nvarchar(50),
        City nvarchar(50),
        State nvarchar(20),
        ZIP nvarchar(20),
        Department nvarchar(50) )
```

Notice that the WITH statement defines the columns of the pseudo-table in the XML data. Of course, as long as you can retrieve data as rows, you can do anything with it in SQL. For example, you might use an INSERT INTO ... SELECT * FROM OPENXML ... command to extract the rows from the XML and add them to an existing table. Figure 11.15 shows some of the sample rows retrieved from the XML data.

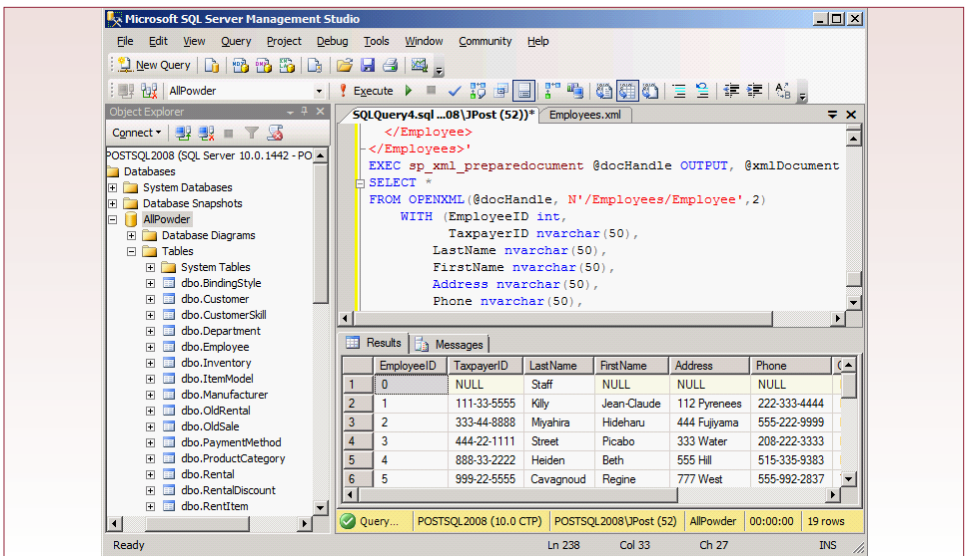


Figure 11.15

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. 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. Employees would connect to the database using the Web interface. 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 employee productivity declined or improved.

1. Check the performance of the database using an Internet connection from off-site. If possible, try it with a cable-modem connection and with a dial-up connection. Is the performance fast enough?
2. Outline the security issues involved in enabling employees to access the database from home over the Internet.
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 to access the application that handles sales, orders, and management reports. However, with the Web-based forms, the process is straightforward. On the other hand, the company deals with a huge number of parts, and it seems like vendors constantly change descriptions and prices. The company is trying to work with the vendors to connect to their databases and at least be able to retrieve replicated materialized views.

1. Set up a small new database that would be created by a vendor to hold information on parts. Replicate the table as read only so the Offshore Speed company can subscribe to it to automatically receive changes on a regular basis.
2. Some of the company's partner firms would like to receive files that they can read into their databases or into Excel. Set up a procedure that will create text files with basic order data for a selected partner.
3. Create a Web page that customers can use to check on the status of their orders. You should create a separate password for the customers that will be stored within the Customer table. Verify that the password and order number are correct before displaying the 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. Describe any distributed features or database links that will be useful to the project and list any problems you might encounter.
2. Create a replica and 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 form and response page that enables customers (or employees) to enter some identifier and receive additional information. For example, a customer might select a product category and receive a list of products in that category.
5. Create a second database and build a link to that database, so at least one form operates using data in the second database.

Chapter 12

Physical Database Design

Chapter Outline

Storing Data, 243

Lab Exercise, 243

All Powder Board and Ski Shop, 243

Data Clusters, 246

Exercises, 247

Final Project, 247

Objectives

- Create partitions.

Storing Data

SQL Server handles most data storage tasks automatically and optimizes retrievals based on the types and amount of data stored. You should generally let SQL Server determine the optimal way to store and retrieve the data. In cases of huge databases, the best recommendations are to (1) use RAID drives, and (2) use server clusters that automatically balance the load. Also, use the Tuning Advisor to recommend the use of indexes—which are based on B+trees. As explained in Chapter 10, you can also gain some control over data storage by defining file groups on different devices. About the only other control option you have is the specification of partitions and recommendations for clusters.

Lab Exercise

All Powder Board and Ski Shop

The main text explains that data can be stored using several methods, including sequential, B+trees, and direct or hashed storage. For the most part, SQL Server stores data similar to linked lists using the data pages. By default, SQL Server indexes any primary key columns, and the indexes are stored and searched using B+trees. This approach is the best general storage method, but sometimes you might need more control over how the data is stored and retrieved.

SQL Server handles other storage methods through partitions and clusters. Partitions enable you to store different parts of a table in different locations. This trick can be useful even for relatively small databases. In the All Powder case, the inventory data tends to change every year as new products are introduced. You do not want to delete the old data because the managers want to go back and look at sales in various categories. On the other hand, you do not need it taking up space on the main disk drives—because the old models are no longer available for sale or rental. A partition enables you to move the older data to a different disk drive.



Activity: Create Data Partitions

Partitioning enables you to split a data table into multiple pieces. Each piece contains the same types of data (same column names and same data types), but are placed in different locations (file groups). The filegroup option enables you to store one part of the table in one location (disk drive) and the rest in other locations. Placing data in different filegroups also improves query performance because the DBMS can

restrict the search to a single partition. It also improves backup and recovery operations since you can tell the DBMS to operate on a filegroup or partition at a time. Finally, the partition can be invisible to the query system and the users. Existing queries and applications will continue to work correctly with no changes.

Partitions are defined based on the data contained in the row. Oracle supports three types of partitions: range, list, and hashed. A couple of composite types are also supported, but they are not covered here because they are simply combinations of the three base types. Range partitions are easy to understand and are often

Action

Create three new filegroups with one new file in each.

Create a partition function based on an integer year.

Create a partition scheme.

Define a new ItemModel table that assigns data to the partitions based on ModelYear.

```
ALTER Database AllPowder
  Add FileGroup fgModel01;
ALTER Database AllPowder
  Add FileGroup fgModel02;
ALTER Database AllPowder
  Add FileGroup fgModel03;
go

ALTER Database AllPowder
  Add File
  (
    Name = Item01,
    Filename = 'C:\Program Files\Microsoft SQL Server\MSSQL10.
MSSQLSERVER\MSSQL\DATA\AP_Item01.ndf',
    Size = 20MB,
    FileGrowth = 5MB
  ) TO FileGroup fgModel01;
ALTER Database AllPowder
  Add File
  (
    Name = Item02,
    Filename = 'C:\Program Files\Microsoft SQL Server\MSSQL10.
MSSQLSERVER\MSSQL\DATA\AP_Item02.ndf',
    Size = 20MB,
    FileGrowth = 5MB
  ) TO FileGroup fgModel02;
ALTER Database AllPowder
  Add File
  (
    Name = Item03,
    Filename = 'C:\Program Files\Microsoft SQL Server\MSSQL10.
MSSQLSERVER\MSSQL\DATA\AP_Item03.ndf',
    Size = 20MB,
    FileGrowth = 5MB
  ) TO FileGroup fgModel03;
go
```

Figure 12.1

used for date or ID columns. You choose a data column and split the rows based on ranges of data. For the case of the ItemModel table, you could partition the data based on the ModelYear. You might create a new partition for each year, or you can use groups: Old (ModelYear < 2000), Recent (ModelYear >= 2000 and ModelYear < 2006), New (ModelYear >= 2006).

Of course, you need to create the three filegroups, or at least three new files, before running this command. You can use the Management Studio to create the filegroups and files, however, it is almost as easy to write the commands in a query. Figure 12.1 shows the commands needed to create three files and three filegroups. The commands first define each new filegroup and then create a new file to assign to each group. You cannot use a filegroup unless at least one file is associated with it. Notice that all three files are created in the same folder in this example—purely for convenience. In reality, you would define each file on a different disk drive with different characteristics. For example, current data, with heavy access needs, should be stored on a fast drive. Older data could be stored across a network on a less expensive, slower drive.

```
CREATE PARTITION FUNCTION ItemModelPF (int)
AS RANGE LEFT FOR VALUES (2000, 2006);
go

CREATE PARTITION SCHEME ItemModelPS AS
PARTITION ItemModelPF TO (fgModel01, fgModel02, fgModel03);
Go
```

Figure 12.2

The next step is to define a partition function that defines how you want the data to be split. The partition function takes data from a single column and performs a series of less-than-or-equal-to comparisons (RANGE LEFT) or greater-than comparisons (RANGE RIGHT). Essentially, the function lists the split points. Figure 12.2 shows a partition function where the year is stored as an integer. Note that if you want to partition on dates, the split points are complicated if you want to use a RANGE LEFT command. Because RANGE LEFT uses less-than-or-equal-to, you have to specify the exact date and time, but times are measured in fractions of a second, so it is much easier to use a RANGE RIGHT command for date columns. In the example, notice that you need to specify two split points (2000 and 2006) to get three partitions.

After the partition function is defined, you need to create a partition scheme that matches the function to the filegroups. Figure 12.2 also shows the partition scheme. Basically, you give the scheme a name, provide the name of the partition function and list the filegroups to be used for each partition.

Finally, you can run the CREATE TABLE command to build the ItemModel2 table. To save time, you can ignore the referential integrity constraints. Note that because the partition function does not use the ModelID column, you should not declare ModelID as the primary key. Figure 12.3 shows the basic CREATE TABLE command. The only real difference lies at the end with the addition of the ON ItemModelPS(ModelYear) statement that passes the ModelYear value to the

Figure 12.3

```
CREATE TABLE ItemModel2(
    ModelID nvarchar(50) NOT NULL,
    ManufacturerID int NULL,
    Category nvarchar(50) NULL,
    Color nvarchar(50) NULL,
    Cost money NULL,
    ModelYear int not NULL,
    Graphics nvarchar(50) NULL,
    ItemMaterial nvarchar(50) NULL,
    ListPrice money NULL,
    Style nvarchar(50) NULL,
    SkillLevel int NULL,
    WeightMax real NULL,
    WeightMin real NULL,
    WaistWidth real NULL,
    EffectiveEdge real NULL,
    BindingStyle nvarchar(50) NULL,
    RentalRate money NULL
) ON ItemModelPS (ModelYear);
go
```

partition scheme. You can use an INSERT INTO command to transfer the data from the existing table into the new one. As data is added to the table, the partition function and scheme will transfer the data to the appropriate filegroup. The power of the partition is that nothing else changes. Queries work just as they did without the partition, and your applications run just as before.

Data Clusters



Activity: Create Data Clusters

Clusters are different from partitions—the goal is to store related data close together. Really close together. Remember that disk drives are the slowest component of the computer (not counting interfaces with people), because they rely on mechanical elements. Data that is stored in different locations on the drive take time to retrieve because the drive head has to wait for the sector to spin around. The goal of clustering is to reduce this delay by storing related data together so that it can be retrieved in one pass. Filegroups are one way to store data together—but only at a coarse level. When the SQL Server goes to the engine, it retrieves data in pages (a small group of bytes read at the same time). You can sometimes improve performance by asking SQL Server to cluster data that is always needed at the same time. In particular, it is helpful to keep values in primary keys close together.

Figure 12.4 shows that it is straightforward to create a clustered index on a table. When you create a clustered index, you are effectively telling SQL Server how to physically store the data. Consequently, if you try to create the index shown, you will receive an error message. Because a primary key was specified when the table was created, SQL Server automatically defined a clustered index on that column. If you want to specify a different cluster, you must delete primary key index. Data can only be stored physically in one way. In most cases, the database will run most efficiently by clustering on the primary key.

Note that it is also possible to combine clustering and partitions—so that clustered indexes are stored separately on each partition. To implement this combination, you define a primary key or clustered index as usual, but add the ON partition_scheme (column) clause at the end.

You could try to test the performance of your database with a few of these changes, but you need millions of rows of data, and a variety of hard drives to have any chance at perceiving a difference.

Figure 12.4

```
CREATE CLUSTERED INDEX idxInventory  
ON Inventory (SKU, ModelID)
```

(Error: Only one clustered index per table.)

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. Assuming the company becomes substantially larger, what storage strategies would be useful?
2. 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. Identify the tables that could be improved using partitions or clusters. Explain your reasoning.
2. Partition the project table into three 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. Identify the tables that could be improved using partitions or clusters. Explain your reasoning.
2. Partition the Situation and ProposedSolution tables into three segments based on the date.
3. Partition the Crew table into four regions based on Country.

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. Identify the tables that could be improved using partitions or clusters. Explain your reasoning.
2. Create a partition on at least one table.