

PART i
**DATABASE BASICS AND
STRUCTURED QUERY
LANGUAGE**

COPYRIGHTED MATERIAL

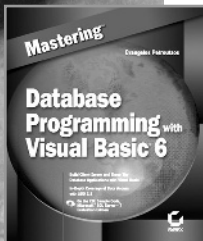


Chapter 1

DATABASE ACCESS: ARCHITECTURES AND TECHNOLOGIES

The first chapter in a typical computer book is an introduction to the book's topic. So, this chapter should be an introduction to databases, but it isn't. Databases are the broadest and most diverse area of computer programming. Before I can give you very much detail on what a database is, how to design one, and then how to program it, I must explain some of the key concepts in this field and the numerous acronyms that are used heavily in this book.

In my attempt to explain all the data access-related technologies at the beginning of the book, I may have oversimplified things. This chapter is for readers who are not comfortable with the various acronyms like *OLE DB*, *ADO*, terms like *n-tiers*, and so on. If you know the difference between *OLE DB* and *ADO*, you can skip this chapter and jump to the next chapter, where I discuss the structure of databases.



Adapted from *Mastering™ Database Programming with Visual Basic® 6* by Evangelos Petroutsos

ISBN 0-7821-2598-1 896 pages \$39.99

DATABASES AND DATABASE MANAGEMENT SYSTEMS

A *database* is a complex object for storing structured information, which is organized and stored in a way that allows its quick and efficient retrieval. We put a lot of effort into designing a database so that we can retrieve the data easily. The information is broken into *tables*, and each table stores different entities (one table stores customer information, another table stores product information, and so on). We break the information into smaller chunks, so that we can manage it easily (divide and conquer). We can design *rules* to protect the database against user actions and ask the DBMS to enforce these rules (for example, reject customers without a name). These rules apply to all the items stored in the customers table; the same rules don't apply to the products table and the orders table, of course.

In addition to tables, we define *relationships* between tables. Relationships allow users to combine information from multiple tables. Let's say you store customer information in one table and sales information in another table. By establishing a relationship between the two tables, you can quickly retrieve the invoices issued to a specific customer. Without such a relationship, you would have to scan the entire invoice table to isolate the desired invoices. This view of a database, made up of tables related to one another, is a *conceptual* view of the database. And the database that relies on relationships between tables is called *relational*.

The actual structure of the database on the disk is quite different. In fact, you have no idea how data is stored in the database (and you should be thankful for this). The information is physically stored into and recalled from the database by a special program known as a *database management system* (DBMS). DBMSs are among the most complicated applications, and a modern DBMS can instantly locate a record in a table with several million records. While the DBMS maintains all the information in the database, applications can access this information through statements made in *Structured Query Language* (SQL), a language for specifying high-level operations. These operations are called *queries*, and there are two types of queries: *selection queries*, which extract information from the database, and *action queries*, which update the database. How the DBMS maintains, updates, and retrieves this information is something the application doesn't have to deal with.

Database Access: Architectures and Technologies 5

Specifically, a DBMS provides the following functions:

- ▶ A DBMS allows applications to define the structure of a database with SQL statements. The subset of SQL statements that define or edit this structure is called *Data Definition Language* (DDL). All DBMSs use a visual interface to define the structure of a database with simple point-and-click operations, but these tools translate the actions of the user into the appropriate DDL statements. SQL Server, for example, allows you to create databases with a visual tool, the Enterprise Manager, but it also generates the equivalent DDL statements and stores them into a special file, called a *script*.
- ▶ A DBMS allows applications to manipulate the information stored in the database with *SQL statements*. The subset of SQL statements that manipulates this information is called *Data Manipulation Language* (DML). The basic data-manipulation actions are the insertion of new records, modification and deletion of existing ones, and record retrieval.
- ▶ A DBMS protects the integrity of the database by enforcing certain rules, which are incorporated into the design of the database. You can specify default values, prohibit certain fields from being empty, forbid the deletion of records that are linked to other records, and so on. For example, you can tell the DBMS not to remove a customer if the customer is linked to one or more invoices. If you could remove the customer, that customer's invoices would be "orphaned." In addition, the DBMS is responsible for the security of the database (it protects the database from access by unauthorized users).



NOTE

The terms "records" and "fields" are not used in the context of relational databases. We now talk about "rows" and "columns." I'm using the old-fashioned terms because most readers who are new to relational databases are probably more familiar with records and fields. If you have programmed older ISAM databases, or even random-access files, you're probably more familiar with records and fields. I will drop the older terms shortly.

SQL Server is a database management system and not a database. An *SQL Server database* is a database maintained by SQL Server. SQL is a universal language for manipulating databases and is supported by all



6 Chapter One

DBMSs—we'll examine it in detail in Chapter 3, "Structured Query Language." SQL retrieves selected records from the database and returns them to the client. The set of records returned by an SQL statement is called a *cursor*. If another user changes some records in the database, those changes will not be reflected in the existing cursors. We need a more complicated mechanism that will synchronize the data in the database and the client computer, and this mechanism is *ActiveX Data Objects (ADO)*. We'll get to ADO soon, but first let's discuss Microsoft's view of data access. We'll look at the big picture first, and then at the individual components.

WINDOWS DNA

The one term you'll be hearing and reading about most frequently in association with Windows 2000 is *DNA*. DNA stands for *Distributed interNet-Architecture*, and it's a methodology for building distributed applications. A *methodology* is a set of rules, or suggestions, and not a blueprint for developing applications; it's a recommendation on how to build distributed applications. Because this recommendation comes from Microsoft, you can consider it a very clear hint of the shape of things to come. Follow these recommendations and your applications will not be outdated soon.

A *distributed application* is one made up of multiple components that run on different machines. These machines can be interconnected through a local area network (LAN)—or a few machines on a LAN and a few more machines on the Internet. To make things even more interesting, throw into the mix a second LAN, located a few thousand miles away. So, in effect, DNA is about building applications for the Internet. If you want to understand how all the pieces fit together, why Microsoft is introducing new access technologies, and why it chooses weird acronyms to describe them, you should start with the big picture.

The big picture starts with the realization that not all information is stored in databases. When most of us are talking about data, we think of databases, rows, and columns—well-structured data that can be easily retrieved. But not all information can be stored in databases. A lot of information is stored in e-mail folders, text documents, spreadsheets, even audio and video files. The ultimate data-access technology is one that can access any information, from anywhere, whether it be a database, an electronic mailbox, a text file, even a handheld device. Ideally, we should be able to access information in a uniform way, no matter where this information resides. And we should also be able to access it from anywhere, meaning the Internet.

Universal Data Access

Microsoft uses the term *Universal Data Access* to describe this idea. The premise of Universal Data Access is to allow applications to efficiently access data where it resides, through a common set of tools. There's nothing new about accessing diverse sources of information today, but how is it done? In most cases, we replicate the information. Quite often, we transform the information as we replicate it. The problem with this approach is that we end up with multiple copies of the same information (a highly expensive and wasteful practice).

At a high level, Universal Data Access can be visualized as shown in Figure 1.1. *Data providers*, or *data stores*, store information, and their job is to expose the data through data services. *Data consumers* receive and process the data. Finally, *business components* provide common services that extend the native functionality of the data providers.

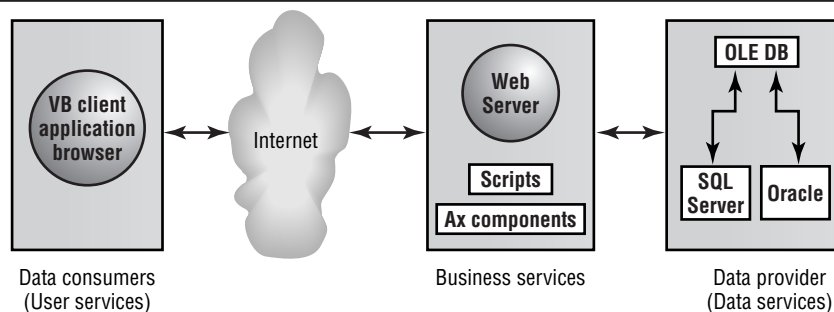


FIGURE 1.1: Universal Data Access

A data provider can be a database management system like SQL Server, but it doesn't necessarily need to be. Eventually, every object that stores data will become a data provider.

ADO 2.5, which is currently distributed with Windows 2000, supports a few special objects for accessing semi-structured data. *Semi-structured data* is the data you retrieve from sources other than database rows, such as folders and their files, e-mail folders, and so on.

For the purposes of this book, data providers are DBMSs. The data consumer is an application that uses the data. This application is usually called a *client* application, because it is being served by the data provider. The client application makes requests to the DBMS, and the DBMS carries out the requests. The data consumer need not be a typical client application



8 Chapter One

with a visible interface. In this book, however, you'll learn how to build client applications that interact with the user. Finally, the *service components* are programs that read the data from the data source in their native format and transform it into a format that's more suitable for the client application. Universal Data Access requires four basic service components, which are:

Cursor Service The *UDA cursor* is a structure for storing the information returned by the data source. The cursor is like a table, made up of rows and columns. The cursor service provides an efficient, client-side cache with local scrolling, filtering, and sorting capabilities. The cursor is usually moved to the client (that is, the address space where the client application is running), and the client application should be able to scroll, filter, and sort the rows of the cursor without requesting a new cursor from the DBMS. Figure 1.2 shows a client application for browsing and editing customer data. A cursor with all customers (or selected ones) is maintained on the client. The scrollbar at the bottom of the *Form*, which is a Visual Basic control, allows the user to move through the rows of the cursor. The fields of the current row in the cursor are displayed on the *Form* and can be edited.

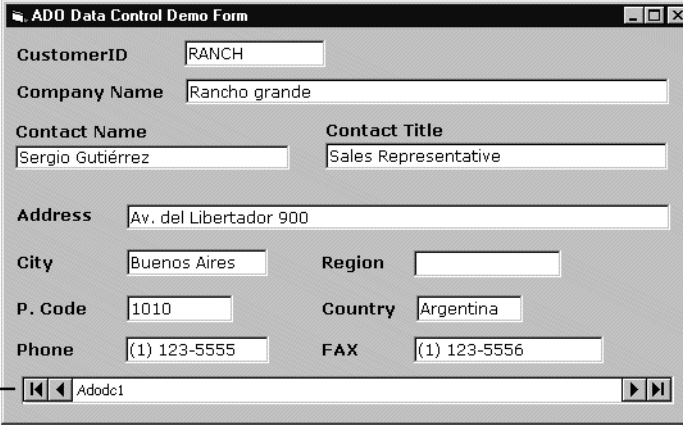


NOTE

For readers who are already familiar with SQL Server, I must point out that the cursors you create with T-SQL statements are different than UDA cursors. The SQL Server cursor contains raw information (the rows extracted from the database with the SQL statement). The UDA cursor contains not only data, but the functionality to manipulate the data. This functionality is implemented mostly with methods for sorting and filtering the rows, navigational methods, and so on.

Synchronization Service This service updates the database with the data in the local cursor. This service must be able to update the database instantly, as the user edits the data in the cursor, or in batch mode. As you will see later in the book, there are two major types of cursors: those that reside on the server and those that reside on the client. The rows of a client-side cursor are moved to the client and can't be synchronized to the database at all times. It is possible for another user to edit the same rows in the database while the client application is processing those rows in the client-side cursor.

VB FORM



PRINI	Princesa Isabel Vinhos	Isabel de Castro	Sales Representative
QUEDE	Que Delicia	Bernardo Batista	Accounting Manager
QUEEN	Queen Cozinha	Lúcia Carvalho	Marketing Assistant
QUICK	QUICK-Stop	Horst Kloss	Accounting Manager
RANCH	Rancho grande	Sergio Gutiérrez	Sales Representative
RATTC	Rattlesnake Canyon Groc	Paula Wilson	Assistant Sales Represer
REGGC	Reggiani Caseifici	Maurizio Moroni	Sales Associate
RICAR	Ricardo Adocicados	Janete Limeira	Assistant Sales Agent
RICSU	Richter Supermarkt	Michael Holz	Sales Manager
ROMEY	Romero y tomillo	Alejandra Camino	Accounting Manager
SANTG	Santé Gourmet	Jonas Bergulfsen	Owner
SAVEA	Save-a-lot Markets	Jose Pavarotti	Sales Representative

CURSOR

FIGURE 1.2: The cursor service is encapsulated into the control at the bottom of the Form.

Shape Service This service allows the construction of hierarchically organized data. A plain cursor is made up of rows extracted from the database. The data may have come from one or more tables, but it appears as another table to the client; in other words, it has a flat structure. A hierarchical, or shaped, cursor contains information about the structure of the data. A hierarchically organized structure in Access is shown in Figure 1.3. The outer table contains customer information. Each customer has several invoices. Clicking the plus sign in front of the customer's name opens a list of the invoices issued to that customer. Finally, each order has one or more detail lines, viewed by clicking the plus sign in front of an invoice's number.



10 Chapter One

Customer	Company Name	Contact Name	Contact Title	Address			
RATTC	Rattlesnake Canyon Grocery	Paula Wilson	Assistant Sales Represent.	2817 Milton Dr.			
REGGC	Reggiani Caseifici	Maurizio Moroni	Sales Associate	Strada Provinciale 124			
RICAR	Ricardo Adocicados	Janete Limeira	Assistant Sales Agent	Av. Copacabana, 267			
RICSU	Richter Supermarkt	Michael Holz	Sales Manager	Grenzacherweg 237			
ROMEY	Romero y tomillo	Alejandra Camino	Accounting Manager	Gran Via, 1			
Order ID	Employee	Order Date	Required Date	Shipped Date	Ship Via	Freight	
10281	Peacock, Margaret	14-Sep-94	28-Sep-94	21-Sep-94	Speedy Express	\$2.94	Romer
10282	Peacock, Margaret	15-Sep-94	13-Oct-94	21-Sep-94	Speedy Express	\$12.69	Romer
10306	Davolio, Nancy	17-Oct-94	14-Nov-94	24-Oct-94	Federal Shipping	\$7.56	Romer
10917	Peacock, Margaret	01-Apr-96	29-Apr-96	10-Apr-96	United Package	\$8.29	Romer
Product	Unit Price	Quantity	Discount				
Nord-Ost Matjeshering	\$25.69	1	0%				
Camembert Pierrot	\$34.00	10	0%				
*	\$0.00	1	0%				
11013	Fuller, Andrew	09-May-96	06-Jun-96	10-May-96	Speedy Express	\$32.99	Romer
oNumber)						\$0.00	
SANTG	Santé Gourmet	Jonas Bergulfsen	Owner		Erling Skakkes gate 78		
SAVEA	Save-a-lot Markets	Jose Pavarotti	Sales Representative		187 Suffolk Ln.		

FIGURE 1.3: Viewing the records of a hierarchical structure with Access

Remote Data Service This service moves data from one component to another in a multitier environment. You'll understand what this service does when you read about tiers, later in this chapter. For example, in a web page that queries a database (a page that searches for books with title keywords, author names, and so on), the user enters the search criteria on the page, which is displayed in the browser. This information must be moved to the web server, then to the database. Obviously, you can't assume that the browser maintains a connection to the database. When it needs to query the database, it must call the appropriate program on the web server, passing the user-supplied keywords as arguments. A special program on the web server will intercept the values passed by the web page, and it will contact the database and extract the desired rows. The result of the query, which is a cursor, must be moved back to the client. Moving information from one process to another is called *marshalling*, and this is where the remote data service comes in, translating the data before passing it to another component. In later chapters you'll see how to use remote data services to write web pages bound to the fields of a remote database.

More services may be added in the future. Throughout this book, we'll discuss how these services enable you to write client-server applications. The cursor service, for example, is implemented in the Recordset object, which is a structure for storing selected records. You can use the Recordset object's properties and methods to navigate through records and

manipulate them. To navigate through the recordset, for example, you use the MoveNext method:

```
RS.MoveNext
```

RS is a Recordset object variable that represents the cursor on the client.

To change the Address field of the current record in the cursor, you use a statement like the following one:

```
RS.Fields("Address") = "10001 Palm Ave"
```

The cursor service is responsible for scrolling and updating the local cursor (the copy of the data maintained on the client). Depending on how you've set up the Recordset object, you can commit the changes immediately to the database, or you can commit all edited records at a later point. To commit the changes to the database, you can call either the Update method (to commit the current record) or the UpdateBatch method (to commit all the edited records in batch mode). Updating the database takes place through the synchronization server. The component services are totally transparent to you, and you can access them through the ADO objects (the Recordset object being one of them).

To summarize, Universal Data Access is a platform for developing distributed database applications that can access a diverse variety of data sources across an intranet or the Internet. You can think of Universal Data Access as the opposite of a universal database that can hold all types of information and requires that users actually move the information from its original source into the universal database. Let's see how this platform is implemented.

ADO and OLE DB

The two cornerstones of Universal Data Access are ActiveX Data Objects (ADO) and OLE for Databases (OLE DB). OLE DB is a layer that sits on top of the database. ADO sits on top of OLE DB and offers a simplified view of the database. Because each database exposes its functionality with its own set of application programming interface (API) functions, to access each database through its native interface, you'd have to learn the specifics of the database (low-level, technical details). Porting the application to another database would be a major undertaking. To write applications that talk to two different databases at once (SQL Server and Oracle, for instance), you'd have learned two different APIs and discovered the peculiarities of each database, unless you use OLE DB and ADO.



12 Chapter One

OLE DB offers a unified view of different data providers. Each database has its own set of OLE DB service providers, which provide a uniform view of the database. ADO hides the peculiarities of each database and gives developers a simple conceptual view of the underlying database. The difference between ADO and OLE DB is that OLE DB gives you more control over the data-access process, because it's a low-level interface. As far as Visual Basic is concerned, OLE DB uses pointers and other C++ argument-passing mechanisms, so it's substantially more difficult to use than ADO. Actually, most C++ programmers also use ADO to access databases because it offers a simpler, high-level view of the database.

Figure 1.4 shows how your application can access various databases. The most efficient method is to get there directly through OLE DB. This also happens to be the most difficult route, and it's not what VB programmers do. The next most efficient method is to go through ADO, which makes the OLE DB layer transparent to the application. You can also get to the database through Open DataBase Connectivity (ODBC), which is similar to OLE DB, but it's an older technology. If you can program ODBC, then you can program OLE DB, and there's no reason to use ODBC drivers. Many of you are already familiar with Data Access Objects (DAO) and Remote Data Objects (RDO). These are older technologies for accessing databases through ODBC. In a way, they are equivalent to ADO. These components, however, will not be updated in the future, and you should use them only if you're supporting database applications that already use DAO or RDO.

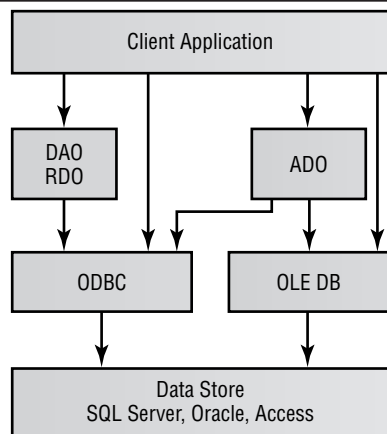
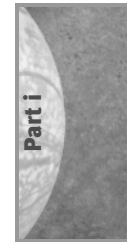


FIGURE 1.4: How client applications communicate with databases

There was a time when DAO was the only way for VB programmers to program databases, and as a result too many DAO-based applications are in use today (and will remain in use for a while). In fact, most VB books on the market still focus on DAO in discussing Visual Basic's data-access capabilities. However, it's an outdated technology, and you should not base any new project on DAO. I wouldn't be surprised if the ADO data control takes the place of the DAO data control in the toolbox of the next version of Visual Basic.



The ADO Objects

Let's switch our attention to the ADO objects. You'll find all the information you need about ADO in the following chapters, so this is a very brief overview to show you how the ADO object model reflects the basic operations we perform on databases. A client application performs the following:

- ▶ Establishes a connection to the database
- ▶ Executes commands against the database
- ▶ Retrieves information from the database

ADO's basic objects correspond to these operations, and they are appropriately named Connection, Command, and Recordset. The *Connection* object represents a connection to the database. To specify the database you want to connect to, set the Connection object's properties and then call the Open method to actually establish the connection. With the visual database tools, you don't even have to set any properties. You specify the database you want to connect to with point-and-click operations, and VB will prepare the appropriate Connection object for you.

Connection objects are expensive in terms of resources, and establishing a new connection is one of the most resource-intensive operations. It's crucial, therefore, to create a single Connection object in your application and use it for all the operations you want to perform against the database. If you need to connect to multiple databases, however, you must create one Connection object for each database. (This statement isn't universally true. There are situations, as in web applications, where you can't afford to maintain a Connection object for each viewer. As far as client applications are concerned, however, the rule is to establish a connection and maintain it during the course of the application.)

Once you've established a connection to the database, you can execute commands against it. A command can be an SQL statement or the name

14 Chapter One

of a stored procedure. *Stored procedures* are applications written in Transact-SQL (T-SQL, the programming language of SQL Server) and are usually called with arguments. To execute an SQL statement or a stored procedure, you must set up a Command object and then call its Execute method to execute the command. The Command object contains the SQL statement or the name of the stored procedure as well as the required arguments. If the command retrieves information from the database, the results are stored in a Recordset object, and you can access them from within your application through the methods and properties of the Recordset object.

Now that you've seen how an application communicates with the database, we'll turn our attention to the formerly ubiquitous client-server architecture. This architecture is different than DNA, but you need a solid understanding of client-server architecture before you adopt more complicated architectures for your applications. If you're developing database applications to run on a LAN, client-server architecture is adequate.

CLIENT-SERVER ARCHITECTURE

Client-server architecture is based on a simple premise: Different computers perform different tasks, and each computer can be optimized for a particular task. It makes sense, therefore, to separate the DBMS from the client application. In a networked environment, the DBMS resides on a single machine. However, many applications access the database, and all clients make requests from the same database. The program that accepts and services these requests is the DBMS, and the machine on which the DBMS is running is the *database server*. The client applications do not know how the data is stored in the database, nor do they care.

In client-server architecture, the application is broken into two distinct components, which work together for a common goal. These components are called *tiers*, and each tier implements a different functionality. The client-server model involves two tiers. As you will see later in this chapter, you can—and often should—build applications with more than two tiers.

Client-server became very popular because much of the processing is done on the client computer, which can be an inexpensive desktop computer. The more powerful the client is, the more processing it can do. Two clients may receive the same data from the client—sales by territory, for instance. One computer can do simple calculations, such as averages, while another, more powerful, client might combine the data with a mapping application to present complicated charts.

The Two-Tier Model

The first tier of a client-server application is the client tier, or *presentation tier*, which runs on the client. This tier contains code that presents data and interacts with the user, and it is usually a VB application. You can also build client tiers that run in a browser—these are web pages that contain controls, which are similar to the basic VB controls and allow the user to interact with the database. Figure 1.5 shows a simple client application for browsing and editing customers. This is a VB Form with TextBox controls that display the current customer's fields. Figure 1.6 shows a web page that does the same. It contains Text controls, which are bound to the customer fields in the database. The VB client application relies on the cursor service, while the web page relies on the remote data service.



CustomerID	RANCH			
Company Name	Rancho grande			
Contact Name	Sergio Gutiérrez		Contact Title	Sales Representative
Address	Av. del Libertador 900			
City	Buenos Aires	Region		
P. Code	1010	Country	Argentina	
Phone	(1) 123-5555	FAX	(1) 123-5556	

FIGURE 1.5: A VB client application for viewing and editing customers

The client application requests data from the database and displays it on one or more VB Forms. Once the data is on the client computer, your application can process it and present it in many different ways. The client computer is quite capable of manipulating the data locally, and the server is not involved in the process. If the user edits the fields, the application can update the database as well. The communication between the client and the server takes place through ADO, which makes it really simple to extract data from and update the database.

16 Chapter One

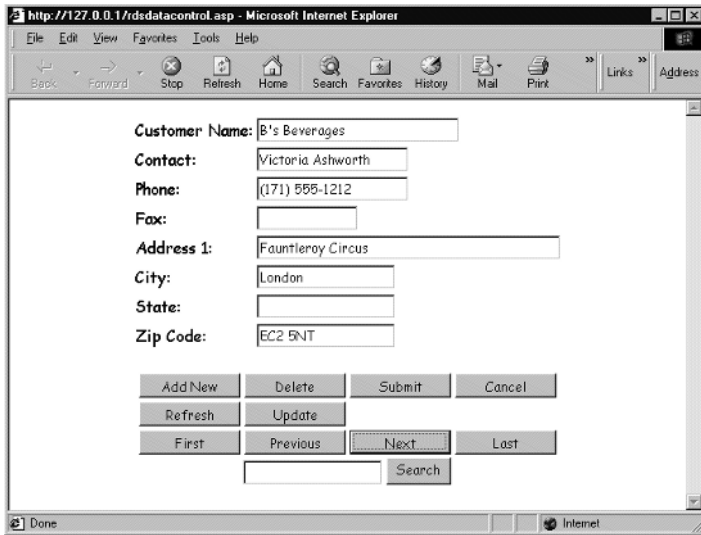


FIGURE 1.6: A web page for viewing and editing customers

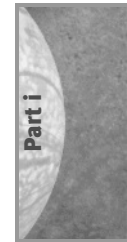
The second tier is the *database server*, or DBMS. This tier manipulates a very complex object, the database, and offers a simplified view of the database through OLE DB and ADO. Clients can make complicated requests like “Show me the names of the customers who have placed orders in excess of \$100,000 in the last three months,” or “Show me the best-selling products in the state of California.” The DBMS receives many requests of this type from the clients, and it must service them all. Obviously, the DBMS can’t afford to process the data before passing it to the client. One client might map the data on a graph, another client might display the same data on a ListBox control, and so on. The server’s job is to extract the required data from the tables and furnish them to the client in the form of a cursor. It simply transmits a cursor to the client and lets the client process the information. The more powerful the client, the more it can do with the data. (As you will see later in this chapter, in the discussion of stored procedures, certain operations that are performed frequently, or require the transmission of a very large number of rows to the client, can be carried out by the server.)

By splitting the workload between clients and servers, we allow each application to do what it can do best. The DBMS runs on one of the fastest machines on the network. The clients don’t have to be as powerful. In fact, there are two types of clients: thin and fat clients.

Thin and Fat Clients

Thin clients are less-powerful computers that do very little processing on their own. A browser is a thin client: Its presentation capabilities are determined by the current version of HTML. The benefits of thin clients are their cost (any computer that runs Internet Explorer or Netscape Navigator is good enough) and their connectivity (they can access the database server from anywhere). Another very important—and often overlooked—feature of thin clients is that their presentation capabilities don't vary. A client application that runs within a browser will run on virtually all computers. Thin clients are easy to maintain too, a fact that can lower the cost of deployment of the application.

A *fat client* is a desktop computer with rich presentation features. Because client applications that run on fat clients are far more flexible and powerful, they require more expensive computers to run, and their interfaces can't be standardized. You can make them as elaborate as the available hardware permits.



The Three-Tier Model

The two-tier model is a very efficient architecture for database applications, but not always the best choice. Most programmers develop two-tier applications that run on small local area networks. The most complete form of a database application, however, is one that involves three tiers.

In two-tier or client-server architecture, the client talks directly to the database server. Every application that connects to SQL Server or Oracle, and retrieves some information, like customer names or product prices, is a client-server application. The role of the database server is to access and update the data. Everything else is left to the client. In other words, the client is responsible for presenting the data to the user, parsing user input, preparing the appropriate requests for the database server, and finally implementing the so-called *business rules*. A business rule is a procedure specific to a corporation. Your corporation, for example, may have rules for establishing the credit line of its customers. These rules must be translated into VB code, which will be executed on the client. It is also possible to write procedures that will be executed on the server, but you can't move all the processing back to the server.

Business rules change often, as they reflect business practices. New rules are introduced, and existing ones are revised, which means that the code that implements them is subject to frequent changes. If you implement business rules on the client, you must distribute new executables to

18 Chapter One

the workstations and make sure all users on the network are using the latest version of the client software (that is, your applications). If business rules are implemented on the server, you don't have the problem of redistributing the application, but you place an additional burden on the server, tying it up with calculations that it's not optimized for or that could be performed on another machine.

This leads naturally to the introduction of a third tier, the middle tier. The *middle tier* is an object that sits between the client application and the server. It's a Class (or multiple Classes) that exposes several methods and isolates the client from the server. If many clients need to calculate insurance premiums, you can implement the calculations in the middle tier. Client applications can call the methods of the objects that reside on the middle tier and get the results. The client application need not know how premiums are calculated or whether the calculations involve any database access. All they need to know is the name of one or more methods of the objects that run on the middle tier.

The main advantage of the middle tier is that it isolates the client from the server. The client no longer accesses the database. Instead, it calls the methods exposed by the objects in the middle tier. A client application will eventually add a new customer to the database. Even this simple operation requires some validation. Is there a customer with the same key already in the database? Did the user fail to supply values for the required fields (we can't add a customer without a name, for example)? Adding orders to a database requires even more complicated validation. Do we have enough items of each product in stock to fill the order? And what do we do if we can only fill part of the order?

A well-structured application implements these operations in the middle tier. The client application doesn't have to know how each customer is stored in the database if it can call the `AddCustomer()` method passing the values of the fields (customer name, address, phone numbers, and so on) as arguments. The middle tier will actually insert the new information to the database and return a `True` value if all went well, or an error message if an error occurred.

Likewise, the client application can pass all the information of the invoice to the middle-tier component and let it handle the insertion of the new invoice. This action involves many tables. We may have to update the stock, the customer's balance, possibly a list of best-selling products, and so on. The middle-tier component will take care of these operations for the client. As a result, the development of the client application is greatly simplified. The client will call the `NewInvoice` member

passing the ID of the customer that placed the order, the products and quantities ordered, and (optionally) the discount. Or you may leave it up to the middle tier to calculate the discount based on the total amount, or the items ordered.

The `NewInvoice` method must update multiple tables in a transaction. In other words, it must make sure that all the tables were updated, or none of them. If the program updates the customer's balance, but fails to update the stock of the items ordered (or it updates the stock of a few items only), then the database will be left in an inconsistent state. The program should make sure that either all actions succeed, or they all fail. You can execute transactions from within your VB code, but it's a good idea to pass the responsibility of the transaction to a middle-tier component.

As a side effect, the middle tier forces you to design your application before you actually start coding. If you choose to implement business rules as a middle tier, you must analyze the requirements of the application, implement and debug the middle-tier components, and then start coding the client application. While this is "extra credit" if you're only learning how to program databases with VB, or you write small applications to be used by a workgroup in your company, it's more of a necessity if you're working as a member of a programming team. By designing and implementing the middle tier, you are in effect designing the client application itself, and the work you do in the middle tier will pay off when you start coding the client application.

The middle tier can also save you a good deal of work when you decide to move the application to the Web. Sooner or later, you'll be asked to develop a site for your company. If the middle tier is already in place, you can use its components with a web application. Let me describe a sample component. A client application needs a function to retrieve books based on title keywords and/or author name(s). If you specify which of the search arguments are title keywords and which ones are author names, the operation is quite simple. As I'm sure you know, all electronic bookstores on the Web provide a box where you can enter any keyword and then search the database. The database server must use the keywords intelligently to retrieve the titles you're interested in. If you think about this operation, you'll realize that it's not trivial. Building the appropriate SQL statement to retrieve the desired titles is fairly complicated. Moreover, you may have to revise the search algorithm as the database grows.

The same functionality is required from within both a client application that runs on the desktop and a client application that runs on the Internet (a web page). If you implement a `SearchTitles()` function for the



20 Chapter One

client application, then you must implement the same function in VBScript and use it with your web application. If you decide to change implementation of the function, you must recompile the desktop application, redistribute it, and then change the scripts of the web application accordingly. Sooner or later the same arguments will retrieve different titles on different machines.

If you implement the SearchTitles() function as a middle-tier component, the same functionality will be available to all clients, whether they run on the desktop or the Web. You may wish to extend the search to multiple databases. Even in this extreme case, you will have to revise the code in a single place, the middle tier, and all the clients will be able to search both databases with the existing code. As long as you don't add any new arguments to the SearchTitles() function, the client will keep calling the same old function and be up to date.

It is actually possible to write client applications that never connect to the database and are not even aware that they're clients to a database server. If all the actions against the database take place through the middle tier, then the client's code will be regular VB code and it could not contain any database structures. As you can understand, it's not feasible to expect that you can write a "database application without a database," but the middle tier can handle many of the complicated tasks of accessing the database and greatly simplify the coding of the client application.

The Layers of a Three-Tier Application

The three-tier model breaks the components of the application into three categories, or layers, described below. Figure 1.7 shows a diagram of a three-tier application.

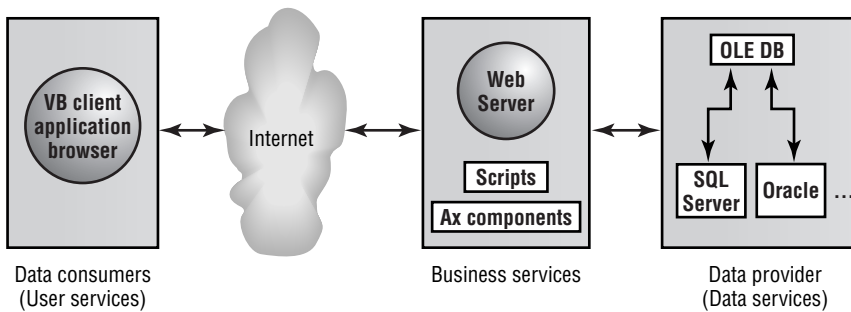


FIGURE 1.7: A three-tier application

Presentation layer This program runs on the client and interacts with the user, primarily presenting information to the user. You will usually develop applications for the presentation layer (unless you're on the business services team), and these applications are frequently called *user services*. By the way, user services are not trivial. They can include advanced data-bound controls and, in many cases, custom data-bound controls. *Data-bound controls* are bound to a field in the database and change value to reflect the field's current value, as the user navigates through the recordset. When a data-bound control is edited, the new value is committed automatically to the database (unless the control is not editable).

Application layer Also known as the *business layer*, this layer contains the logic of the application. It simplifies the client's access to the database by isolating the user services from the database. In addition, you can insert business rules here that have nothing to do with the presentation logic. This layer is designed before you start coding the client application. The components of the application or business layer are frequently called *business services*.

Data layer This layer is the database server, which services requests made by the clients. The requests are usually queries, like "Return all titles published by Sybex in 1999" or "Show the total of all orders placed in the first quarter of 2000 in California." Other requests may update the database by inserting new customers, orders, and so on. The database server must update the database and at the same time protect its integrity (for example, it will refuse to delete a customer if there are invoices issued to that specific customer).

Three-Tier Applications on the Web

The best example of a three-tier application is a web application. Web applications are highly scalable, and two tiers of the application may run on the same computer (the client tier runs on a separate machine, obviously). Even though you may never write applications for the Web, you should understand how web applications interact with viewers.

Figure 1.8 shows a web application that runs in a browser and contacts a web server and a database server to interact with the user. The first tier—the presentation layer—is the browser, which interacts with the



22 Chapter One

user through HTML documents (web pages). A web page may contain controls where the user can enter information and submit it to the server. The web page, therefore, is the equivalent of a VB Form. Where your VB application can read the controls' values the moment they're entered, the values of the controls on a web page must be passed to the server before they can be processed. (It is possible to do some processing on the client, but client-side scripting is beyond the scope of this book).

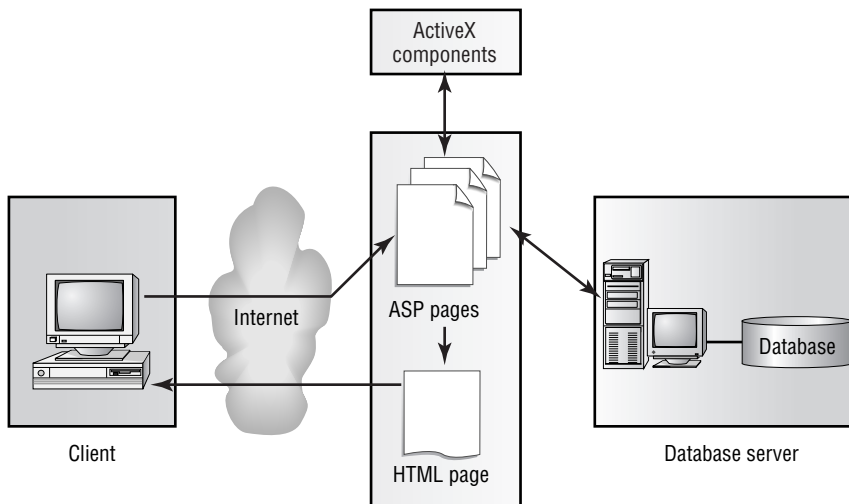


FIGURE 1.8: A web application is a typical example of a three-tier application.

All requests are channeled by the browser to the web server. *Internet Information Server (IIS)* is Microsoft's web server and requires Windows NT or Windows 2000, Server Edition. Most of the examples in this book will work with the Personal Web Server, which comes with Windows 98. IIS is the middle tier (the application layer). The web server's role is to generate HTML documents and send them to the client. If the web server needs to access a database, it must contact a DBMS through an ActiveX component. The programs on the web server are active server pages, written in VBScript.

The DBMS, finally, is the data layer of the application.

Notice that the tiers of a web application need not reside and execute on different machines. The DBMS may be running on the same machine as the web server. For testing purposes, you can run all three tiers on the same computer, but when you deploy the application, you will install the client application on multiple workstations. The web server and the DBMS

are frequently installed on the same machine, but they run as two separate processes. Even though they're on the same computer, the DBMS will authenticate the web server and will not allow it to view information or invoke procedures unless it has *the* appropriate privileges. As the site grows, you may have to use multiple databases and/or multiple web servers.



ACCESS AND CLIENT-SERVER APPLICATIONS

Many of you readers are probably wondering whether you can develop client-server applications with Access. Access is not a database server. When you contact Access and open a table, the entire table is uploaded to the client computer's memory. (If the table is large, it's uploaded in segments, but the processing takes place on the client.) If five users on the network are all accessing the same information, then five copies of the same table(s) are on the clients, plus the original. To describe Access applications, the term *file-based database* is used, but I prefer the (older) term *desktop database*. To use an Access database, you must have Access or compatible software, such as Excel or a Visual Basic application, installed on the client.

One of the most important differences between Access and SQL Server is how they handle concurrency. SQL Server maintains a single copy of the data. Because all clients must go through the DBMS, SQL Server knows when a record is being edited and prevents other users from deleting it or even reading it. Access must compare the changes made on the client to the original and then decide whether other users can access a row. If a user opens a table and selects a row to edit, no other user can edit the same row. This is a nice feature, unless the user gets an important call or goes out to lunch. Then the row will remain locked indefinitely. As the number of users grows, the overhead is overwhelming, and it's time to upsize the database to SQL Server.

Access 2000 can be used for developing client-server applications, but this feature of Access relies on SQL Server technology. Microsoft has released the Microsoft Data Engine (MSDE) component, which is a client-server data engine. MSDE is fully compatible with SQL Server; it's actually based on the same data engine as SQL Server, but it's designed for small workgroups. You can use MSDE to develop client-server applications with Access 2000 (or VB, for that matter) that can "see" SQL Server databases. These applications are fully compatible with SQL Server and will work with SQL Server if you change the connection information.



SQL SERVER

Quite a few of you are familiar with Access, and you may have even developed database applications with it. As I mentioned earlier, however, Access is a desktop database. It can't be scaled up, and it can't accommodate many simultaneous users. To develop real database applications, you should move to SQL Server, which is Microsoft's DBMS. It's highly scalable and you can use it to develop applications for everything from small networks to thousands of users.

Until recently, Microsoft was pushing Access databases with Visual Basic. Now VB6 comes with all the drivers and tools you need to access SQL Server databases, and the next version of VB will probably rely heavily on SQL Server. So, this is an excellent time to move up to SQL Server. The current version of SQL Server (version 7) runs under Windows 98 and can be easily deployed in a small network. Even if you have no prior experience with SQL Server, I urge you to install the Evaluation Edition of SQL Server on your computer and use the same machine for development and as a database server.



NOTE

Nearly all of this book's examples will work on a stand-alone computer running Windows 98, but I recommend using Windows NT or Windows 2000.

There are two ways to use SQL Server: as a powerful substitute for Access or as a powerful DBMS (which is what SQL Server is). You can write an application that works with Access, then change its connection to the same database on SQL Server, and the application will work. I know some programmers who upsized their Access database to SQL Server and then changed their DAO-based VB code to work with SQL Server. By the way, converting an application based on DAO to work with ADO is not trivial, but if you write applications based on ADO, you can manipulate Access and SQL Server databases with nearly the same code.

SQL Server has a few unique features that you can't ignore. To begin with, it has its own programming language, T-SQL. T-SQL is an extension of SQL and it's so powerful that it can do just about everything you can do with VB. T-SQL has no user interface, but it supports many data-manipulation functions (similar to the functions of VB) and flow-control statements. It can also access the tables of a database through SQL. In essence, T-SQL combines the power of SQL with the structure of more

traditional programming languages. If you don't care about a user interface, you can use T-SQL to implement all of the operations you'd normally code in VB. The advantage of T-SQL is that it's executed on the server and can manipulate tables locally. To do the same with VB, you'd have to move information from the server to the client and process it there. Stored procedures are faster than the equivalent VB code and they standardize client applications, since all clients will call the same procedure to carry out a task.

In effect, it's quite acceptable to implement business rules as stored procedures. Chapters 4 and 5 discuss how to take advantage of stored procedures from within your VB code. I think stored procedures are one of the best reasons to switch from Access databases to SQL Server. A good VB programmer implements the basic operations of the application as functions and calls them from within the application. Practically, you can't implement every data-access operation as a stored procedure, and I urge you to do this. Stored procedures become part of the database and can be used by multiple applications, not just the client application.



WRITE BETTER CLIENT APPLICATIONS WITH STORED PROCEDURES

If you implement the NewInvoice stored procedure to add new invoices to a database, then you can call this stored procedure from within any VB application that needs to add invoices to the database. If you implement the same operation as a method of a middle-tier component, then you can call this method from within any application—including the Office applications. Because middle-tier components are implemented as Classes, they can be called by any COM-enabled application. In simple terms, this means that every programming language that supports the CreateObject() function can call the methods of the middle-tier component. You will see how to create a script to add orders to the database. If you distribute the application, users don't have to go through the visible interface of the application to add new invoices. They can write a short script to automate the process.

SQL Server also uses triggers. A *trigger* is a special stored procedure that is executed when certain actions take place. For example, you can write a procedure to keep track of who has deleted a record and when. Triggers are added to individual tables and can be invoked by three different actions: insertions, deletions, and updates. We'll discuss stored procedures and

triggers in Chapter 4, and you'll see how you can simplify your VB code by implementing certain operations as stored procedures.

SQL Server Tools

Many of you may not be familiar with SQL Server, so this introduces you to its basic tools. If you don't have access to SQL Server on your company's network, you can install the desktop version on a local machine and use it as a development platform as well. The following section describes how to install SQL Server and related tools on your computer. If you haven't purchased SQL Server yet, you can use the Evaluation Edition on the companion CD, but it will expire three months after installation. For more information on ordering SQL Server 7, visit the Microsoft website at www.microsoft.com/sql.



NOTE

Although SQL Server 2000 is now available, SQL Server 7 is still used more often and so is referenced throughout this book. If you're using SQL Server 2000, you may find new or optimized features not mentioned here, but the functionality of the examples should be the same.

Installation

Installing SQL Server is fairly straightforward. Of the available installation options, select the Desktop version. To keep it simple, install SQL Server on the same machine you will use to develop your applications in the course of reading this book. This is a client-server configuration, as SQL Server is a separate program that must be running in order to service client requests. Whether it's running on the same or a different machine, it makes no difference to your application.

If you plan to install and configure SQL Server on a local area network, please consult the product documentation. This is the job of the database's administrator (DBA), who is responsible for maintaining the database as well. SQL Server is nothing like Access, and you really need a DBA to take care of the day-to-day operations.

SQL Server 7 runs under both Windows 95/98 and Windows NT/2000. So, you can really learn how to develop database applications with Visual Basic and SQL Server with a typical desktop system. The Server version of SQL Server that runs under Windows 2000 supports

additional features, of course, like full-text search support, replication, and more, but these features are not discussed in this book. You can also install the *Microsoft English Query*, a component that allows you to query the database with English-language statements like “How many orders were placed in the 1999?” or “Show the titles of all books written by T. S. Eliot.” The English Query is not a ready-to-use utility, but an environment that must be customized for each database. It’s an advanced topic and has very little to do with database programming, so it’s not covered in this book. However, it’s a very interesting program, and you should probably take a look at the sample application after you have mastered SQL and database programming.

Once SQL Server has been installed, a new command is added to the Programs menu: SQL Server 7. This command leads to another menu with a few options, including Microsoft SQL Server 7, which leads to a submenu listing SQL Server’s tools. The most important tools, which are also relevant to this book’s contents, are presented briefly in the following sections.

SQL Server Service Manager

This tool allows you to start and stop SQL Server. To start SQL Server, select Start > Programs > SQL Server 7.0 > Microsoft SQL Server 7.0 > Service Manager, which opens a window where you can start and stop SQL Server. Select the MSSQLServer service in the services box and then click Start. If you’d rather have SQL Server autostart every time you turn on your computer, check the option “Auto-start Service when OS starts.”

When SQL Server is running, a small icon with a green arrow is added to the system tray. If you attempt to connect to SQL Server from within a client application while SQL Server is not running, you will get an error message to the effect that there’s a problem with your network. At this point you must stop the application, start SQL Server through the Service Manager, and then restart the VB application.

Enterprise Manager

The Enterprise Manager, shown in Figure 1.9, is a visual tool that allows you to view and edit all the objects of SQL Server. This is where you create new databases, edit tables, create stored procedures, and so on. You can also open a table and edit it, but the corresponding tools are not nearly as user-friendly as the ones that come with Access. SQL Server databases shouldn’t be manipulated directly. Only the DBA should open tables and examine or edit their contents.



28 Chapter One

Visual Basic includes several visual database tools (discussed in Chapter 6) that allow you to view the structure of your databases, create and edit tables, create and debug stored procedures, and more. Much of what you can do with Enterprise Manager can be done with the visual database tools, except for adding new users, setting user rights, and similar operations. Again, these tasks are the responsibility of the DBA. You will see how to set up a user's profile so that the specific user can execute stored procedures only and other simple tasks in a later chapter. This topic is discussed in Chapter 4, "Transact-SQL."

Expand the folder with the name of the server (TOSHIBA in Figure 1.9) in the left pane, and you will see five folders.

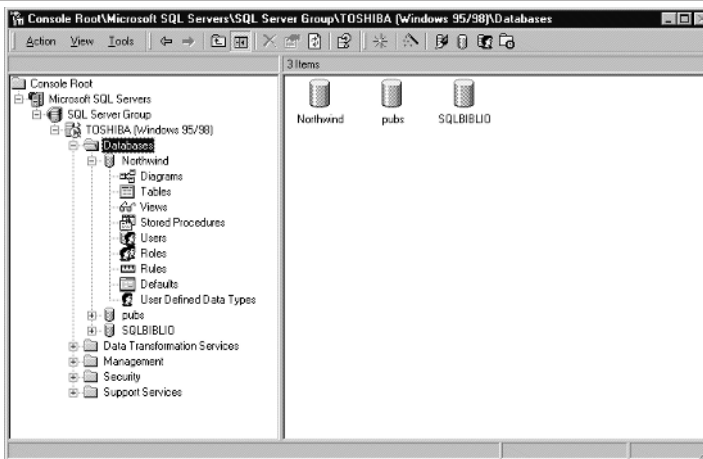


FIGURE 1.9: The SQL Server Enterprise Manager window

Databases

This folder contains a subfolder for each database. If you select a database here, you will see a list of objects, described below, that are specific to that database.

Diagrams A diagram is a picture of the database's structure, similar to the one shown in Figure 1.10. You can manipulate the very structure of the database from within this window, which shows how the various tables relate to each other. You can add new relationships, set their properties, add constraints for the various fields (for example, specify that certain fields must be positive), enforce referential integrity, and so on. Don't

Database Access: Architectures and Technologies 29

worry if you are not familiar with these terms; they are discussed in detail in the first few chapters of the book.

To create a new database diagram, right-click the right window and select New diagram from the shortcut menu. A Wizard will prompt you to select the tables to include in the diagram, and then it will generate the diagram by extracting the information it needs from the database itself. You will find more information on creating tables and diagrams in Chapter 2.

Tables A *table* consists of rows and columns where we store information. Databases have many tables and each table has a specific structure. You can edit the columns of each table through the Design window, shown in Figure 1.11. To open the Design window of a table, right-click the table's name and select Design from shortcut menu.

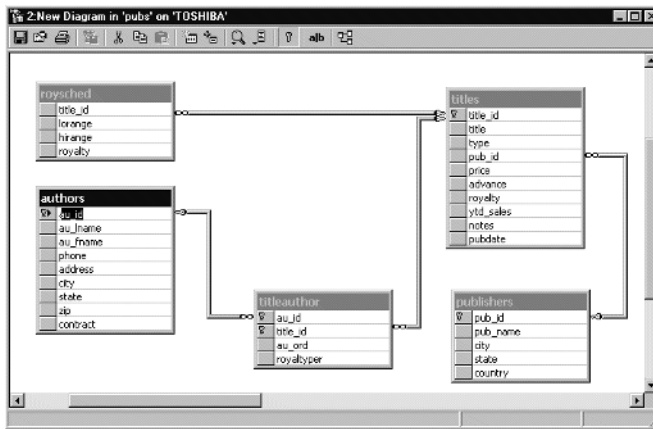


FIGURE 1.10: A database diagram shows the structure of its tables and the relationships between them.

Column Name	Datatype	Length	Precision	Scale	Allow Nulls	Default Value	Identity	Identity Seed	Identity Increment
title_id	int (varchar)	5	0	0	<input type="checkbox"/>		<input checked="" type="checkbox"/>		
title	varchar	80	0	0	<input type="checkbox"/>		<input type="checkbox"/>		
type	char	12	0	0	<input type="checkbox"/>	(UNDECIDED)	<input type="checkbox"/>		
pub_id	char	4	0	0	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>		
price	money	8	19	4	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>		
advance	money	8	19	4	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>		
royalty	int	4	10	0	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>		
ytd_sales	int	4	10	0	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>		
notes	varchar	200	0	0	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>		
pubdate	datetime	8	0	0	<input type="checkbox"/>	(getdate())	<input type="checkbox"/>		

FIGURE 1.11: The Design window of the titles table

30 Chapter One

Views A *view* is a section of a table, or a combination of multiple tables, and contains specific information needed by a client. If the Customers table contains salary information, you probably don't want every application to retrieve this information. You can define a view on the table that contains all the columns except for the salary-related ones. As far as the client application is concerned, the view is just another table. SQL Server's views are based on SQL statements and they're equivalent to Access queries.

Most views are editable (a view that contains totals, for example, can't be edited). To open a view, select Views in the left pane of the Enterprise Manager, then right-click the desired view's name in the right pane and select Return All Rows from the shortcut menu. The view's rows will appear on a grid, where you can edit their fields (if the view is updateable). To refresh the view, click the button with the exclamation mark in the window's toolbar.

Stored Procedures A *stored procedure* is the equivalent of a VB function, only stored procedures are written in T-SQL and they're executed on the server. In this folder, you see the list of stored procedures attached to the database and their definitions. You can create new ones as well, but you can't debug them. To edit and debug your stored procedures, use either the Query Analyzer (discussed in the next section) or the T-SQL Debugger, a tool that comes with VB. Actually, the Stored Procedure Properties window, which will appear if you double-click a procedure's name, contains the definition of the procedure and a button named Check Syntax. If you click this button, the Enterprise Manager will verify the syntax of the stored procedure's definition. It points out the first mistake in the T-SQL code, so it doesn't really qualify as a debugging tool.

Users In this folder, you can review the users authorized to view and/or edit the selected database and add new users. By default, each database has two users: the owner of the database (user *dbo*) and a user with seriously limited privileges (user *guest*). To view the rights of a user, double-click their name. On that user's Properties dialog box, you can assign one or more *roles* to the selected user (instead of setting properties for individual users, you create roles and then assign these roles to the

users). If you click Permissions, you will see the user's permissions for every object in the database, as shown in Figure 1.12. It's a good idea to create a user called *application* (or something similar) and use this ID to connect to the database from within your application. This user will impersonate your application, and you can give this user all the rights your application needs.

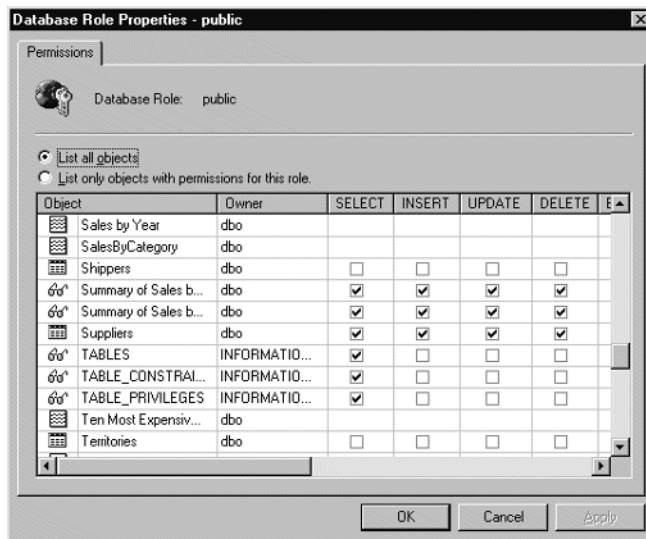


FIGURE 1.12: Setting user permissions for the various objects of a database

Roles When you select the Roles item in the right pane, you will see a list with the existing roles. A role is nothing more than a user profile. If multiple users must have common privileges, create a new role, set permissions to this role, and then use it to specify the permissions of individual users.

Rules SQL Server allows you to specify rules for the values of individual fields of a table. These rules are called *CHECK constraints* and they are specified from within the Database Diagram window. There's no reason to use this window to specify rules, but it's included for compatibility reasons.

Defaults Here you can define the default values for any field. The default values are used when no value is supplied by the user, or the application, for the specific field. It is simpler to specify defaults during the design of the table, than to provide



the code that checks the user-supplied value and supplies a default value if the user hasn't entered a value for a field.

User-Defined Data Types This is where the user-defined data types (UDTs) are specified. SQL Server doesn't allow the creation of arbitrary data structures like Visual Basic does. A UDT is based on one of the existing data types, but you can specify a length (for character and binary types) and, optionally, a default value. For example, you can create a UDT, name it ZCODE, and set its type to CHAR and length to five. This is a shorthand notation, rather than a custom data type. UDTs are useful when you allow developers to create their own tables. You can create data types like FNAME, LNAME, and so on, to make sure that all fields that store names, in all tables, have the same length. When you change the definition of a UDT, the table(s) change accordingly without any action on your part.

Data Transformation Services (DTS)

This folder contains the utilities for importing data into SQL Server and exporting data out of SQL Server. The DTS component of SQL Server allows you to import/export data and at the same time transform it. In Chapter 2, you will see how to use the DTS component to upsize the Biblio sample database, which comes with both Access and Visual Basic.

Management

This folder contains the tools for managing databases. The most important tool is the Backup tool, which allows you to back up a database and schedule backup jobs. These tools are also meant for the DBA, and we are not going to use them in this book.

Security

Here's where the DBA creates new logins and assigns roles to users. We are not going to use these tools in this book.

Support Services

This is where you configure two of SQL Server's support services: the Distributed Transaction Coordinator and SQL Server Mail. The *Distributed Transaction Coordinator* is a tool for managing transactions that span

across multiple servers. We will discuss transactions in detail beginning in Chapter 4, but we won't get into transactions across multiple servers.

The *SQL Server Mail* service allows you to create mail messages from within SQL Server. These messages can be scheduled to be created and transmitted automatically and are used to notify the database administrator about the success or failure of a task. You can attach log files and exception files to the message.



The Query Analyzer

If there's one tool you must learn well, this is it. The *Query Analyzer* is where you can execute SQL statements, batches, and stored procedures against a database. To start the Query Analyzer, select Start > Programs > SQL Server 7.0 > Microsoft SQL Server 7.0 > Query Analyzer. The Query Analyzer uses an MDI interface, and you can open multiple windows, in which you can execute different SQL statements or stored procedures.

If you enter an SQL statement in the Query Analyzer window and click Execute (the button with the green arrow on the toolbar), the window will split into two panes; the result of the query will appear in the lower pane—the Results pane—as shown in Figure 1.13. The statement will be executed against the database selected in the DB box at the top of the window, so make sure you've selected the appropriate database before you execute an SQL statement for the first time. You can save the current statement to a text file with the File > Save As command and open it later with the File > Open command.

In addition to SQL statements, you can execute batches written in T-SQL. A *batch* is a collection of SQL and T-SQL statements. For example, you can enter multiple SQL statements and separate them with a GO statement. Each time a GO statement is reached, the Query Analyzer executes all the statements from the beginning of the file, or the previous GO statement. All the results will appear in the Results pane.



NOTE

SQL statements and batches are stored in text files with the extension `.SQL`. All of the SQL statements and stored procedures presented in this book can be found in a separate SQL file, each under the corresponding chapter's folder on the companion CD-ROM.

34 Chapter One

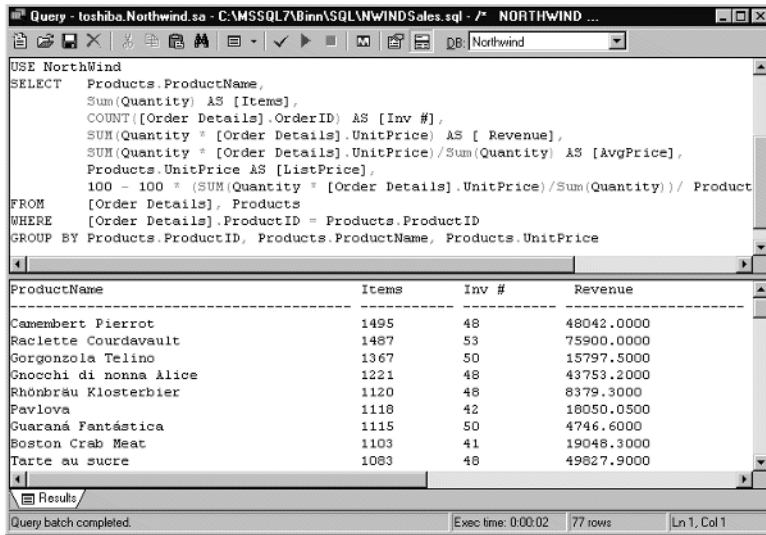
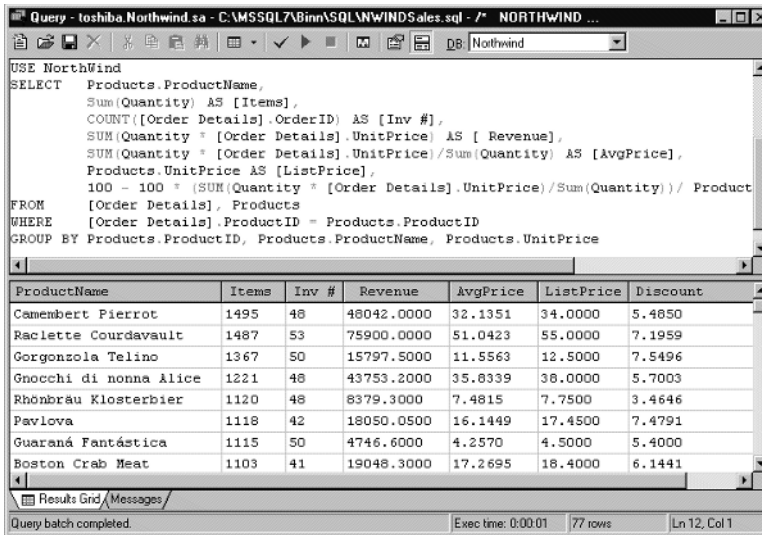


FIGURE 1.13: Executing SQL statements with the Query Analyzer

By default, the Query Analyzer displays the row output produced by SQL Server: the results and any messages indicating the success or failure of the operation. Most people prefer the Grid view, which is shown in Figure 1.14. To activate this view, select Results in Grid from the Query menu. The advantage of this view is that you can change the width of the columns. The grid on the Results Grid tab contains the results of the query, and the messages returned by SQL Server are displayed on the Messages tab.



```

USE NorthWind
SELECT
    Products.ProductName,
    Sum(Quantity) AS [Items],
    COUNT([Order Details].OrderID) AS [Inv #],
    SUM(Quantity * [Order Details].UnitPrice) AS [Revenue],
    SUM(Quantity * [Order Details].UnitPrice) / Sum(Quantity) AS [AvgPrice],
    Products.UnitPrice AS [ListPrice],
    100 - 100 * (SUM(Quantity * [Order Details].UnitPrice) / Sum(Quantity)) / Product
FROM
    [Order Details], Products
WHERE
    [Order Details].ProductID = Products.ProductID
GROUP BY Products.ProductID, Products.ProductName, Products.UnitPrice

```

ProductName	Items	Inv #	Revenue	AvgPrice	ListPrice	Discount
Camembert Pierrot	1495	48	48042.0000	32.1351	34.0000	5.4850
Raclette Courdavault	1487	53	75900.0000	51.0423	55.0000	7.1959
Gorgonzola Telino	1367	50	15797.5000	11.5563	12.5000	7.5496
Gnocchi di nonna Alice	1221	48	43753.2000	35.8339	38.0000	5.7003
Rhonbrau Klosterbier	1120	48	8379.3000	7.4815	7.7500	3.4646
Pavlova	1118	42	18050.0500	16.1449	17.4500	7.4791
Guaraná Fantástica	1115	50	4746.6000	4.2570	4.5000	5.4000
Boston Crab Meat	1103	41	19048.3000	17.2695	18.4000	6.1441

Results Grid / Messages /
Query batch completed. Exec time: 0:00:01 77 rows Ln 12, Col 1

FIGURE 1.14: The Query Analyzer's Grid view

SUMMARY

This chapter was a very broad indeed. It touched a lot of topics, and it probably raised quite a few questions. The following chapters elaborate on all the topics discussed here. Starting with the next chapter, you'll learn how to design databases and how to manipulate them with SQL. Then, you'll see how to use ADO to write database applications that are almost independent of the DBMS you use. Nearly all of this book's application will work equally well with SQL Server and Access databases.



