CHAPTER 22

# Developing Multiuser and Enterprise Applications

IN THIS CHAPTER

## Why This Chapter Is Important

Many people forge right into the application development process with little worry about the scalability of the application. Even a simple application that begins as a single-user application can develop into a multiuser or enterprise-wide application. Unfortunately, the techniques you can get away with in the single-user application can wreak havoc in a network or client/server environment. It is therefore necessary to think about the future when you design any application. Although the initial development process might be more complex, if written properly, the application will survive any growth that it experiences. This chapter focuses on writing applications that transition easily from the single-user environment through the enterprise client/server environment.

## Designing Your Application with Multiuser Issues in Mind

When you develop applications that multiple users will access over the network, you must make sure they effectively handle sharing data and other application objects. Many options are available for developers when they design multiuser applications, and this chapter covers the pros and cons of these options.

Multiuser issues revolve around locking data; they include deciding where to store database objects, when to lock data, and how much data to lock. In a multiuser environment, having several users simultaneously trying to modify the same data can cause conflicts. As a developer, you need to handle these conflicts. Otherwise, your users will experience unexplainable errors.

## Multiuser Design Strategies

There are many methods for handling concurrent access to data and other application objects by multiple users; each one offers both solutions and limitations. It's important to select the best solution for your particular environment.

### Strategies for Installing Access

There are two strategies for installing Access:

- Run Access from a file server across a network.
- Run a separate copy of Access on each workstation.

The advantages of running Access from a file server are that it

- Allows for central administration of the Access software
- Potentially reduces the licensing requirements
- Allows Access applications to be installed on diskless workstations
- Reduces hard disk requirements

File server installations also have *serious* drawbacks, including the following:

- Every time the user launches an Access application, the Access EXE, DLLs, and any other files required to run Access are *all* sent over the network wire to the local machine. Obviously, this generates a significant volume of network traffic.
- Performance is generally degraded to unacceptable levels.

Because the disadvantages of running Access from a file server are so pronounced, I *strongly* recommend that you install Access, or at least the runtime engine, on each user's machine.

### Strategies for Installing Your Application

Just as there are different strategies for installing Access, there are also various strategies for installing your application, such as the following:

- Install both the application and data on a file server.
- Install the data on the file server and the application on each workstation.
- Install the application and the data on a machine running Windows 2003 Terminal Services.

In other words, after you have created an application, you can place the entire application on the network, which means that all the tables, queries, forms, reports, macros, and modules that make up the system reside on the file server. Although this method of shared access keeps everything in the same place, you will see many advantages to placing only the database's data tables on the file server. The remaining objects are placed in a database on each user's machine, and each local application database is linked to the tables on the network. In this way, users share data but not the rest of the application objects.

The advantages of doing this are as follows:

- Because each user has a copy of the local database objects, load time and network traffic are both reduced.
- You can easily back up data without having to back up the rest of the application objects.
- When redistributing new versions of the application, you don't need to worry about overwriting the application's data.
- You can design multiple applications to use the same centrally located data.
- Users can add their own objects (such as their own queries) to their local copies of the database.

In addition to storing the queries, forms, reports, macros, and modules that make up the application in a local database, I also recommend that you store the following objects in each local database:

- Temporary tables
- Static tables
- Semistatic tables

Temporary tables should be stored in the database that's on each workstation because, if two users are performing operations that build the same temporary tables, you don't want one user's process to interfere with the other user's process. You can eliminate the potential conflict of one user's temporary tables overwriting the other's by storing all temporary tables in each user's local copy of the database.

You should also place static lookup tables, such as state tables, on each workstation. Because the data doesn't change, maintenance isn't an issue. The benefit is that Access doesn't need to pull that data over the network each time the application needs it.

Semistatic tables—tables that are rarely updated—can also be placed on the local machine. As with static tables, having these tables in a local database means reduced network traffic and better performance, not only for the user needing the data, but also for anyone sharing the same network wire.

The configuration described throughout this section is illustrated in Figure 22.1.

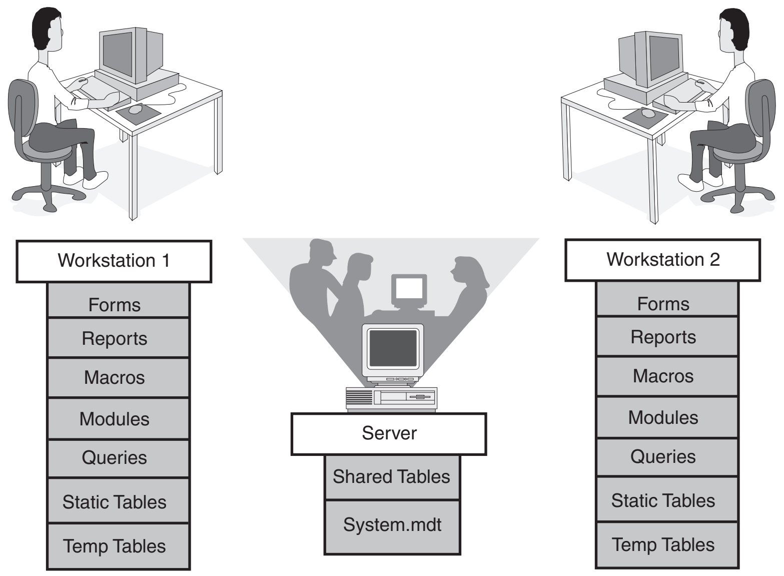

FIGURE 22.1 An example of a configuration with database objects split, storing temporary and static tables locally and shared tables remotely (on the file server).

Terminal Services has emerged as a viable alternative for deployment of an Access application. It addresses both bandwidth and centralization issues. With this option, a Windows 2003 machine runs the Windows 2003 Terminal Services. Client machines then access the server machine using the Terminal Server Client Utility. In this scenario, Access, your application, and the data that it accesses are all installed on the Windows 2003 Server machine. All other machines access the application via user sessions created on the server machine. Keystrokes and mouse events are sent from the client machines to the server machine. The resulting screen image is returned to the client machine. This configuration addresses many of the problems inherent in the two other solutions.

## The Basics of Linking to External Data

Chapter 20, "Using External Data," covers linking to external data, including data not stored in another Access database. Two options are available to you:

- Design the databases separately from the start.
- Include all objects in one database and then split them manually when you're ready to distribute your application.

Chapter 20 covers these two options in detail.

**CAUTION**

Be aware that when you're distributing an application using linked tables, you must write code to make sure the data tables can be located from each application database on the network. The reason is that Access hard-codes the location of linked tables into the application database. If each user has the same path to the file server, this isn't a problem. However, if the path to the file server varies, you need to write a routine that makes sure the tables can be successfully relinked. If they can't, the routine prompts the user for the data's location. Chapter 20 covers this routine.

## Understanding Access's Locking Mechanisms

Although the preceding tips for designing network applications reduce network traffic, they in no way reduce locking conflicts. To protect shared data, Access locks either a record or a page of data as the user edits a record. In this way, multiple users can read the data, but only one user can make changes to it. Data can be locked through a form or through a recordset that isn't bound to a form.

Here are the methods of locking for an Access application:

- Record locking
- Page locking
- Table and Recordset locking
- Opening an entire database with Exclusive Access

With Record locking, the Access Database Engine locks only the record that the user is editing. With Page locking, the Access Database Engine locks the 4K page with the record being edited. On the other hand, in Table and Recordset locking, the Access Database Engine locks the entire table or recordset with the record being edited. With Database locking, the Access Database Engine locks the entire database, unless the user opening the database has opened it for read-only access. In that case, other users can also open the database for read-only access.

It's important to note that the locking scheme you adhere to depends on the source providing the data. If you're using client/server data, you inherit the locking scheme of the particular back end you're using. If you're manipulating Indexed Sequential Access Method (ISAM) data over a network, you get the type of data locking that the particular ISAM database supports. For example, if you're working with a FoxPro database, you can use Record locking or any other locking scheme that FoxPro supports.

**NOTE**

Multiuser development and multiuser issues are covered in extensive detail in *Alison Balter's Mastering Access 2002 Enterprise Development*.

## Understanding the Client/Server Model

Now that you understand the basics of using Access in a multiuser environment, I am going to take things a step further by discussing client/server applications. One of the hot computing terms of the 21st century, *client/server* refers to distributed processing of information. A client/server model involves the storage of data on database servers dedicated to the tasks of processing data and storing it.

The client/server model introduces a separation of functionalities. The *client*, or front end, is responsible for presenting the data and doing some processing. The *server*, or back end, is responsible for storing, protecting, and performing the bulk of the data processing.

With its tools that assist in the rapid development of queries, forms, and reports, Access provides an excellent front end for the presentation of back-end data.

For years, most information professionals have worked with traditional programming languages to process and maintain data integrity in the application. This means that data validation rules must be embedded in the programming code. Furthermore, these types of applications are record-oriented; that is, all records are read into memory and processed. This scenario has several drawbacks:

- If the underlying data structure changes, every application that uses the data structure must be changed.
- Data validation rules must be placed in *every* application that accesses a data table.
- Presentation, processing, and storage are handled by one program.
- Record-oriented processing results in an extraordinary amount of unnecessary network traffic.

## Deciding Whether to Use the Client/Server Model

Client/server technology was not as necessary when there was a clear delineation between mainframe applications and personal computer applications. Today, the line of demarcation has blurred. Personal computer applications are taking over many applications that had been relegated to mainframe computers in the past. The problem is that users are still very limited by the bandwidth of network communications. This is one place where client/server technology can really help.

However, many developers are confused about what client/server architecture really is. Some mistakenly believe that an Access ACCDB database file stored on a file server acts as a database server. This is not the case. (In fact, I have participated in many debates in which other developers have insisted that Access itself is a database server application. Well, it's not.) Access is a front-end application that can process data stored on a back end. In this scenario, the Access application runs on the client machine accessing data stored on a database server running software such as Microsoft SQL Server. Access does an excellent job acting as the client-side, front-end software in this scenario. The confusion lies in Access's capability to act as a database server.

The difference lies in the way that data is retrieved when Access is acting as the front end to a database server versus when the data is stored in an Access ACCDB file. Suppose that you have a table with 500,000 records. A user runs a query based on the 500,000-record table stored in an Access database on a file server. Suppose that the user wants to see a list of all the Californians who make more than $75,000 per year. With the data stored on the file server in the Access ACCDB file format, all records would be sent over the network to the workstation, and the query would be performed on the workstation (see Figure 22.2). This results in significant network traffic.

On the other hand, assume that these 500,000 records were stored on a database server such as Microsoft SQL Server. If the user runs the same query, only the names of the Californians who make more than $75,000 per year would be sent over the network. In this scenario, only the specific fields requested would be retrieved (see Figure 22.3).

What does this mean to you? When should you become concerned with client/server technology and what it can offer you? The following sections present some guidelines as to why you might want to upsize from an Access back end to a SQL Server back end.

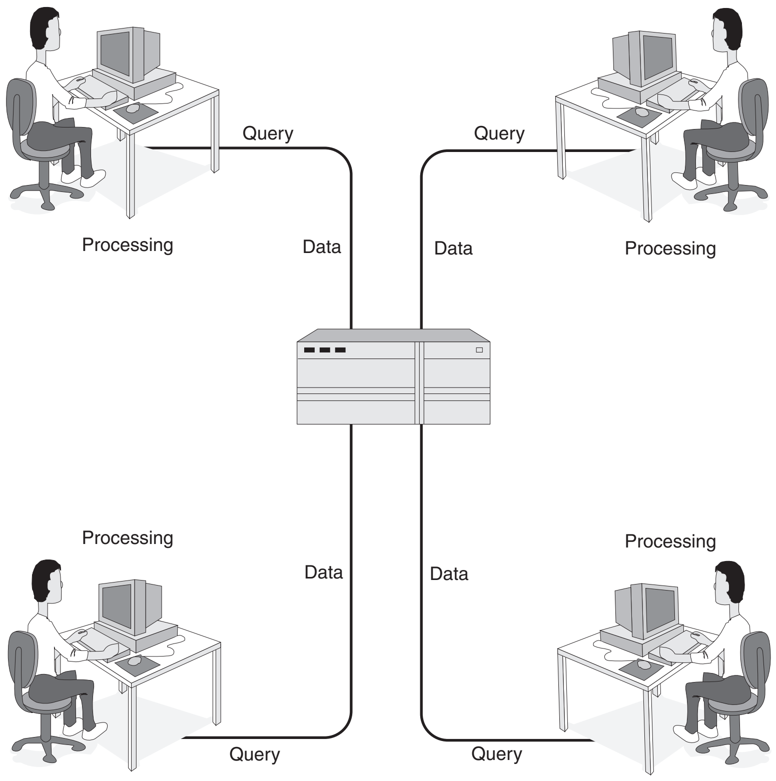

FIGURE 22.2 Access as a front end using data stored in an Access database.

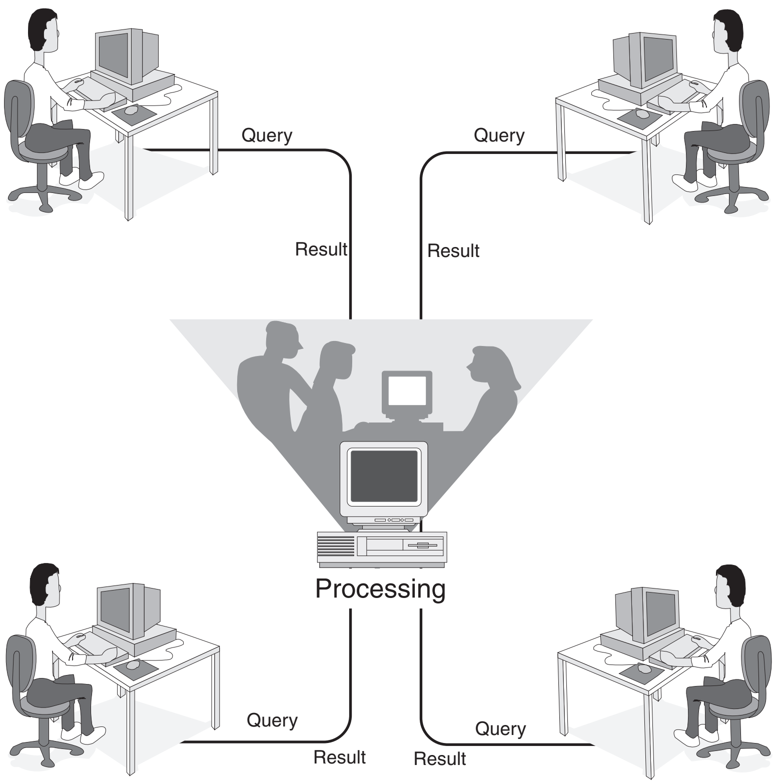

FIGURE 22.3 Access as a front end using a true back end.

## Dealing with a Large Volume of Data

As the volume of data in your Access database increases, you will probably notice degradation in performance. Many people say that 100MB is the magical number for the maximum size of an Access database, but many back-end database servers can handle databases containing multiple gigabytes of data. Although a maximum size of 100MB for an Access database is a good general guideline, it is *not* a hard-and-fast rule. You might find that the need to upsize occurs when your database is significantly larger or smaller than 100MB. The magic number for you depends on all the factors discussed in the following sections, as well as on how many tables are included in the database. Generally, Access performs better with large volumes of data stored in a single table rather than in multiple tables.

## Dealing with a Large Number of Concurrent Users

Just as a large volume of data can be a problem, so can a large number of concurrent users. In fact, more than 10 users concurrently accessing an Access database can degrade performance. As with the amount of data, this is not a magical number. I have seen applications with fewer than 10 users where performance is awful, and I have seen applications with significantly more than 10 users where performance is acceptable. Performance often depends on how the application is designed, as well as what tasks the users are performing.

## Demanding Faster Performance

Certain applications demand better performance than other applications. An Online Transaction Processing (OLTP) system generally requires significantly better performance than a Decision Support System (DSS), for example. Suppose that 100 users are simultaneously taking phone orders. It would not be appropriate for the users of the system to ask their customers to wait 15 seconds between entering each item that is ordered. On the other hand, asking users to wait 60 seconds to process a management report that users run once each month is not a lot to ask (although many will still complain about the wait).

Most back-end database servers can use multithreaded operating systems with multiple processors to handle large volumes of user demand; Access cannot.

## Handling Increased Network Traffic

As a file server in an organization experiences increasing demands, the Access application simply might exacerbate an already growing problem. If the application data is moved to a database server, the overall reduced demands on the network might provide all users on the network with better performance, regardless of whether they are using the Access application.

Probably one of the most exaggerated situations I have seen is one in which all the workstations were diskless. Windows and all application software were installed on a file server. All the users were concurrently loading Microsoft Word, Microsoft Excel, and Microsoft PowerPoint over the network. In addition, they had large Access applications with many database objects and large volumes of data. This was all stored on the file server as well. Needless to say, performance was abysmal. You can't expect an already overloaded file server to handle sending large volumes of data over a small bandwidth. The benefits offered by client/server technology can help alleviate this problem.

## Implementing Backup and Recovery

The backup and recovery options offered with an Access ACCDB database stored on a file server simply do not rival the options for backup and recovery on a database server. Any database server worth its salt sports very powerful uninterruptible power supplies (UPSs). Many have hot-swappable disk drives with disk mirroring, disk duplexing, or disk striping with parity (RAID Level 5). With disk mirroring and duplexing, data can be written to multiple drives at one time, providing instantaneous backups. Furthermore, some database server tape backup software enables backups to be completed while users are accessing the system. Many offer automatic transaction logging. All these options mean less chance of data loss or downtime. With certain applications, this type of backup and recovery is overkill. With other applications, it is imperative. Although some of what back ends have to offer in backup and recovery can be mimicked by using code and replication, it is nearly impossible to get the same level of protection from an Access database stored on a file server that you can get from a database stored on a database server.

### Focusing on Security

Access offers what can be considered the best security for a desktop database. However, it cannot compare with the security provided by most database servers. Database server security often works in conjunction with the network operating system. This is the case, for example, with Microsoft SQL Server 2005 and Windows Server 2003 Enterprise. The user is given no direct rights to the physical database file; it can be accessed only via an Open Database Connectivity (ODBC) data source or an ActiveX Data Objects (ADO) connection. Remember that no matter how much security you place on an Access database, a user can still see or even delete the entire ACCDB file from the network disk.

Offering protection from this potential problem, and others, on a database server is easy. Furthermore, many back-end application database server products offer field-level security not offered within an Access ACCDB file. Finally, many back ends offer integrated security with one logon for both the network and the database.

### Sharing Data Among Multiple Front-End Tools

The Access ACCDB file format is proprietary. Very few other products can read data stored in the Access database format. With a back-end database server that supports ODBC, front-end applications can be written in a variety of front-end application software, all concurrently using the same back-end data.

### Understanding What It All Means

You must evaluate the specific environment in which your application will run:

- How many users are there?
- How much data exists?
- What is the network traffic already like?
- What type of performance is required?
- How disastrous is downtime?
- How sensitive is the data?
- What other applications will use the data?

After you answer these and other questions, you can begin to decide whether the benefits of the client/server architecture outweigh the costs involved.

The good news is that it is not an all-or-none decision. Various options are available for client/server applications using Access as a front end. Furthermore, if you design your application with upsizing in mind, moving to client/server technology will not require you to throw out what you have done and start again. In fact, Microsoft provides an upsizing wizard that makes upsizing to a SQL Server database a relatively painless process. How painless depends on numerous factors, including how complex your queries are, whether your queries include Visual Basic for Applications (VBA) functions, and other factors that are covered later in this chapter, and in detail in *Alison Balter's Mastering Access 2002 Enterprise Development*.

## Understanding the Roles That Access Plays in the Application Design Model

This section takes a look at the many different roles that Access can take in an application design.

### The Front End and Back End as Access ACCDB Files

Earlier in this book, you learned about using Access as both the front end and the back end. The Access database is not acting as a true back end because it is not doing processing. Figure 22.4 shows the architecture in this scenario. The Access application resides on the workstation. Access uses the Access Database Engine to communicate with data stored in an Access ACCDB database file stored on the file server.

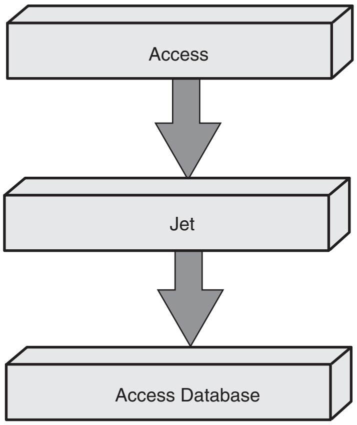

FIGURE 22.4 Access as a front end using an ACCDB file for data storage.

### The Front End as an ACCDB File Using Links to Communicate to a Back End

In the second scenario, you can link the back-end tables to the front-end application database (`.ACCDB`). The process of linking to back-end tables is almost identical to that of linking to tables in other Access databases or to external tables stored in FoxPro, or other database formats. You can also treat the linked tables like any other linked tables. Access uses ODBC to communicate with the back-end tables (see Figure 22.5). Your application sends an Access SQL statement to the Access Database Engine, which translates the statement into ODBC SQL. The Access Database Engine sends this ODBC SQL statement to the ODBC Manager, which locates the correct ODBC driver and passes it the ODBC SQL statement. Supplied by the back-end vendor, the driver translates the statement into the back end's specific dialect. The ODBC Manager sends this now back-end–specific query to the SQL server and to the appropriate database. Although this may seem cumbersome, a properly designed Access front end accessing data stored in a SQL Server database is quite efficient. I have proven this over and over again with enterprise-wide applications written in Microsoft Access.

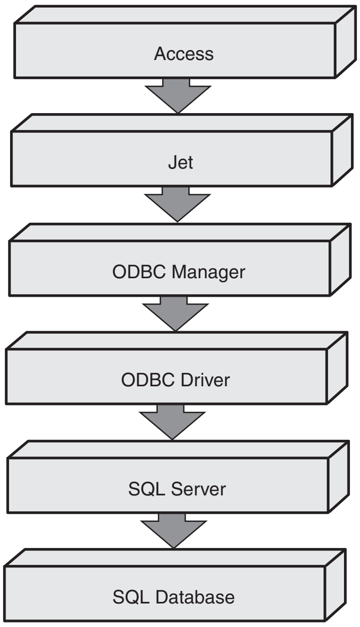

FIGURE 22.5 Access as a front end using links to back-end tables.

### The Front End Using SQL Pass-Through to Communicate to a Back End

If a particular query is running inefficiently, you may want to bypass ODBC and go directly against SQL server. Here are a few reasons why a SQL pass-through query may be the best option available in specific situations:

- Access SQL might not support some operation that the native query language of the back end supports.
- Either the Access Database Engine or the ODBC driver produces a SQL statement that is not optimized for the back end.
- You want a process performed in its entirety on the back end.

As an alternative, you can execute a pass-through query written in the syntax specific to the back-end database server. Although the query does pass through the Access Database Engine, the Access Database Engine does not perform any translation on the query. Neither does ODBC. The ODBC Manager sends the query to the ODBC driver, which passes the query to the back end without performing any translation. In other words, exactly what was sent from Access is what is received by the SQL database. Figure 22.6 illustrates this scenario. Notice that the Access Database Engine, the ODBC Manager, and the ODBC driver are not eliminated entirely. They are still there, but they have much less impact on the process than they do with attached tables.

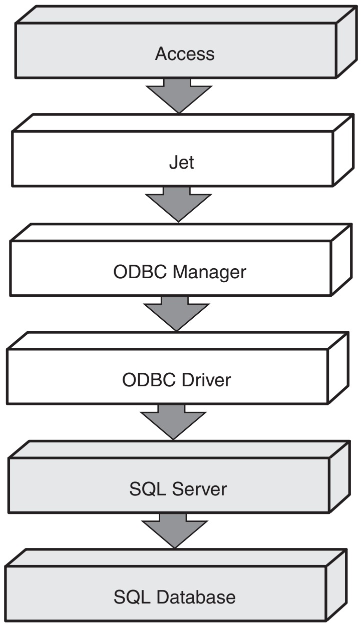

FIGURE 22.6 Access sending a pass-through query to a back-end database.

Pass-through queries are not a panacea, although they are very useful. The results of a pass-through query are not updateable, for example. Furthermore, because pass-through queries are written in the back end's specific SQL dialect, you must rewrite them if you swap out your back end. For these reasons and others, you will generally use pass-through with other solutions.

## The Front End Executing Procedures Stored on a Back End

A *stored procedure* is compiled SQL code stored on a back end. You will generally execute it using ADO or Data Access Objects (DAO) code. You can also execute a stored procedure using a pass-through query. Regardless of what you call it, the code within the stored procedure is written in the SQL native to the back end on which it is stored, and the stored procedure is executed in its entirety on the back end. Stored procedures can return results or can simply execute on the back end without returning data.

## The Front End as a Microsoft Access Data Project Communicating Directly to a Back End

ADP files were introduced in earlier versions of Access. Although for a while they were considered the technology to use, it turned out that ADP files were the database technology *du jour*. In fact, there is no upgrade path for an ADP file in Access 2007; therefore, using ADP files, you cannot take advantage of the features added to Access 2007.

## Learning the Client/Server Buzzwords

People who talk about client/server technology use many terms that are unfamiliar to the average database developer. To get a full appreciation of client/server technology and what it offers, you must have at least a general understanding of the terminology. Table 22.1 lists the most commonly used terms.

TABLE 22.1 Client/Server Terms

| Term | Definition |
|---|---|
| Column | A field. |
| DDL | A data definition language used to define and describe the database structure. |
| Foreign key | A value in one table that must be looked up in another table for validation. |
| Access Database Engine | The native database engine used by Microsoft Access. |
| ODBC (Open Database Connectivity) | A standard proposed by Microsoft that provides access to a variety of back-end databases through a common interface. In essence, ODBC is a translator. |
| OLEDB | A standard for connecting to relational and nonrelational data sources. |
| DAO (Data Access Objects) | A method of manipulating data. It has been replaced by ADO in many databases because it was optimized for accessing Jet databases. |
| ADO (ActiveX Data Objects) | A COM-based object model that allows you to easily manipulate OLE DB data sources. It is the data access methodology that replaces DAO. |
| Primary key | A set of fields that uniquely identify a row. |
| Row | A record. |
| Schema | A blueprint of the entire database. It includes table definitions, relationships, security, and other important information about the database. |
| SQL (Structured Query Language) | A type of data manipulation language commonly used to talk to tables residing on a server. |
| Stored procedures | Compiled SQL statements, such as queries, stored on the database server. They can be called by an application. |
| Transaction | A set of actions that must be performed on a database. If any one action fails, all the actions are discarded. |
| Triggers | Pieces of code that execute in response to an action occurring on a table (insert, edit, or delete). |

Many books are devoted solely to client/server technology; *Alison Balter's Mastering Access 2002 Enterprise Development* focuses entirely on client/server and Web development using Access 2002. Most magazines targeted at developers contain numerous articles on client/server technology. *Access/VB/SQL Advisor* always offers excellent articles on client/server development. Many of the articles are specifically about client/server connectivity using Access as a front end. *Visual Studio Magazine* often contains useful articles as well. Another excellent source of information is the Microsoft Developer Network CD. Offered by Microsoft as a subscription, it includes numerous articles and white papers on client/server technology, ODBC, and use of Access as a front end to a database server.

## Upsizing: What to Worry About

Suppose that your database is using Microsoft Access as both the front end and back end. Although an Access database on a file server might have been sufficient for a while, the need for better performance, enhanced security, or one of the other benefits that a back-end database provides compels your company (or your client's company) to upsize to a client/server architecture. The Access tables already have been created and even contain volumes of data. In this scenario, it might make sense to upsize.

Because all the tables have been designed as Access tables, you must upsize them to the back-end database server. Upsizing involves moving tables from a local Access database (or from any PC database) to a back-end database server that usually runs on UNIX, Windows 2000, and Windows 2003 Server.

Another reason why you might decide to upsize tables from Access to a back-end server is that many developers prefer to design their tables from within the Access environment. Access offers a more user-friendly environment for table creation than most server applications.

Because of the many caveats involved when moving tables from Access to a back end, many people opt to design the tables directly on the back end. If you do design your tables in Access, you can export them to the back end and then link them to your local database, or you can use the Upsizing Wizard to greatly facilitate this process. Regardless of the method that you choose, as you export your tables to the database server, you need to be aware of the issues covered in the following sections.

**NOTE**

If you are updating to a SQL Server database, most of the concerns regarding upsizing are handled by the Upsizing Wizards included as part of Microsoft Access 2000 and above.

### Indexes

When you are exporting a table to a server, no indexes are created. All indexes need to be re-created on the back-end database server. If your database server is running Microsoft SQL Server, you can use the Access 2007 Upsizing Wizard. This wizard will create indexes for server tables in the place where the indexes exist in your Access tables.

### AutoNumber Fields

AutoNumber fields are exported as Long integers. Because some database servers do not support autonumbering, you have to create an insert trigger on the server that provides the next key value. You also can achieve autonumbering by using form-level events, but this approach is not desirable. The numbering will not be enforced if other applications access the data. If you are upsizing to Microsoft SQL Server, the Upsizing Wizard for Access 2007 converts all AutoNumber fields to Identity fields.

### Default Values

Default values are not automatically moved to the server, even if the server supports them. You can set up default values directly on the server, but these values do *not* automatically appear when new records are added to the table unless the record is saved without data being added to the field containing the default value. As with autonumbering, you can implement default values at the form level, with the same drawbacks. If you use the Upsizing Wizard for Access 2007 to move the data to Microsoft SQL Server, the wizard exports default values to your server database.

### Validation Rules

Validation rules are not exported to the server. They must be re-created using triggers on the server. No Access-defined error messages are displayed when a server validation rule is violated. Your application should be coded to provide the appropriate error messages. You also can perform validation rules at the form level, but they are not enforced if the data is accessed by other means. If you use the Upsizing Wizard for Access 2007 to move the data to Microsoft SQL Server, validation rules are exported to the server database where possible.

### Relationships

Relationships need to be enforced using server-based triggers. Access's default error messages do not appear when referential integrity is violated. You need to respond to, and code for, these error messages in your application. You can enforce relationships at the form level, but as with other form-level validations, this method of validation does not adequately protect your data. If you use the Upsizing Wizard for Access 2007 to move the data to Microsoft SQL Server, the wizard sets up all relationships and referential integrity that you have set up in your Access database within the server database.

### Security

Security features that you have set up in Access do not carry forward to the server. You need to reestablish table security on the server. After you set up security on the server, Access is unaware that the security exists until the Access application attempts to violate the server's security. Then the server returns error codes to the application. You must handle these errors by using code and displaying the appropriate error message to users.

### Table and Field Names

Servers often have much more stringent rules than Access does regarding the naming of fields. When you export a table, all characters that are not alphanumeric are converted to underscores. Most back ends do not allow spaces in field names. Furthermore, some back ends limit the length of object names to 30 characters or fewer. If you already have created queries, forms, reports, macros, and modules that use spaces and very long field and table names, these database objects might become unusable when you move your tables to a back-end database server.

### Reserved Words

Most back ends have many reserved words. Reserved words are words used by the back end in its own operations. It is important to be aware of the reserved words of your specific back end. It is quite shocking when you upsize a table and find that field names you have been using are reserved words on your database server. If this is the case, you need to rename all the fields in which a conflict occurs. Once again, this means modifying all the queries, forms, reports, macros, and modules that reference the original field names.

### Case Sensitivity

Many back-end databases are case sensitive. If this is the case with your back end, you might find that your queries and application code don't process as expected. Queries or code that refer to the field or table name by using the wrong case are not recognized by the back-end database and do not process correctly.

### Properties

Most properties cannot be modified on remote tables. Any properties that can be modified are lost upon export, so you need to set them up again when you export the table.

### Visual Basic Code

Certain properties and methods that work on Access tables might not work on remote tables. You therefore might need to make some coding changes after you export your tables.

## Proactively Preparing for Upsizing

If you set up your tables and code modules with upsizing in mind, you can eliminate many of the pitfalls discussed previously. Despite any of the problems that upsizing can bring, the scalability of Access is one of its stronger points. Sometimes resources are not available to implement client/server technology in the early stages of an application. If you think through the design of the project with the possibility of upsizing in mind, you will be pleased at how relatively easy it is to move to client/server technology when the time is right. With the Access 2007 Upsizing Wizard, which is designed to take an Access application and upsize it to Microsoft SQL Server 2000 or Microsoft SQL Server 2005, the process is relatively simple. The upsizing tools for Access 2007 perform a lot of the work involved in upsizing a database, with just the click of a few buttons.

**NOTE**

Client/server development and client/server issues are covered in extensive detail in *Alison Balter's Mastering Access 2002 Enterprise Development*.

**NOTE**

The upsizing wizards available for Access 2000, Access 2002, and Access 2003 are almost identical to the Access 2007 Upsizing Wizard. They therefore afford you the same ease when upsizing from Access to SQL Server.

**CAUTION**

Although the upsizing tools for Access are excellent, they do have their drawbacks. For example, they do not always map the Access field type to the desired SQL Server field type. For this reason, you can opt not to use the wizards. If, despite their shortcomings, you decide to use the upsizing wizards, make sure that you carefully review both the upsizing report and the structure of each table after the wizard upsizes it.

## Using Transaction Processing

*Transaction processing* refers to the grouping of a series of changes into a single batch. The entire batch of changes is either accepted or rejected as a group. One of the most common implementations of transaction processing is a bank automated teller machine (ATM) transaction. Imagine that you go to the ATM to deposit your paycheck. In the middle of processing, a power outage occurs. Unfortunately, the bank recorded the incoming funds prior to the outage, but the funds had not yet been credited to your account when the power outage occurred. You would not be very pleased with the outcome of this situation. Transaction processing would prevent this scenario from occurring. With transaction processing, the whole process succeeds or fails as a unit.

A group of operations is considered a transaction if it meets the following criteria:

- **It is atomic**—The group of operations should finish as a unit or not at all.
- **It is consistent**—The group of operations, when completed as a unit, retains the consistency of the application.
- **It is isolated**—The group of operations is independent of anything else going on in the system.
- **It is durable**—After the group of operations is committed, the changes persist, even if the system crashes.

If your application contains a group of operations that are atomic and isolated, and if, to maintain the consistency of your application, all changes must persist even if the system crashes, you should place the group of operations in a transaction loop. With Access 2007, the primary benefit of transaction processing is data integrity.

## Understanding the Benefits of Transaction Processing

**NOTE**

This code, and all the code in this chapter, is located in the `CHAP22EX.ACCDB` database in the `basTransactions` module on the sample code CD-ROM.

**NOTE**

Any discussion of Access 2007 covered in this section also applies to Access 2000, Access 2002, and Access 2003.

Access 2007 does its own behind-the-scenes transaction processing. The Access Database Engine does this implicit transaction processing solely to improve the performance of your application. As a processing loop executes, Access buffers and then periodically writes the data to disk. In a multiuser environment, the Access Database Engine (implicitly) commits transactions every 50 milliseconds by default. This period of time is optimized for concurrency rather than performance. If you feel that it is necessary to sacrifice concurrency for performance, you can modify a few Windows Registry settings to achieve the specific outcome you want. The next section covers these settings.

Although implicit transaction processing, along with the modifiable Windows Registry settings, generally gives you better performance than explicit transaction processing, it is not a cut-and-dried situation. Many factors affect the performance benefits gained by both implicit and explicit transaction processing:

- Amount of free memory
- Number of columns and rows being updated
- Size of the rows being updated
- Network traffic

If you plan to implement explicit transaction processing solely to improve performance, you should make sure that you benchmark performance using both implicit and explicit transactions. It is critical that your application-testing environment be as similar as possible to the production environment in which the application will run.

## Modifying the Default Behavior of Transaction Processing

Before you learn how to implement transaction processing, take a look at what you can do to modify the default behavior of the transaction processing built in to Access 2007. Three Registry settings affect implicit transactions in Access 2007: `ImplicitCommitSync`, `ExclusiveAsyncDelay`, and `SharedAsyncDelay`. These keys are located in the `\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Jet\4.0\Engines\Jet 4.0` Registry folder.

**TIP**

You can access the Windows Registry using the `RegEdit` utility. To use `RegEdit`, select the Run option from the Start menu and then type **`RegEdit`**. In Windows Vista, you must locate the `RegEdit` utility and then double-click it to run the utility.

The `ImplicitCommitSync` setting determines whether the system waits for a commit to finish before proceeding with application processing. The default is `No`. This means that the system will proceed without waiting for the commit to finish. You generally won't want to change this setting; using `No` dramatically improves performance. The danger of accepting the value of `No` is that you will increase the amount of time during which the data is vulnerable. Before the data is flushed to disk, the user might turn off the machine, compromising the integrity of the data.

The `ExclusiveAsyncDelay` setting specifies the maximum number of milliseconds that elapse before the Access Database Engine commits an implicit transaction when a database is opened for exclusive use. The default value for this setting is 2000 milliseconds. This setting does not in any way affect databases that are open for shared use.

The `SharedAsyncDelay` setting is similar to the `ExclusiveAsyncDelay` setting. It determines the maximum number of milliseconds that elapse before the Access Database Engine commits an implicit transaction when a database is opened for shared use. The default value for this setting is `50`. The higher this value, the greater the performance benefits reaped from implicit transactions, but also the higher the chances that concurrency problems will result. These concurrency issues are discussed in detail in *Alison Balter's Mastering Access 2002 Enterprise Development*.

In addition to the settings that affect implicit transaction processing in Access 2007, an additional Registry setting affects explicit transaction processing. The `UserCommitSync` setting controls whether explicit transactions are completed synchronously or asynchronously. With the default setting of `Yes`, control doesn't return from a `CommitTrans` statement until the transactions are actually written to disk, resulting in synchronous transactions. When this value is changed to `No`, a series of changes is queued, and control returns before the changes are complete.

You can modify the values of these Registry settings and other Access Database Engine settings by using `Regedit.exe` (the Registry Editor) for Windows Vista, and Windows 2003. Changes to this section of the Registry affect all applications that use the Access Database Engine. If you want to affect only your application, you can export the Microsoft Jet portion of the Registry tree and import it into your application's Registry tree. You then can customize the Registry settings for your application. To force your application to load the appropriate Registry tree, you must set the `INIPath` property of the `DBEngine` object.

A much simpler approach is to set properties of the ADO `Connection` object; you can specify new settings at runtime for all the previously mentioned Registry entries as well as for additional entries. A further advantage of this approach is that it will modify (temporarily) Registry entries for any machine under which your application runs. Any values you change at runtime temporarily override the Registry values that are set, enabling you to easily control and maintain specific settings for each application. This code illustrates how you modify the `ExclusiveAsyncDelay` and `SharedAsyncDelay` settings using properties of the `Connection` object:

```
Sub ChangeOptions()
    Dim cnn As ADODB.Connection
    Set cnn = CurrentProject.Connection

    cnn.Properties("JET OLEDB:Exclusive Async Delay") = 1000
    cnn.Properties("JET OLEDB:Shared Async Delay") = 50
End Sub
```

## Implementing Explicit Transaction Processing

Now that you are aware of the settings that affect transaction processing, you are ready to learn how to implement transaction processing. Three methods of the `Connection` object (covered in Chapter 15, “What Are ActiveX Data Objects, and Why Are They Important?”) control transaction processing:

- `BeginTrans`
- `CommitTrans`
- `RollbackTrans`

The `BeginTrans` method of the `Connection` object begins the transaction loop. The moment `BeginTrans` is encountered, Access begins writing all changes to a log file in memory. Unless you issue the `CommitTrans` method of the `Connection` object, the Access Database Engine never actually writes the changes to the database file. After the `CommitTrans` method is issued, the Access Database Engine permanently writes the updates to the database object. If a `RollbackTrans` method of the `Connection` object is encountered, the log-in memory is released. Listing 22.2 shows an example of how transaction processing works under Access 2007. Compare this to Listing 22.1.

LISTING 22.2 Transaction Processing in Access 2007 Using `BeginTrans`, Logging, `CommitTrans`, and `RollbackTrans`

```
Sub IncreaseQuantityTrans()
    On Error GoTo IncreaseQuantityTrans_Err
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset
    Dim boolInTrans As Boolean

    boolInTrans = False
   Set rst = New ADODB.Recordset

    Set cnn = CurrentProject.Connection
```

LISTING 22.2 Continued

```
    rst.ActiveConnection = cnn
    rst.CursorType = adOpenKeyset
    rst.LockType = adLockOptimistic
    rst.Open "Select OrderId, Quantity From tblOrderDetails"

    'Begin the Transaction Loop
    cnn.BeginTrans
        boolInTrans = True
        'Loop through recordset increasing Quantity field by 1
        Do Until rst.EOF
            rst!Quantity = rst!Quantity + 1
            rst.UPDATE
            rst.MoveNext
        Loop
        'Commit the Transaction; Everything went as Planned
    cnn.CommitTrans
    boolInTrans = False

IncreaseQuantityTrans_Exit:
    Set cnn = Nothing
    Set rst = Nothing
    Exit Sub

IncreaseQuantityTrans_Err:
    MsgBox "Error # " & Err.Number & ": " & Err.Description
    'Rollback the Transaction; An Error Occurred
    If boolInTrans Then
        cnn.RollbackTrans
    End If
    Resume IncreaseQuantityTrans_Exit
End Sub
```

This code uses a transaction loop to ensure that everything completes as planned or not at all. Notice that the loop that moves through the recordset, increasing the `Quantity` field in each record by `1`, is placed in a transaction loop. If all processing in the loop completes successfully, the `CommitTrans` method executes. If the error-handling code is encountered, the `RollbackTrans` method executes, ensuring that none of the changes are written to disk. The `boolInTrans` variable is used to determine whether the code is within the transaction loop. This ensures that the error handler performs the rollback only if an error occurs within the transaction loop. If the `CommitTrans` method or the `RollbackTrans` method is issued without an open transaction, an error occurs.

## Practical Examples: Getting Your Application Ready for an Enterprise Environment

Splitting the application code and data is the first step toward making your application enterprise ready. Consider placing the application data on the network and the application code on each workstation. If you think that the number of users, required security, or volume of data stored in the application warrants client/server technology, consider using one or more of the client/server techniques covered in this chapter. Finally, think about whether any application processes warrant transaction processing. If you feel that client/server technology or transaction processing is a necessary ingredient to your application, learn more about these techniques from a source such as *Alison Balter's Mastering Access 2002 Enterprise Development*.

## Summary

Many people think that the transition of a simple Access application to a multiuser or client/server environment is a simple one. I strongly disagree. There are several things to think about when moving an application from a single-user environment to a multiuser environment, and even more things to think about when moving to a client/server environment. The more you think about these potential evolutions when you first design and build your application, the fewer problems you'll have if your application data has to be upsized.

This chapter exposed you to multiuser techniques. It explained client/server technology and when you need it. It also described the various roles that Access plays in the application design model. Finally, you learned about a technique that is important within an enterprise application: transaction processing.

The chapter is intended to be an introduction to these important topics. All the topics in this chapter are covered in detail in *Alison Balter's Mastering Access 2002 Enterprise Development* (which applies to Access 2007 as well).