1

SQL Server 2000 – Particulars and History

So you want to learn something about **databases** – **SQL Server** in particular? That's great because databases are pervasive – they are everywhere, though you may not have really thought about this up until now.

In this chapter, we'll be looking at some of the different varieties and brands of databases available both today and throughout history. We'll examine some of the advantages and disadvantages of each, and how they fit into the grand picture of life, so to speak.

Moving on from there, we'll take a look at SQL Server 2000 specifically – the different editions that are currently available and what each one does or doesn't include.

In addition, we'll take some time to examine the database development process and how it fits into your overall development cycle. In this same section, we'll talk at an entry level about some system architecture issues and how these affect our database thinking.

This book is focused on trying to successfully prepare you to develop applications using SQL Server 2000. It should also act as good preparation for some of the exams in the Microsoft certification process, so we'll round off the chapter with a brief look at this.

A Brief History of Databases

SQL Server is an **RDBMS** – or **Relational Database Management System**. RDBMS systems are at the pinnacle of their popularity at the moment. Using an RDBMS as the basis for data storage is plainly "the way it's done" for most applications nowadays – but it wasn't always this way.

In this section, we're going to take a look back in time and examine some of the other databases used in the past. We'll try not to dwell on this "Old News", but it's critical to understand where database technology has come from if you want to understand where you're going today, and why.

Types of Databases

Databases are not just limited to the computer-based systems that we typically think about when we hear the term – they are much, much more. A database is really any collection of *organized* data. Even Webster's dictionary puts a qualifier on any computer notion:

> *Database: A usually large collection of data organized especially for rapid search and retrieval (as by a computer).*

The file drawers in your office are really something of a database (that is, if they are better organized than mine at home). In fact, databases have existed throughout most of the history of the "civilized" world, going back to the days of the early philosophers and academics (Socrates, Aristotle, Hippocrates, etc.).

That being said, there's a reason why databases are so closely associated with computers. It's because, for most database situations (virtually, but not quite, all of them), computers are simply the fastest and most efficient way to store data. Indeed, the term database is thought to have originated from the computing community in 1962 or so.

Databases, then, fall into a number of common categories:

- **Paper-based**: These, although often not thought of as databases, probably still make up the largest proportion of databases in the world today. There are literally billions and billions of tons of paper out there that are still meticulously organized, but haven't been anywhere near a computer.
- **Legacy mainframe** – often **VSAM** (**Virtual Storage Access Method** – common to IBM mainframes) databases: Don't underestimate the number of legacy mainframes still out there, and their importance. Connectivity to host systems and the vast amounts of data they still contain is one of the major opportunity areas in database and systems development today. There are still many situations where I recommend a host system solution rather than a client-server or web-based model. It's worth noting though that I still believe in using a true RDBMS – albeit one that's located on a host system.
- **dBase** and other file-based databases: Typically, these include any of the older **Indexed Sequential Access Method** – or **ISAM** – databases. These normally use a separate file for each table, but the ISAM name comes from the physical way the data is stored and accessed more than anything else. Examples of ISAM databases that are still in widespread legacy use – and even in some new developments in certain cases – include dBase, FoxPro, Excel, Paradox, and Access. (Yes, Access is an ISAM with a relational feel and several relational features – it is not, however, a true relational database system.) These systems had most of their heyday well before RDBMS systems. (There is something of a paradox in this since RDBMS systems appeared first.) These systems are still quite often great for small, stand-alone databases where you will never have more than a small number of users accessing the data at a time.

- **RDBMS systems**: Data for the masses, but with much better data integrity. These systems do more than just store and retrieve data. They can be thought of as actually caring for the integrity of the data. Whereas VSAM and ISAM databases typically store data very well, the database itself has no control over what goes in and out (OK, Access has some, but not like a true RDBMS). The programs that use the database are responsible for implementing any data integrity rules. If five programs are accessing the data, you'd better make sure that they are all programmed correctly. RDBMS systems, on the other hand, take the level of responsibility for data integrity right down to the database level. You still want your programs to know about the data integrity rules to avoid getting errors from the database, but the database now takes some of the responsibility itself and the data is much safer.
- **Object-oriented databases**: These have been around for a while now, but are only recently beginning to make a splash. They are really a completely different way of thinking about your data and, to date, have only found fairly specialized use. Examples would be something similar to a document management system. Instead of storing the document in several tables, the document would be stored as a single object, and would have properties whose state would be maintained. **ODBMS** systems often provide for such object-oriented concepts as inheritance and encapsulation.

RDBMS systems are clearly king these days. They are designed from the ground up with the notion that they are not going to be working with just one table that has it all, but with data that relates to data in completely different tables. They facilitate the notion of combining data in many different ways. They eliminate the repetitive storage of data and increase speed in transactional environments.

The Evolution of Relational Databases

E.F. Codd of IBM first introduced the principles behind relational database structures and a Structured English QUEry Language – or SEQUEL – back in the late 1960's (the name was later shortened to just **Structured Query Language** or **SQL**). The concept was actually pretty simple – increase data integrity and decrease costs by reducing repetitive data as well as other database problems that were common at the time.

Nothing really happened in the relational world as far as a real product was concerned until the mid to late 70's, though. Around that time, companies such as Oracle and Sybase became the first to create true relational database systems. It might surprise you to learn that these systems got their start in mainframe – not client-server – computing. These systems offered a new way of looking at database architecture and, since they ran on multiple platforms, they also often offered a higher potential for sharing data across multiple systems.

In the 80's, the **American National Standards Institute** (**ANSI**) finally weighed in with a specification for SQL, and **ANSI-SQL** was born. This was actually a key moment in RDBMS computing because it meant that there would be better compatibility between vendors. That, in turn, meant that more of the expertise built up in one RDBMS was also usable in a competing system. This has greatly aided the process of trying to increase the number of developers in the SQL community. The ANSI specification called for several different levels of compliance. Most of the major RDBMS products available today are classified as being **Entry-Level ANSI compliant** (like SQL Server, for example). Entry-level ANSI compliance means that a database meets the basic defined ANSI standards for the SQL syntax.

ANSI compliance is a double-edged sword. I'm going to encourage you to make use of ANSI compliant code where feasible – it's particularly important if you may be migrating your code between different database servers. But you also need to realize that many of the performance and functionality features that each of the high performance database vendors offers are not ANSI compliant. Each vendor extends their product beyond the ANSI spec in order to differentiate their product and meet needs that ANSI hasn't dealt with yet. For example, SQL Server 2000 has expanded on the basic SQL with its own additions, which are called T-SQL. This leaves you with a choice – ANSI compliance or performance.

Use ANSI compliance not as a religion but, rather, where it makes sense. Go for ANSI code where it means little or no difference in performance (such as with queries), but also don't be afraid to make judicious use of specialized features that may offer some functionality or performance gain that ANSI can't give you. Just document these areas where you use them so that, if you are faced with porting to a new RDBMS, you know where to look for code that may not run on the new system.

Microsoft SQL Server (referred to in this book as simply **SQL Server**) was originally born from Sybase SQL Server (referred to in this book simply as **Sybase**). Microsoft partnered with Sybase in 1989 to develop a version of SQL Server for, of all things, OS/2. SQL Server was migrated to Windows NT back in 1993 with version 4.2. The relationship ended with the release of version 6.0. From 6.5 forward, SQL Server has been a Microsoft-only product. The highly successful version 7.0 was essentially a complete rewrite of the product and was the first version available for Windows 9x (there was now virtually no Sybase code left in SQL Server). Finally, we reach today's version – SQL Server 2000.

While there are unmistakable similarities, there are now substantial differences in implementation and feature support between version 4.21 (the oldest version you're actually likely to find installed somewhere) and version 2000. Version 6.0 added such details as cursor support. Version 6.5 added distributed transactions, replication, and ANSI compatibility. The rewrite with version 7.0 enabled the loss of problem areas such as the devices defined for data storage.

About SQL Server 2000

SQL Server 2000 comes with far more than just the usual RDBMS – it has additional components that would, for many products, be sold entirely separately or with add-on pricing. Instead, Microsoft has seen fit to toss in these extras at no additional charge.

SQL Server 2000 is now available in five **editions** (CE, Personal, Desktop Engine, Standard, Developer, and Enterprise), which are discussed in more detail later.

There is also an Enterprise Evaluation Edition, which can be downloaded from the Web for a 120 day trial period.

The full suite that makes up SQL Server 2000 includes:

System/Subsystem	Description	Editions
SQL Server 2000 (the main RDBMS)	This is the "guts" of the system, and is required for anything else to work. It is a very robust relational database system. With the exception of the Desktop Engine, which only has the main RDBMS, you will find that this part of the system also includes several different services and utilities, such as the **SQL Server Agent** (Scheduler), the **Distributed Transaction Coordinator** (**DTC**), the **SQL Server Profiler** (trouble-shooting), and the **Enterprise Manager** (**EM**) – one of the best built-in management tools in the business, regardless of price-range. If you're coming from the Access world or some other desktop database, strap on your seatbelt, because you have just seen a glimpse of what's possible.	Desktop Engine Personal Standard Developer Enterprise

System/Subsystem	Description	Editions
Full-Text Search	This is an optional part of the main installation. If you want this functionality, you need to actively choose it – it's not installed by default. **Full-Text Search** provides the functionality to support more robust word lookups. If you've used an Internet search engine and been left in awe of the words and phrases that you can find, Full-Text Search is the tool for you. It ranges from being able to quickly locate small phrases in large bodies of text to being able to tell that "drink" is pretty much the same word as "drunk" or that "swam" is just the past tense of that word "swim" you were looking for. This one is not available in the Personal version (Win 9x) of SQL Server. We'll look at it extensively in Chapter 26.	Personal (except Win 9x) Standard Developer Enterprise
English Query	Featured in Chapter 27, **English Query** (**EQ**) allows you to develop applications for even the most non-technical of users. EQ allows users to ask questions or give commands in plain English and have them translated into a query that's usable by SQL Server. A great tool, but keep in mind that this is a completely separate installation from the rest of SQL Server.	Personal Standard Developer Enterprise
Analysis Services	Yet another tool that isn't part of the main installation, but gives great extras to the product. Analysis Services comprises of **OLAP** (**On Line Analytical Processing**), data warehousing, and data mining tools. It's something that many companies try to do from their main server – we'll look into why that's a mistake and how to make use of SQL Server's Analysis Services in detail in Chapter 25. The editions listed on the right all support OLAP, although only Enterprise and Developer have the full Analysis Services functionality. The Standard and Personal editions only contain the main functionality, which consists of Analysis Services itself, custom rollups, data mining, and actions (end user operation on data). Additionally, Analysis Services can only be installed on Windows NT/2000.	Personal Standard Developer Enterprise
Replication	This function allows data to be replicated to another SQL Server instance, usually found on another server as part of a recovery strategy, or to a remote server in another physical location, to reduce data transfer traffic. Replication is covered in Chapter 23.	Desktop Engine Personal Standard Developer Enterprise

Table continued on following page

System/Subsystem	Description	Editions
Data Transformation Services	**Data Transformation Services** (**DTS**) has expanded enormously in SQL Server 2000. A great range of different functionality in transforming data, either within a database, or transferring information in or out, is now available, including the ability to customize tasks and workflows. DTS is a greatly under-utilized product that reduces the need for companies to use a programmatic approach to transform data (for example, with Visual Basic), or even a basic Bulk Copy Program (bcp). We will cover DTS in Chapter 22.	Personal Standard Developer Enterprise

There are a few additional differences between the various editions of SQL Server 2000. These include:

- **Symmetric Multiprocessing** (**SMP**): Support for SMP has increased a great deal through the different editions of SQL Server 2000 (though Win 98 and NT4 Workstation can't support this). There is support for up to four processors in the Standard edition if installed on NT Server or Enterprise, and support for up to 32 processors with the Enterprise edition if it is installed on Windows 2000 Datacenter Server.

  SMP distributes the workload of the server over multiple processors symmetrically – that is, it tries to balance the load as opposed to running on just one CPU per process.

- **Clustering Support** (Enterprise/Developer editions only): Clustering allows load-balancing across servers and automatic fail-over support (if one server dies, another one automatically picks up where the other left off). Currently, you can only cluster two servers with all operating systems, with the exception of Windows NT Enterprise edition, Windows 2000 Enterprise edition, and Windows 2000 Datacenter edition which can have up to four cluster servers.

Which Edition Should You Use?

The answer to this is like the answer to most things in life: it depends.

Each of the various editions has a particular target "market" that it's designed for. Usually, I find some exceptions to the rules on how things should best be used but, for these products, I would say that what Microsoft designed them for really is their best use. Let's take a quick look at the editions, one by one. The following section gives my summary of each edition. Obviously, Microsoft makes its own comparisons, which it might be useful for you to see. Don't forget that there is a Microsoft slant to all of these.

- http://www.Microsoft.com/sql/productinfo/sqlcompdata.htm – gives an overall comparison on data warehousing
- http://www.Microsoft.com/sql/productinfo/sqlcompecom.htm – gives an overall comparison on e-commerce
- http://www.Microsoft.com/sql/productinfo/sqlcomplob.htm – gives an overall comparison on Line-of-Business
- http://www.Microsoft.com/sql/productinfo/feaover.htm – gives a features overview with links to specific areas

Windows CE Edition

The **Windows CE Edition** will be used on Windows CE devices. It will be extremely limited in its functionality as, obviously, these devices have an extremely limited capacity. Applications using Windows CE and SQL Server are still quite limited at present and it's really only possible to have any sort of useful application built on the more expensive CE products.

Desktop Engine Edition

The **Desktop Engine Edition** of SQL Server 2000 was known as the Microsoft Data Engine (MSDE) in SQL Server 7.0. Don't get confused by thinking that this is still the same version as the SQL Server 7.0 Desktop version. It isn't. Desktop with SQL Server 7.0 is now Personal with SQL Server 2000. The Desktop Engine Edition consists only of the main RDBMS. It has none of the administration tools – not even the Enterprise Manager or Query Analyzer.

Contrary to popular belief, this is not a different version of SQL Server – all editions use the same binary executable that the Enterprise Edition uses. The difference is more in what auxiliary services are supported.

This edition is small and freely distributable, and Microsoft is pushing it as the new database engine for Access (replacing Jet). This makes it great for salespeople who need a database to take on the road with them.

Personal Edition

The **Personal Edition** is a rename of the Desktop Edition found with SQL Server 7.0 (not to be confused with the Desktop Engine Edition in SQL Server 2000). It was created to serve a couple of purposes: to provide a more robust desktop database solution than that provided with Access (even on Windows 9x); and to provide a version of SQL Server that could be used in "unplugged" situations. The latter is the big advantage – it is proving to be really popular in remote situations, like for sales reps who are on the road all the time. They can have their own version of the customer database and just "synchronize" using replication when they are able to connect back up with the network.

The Personal Edition is excellent when you want a small stand-alone database or when you have the need to be disconnected from a central data source, but want to be able to take some of that data with you. You could also use the Personal Edition to run a small server on Windows NT/2000 – this latter configuration even has support for multiple processors. Keep in mind though that, even with multiprocessing active, there is no support for parallel queries (which run different parts of the same query at the same time).

I'm told (I haven't tried it myself) that several of the tools that are not supposed to work with the Personal Edition actually do work just fine – particularly if you're running under NT Workstation. I strongly discourage you from implementing things this way. If you need the extra features, then use the right O/S to support them. Otherwise, you may find that everything works OK for a while, but you'll also find that you have no support from Microsoft when you want to ask why something broke.

Confusingly, the Personal Edition cannot be bought, but if you buy the Standard or Enterprise Editions you'll get it for free. It is part of the Client Access License (CAL) and client software. Useful if you are running Windows 2000 Professional or Windows 98 and you need access to the GUI Admin tools (Standard and Enterprise editions don't run on Win2K Pro or 98, and no edition supports 95).

Standard Edition

The **Standard Edition** is the mainstream edition of SQL Server. This is the edition that's going to be installed for the majority of SQL Server users. The Standard Edition supports multiprocessing with up to four CPUs and 2GB of RAM. However, it doesn't support some of the more advanced features. For example, only a subset of the Analysis Services features is supported. You need to purchase a separate license for each Standard Edition instance you install on a machine.

Developer Edition

Readers of this book will probably see the **Developer Edition** as the default installation. Enterprise and Standard Editions should be seen as the production server solution, with developing and testing of applications performed on the Developer Edition. This has all the features of the Enterprise edition and, therefore, once a solution has been developed using the Developer Edition, there should be no problems in moving this to a production environment.

The only difference between this edition and the Enterprise Edition is the licensing of the product – the Developer Edition can only be used as a development environment.

Enterprise Edition

To run the **Enterprise Edition**, you must have NT Enterprise Edition, Windows 2000 Advanced Server, or Windows 2000 Datacenter Server installed. SQL Server Enterprise Edition adds support for multiprocessing with up to 32 CPUs, there is support for clustering (where two separate servers provide fail-over and can otherwise share a workload), and it allows for HTTP access to OLAP cubes (cubes are fully described in Chapter 25).

Whether to go with Enterprise Edition or not is usually an easy choice because the outcome is almost always decided for you, based on a requirement for one of the Enterprise Edition features, or costs and licensing (the per-processor license for the Enterprise Edition is four times the price of that for the Standard Edition). If you need clustering, then you need the Enterprise Edition. Enterprise edition special features include:

- Clustering
- Distributed partitioned views
- Indexed views
- Partitioned cubes
- Support for more than 4GB of RAM
- Log shipping (a fail-over strategy)
- Support for more than 4 CPUs

In addition, there is a long list of more obscure features that are only supported in the Enterprise edition, but it would be rare that one of those is needed if you don't also need one of the above items. Basically, if you need one of these, then you need Enterprise edition – it's that simple.

Hardware and OS Requirements

The stated **minimum hardware requirements** for SQL Server are pretty easy to reach these days:

- Pentium 166 or better (Alpha is no longer an option, and Microsoft has stated that there will be no future development for that platform).
- At least 64MB memory, although 128MB is recommended for the Enterprise Edition.

  You can get away with only 32MB for the Desktop Engine and Personal Editions on all but Windows 2000 – where 64MB is required.
- Between 95MB hard disk space (minimum installation) and 270MB (full installation).

  You will also need a further 50-130MB if you want to install Analysis Services, and another 80MB for English Query.

  The Desktop Engine Edition requires only 44MB.
- Enterprise and Standard Editions run on Windows NT Server version 4.0 with Service Pack 5 (SP5) or later, Windows NT Server 4.0 Enterprise Edition with SP5 or later, Windows 2000 Server, Windows 2000 Advanced Server, and Microsoft Windows 2000 Datacenter Server.

  Developer Edition runs on the operating systems listed above for the Enterprise and Standard Editions, as well as on Windows 2000 Professional and Windows NT Workstation 4.0 with SP5 or later.

  Personal Edition and Desktop Engine run on the operating systems listed above for the Enterprise and Standard Editions, as well as on Windows 98 (Second Edition if the computer doesn't have a network card), Windows Millennium Edition, Windows 2000 Professional, and Windows NT Workstation 4.0 with SP5 or later.

  This information can be found at http://www.microsoft.com/sql/productinfo/sysreq.htm.
- VGA Video in 800 x 600 mode (some of the graphical tools require it)
- IE 5.0 or later

In reality, you'll want to have a bit beefier machine than the recommendation. Even for a stand-alone development server, I recommend a minimum of 128MB of RAM and a Pentium II 500 or better processor. For production systems, no less than 256MB of RAM even for the smallest systems – and more likely 512MB to 2GB.

Building Database Connected Systems

At this juncture, we're ready to go into the holy-war territory of architectural issues. It seems like everybody's got an idea of what the best architecture is for everything.

Before we even get too deep into this I'll give you my first soapbox diatribe.

The perfect architecture to use is the one that is right for the particular solution you are working towards. There are very few easy answers in life, and what system architecture to use for a project is rarely one of them. Don't allow anyone to mislead you into thinking, "You should always use n-tier architecture", or that, "The mainframe is dead – anyone who installs a host-based system today is nuts!"

I have a definite belief in the power and flexibility of the n-tier approach we'll be talking about shortly – but don't believe for a minute that I think it's the only solution. The moment you let yourself be backed into thinking one approach is the right one for everything, will be the moment that you start turning out sub-standard work. Both traditional client-server and host technology still very much have their place, and I'll try to address some of the "wheres" and "whys" as we go through this section.

There have been a few models to come around through the years, but today we usually group things depending on how they handle three groups of **services**:

- **User Services**: This usually includes aspects like drawing the user interface (UI) and basic formatting and field rules. An example of facets that might be handled by User Services is proper formatting of a date – including making it known that a given field is a date field and pre-validating that any value entered into this field is actually a date. User services is all about presentation and making sure that each field has at least the type of data it's supposed to have in it.
- **Business Services**: This part knows about various business rules. An example of a business service is one that connects with your credit card company to validate a customer's credit card purchase. In a 3-tier or n-tier system, the Business Services objects may reside on their own server, be split across several servers, or, in smaller installations, share a server with Data Services.
- **Data Services**: This is all about storage and retrieval of data. Data services know about data integrity rules (say, that an inventory value can't go below zero), but don't care where the credit card approval came from. This is where SQL Server lies.

Let's take a look at some of the classic architectures used, both past and present, and see how our services fit in.

Single Tier (Host) Systems

This is the old mainframe and mini-computer model. There was virtually no logic at the desktop – instead, there was a dumb terminal. All that was sent down the wire to the terminal was the screen layout information, which included the data to display, of course.

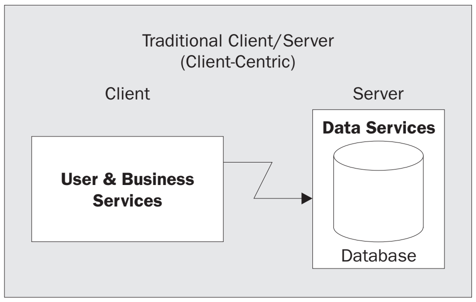
Advantages	**Disadvantages**
Requires very little bandwidth on your network in order to have fast response times – great for international or WAN situations where bandwidth can be expensive.	Very expensive hardware-wise. In the old days, these often required special plumbing for cooling water – although I'm not aware of any systems which still require this being produced today.
Also exceptionally reliable. You'll find mainframes out there that haven't been "down" in years, literally.	Typically proprietary in nature – much more difficult to share information with other systems.
Deployment of new software is extremely easy. Just install on the host system, and every user has the new version – no running from machine to machine for the upgrade.	Very limited number of "off-the-shelf" software packages available. Since the number of potential customers is few, the cost of these packages tends to be extremely high.

2-Tier Architecture (Client-Server)

2-tier, or **client-server** systems, first started becoming popular in the early 90's. There were actually two sub-types to this architecture: client-centric (smart client) and server-centric (smart server).

Client-Centric

The **client-centric** version of client-server was based on the notion that PCs are cheap (the driving force behind most client-server development) and that you're going to get the most power when you distribute the computing requirements as much as possible. As such, whenever possible, only the data services part was performed on the server. The business and UI side of things was performed at the client – thus ensuring that no one system had to do all that much of the work. Every computer did its fair share (at least, that was the idea).

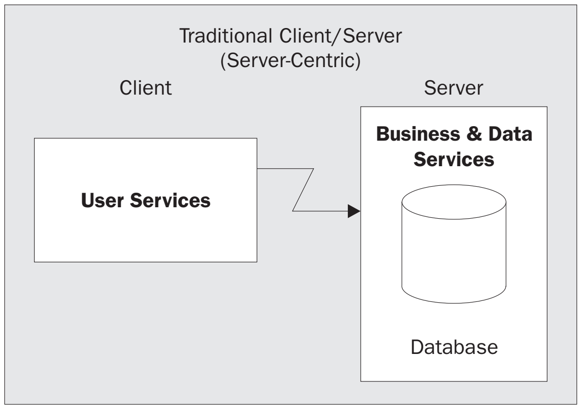


The big problem with client-centric client-server was (and still is) bandwidth. If all the business logic is on the client, then there tends to be a very large number of round trips (network sends and receives) between the client and the server. Frequently, large chunks of raw data are sent to the client – quickly clogging the network and slowing down everyone else trying to get their own huge blocks of data back and forth.

Advantages	Disadvantages
Distributes the workload to a large number of relatively cheap clients.	Is a terrible bandwidth hog – clogs networks up very quickly.
If you have one user who needs more speed, you can purchase a faster system for just them rather than a large expensive host system that everyone is going to take a piece of.	Installations are time consuming and difficult to coordinate. New software or versions of software must be installed on multiple machines. Version upgrades can be particularly problematic since old clients are not always compatible with the new server components and vice versa. All clients may have to be upgraded at one time, which can create quite a serious logistics problem.
The same money that buys the computing power on the client side also buys power for other productivity applications, such as word processing and spreadsheet applications.	Each client, depending on the vendor, may need a separate license for each seat or connection. This can increase costs.

Server-Centric

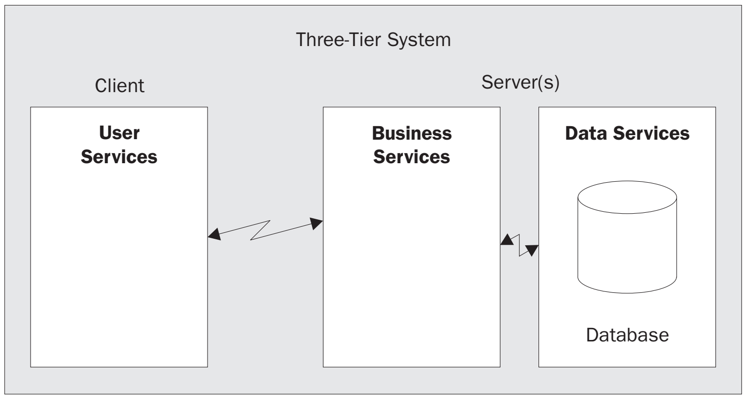
This lives on the notion that computing power is cheaper in PCs than in host systems, but tries to gain some of the advantages of centralized systems. Only user services are distributed to the client. Only information that actually needs to be displayed on the screen is sent to the client. Business and data services remain at the server. Network bandwidth is far more host system like.



Advantages	Disadvantages
Some upgrades can be done entirely at the server level.	Other upgrades still require a "touch" on every client computer – upgrades and new installs are both very tedious and difficult logistically.
A large number of homogeneous products are available off-the-shelf – pre-made software is cheap.	Long-running and heavy-load jobs by one user affect all users.
Since only the information to be displayed is sent on the network, there is little network bandwidth used compared to the client-centric model.	Large servers grow exponentially in price. Some are every bit as expensive as host systems.
	Though the model starts to look like a host system model, there is usually considerably more downtime.

Three-Tier

This model, and the closely related one that follows (n-tier), are the much-hyped architectures of today. If you hear someone talking about how everything needs to be done one way regardless of what it is – they are almost certainly talking **three-tier** or n-tier computing.



This model takes the approach of breaking up all three service levels into completely separate logical models. Clients are responsible for UI issues only – just as they were under server-centric client-server. The difference is that the business and data services are logically separated from each other. In addition, this approach moves the logical model into a distinctly separate realm from the physical model. This means that business and data services can run on the same server, but do not have to. This adds a significant level of stability and scalability since you can split the workload onto two (and, depending on how it's done, more) servers. In addition, this model has a tendency to be more extensible, since changes and additions affect smaller pieces of code (instead of one huge build of everything, you can just rebuild the affected components).

Since everything is component based, you can, if you use DCOM, take an approach where you distribute the components over many servers. If you use Microsoft Transaction Server (MTS), or COM+, which comes with Windows 2000, you can even keep copies of the same component on multiple servers for load balancing.

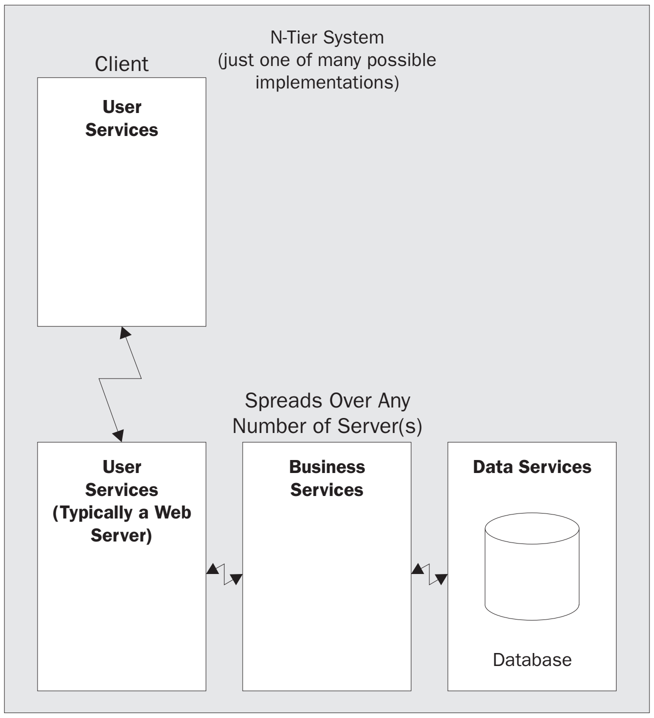
Advantages	**Disadvantages**
Some upgrades can be done entirely at the server level.	Other upgrades still require a "touch" on every client computer – upgrades and new installs are both very tedious and difficult logistically.
An increasing number of homogeneous products are available off the shelf – pre-made software is cheap.	Typically there is still considerably more downtime than with a host system.
Since only the information to be displayed is sent on the network, there is little network bandwidth used compared to the client-centric model. The load may, however, be higher between the business-logic and data-services systems if they are on different servers.	Performance can be degraded due to COM and marshalling, especially across servers or even networks. This includes any access over the Internet.

Table continued on following page

Advantages	Disadvantages
Allows for (actually encourages) component-based development, which *can* increase reusability.	There is a much greater need for security and infrastructure. For example, MTS/COM+ for the whole process of looking after your COM modules, or MSMQ if you are using any sort of messaging.
Two medium servers are often cheaper than one large server. The separation of business and data services makes two servers an option.	

N-Tier

N-tier is essentially like 3-tier and, theoretically, the best of all worlds. Frankly, I like this model a lot, but I still have to caution you about taking the "one size fits all" approach. This model gets serious about implementing what looks like a three-tier model logically, but instead breaks the components down to their smallest reasonable logical units of work. If the data services layer is done properly, even the database can be spread across multiple servers and moved around as needed. The only impact is on the data services components that provide access to the moved data. The business services components are oblivious to the move (less re-development here folks!), since they only need to know the name of the data services component that supplies the data and what specific method to call.


N-Tier System
(just one of many possible implementations)
Client
User Services
Spreads Over Any Number of Server(s)
User Services (Typically a Web Server)
Business Services
Data Services
Database

Similarly, you can change UI implementations fairly easily – you only need to redevelop the UI. You still call the same business logic components regardless of what UI (say, web vs. true client?) you are using.

Advantages	Disadvantages
Even more upgrades can be done entirely at the server level.	Often increases the number of network connections (which are frequently the slowest and most unreliable part of your system).
An increasing number of homogeneous products are available off-the-shelf – pre-made software is cheap.	Typically there is still considerably more downtime than with a host system.
Since only the information to be displayed is sent on the network (rather than an entire, not yet filtered, data set), there is little network bandwidth used compared to the client-centric model. The load may, however, be higher between the business-logic and data-services systems if they are on different servers.	The more layers of course, the more marshalling of data and requests, and so speed limitations can creep in.
Allows for (actually encourages) component-based development, which *can* increase reusability.	As with 3-tier, more infrastructure with MTS/COM+ and MSMQ.
Multiple medium servers are often cheaper than one large server. The separation of business and data services makes two servers an option.	

About .NET

.NET (pronounced "dot net") is a new development strategy being pushed by Microsoft. The latest updates of Microsoft's server products (including SQL Server 2000) are now being called the **.NET servers**. Microsoft will also be producing new versions of Visual Studio that will seek to simplify implementing .NET.

.NET is, as of this writing, still not a fully rounded-out vision, but it does have a number of proposed notions that are important for your architecture considerations. For now, what I would say is that it is important to recognize that all of the .NET servers are increasing their support for XML, and that they are doing a better job of talking to each other. For your component development, there are changes coming that will allow you to do an in-place replacement of a component without having to bring down your server to do it (remember I said, "There is typically still more downtime than a host system" when talking about n-tier?). This addresses one of the serious problems in development today – how to perform system upgrades gracefully.

More information will be coming about .NET over the next few years. The first implementations will only be the beginning of what appears to be a coming revolution in development on the Microsoft Platform. Microsoft has even indicated that it may take .NET onto Linux (could you imagine?). We'll see! In the mean time, watch .NET closely – not only for its first arrival, but also as it matures.

Data Access Models

Certainly one of the biggest issues to deal with these days is how to access your database within the various options in Microsoft's "Alphabet Soup" of **data access architectures**. Which models are available in what circumstances depends primarily on the version of SQL Server, and the choice and version of your programming language.

There are four different access models that Microsoft considers as being current for accessing SQL Server. These are:

- **ADO: ActiveX Data Objects**. The new RDO – or at least that's what Microsoft would like you to believe. Don't – ADO is its own animal. Each new version seems to improve performance and add features. Unlike RDO, which was based on ODBC, ADO is based on OLE DB. This provides a level of flexibility that ODBC alone can't offer, but it comes with more than just a few headaches (note that ADO can still indirectly use ODBC for connectivity by using the OLE DB provider for ODBC). RDO had a few features that will never be in ADO, but ADO has some really cool features – with persistent recordsets, filters, sorts (without going back to the server), and others – that RDO never had.

  It's tough to take just a brief look at ADO, but since I have to, I'll say this much: ADO is now competitive, if not faster, in terms of speed compared with RDO, and has a very robust feature set. It is still nowhere near RDO in reliability as of writing this, but Microsoft has and continues to make a substantial investment in ADO and OLE DB. This is where they are saying the future is, and I would suggest any new non-C++ based development uses this access method (there'll be more for the C++ crowd shortly).

- **ODBC: Open Database Connectivity**. If you've been developing for any length of time at all (say, for more than a week), you have to have heard of ODBC. It is a Microsoft-pushed standard, but it is most definitely a standard – and a very good one at that. ODBC provides a way of gaining cross-platform access to database information. It is quite fast (often as fast as or faster than the native driver for your database) and allows you to use most of the mainstream standard SQL statements regardless of what the back-end expects for syntax. In short, ODBC is very cool. The major shortcoming for ODBC is that it is very much oriented to tabular data (columns and rows), and doesn't deal with non-standard data such as a directory structure or a multi-sheet database.

- **OLE DB**. The primary competitor to ODBC at this point, OLE DB is an attempt at having an open standard to communicate with both tabular and non-tabular data. OLE DB uses what is called a **provider**. A provider is a lot like an ODBC driver except that it is relatively self-describing. That is, it is able to tell the application that uses it what kind of functionality it supports. As mentioned earlier, OLE DB is the foundation under ADO. It is very fast indeed when not being used with ADO (that extra layer adds some overhead) but, since it deals with a number of items that aren't compatible with VB, you'll typically only see OLE DB being used directly by C++ programmers. It is far more of a pain to program in than ADO, but it is much faster by itself than when used in conjunction with ADO. If you're a non-C++ programmer, stick with ADO at this point. If you're a C++ programmer though, you're going to need to figure out whether the extra speed is worth the hassle.

- **Java Database Connectivity: (JDBC)**. OK, so I said there were five, and there are, but I have to say that JDBC is still something of an outcast at this point. It is used primarily in the web arena and by non-Microsoft users. The last time I saw much of anything done with it, it was fairly simple to use, but had very little functionality and was very slow. In short – unless for some reason you absolutely have to, don't go there.

In addition to these current object models, there are a few others that you may run across and should know about:

- **RDO: Remote Data Objects**. This was the speed leader for a couple of years. DAO and ODBC used to be the only options for VB programmers. DAO was easier than ODBC to develop in, but it was slow, and the object model was still too deep and complex. ODBC on the other hand, was fast, but required tons of code just to get ready to make a connection.

  RDO recognized that most of the "set-up" code for ODBC was always the same, or was predictable based on a few other settings. It was created by using a very thing wrapper around ODBC that substantially simplified the code while giving most of the performance benefit of ODBC.

- **DB-Lib**. Prior to version 7.0, this was the native way in which SQL Server did all of its talking between the main host and client and utility applications (SQL Server now uses OLE DB natively in this role). It is still actively supported, but will only be enhanced to the extent necessary to maintain backward compatibility as SQL Server moves forward. Microsoft has said that it will pull support for this access method at some point in the future, but it also acknowledges that there are too many legacy applications out there using DB-Lib to figure on dropping support for it anytime soon.

- **VB-SQL**. This was only briefly available, but still found its way into several applications. This was based on an old wrapper that was written for VB to make many DB-Lib functions available for VB programmers.

  If you're still using VB-SQL, move off it as soon as possible. It is slow and, if it breaks, you'll get no help with it.

- **DAO: Data Access Objects**. This is actually native to Microsoft Access (more specifically, the Jet database that is at the heart of Access). There are a lot of applications written in VB and Access that use this technology. Too bad! This object model can be considered clunky, slow, and just plain outdated (believe me, I'm being nice and not saying what I really think). It's still the fastest way to access things if you're using a Jet (Access) database but, if you're using this technology to access SQL Server, I would suggest putting some serious effort into migrating away from it as soon as possible. Microsoft was calling DAO a "legacy" model more than a year before the end of the Office 97 lifecycle. They want people to stop using it, and I have to agree with them.

There are several books out there on accessing SQL Server and the data access side of the database relationship – I'm going to leave you to look through those for more information, but I will recommend that you check out Bill Vaughn's *The Hitchhikers Guide to Visual Basic and SQL Server,* Microsoft Press, ISBN 1-572318-48-1. It is the relative bible of the connectivity side of Visual Basic programming. Another source you may want to check out is *Professional Visual Basic 6 Databases*, Wrox Press, ISBN 1-861002-02-5. C++ programmers should look at *Visual C++ Database Programming Tutorial*, Wrox Press, ISBN 1-861002-41-6.

Microsoft Certification

There are three SQL Server related exams that participate in different certifications offered by Microsoft. These include exams on **development**, **administration**, and **data warehousing**. Note that, at the time of writing, there are only exams for SQL Server 7.0. The SQL Server 2000 exams are due to go to Beta in January 2001. If you are interested in becoming certified visit http://www.microsoft.com/technet/training/default.asp, where there is a wealth of information on how to study and get up to date.

Each of these exams has some relevance to the MCP, MCSD, MCDBA, and MCSE certifications. Indeed, the development and administration exams are core to the MCDBA certification process.

This book was not purposely written to address any Microsoft exam – it is focused on trying to successfully prepare you to develop applications using SQL Server 2000. Even so, the development exam (70-029 for SQL Server 7 and 70-229 for SQL Server 2000) was written to try and test to see if you are ready to do just that – and almost everything I can think of that's covered in the exam is addressed somewhere in this book.

I am not going to tell you that if you read this book you will pass that exam. I participated in the authoring of that exam, and I have had to take it myself – it is one seriously nasty exam. Still, the topics covered in this book happen to speak right to the heart of the exam – if you do well going through this book, the odds are you'll do just fine on the exam (sorry folks – this isn't exam cram so no guarantees on that!).

Summary

We've made a start and talked briefly about where the database world has been, database access, and a few other miscellaneous items. This is really just conducting some bookkeeping and preparation to get you ready to go.

In our next few chapters, we're going to take a deeper look into many of the basics of building and making use of a SQL Server database. I strongly encourage you to run through the many examples in this book. As I mentioned earlier, there are very few concepts in this book that do not have specific examples associated with them – take advantage of that and you'll be taking full advantage of this book.