Designing Scalable .NET Applications

JOACHIM ROSSBERG AND RICKARD REDLER

Apress™

ISBN (pbk): 1-59059-214-X

Printed and bound in the United States of America 12345678910

Technical Reviewer: Phil Pledger

Editorial Board: Dan Appleman, Craig Berry, Gary Cornell, Tony Davis, Steven Rycroft, Julian Skinner, Martin Streicher, Jim Sumser, Karen Watterson, Gavin Wray, John Zukowski

Assistant Publisher: Grace Wong

Project Manager: Tracy Brown Collins

Copy Editor: Ami Knox

Production Manager: Kari Brooks

Production Editor: Laura Cheu

Compositor and Artist: Kinetic Publishing Services, LLC

Proofreader: Thistle Hill Publishing Services

Indexer: Rebecca Plunkett

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, email orders@springer-ny.com, or visit http://www.springer-ny.com. Outside the United States: fax +49 6221 345229, email orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, email info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Downloads section. You will need to answer questions pertaining to this book in order to successfully download the code.

CHAPTER 8

Data Storage Design and SQL Server

All applications you build store data in some way or another. It has become vital for enterprises to increase efficiency and reduce Total Cost of Ownership (TCO) for data storage as well as all other infrastructure components.

Many factors must be considered when planning storage: Do you need data to be available 24 hours a day? What recovery times can you accept if data is compromised in any way? How much will your data grow with time? Are you using your data storage architecture effectively as it is?

It is not only your applications that need to be designed carefully. When it comes to storing data, you need to weigh an increasing number of issues, not only for the application you are currently building, but also for the entire enterprise data storage design. Considering how much important data is actually stored at a given company, it is surprising that many do not take it as seriously as they should.

Many enterprises create new database servers for almost every new application. This has resulted in data being scattered all over such companies and storage areas not being fully utilized. The scattered data has made manageability harder and more costly than it should be. By not using data storage effectively, a lot of investment is wasted, costing money in the long run.

Availability, scalability, and security have been emphasized throughout this book. These topics are important not only when building applications, but also when it comes to designing storage architecture. You can use various technologies to increase these aspects of an application, but they have some trade-offs that you need to consider. Availability always costs money. Redundant technology comes with a rather hefty price tag, and this is true in development and in data storage alike. When you try to increase scalability, availability can suffer. This requires you to purchase even more hardware to avoid problems.

Security most often affects performance and manageability due to the added overhead and complexity of secure solutions. The more you tighten security, the more performance and manageability suffers as a direct result.

You need to design your storage solution to meet goals of lowered TCO, high availability, great scalability, tight security, and, of course, simplified management. That is quite a lot to live up to, as you can see. When you design your storage solution, you must plan for enabling your organization to quickly create new business applications, and at the same time provide security and minimize the potential of

data loss. You must also make sure new applications can be implemented with minimum disruption to your existing business. The decisions you make for the data storage architecture have an impact on how you design your future business projects.

The tasks facing a data storage team are not easy ones, so first we will take a look at some of the technologies for storing data and see how they can ease the burden. Then we will move on to choosing and implementing a logical design for a storage system before diving into SQL Server.

This chapter will explore the most common storage technologies available to you. It will also present some benefits and concerns about different storage designs so that you can more easily choose what suits your solution. We will finish the chapter with a presentation of SQL Server—examining its architecture and how it fits into your storage policy.

Three Storage Technologies

There are basically three different storage technologies you can use: Storage Area Networks (SANs), network-attached storage (NAS), and direct-attached storage (DAS). They each have their advantages and disadvantages, naturally; but by combining them, you can perhaps overcome their individual shortcomings. We will look at these in more detail in the following sections.

Storage Area Networks (SANs)

A SAN is a specialized network, the sole purpose of which is to provide access to high performance and highly available storage subsystems. This solution is quite interesting. The SAN is constructed of several devices, all interconnected by fiber or copper wiring. The subsystem is available to multiple applications at the same time, just like a network-attached storage appliance. The major difference is that it provides higher scalability and performance. Even though a DAS is still faster than a SAN, this performance gap is diminishing constantly with the evolution of SAN technology. (For more on DAS, see the section "Direct-Attached Storage.")

The SAN concept is quite simple. An external RAID (short for *redundant array of independent disks,* sometimes called *redundant array of inexpensive disks*) cabinet has a connection directly from the Host Bus Adapter (HBA) to the external RAID subsystem. The SAN, on the other hand, connects the HBA to a switch instead, providing other servers access to the data storage that way (see Figure 8-1).

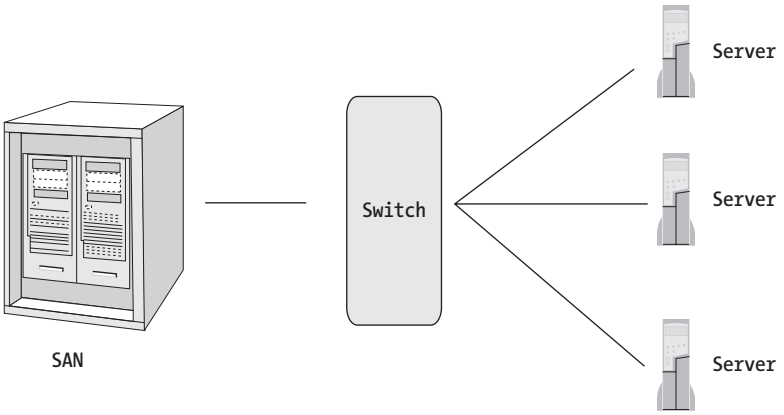


Figure 8-1. A SAN connects the HBA to a switch.

Next we will take a closer look at NAS and DAS, after which we will introduce you to the logical designs of storage solutions.

Network-Attached Storage (NAS)

A network-attached storage system is a file server, to put it simply. This solution is built on an operating system, often designed for the sole purpose of providing file services. It thereby provides a flexible and scalable solution for most enterprises. Many applications can access a NAS appliance over the LAN by using protocols like TCP/IP. Since this solution is accessed across a LAN, performance might be slow. When compared to a DAS system, performance is lower, but with the help of Gigabit Ethernet you can decrease this performance hit.

Windows Storage Server 2003

Microsoft has developed a dedicated file server based on Windows Server 2003. This edition of the Windows Server family is called *Windows Storage Server 2003* and is only available through OEMs. The Storage Server is enhanced to offer a scalable, available, quickly recoverable solution for enterprise file servers.

Besides being a file server, the Windows Storage Server 2003 also acts as a gateway to SANs, so you are not limited to the features offered in the server itself. To learn more about this server, direct your Web browser to `http://www.microsoft.com/storage`.

Direct-Attached Storage (DAS)

As the name implies, direct-attached storage refers to a storage device directly attached to a server, specifically local hard drives and RAID systems attached with an IDE or SCSI interface to a computer.

This solution is fast and allows great performance on the server it is attached to, and often only the attached server can access it. This could be a great solution for a small SQL Server perhaps, but if you are planning a larger enterprise application with clustering, this is not a good choice, because many servers might need to access it to provide redundancy.

Logical Design

We feel that it is better for an enterprise to develop a storage architecture that all projects comply with, rather than letting every project develop its own strategy. With a well-thought-out plan for storage, teams working on new projects can spend time on designing and building applications without bothering with planning how to store data. Furthermore, the more you centralize your storage, the more you can simplify management and decrease complexity.

One of the buzzwords these days is *consolidation*. Everything should be consolidated: databases, servers, applications, and so on. This is not a bad idea if you look behind the hype. Take storage, for example: Many companies have little control over where all data is located, since a lot of smaller databases are distributed around their networks. If you would have a strategy or a policy that dictates all data be located at the same place, you could utilize your storage space better, as you will see later in this chapter in the section "Centralized Model." SANs and NASs are great for accomplishing this.

To illustrate this, we will now show you some examples of how you can develop your logical design for storage purposes.

Distributed Model

In a distributed model solution, such as the one shown in Figure 8-2, all servers have their own data stored on a DAS. An example that fits this model is a SQL Server that has its data files and log files on local RAID systems.

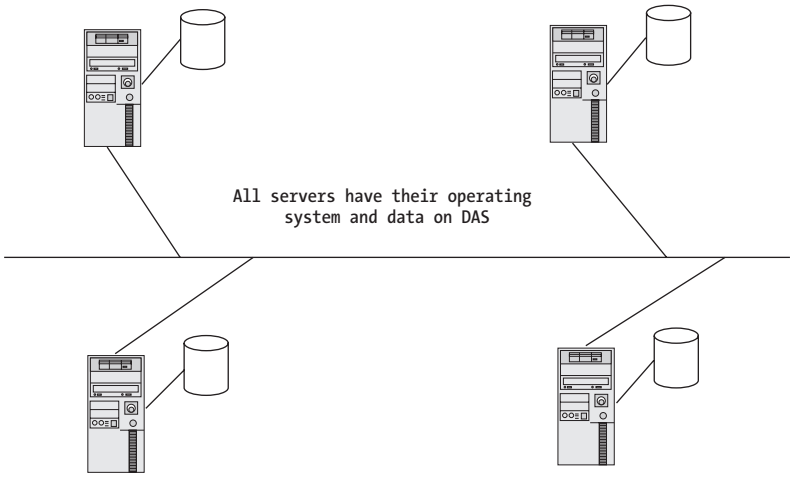


Figure 8-2. A distributed storage design

Although there are several benefits, such as flexibility, to this design, the disadvantages can outnumber these benefits. If you deploy this kind of solution, all projects need to have expertise in data storage applied during the design phase. The same storage decisions have to be made again and again, and this process is costly.

Because data will be scattered in data islands all over the network, manageability will suffer. As you saw earlier, this situation can also lead to underutilization of the total storage capacity in the enterprise. This will inevitably lead to greater TCO than is necessary, making it an expensive solution in the end. Although this might seem like a cost-effective path to follow for a single project, the total cost for your company will be greater as time goes by.

Centralized Model

The opposite of a distributed solution is a completely centralized one, where all data is located on the SAN (see Figure 8-3). In this model, the operating system and application files are the only things placed on local hard disks. By using a powerful SAN, you can supply a solution that can fulfill the business requirements of an enterprise.

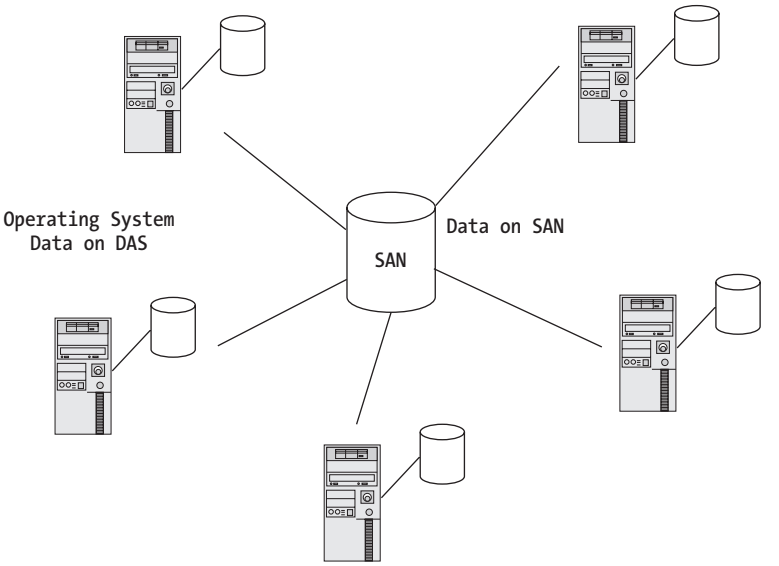


Figure 8-3. A centralized storage design

The advantages of this solution are numerous. Since you keep all data in a central place you can also manage it easier. This lowers the TCO and makes it less expensive in the long run. You must be aware that the startup costs are high when implementing this design, however, but over time it is worth it. This is why it is important for an enterprise to have a clear view of how data should be stored from the start. If no guidelines exist, many projects will continue to invent their own solution every time. In the end, this will be more expensive than if a clear policy exists, even if the chosen policy might have a high initial implementation cost.

Another benefit is that you can provide high security for the enterprise data, since you have it all in one place. This security does not come without manual labor, however, but is easier to set up with a centralized solution.

A disadvantage with this solution, besides initial costs, is that you cannot distribute your data store to separate geographic locations. So if you are unlucky, you can lose a lot of data if the geographic placement of the SAN is destroyed. Without a doubt, this is sure to change in the near future; in the meantime you do have opportunities available to back up data to a separate geographic location, so you can overcome this problem.

A Middle-Way Design

The last example is a middle-way data storage design. Here you use various technologies to store your data. Some of it is stored on DAS, some on SAN, and the rest on NAS appliances. This solution is very flexible, and offers several possibilities for

your projects. You can let team leads on each project decide what best serves their purposes.

This could be a more complex design to implement, however. Many design considerations have to be made, just as with the distributed model, making the process long and complicated before it can be implemented. Without proper guidelines, it will also be costly. Since this solution involves a lot of storage technologies from different vendors, TCO risks being higher than a centralized solution.

Choosing Your Storage Solution

Once you have determined the requirements for your storage solution, you must decide how you want to go about meeting those requirements.

Reasons for Choosing a DAS

A DAS is a popular choice for storing data. It is a solution with a low entry cost, because hard drives are cheap these days. It is tempting for a department or a workgroup to choose this technique, since it is easy to implement. A small organization or department of a larger organization often does not want to spend time and money on shared storage or on the enhanced availability and higher performance that comes with it. Our view is that this might harm the organization in the end. A large number of DAS solutions might very well be complex and expensive to manage. The hardware often comes from different vendors, so standardization might suffer. This could lead to isolated data islands that might not be accessible to more than the server where the hardware is attached. From an enterprise view, such a solution is not what you should aim for.

On the other hand, a DAS solution is easy to launch. Most administrators feel at home with this kind of technology. Another issue is performance. Because DAS is directly attached to the server, performance is great. Depending on the data, it can also be hard to get more than one server to access it at the same time, so a SAN or NAS appliance might be a better choice if you need the data to be available for many servers or applications. And in a large enterprise, you can bet that this is the case. It is true, however, that a database with database files stored on a DAS can be accessed by many applications, but then you will have the resultant manageability issues to deal with. You must be aware that access to the data is limited to the number of physical network connections the server can handle at the same time, however.

We would recommend using DAS when you need to have a database close to your Web server, and for use as a caching server—that is, a server that you only use for data immediately needed for your Web solution, thereby providing fast access

to this data. All your user data, customer data, product data, and so on should be stored on a more permanent basis in a centralized data store like a SAN. The contents of your user sessions information, shopping baskets, and the like should be stored in your DAS caching database. This way you provide fast access of such info for your Web solution.

Reasons for Choosing a NAS

Network-attached storage is a scalable and flexible way to fulfill your enterprise file-sharing needs. This technology, like DAS, offers great simplicity, and furthermore most administrators know about it. You can easily administer security and access control for this solution. Access to a network-attached storage system is limited to the LAN, however, and if you do not want to use Gigabit Ethernet, this can be a potential bottleneck.

Compared to a SAN, a NAS system is simpler and less expensive. SANs require additional hardware and cables, and a lot more effort has to go into the design of it. The fact that you attach it directly to the LAN infrastructure also makes it complex to implement.

In addition, you might run into problems with your database applications, because not all of them support storing database or log files on network-attached storage appliances.

Your backup solution is something you also have to consider. It must allow you to back up over the LAN; otherwise, you might run into trouble. Even if it allows you to back up over a LAN connection, you must plan your backup schedule carefully. You do not want to run your backups when network traffic mostly consists of business-critical operations.

Reasons for Choosing a SAN

SANs make it possible to implement highly available storage subsystems to multiple hosts. *SAN fabrics* are networks that connect hosts to storage devices. By using fabric switches, you can connect separate SAN fabrics, or *SAN islands* as they are sometimes called, to each other.

One of the benefits of SAN is that you can expand it quite easily. You can add new storage devices to your SAN when you need to without interrupting the existing structure in the meantime. This means you can scale the SAN when the need for it arises.

Another benefit is that backup and restore operations are done over the SAN fabric. This way you get high speed on these actions and at the same time minimize impact on the LAN. If you want to implement a server-free backup, you can

do so by using the appropriate hardware, but you should be aware that this takes a few extra configurations to maintain integrity of your application data.

It is also possible to manage the SAN from one place, which makes manageability easier. You can pool storage together and allocate it to those servers that need it.

One of the problems with a distributed model has been that it often has a lot of excess storage, and thus does not utilize resources as well as it could. With a SAN, the excess storage can be fully used, and nothing is wasted.

All this might sound great, but there are, of course, disadvantages. A SAN is very expensive to implement, as you have seen. Luckily, most of the expense occurs at the initial implementation, and the centralized management as well as the reduction of wasted storage and greater flexibility makes the TCO decrease over time.

NOTE *One of our customers has implemented a SAN over the last year. Although it has cost a lot of money, it seems like administrators and users alike are satisfied with the solution. At the same time, the customer has implemented SharePoint Portal Server and now has a truly centralized solution in place.*

A SAN also demands a lot more from your administrators, since they might need to learn a new technology. If a SAN is not already implemented in the company, once you start planning for it, you soon find that it is a complex solution. A SAN requires specialized knowledge to design, administer, and maintain.

If you could choose, a SAN would be perfect for your applications, but unfortunately the choice is not always yours to make. Implementing a SAN requires an enterprise to develop a new storage architecture and policy. Since the initial costs are high, many might hesitate at this, especially since the long-term perspective often is forgotten in the cost-aware times in which we are living.

Consider Combinations

Rather than choosing just one the preceding technologies, you could use various combinations of these technologies. Many new NAS devices also handle access to SANs (see Figure 8-4). This increases their scalability, while still offering the benefits of network-attached storage. You can also improve backups and restores, because this setup lets you perform these operations to backup devices on the SAN. This way, you get server-free backup of your network-attached storage devices as well.

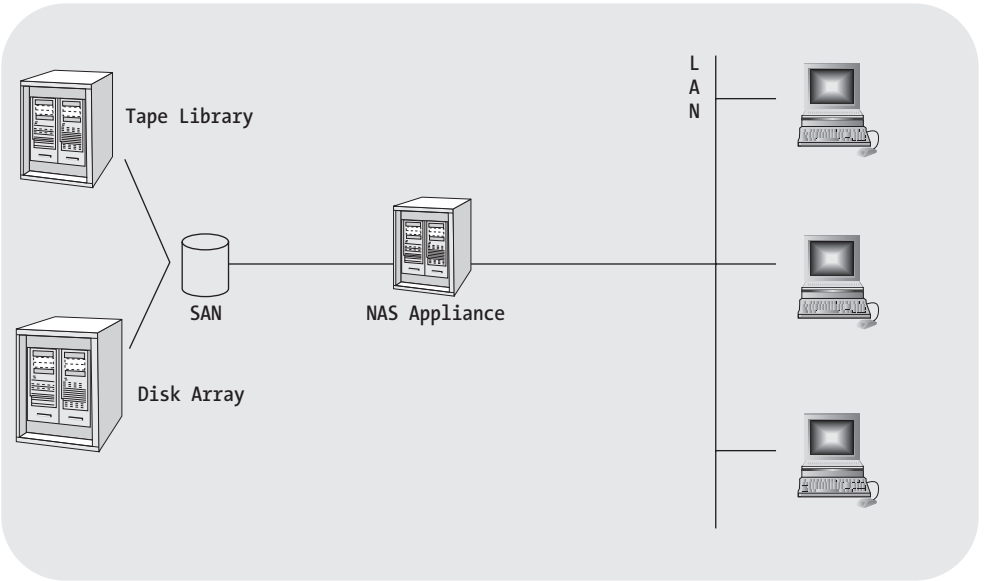


Figure 8-4. A NAS appliance that also provides access to SANs

By using this combination design, you can take advantage of the best features from the different storage technologies. At the same time, you get a great deal of flexibility and high performance. This combination is not perfect, however. The price tag is not something everybody could afford. You must also be aware of the increasing complexity in your solution, since you mix hardware from different vendors. Naturally, this affects operability and manageability.

You have to consider what kind of storage solution you actually need and the requirements your business has of your storage before you run out and purchase anything. Design is always something you should not take lightly, and it is often worthwhile to take a little extra time to think about it early on.

Next we will move on to our discussion of SQL Server. SQL Server is a powerful database manager, and it is important to know how it works and how it can be used in a storage policy. We will show you the different versions you can use, examine the architecture of SQL Server, and share some tips for working with SQL Server that we have learned over the years.

Introduction to SQL Server

We have been working with SQL Server since version 6.5 or so. Back then, we would never have suggested to a customer that they should use SQL Server in a business-critical application. SQL Server has fortunately grown since then, and with the 2000 edition, it has become one of our most commonly suggested applications to customers these days. We cannot wait for the next version to come out, as it looks very promising on paper.

SQL Server still has a reputation to fight against, however. A lot of people think it cannot perform as well as many of its competitors. That is not true if it is configured correctly, and if it is deployed on a good platform.

TIP *For those interested in SQL Server performance, check out* `http://www.tpc.org` *to see the latest test results and database system configurations.*

SQL Server's ease of use has also been an issue over time. The general misconception has been that something that is easy to use with a graphical user interface is as worthless as a Steven Seagal movie. This in effect has led many companies to give someone like a department secretary the responsibility of setting up and managing a SQL Server, while the database administrator handles an Oracle database or the like. Now we do not intend any disrespect to the secretary, but this is not a good solution, since getting great performance out of SQL Server requires the skills of a trained and experienced database administrator.

Many database administrators have not had an interest in working with SQL Server because it is so easy to use and not much configuration has to be done once it is set up. Setting up a new SQL Server takes a good deal of planning, however, and knowledge about how to do it the right way.

Fortunately, much of this attitude has changed, and nowadays trained IT staff handle most SQL Servers in enterprises.

Several versions of SQL Server are available. To help you choose the most suitable for your solution, we will present an overview of the different versions and their intended use in the following sections.

SQL Server Editions

SQL Server, like Windows Server, comes in various editions to suit a large number of uses:

- SQL Server 2000 Enterprise Edition
- SQL Server 2000 Enterprise Edition (64-bit)
- SQL Server 2000 Standard Edition
- SQL Server 2000 Developer Edition
- SQL Server 2000 Personal Edition
- SQL Server 2000 Desktop Engine (MSDE)
- SQL Server 2000 Windows CE Edition (SQL Server CE)

We will explore each edition and see what its intended use is. As you learn about each edition, you will soon see that now you actually do have a version for your department secretary, another for your developers, and others for your database administrators. Now you do not have to compromise—you can have the database you need for every occasion. Let us start with a look at the Enterprise Edition.

SQL Server 2000 Enterprise Editions

SQL Server 2000 Enterprise Edition is the top-of-the-line of SQL Server. It includes all features that SQL Server offers. The Enterprise Edition comes in both 32-bit and 64-bit versions, so you can use it on most hardware available. This edition includes tools that let you not only store data, but also extract and analyze it.

SQL Server 2000 Enterprise Edition is a suitable solution for large Web sites, Online Transaction Processing (OLTP) systems, and data warehouses. Good platforms for running the Enterprise Edition are Windows Server 2003 Datacenter and Windows 2000 Server Datacenter. You should definitely let an experienced database administrator handle this edition.

SQL Server 2000 Standard Edition

SQL Server 2000 Standard Edition includes the core functionality of SQL Server. It does not include all the advanced tools that the Enterprise Edition does, but it does have some great tools that let you analyze data, too.

The Standard Edition is intended for small and medium-sized organizations. Recommended platforms include Windows Server 2003 Enterprise Edition and Windows Server 2003 Standard Edition, and the equivalent Windows 2000 Server Editions. We still recommend an experienced database administrator for handling this edition, however.

SQL Server 2000 Developer Edition

Basically, SQL Server 2000 Developer Edition is the Enterprise Edition in disguise. It is intended for developers to use when new applications and solutions are developed. By offering all the features of SQL Server, a developer can use it to develop all kinds of applications. It also includes licenses and download rights for the CE Edition, just to make everything SQL Server has to offer available at a reasonable price.

NOTE *Microsoft has cut the price down to $49 at the time of this writing, so this edition is affordable. One of the major reasons for this price cut is the threat posed by free database tools like mySQL.*

SQL Server 2000 Personal Edition

Most companies have mobile users, such as salespeople, who on their business trips need to have large sets of data available. SQL Server 2000 Personal Edition is the ideal choice to install on a laptop, for it lets users sync data with the company database when they are at the office.

NOTE *The Personal Edition is only available with the Enterprise and Standard Editions. It is not available separately.*

The Personal Edition can also be of use for the department secretary (you knew this was coming) when he or she requires an application that needs to store data on a local database. It does not offer the scalability and flexibility of the previously mentioned editions, but it is nevertheless a great tool in some cases.

SQL Server 2000 Desktop Engine

The Desktop Engine is a redistributable version of the SQL Server Database Engine (MSDE). It is not available separately. On the Enterprise, Standard, and Developer Edition CD-ROMs it is included in the MSDE catalog in the root directory. Developers can include it with their applications when local data is needed and requiring a stand-alone database would be overkill.

TIP *Check* `http://www.microsoft.com/sql/` *for more information about obtaining MSDE, and the licensing issues that may come with it. There you will find all the information you need about the various SQL Server editions.*

If you need to, you can use SQL Server Enterprise Manager to access the Desktop edition, and this way get a graphical user interface to it. Otherwise, this edition does not include any GUI.

SQL Server 2000 Windows CE Edition

SQL Server 2000 CE Edition is intended for use on PDAs and other Windows CE devices. It has a small footprint to enable it to run smoothly on these devices, but it still provides tools that let you store data on a Pocket PC, for instance. It also includes APIs so you can develop applications for your Windows CE devices that use a full-fledged SQL Server as a data store, instead of flat files or any other solution.

Table 8-1 shows the system requirements of the various SQL Server Editions.

Table 8-1. SQL Server 2000 Editions Overview

Edition	Operating System	Scalability
Enterprise Edition (64-bit)	Windows Server 2003 Enterprise Edition; Windows Server 2003 Datacenter Edition	Max 64 CPUs, max 512GB RAM, max 1,048,516TB database size
Enterprise Edition (32-bit)	Windows Server 2003 Standard Edition, Enterprise Edition, Datacenter Edition; Windows 2000 Server, Advanced Server, Datacenter Server; Windows NT 4.0 Server, Enterprise Edition	Max 32 CPUs, max 64GB RAM, max 1,048,516TB database size
Standard Edition	Windows Server 2003 Standard Edition, Enterprise Edition, Datacenter Edition; Windows 2000 Server, Advanced Server, Datacenter Server; Windows NT 4.0 Server, Enterprise Edition	Max 4 CPUs, max 2GB RAM, max 1,048,516TB database size
Developer Edition	Windows Server 2003 Standard Edition, Enterprise Edition, Datacenter Edition; Windows XP Professional; Windows 2000 Professional; Windows 2000 Server, Advanced Server, Datacenter Server; Windows NT 4.0 Server, Enterprise Edition	Max 32 CPUs, max 64GB RAM, max 1,048,516TB database size
Personal Edition	Windows Server 2003 Standard Edition, Enterprise Edition, Datacenter Edition; Windows XP Professional; Windows 2000 Professional; Windows 2000 Server, Advanced Server, Datacenter Server; Windows NT 4.0 Server, Enterprise Edition	Max 2 CPUs, max 2GB RAM, max 1,048,516TB database size

Table 8-1. SQL Server 2000 Editions Overview (continued)

Edition	Operating System	Scalability
Desktop Engine	Windows Server 2003 Standard Edition, Enterprise Edition, Datacenter Edition; Windows XP Professional; Windows 2000 Professional; Windows 2000 Server, Advanced Server, Datacenter Server; Windows NT 4.0 Server, Enterprise Edition	Max 2 CPUs, max 2GB RAM, max 2GB database size
Windows CE Edition	Windows CE 2.11 or later, Handheld PC Pro (H/PC Pro), Palm-size PC (P/PC), Pocket PC	Max 1 CPU, max 2GB database size

SQL Server Architecture

We have already mentioned that SQL Server is quite easy to use, but this can also be one of its drawbacks when it comes to people's perceptions of this product. As long as you understand what it does and when it behaves in a wrong way, we argue that there is no value in having to tune a database server manually when you can let the built-in logic do it. You want your database administrators and IT staff spending time on other stuff, like designing, planning, and deploying applications. Obviously such staff would still need to provide maintenance and surveillance of your existing SQL Server system; if it behaves strangely, they must have the knowledge to troubleshoot it. This should not be their main occupation, however. A database that requires IT staff to spend hours and hours just to get it up and running takes too much time from the IT staffs' other tasks.

To get a better understanding of SQL Server, we will outline its architecture in the following sections.

Memory Management and Memory Architecture

Memory management is one of the things that in SQL Server requires little or no manual work, at least not in most cases. SQL Server by default dynamically allocates and deallocates memory as needed. This way SQL Server itself optimizes memory for best performance, based on the amount of physical memory it has to work with.

SQL Server uses the virtual memory in Windows for two main components (see Figure 8-5):

- The memory pool
- Executable code

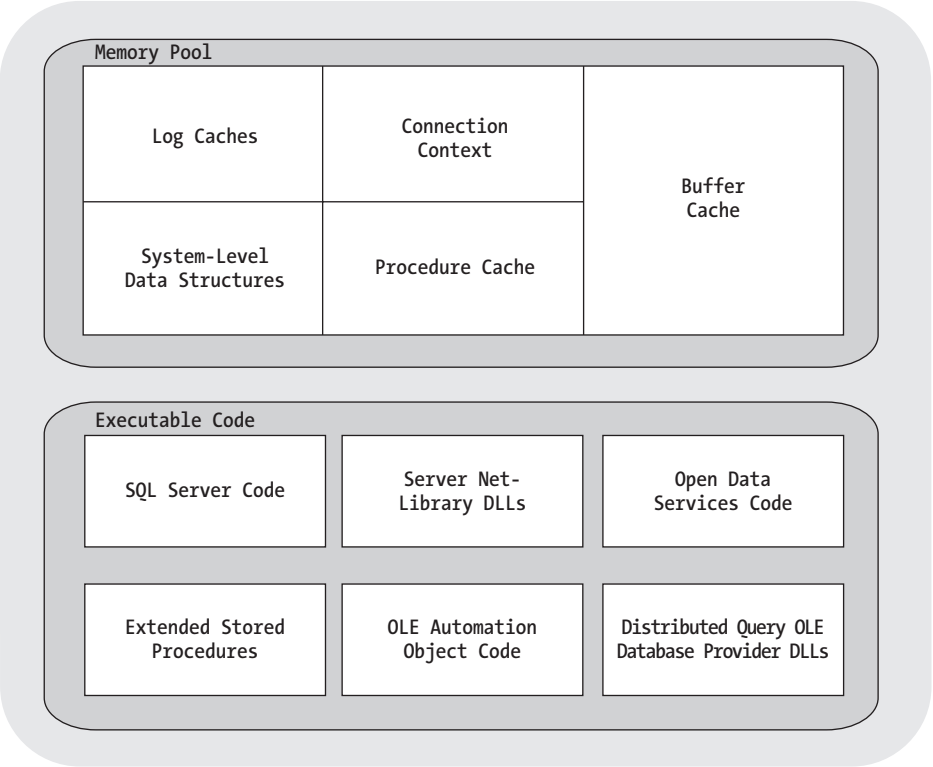


Figure 8-5. SQL Servers' two main components in the SQL Server address space

The memory pool is an area of memory from which some objects allocate their memory. These components are as follows:

- *System-level data structures:* These hold data that is global to the SQL Server instance, such as database descriptors and the lock table.
- *Buffer cache:* This cache stores data and index pages read from a database. If these items need to be accessed again, they can be found in the cache, which speeds up processing.
- *Procedure cache:* This holds the execution plans so that they can be reused for stored procedures and Transact-SQL statements.
- *Log cache:* The log cache is used to hold pages that are read from and written to the log. Each database has one log cache.
- *Connection context:* This constitutes a set of data structures that keeps a record of the current state of a connection.

Determined by the number of user requests, the size of the memory pool varies, simply because the objects within it vary in size. When a new database is defined, the different data structures are allocated. The same goes for when a table or view is referenced.

The buffer cache, log cache, and procedure cache are constantly managed and adjusted by SQL Server to optimize performance so it fits the current workload. So if the procedure cache needs to have more memory, SQL Server dynamically increases the whole memory pool size. It does so by allocating more memory from the physical memory. If no more physical memory exists, it tries to resize the other objects in the memory pool.

You can control the size of the memory pool by setting minimum and maximum values for it. You do not need to control the size of the objects within the memory pool, as SQL Server handles this best alone.

The executable code is more or less the SQL Server engine. It includes the code for the engine itself, as well as the DLLs and executables for Open Data Services and Net-Libraries.

Let us take a closer look at the buffer cache, since this is an important part of SQL Server. The buffer cache consists of a number of memory pages that are initially free when SQL Server starts up. When SQL Server starts reading a page from disk, this page is stored in the first page of the buffer cache's free list. When this page is read or modified by the same process or another, it is read from the cache instead of from disk. This reduces the amount of physical I/O required for the operation, and performance increases.

A page that has been modified in the buffer cache but not yet written to disk is called a *dirty page*. Whether a page is dirty or not is written to each buffer page's header information. A reference counter also resides in the header, which is incremented and decremented with each reference to the page. The buffer cache is scanned periodically, and if a page has been referenced less than three times since the last scan and is not dirty, the page is considered to be free. It is then added to the free list, which we talked about in Chapter 4. If the page is dirty, the modifications are written to disk first. This work is done by the worker processes of SQL Server.

To prevent the free list from being too small, a worker process called the *lazywriter* periodically checks to see that the free list does not fall below a certain size. The size depends on the size of the buffer cache. If the free list is becoming too small, the lazywriter scans the cache and reclaims unused pages and pages that have a reference counter set to zero.

NOTE *The lazywriter is mostly used in very I/O-intense systems, because the other threads can handle these tasks on less heavily used systems.*

To allow the buffer cache to have as much memory as possible, you should see to it that your system has enough physical memory (RAM) for the task it performs. Memory is cheap these days, so this does not have to be too expensive even for small companies.

Memory Configuration in SQL Server

As we have mentioned, SQL Server dynamically adjusts the memory for the memory pool. It not only allocates memory for its parts, but it also deallocates memory if other applications need it. But it only deallocates memory if another process asks for it; otherwise it maintains memory at the current size.

The virtual memory SQL Server uses is maintained at 5MB less than the available physical memory. This stops excessive memory swapping to disk, while still giving SQL Server the largest memory pool possible. SQL Server always tries to keep 5MB of free physical memory on the system by deallocating memory from its pool, if another process tries to access these megabytes.

To make sure SQL Server always has a certain minimum memory pool, you can set the *min server memory* option so SQL Server does not release memory less than this value. The min server memory option is always set in megabytes.

You can also set the *max server memory* option so that SQL Server does not steal too much memory from other applications running on the server.

Carefully use these options so that no application (including SQL Server) starts any excessive swapping. You can keep an eye on this by using the Performance MMC (see Figure 8-6) and the Pages/sec counter of the memory object. This counter shows the paging on the system. A value between 0 and 100 is considered normal.

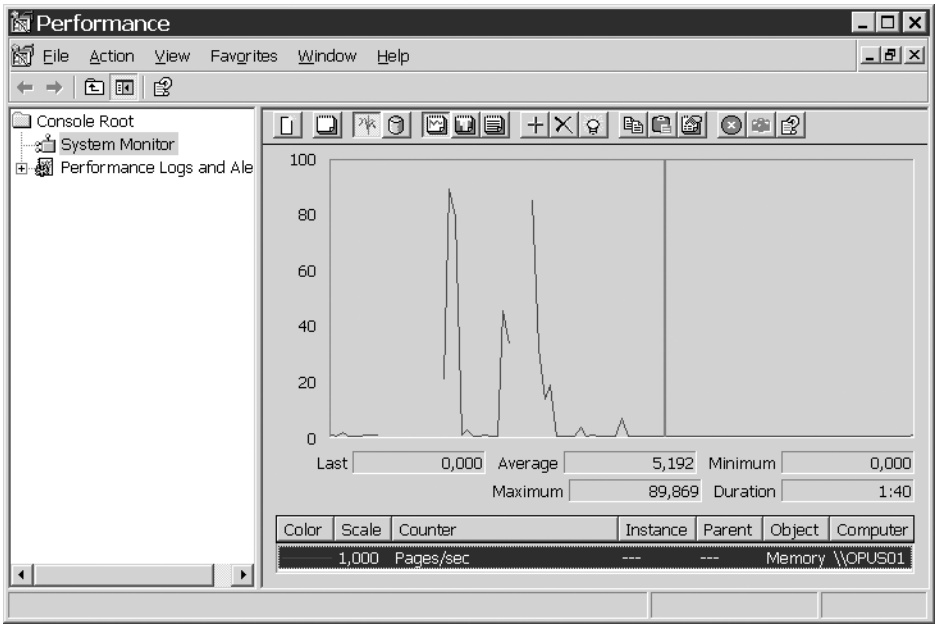


Figure 8-6. The Memory: Pages/sec counter in the Performance MMC

The best approach is to let SQL Server handle memory usage and not to set the manual options. This is especially true when only SQL Server runs on the machine. When you have other applications running, you might need to manually configure these settings, however. With all the consolidation going on these days, it is not unlikely that SQL Server has to share room with other applications, so it could be nice to know these options are available. The following code shows how to set these values from Transact-SQL:

```
sp_configure 'show advanced options', 1
go
sp_configure 'min server memory', 256
go
RECONFIGURE WITHOUT OVERRIDE
go
```

You can also set minimum and maximum memory values from Enterprise Manager if you want to, as shown in Figure 8-7.

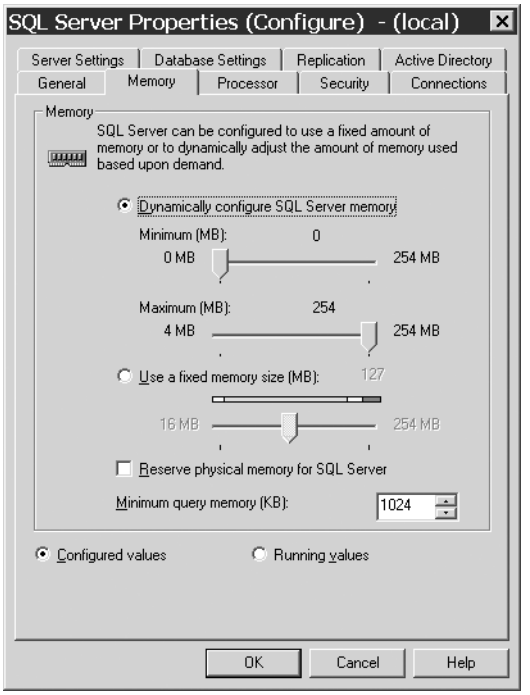
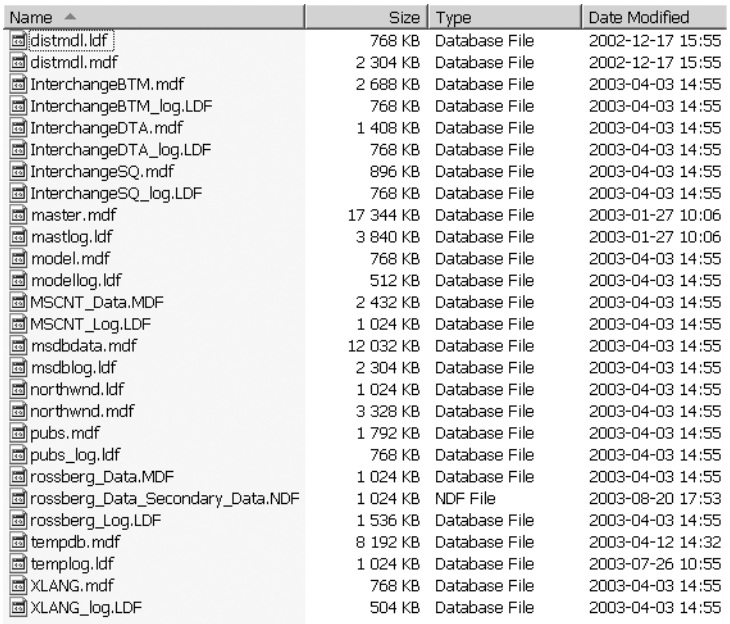


Figure 8-7. Setting the memory pool size from SQL Server Enterprise Manager

Data Files and Log Files

No database manager would be of any value if you did not store your data somewhere. SQL Server stores its data in a set of operating system files. Each file is made up of smaller parts called *pages*. There are three kinds of files that SQL Server uses (see Figure 8-8):

- Primary data files
- Secondary data files
- Log files



Name	Size	Type	Date Modified
distmdl.ldf	768 KB	Database File	2002-12-17 15:55
distmdl.mdf	2 304 KB	Database File	2002-12-17 15:55
InterchangeBTM.mdf	2 688 KB	Database File	2003-04-03 14:55
InterchangeBTM_log.LDF	768 KB	Database File	2003-04-03 14:55
InterchangeDTA.mdf	1 408 KB	Database File	2003-04-03 14:55
InterchangeDTA_log.LDF	768 KB	Database File	2003-04-03 14:55
InterchangeSQ.mdf	896 KB	Database File	2003-04-03 14:55
InterchangeSQ_log.LDF	768 KB	Database File	2003-04-03 14:55
master.mdf	17 344 KB	Database File	2003-01-27 10:06
mastlog.ldf	3 840 KB	Database File	2003-01-27 10:06
model.mdf	768 KB	Database File	2003-04-03 14:55
modellog.ldf	512 KB	Database File	2003-04-03 14:55
MSCNT_Data.MDF	2 432 KB	Database File	2003-04-03 14:55
MSCNT_Log.LDF	1 024 KB	Database File	2003-04-03 14:55
msdbdata.mdf	12 032 KB	Database File	2003-04-03 14:55
msdblog.ldf	2 304 KB	Database File	2003-04-03 14:55
northwnd.ldf	1 024 KB	Database File	2003-04-03 14:55
northwnd.mdf	3 328 KB	Database File	2003-04-03 14:55
pubs.mdf	1 792 KB	Database File	2003-04-03 14:55
pubs_log.ldf	768 KB	Database File	2003-04-03 14:55
rossberg_Data.MDF	1 024 KB	Database File	2003-04-03 14:55
rossberg_Data_Secondary_Data.NDF	1 024 KB	NDF File	2003-08-20 17:53
rossberg_Log.LDF	1 536 KB	Database File	2003-04-03 14:55
tempdb.mdf	8 192 KB	Database File	2003-04-12 14:32
templog.ldf	1 024 KB	Database File	2003-07-26 10:55
XLANG.mdf	768 KB	Database File	2003-04-03 14:55
XLANG_log.LDF	504 KB	Database File	2003-04-03 14:55

Figure 8-8. The different files in SQL Server

Primary data files contain different database information like startup information for the database, pointers to the other files of the database, system tables, and objects. These files can also contain database data and objects. There is one primary data file for every database, and the file extension through which you can identify this file is .mdf.

Secondary data files, which sport the extension .ndf, are an option for every database. You do not need to have one of these, but can instead use the primary data file if you want. Secondary data files hold data and objects that are not in the primary file. Here we find tables and indexes. The reason you might want to use secondary files is that you can spread the data on separate disks, thereby improving I/O performance.

Log files are files that hold all the information about the transactions in the database, and they cannot be used to hold any other data. You can have one or more log files (like the ones shown in Figure 8-8 with an extension of .ldf), but our recommendation is to use one only. This is because the log file is written to sequentially, and hence does not benefit from being separated on several physical disks. To enhance performance, you should instead try to use RAID 1+0, and place the log file on such hardware.

One important thing to understand about the log file is that it actually consists of several files, regardless of whether you use only one or not. These files are called *virtual log files* (VLFs) and are created by SQL Server (see Figure 8-9). Of

these virtual log files, only one is active at a time. When a backup of the log is performed, all inactive VLFs are cleared. The active VLF is never cleared, so contrary to what you might have been told, a log backup does not clear the whole log. But do not worry, this is not a problem.

Figure 8-9. The virtual log files in a log file

If you have too many VLFs, the overhead of maintaining them can degrade performance. To view how many VLFs exist in your databases, you could use an undocumented DBCC command, which returns as many rows as there are VLFs in the database. Execute DBCC LOGINFO and have a look at how your database appears. In Figure 8-10, we have run DBCC LOGINFO against the Northwind database on our system. As you can see, a total of four VLFs exist in our database.

```
dbcc loginfo
```

	FileId	FileSize	StartOffset	FSeqNo	Status	Parity	CreateLSN
1	2	253952	8192	23	2	128	0
2	2	253952	262144	22	0	64	0
3	2	253952	516096	21	0	64	0
4	2	278528	770048	24	2	128	0

Figure 8-10. Executing DBCC LOGINFO against the Northwind database

To reduce the number of VLFs, you should perform frequent log backups. You also do not want to forget to shrink the log file through the command DBCC SHRINKFILE (logfilename, truncateonly), otherwise the number of VLFs will not be reduced.

When designing a log file, you should make a good estimate for the size of the log file and create it with that size. If auto-growth is enabled on a small log file, new VLFs are created every time the log is increased. So it is important to make it as close to the size you will need as possible.

Filegroups

A simple database might only hold a primary data file and a log file. A more complex and larger database, on the other hand, often consists of secondary files and more log files as well.

By building *filegroups,* you can create your secondary data files in these groups for data placement purposes. When you create your database, you specify in which filegroup you want to place your database objects. This way, you can have control of where you place your objects, since you also know on which disk the filegroups are placed. You can even create a table or other object across multiple filegroups to increase performance. (This obviously is true only when the filegroups are created on separate disks.)

To better understand this, imagine a database that has two tables, Table_A and Table_B. The database also has two RAID disk arrays, as you can see in Figure 8-11. Table_A is constantly accessed sequentially with read-only requests, whereas Table_B is accessed only occasionally with write requests (perhaps only once every minute or so).

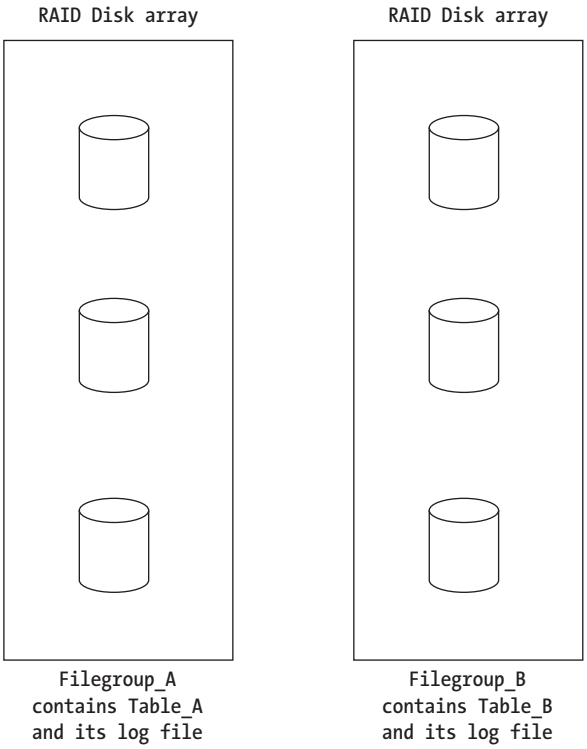


Figure 8-11. Two tables are separated on two different filegroups and then placed on different disk arrays.

In this scenario, it would be great for performance to place a filegroup on each disk array and then place each table in separate filegroups. This way you separate the heavily accessed table from the less accessed one, which improves performance. If one of the disk arrays had more disks than the other, you should also place the read-only table on that array, because doing so also improves read performance.

If some of your disks are too heavily accessed despite this, you could create only one filegroup and place it on both disk arrays. You then place both tables on this filegroup, which spreads I/O on both arrays. Table data will be written to both disk arrays, because the filegroup spans the two.

Since the best way to optimize disk I/O is to distribute your data on as many disks as possible, you should consider this during the *physical design phase* (as you will see later in the section "Physical Design"). By using filegroups, we can even further distribute I/O evenly, because when the data is written to a table on a filegroup that spans many disks, the data is spread in proportion across all the files in the filegroup.

You can use three kinds of filegroups:

- Primary filegroups
- User-defined filegroups
- Default filegroups

The primary filegroup contains the primary data file. It also contains all the files not specifically put into another filegroup. All system tables are always placed on the primary filegroup. The primary data file must also always be placed in the primary filegroup.

User-defined filegroups are the filegroups you create yourself.

The default filegroup is where all tables and indexes are placed if not specifically put in another filegroup. The primary filegroup is the default filegroup, if you do not specify otherwise. You can, if you want, switch the default filegroup with the ALTER DATABASE command in Transact-SQL.

Another benefit of using filegroups is that you can perform backups on separate filegroups independently of each other. You can also restore them separately. This can speed up backup and restore operations considerably.

Automatic File Growth

A cool feature of SQL Server is that you do not have to expand the files manually when they run out of space. SQL Server handles this by itself, if you specify this when creating the file (see Figure 8-12).

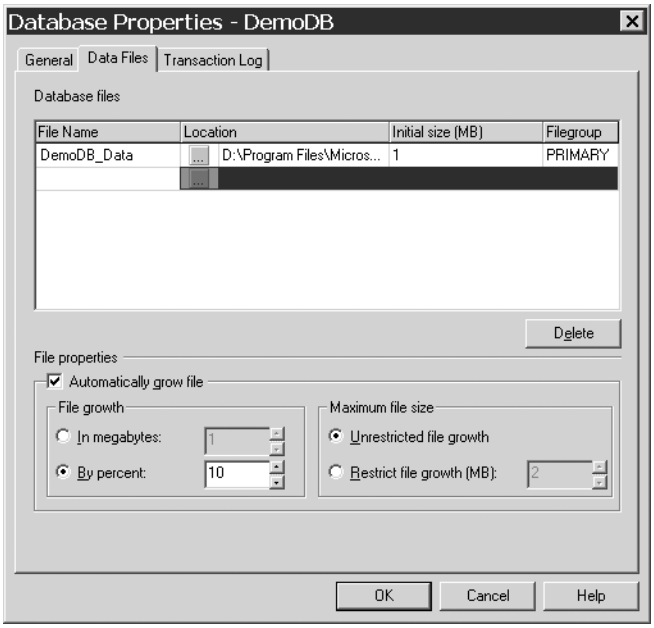


Figure 8-12. Specifying automatic file growth and maximum file size in Enterprise Manager

TIP *If you use this option, you should probably also remember to specify the maximum size of the file. You do not want to run out of space on a disk, which could happen if you forget to set this option.*

As you can see, you need to do some planning before you set these options, but with a little careful thinking, it will prove to be a great help for you.

Lock Management

If you have a database accessed by many users or applications, you need an option to prevent the same data from being modified by two or more processes at the same time. By using locks, you can make sure a process has a dependency on a resource, and as a result have access rights to it. This means other processes cannot modify the resource until the first process releases the lock. Locking ensures that multiple users can read and write to the same database without getting inconsistent data and without overwriting each other's modifications by accident. All locks are managed on a per–SQL Server connection basis.

TIP *If you want to learn more about SQL Server and the way it handles locks, please see the document "Understanding Locking in SQL Server" at the following URL:* `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/acdata/ac_8_con_7a_7xde.asp`.

Different levels of locks can be used with SQL Server. They vary in granularity from the finest, which is the Row Identifier (RID) lock, to the coarsest, which is the database lock. Table 8-2 describes the locks from the coarsest to the finest. The finer the lock, the more concurrency among users you have.

Table 8-2. Lock Levels in SQL Server

Lock Type	Description
Database	Simply locks the entire database. No other user is allowed in the database.
Table	Locks a table, including indexes.
Extent	Locks a 64KB unit. An extent is a contiguous group of eight pages.
Page	A page is 8KB and a page lock locks 8KB.
Key	This is a row lock in an index.
RID (Row Identifier)	Locks a single row in a table.

NOTE *The finer the lock you use, the more overhead you need to maintain it. So consider carefully the kind of lock to use.*

There are also different kinds of locks you can use. SQL Server uses six different kinds of modes, as you see in Table 8-3.

Table 8-3. The Six Modes of SQL Server Locks

Mode	Description
Shared	Used for read-only operations
Exclusive	Used when modifying data
Update	Used on updatable resources
Intent	Used to establish a locking hierarchy
Schema	Used for instance when modifying tables or the schema
Bulk update	Used for bulk-copying data into a table

Luckily, you do not have to consider which mode to use since SQL Server does this for you. It chooses the most cost-effective lock for the current operation.

Threads in SQL Server

Now we have come to the final part of the SQL Server architecture—threads. SQL Server 2000 uses Windows threads and fibers to execute tasks. We have covered threads earlier (in the section "Threads" in Chapter 4), but the fiber concept is new. Before we take a look at fibers, you will need to understand the concept of *context switching* and why this occurs. When SQL Server does not use fibers, it uses threads. These are distributed evenly across the available CPUs on the system. If you want, you can specify which CPUs SQL Server should use, but in most cases SQL Server handles this better than you can. When one thread is moved off a CPU and another is moved on, a context switch occurs. This is a very performance-costly operation, since the switch is between user mode and kernel mode (we covered this switching process earlier in Chapter 4). To get the most out of your system, you should minimize context switching, and fibers is one way of doing this.

A *fiber* is a subcomponent of a thread, and is something you must enable for your SQL Server to use. Fibers are handled by code running in user mode. This means that switching fibers, and thereby switching tasks, is not as costly as switching threads. The reason for this is that the switch does not occur from user to kernel mode, but instead takes place in only one mode.

As opposed to threads, fibers are handled by SQL Server and not by the operating system. One thread is allocated per CPU available to SQL Server, and one fiber is allocated per concurrent user command. There can be many fibers on one thread, and fibers can also be switched on the thread. During the switch, the thread still remains on the same CPU, so no context switch occurs. By reducing the number of context switches, performance is increased.

NOTE *Use fibers only if the server has four or more CPUs, and only use them when the system has more than 5000 context switches per second.*

You can use SQL Server Enterprise Manager to enable fiber mode (see Figure 8-13), or you can run sp_configure to set the lightweight pooling option to 1.

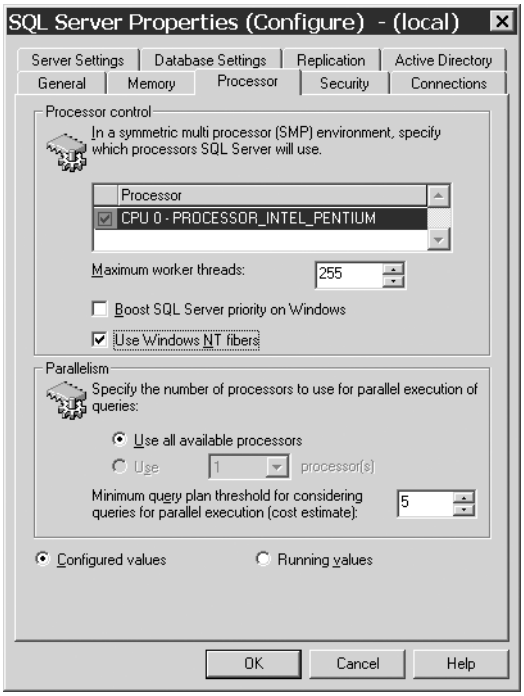


Figure 8-13. Manually enabling fiber mode from Enterprise Manager

Worker Processes

SQL Server keeps a pool of threads or fibers for all user connections. These threads or fibers are called *worker processes*. You can limit the number of worker processes by setting a maximum value for this. When this value is reached, SQL Server begins thread pooling. As long as the maximum worker process value has not been reached, SQL Server starts a new thread for the user commands, but as soon as the limit is reached and thread pooling has begun, each new request has to wait for a thread to be available. You should not set the maximum value too high (default is 255) because the more worker processes you have, the more resources are consumed.

This eventually affects performance. Experiment with it in the lab, and find the best setting for your system.

Database Design

Designing a database means more than just jotting down some objects on a piece of paper and trying to create tables from them. It requires an understanding of the business functions you want to model, as well as how you can implement them by using database concepts and features. As you saw in Chapter 1, Object Role Modeling (ORM) can be a great way of achieving your database design.

When you are beginning to design your database, you should consider performance issues. Even though you can change the database design once it is implemented, this is not a desirable way of doing things. It costs both time and money to go back and redesign a database at a later stage.

During database design, you should consider the purpose of the database. An Online Transaction Processing (OLTP) database will not be normalized the same way that you would normalize a decision support database, for instance. The more update-intensive processing of the transactions in an OLTP database might often require a more normalized design than a decision support database, since the latter often has less redundant updates. The decision support database might actually be more efficient if you do not normalize it too much. By having fewer joins, performance is better and data will be easier to understand.

You must always create a database plan that will fit your purpose. You should also get an overview of the security requirements for the database. Do you need to set up Active Directory Application Mode, or ADAM, (as discussed in more detail in the section "Windows Authentication"), or can you manage with your existing Active Directory? If you are not going to have many users, you can perhaps use SQL Server authentication and skip a directory altogether.

Another aspect you should consider is how much data you will store, and how much you believe this data will grow over time. This affects how you plan your hardware. You also need to address scalability and reliability when designing your database. Do you need to cluster the database, or can you live with some outages of your SQL Server? You should also start planning a disaster recovery plan as early as possible.

TIP *A good rule of thumb, as always, is to dedicate enough time for the design so the project later on will not have any performance or security issues. The time needed will have to be estimated based on the kind of application you are going to build, so it is hard to give exact advice here.*

In the best-case scenario, your company has a policy for data storage so you do not have to make these decisions for every new project, since they take some time to work through.

Database design involves two things:

- Logical design
- Physical design

We will explore these in more detail in the following sections.

Logical Design

During the logical design phase, you model your business requirements and data using tables, constraints, and other database components (see Figure 8-14). You do this preferably by using ORM. (At this stage, you do not yet think about how you will store data physically.)

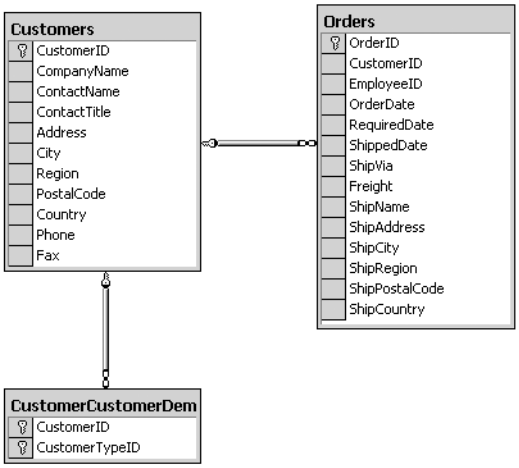


Figure 8-14. A logical database design

One of the first things you do during logical design is to remove as much redundancy as you can from the database. This process, called *normalization,* breaks down the data into smaller parts; for example, the name of an employee can be broken down into first name and last name.

You also need to consider the integrity of your data during this phase. By using primary keys, check constraints, foreign keys, default values, and other database features, you can make sure your data is not compromised, and these features also help you maintain your business requirements.

TIP *Use fixed-size columns when you need to store only small values in such columns. If you have fewer than 5 characters per cell in a column, you should use a fixed column size like char(4). This enhances the way a data row is read and hence increases performance.*

If a column is to include more than 20 characters in each cell, you should use a variable size like varchar(100).

Anything in between is difficult to advise on, so try different sizes in a test lab before deciding.

Please remember that a bad logical design most often results in a bad physical design, which in the end negatively affects performance.

Physical Design

The physical database design phase involves mapping the logical design to your physical hardware. You do this by taking advantage of the hardware and software features available to you. In other words, this is where you allow the data to be physically accessed and maintained as quickly as possible.

NOTE *Indexing your data is also part of this design phase. We will cover indexing later on in this chapter in the section "Index Tuning."*

When you do your physical design, you often benefit if your company has a data storage policy. Otherwise, you must decide where to store your data: on a DAS, a NAS appliance, or a SAN. Depending on the business requirements for your solution, you perhaps must deviate from corporate standards, and during the design stage is the time to determine these things.

Optimizing Performance

Now that you know how the internals of SQL Server work and how database design affects performance, we are going to introduce you to some performance tuning topics of interest.

Database Performance and I/O Configuration Options

Here we will discuss some of the performance counters you might want to keep an eye on to ensure your SQL Server is working like it should.

When SQL Server reads data from tables, it uses Windows system I/O calls to perform this. SQL Server decides when and how this access is performed, but the operating system actually performs the work.

Disk I/O is the most frequent reason for performance bottlenecks, so you should not forget to monitor this activity on your server. As always, use System Monitor (or Performance MMC, as it is called in Windows XP/Windows Server 2003) to monitor your system.

Figure 8-15 shows two performance counters:

- *PhysicalDisk: % Disk Time,* which is the percentage of time the disk is busy with read/write operations.

- *PhysicalDisk: Current Disk Queue Length,* which shows the number of system requests waiting for disk access.

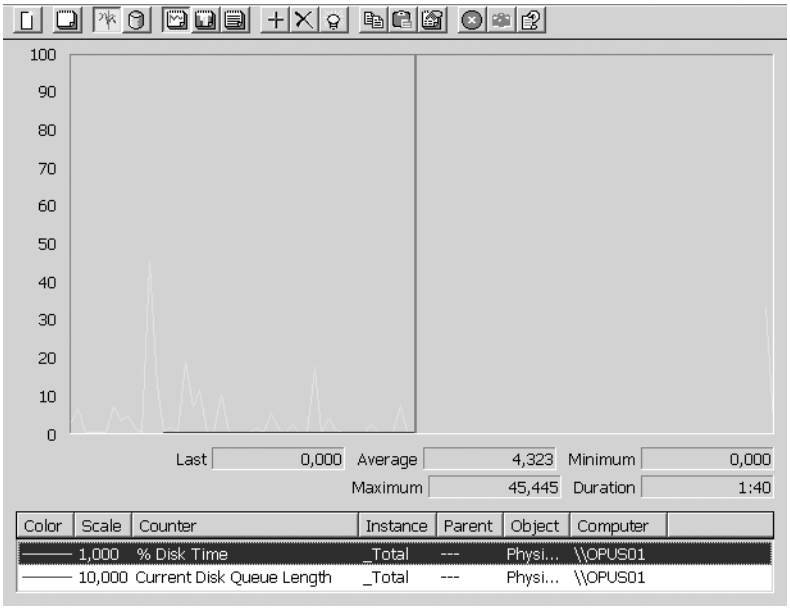


Figure 8-15. Two performance counters that are used to monitor disk access

If the PhysicalDisk: % Disk Time value is above 90 percent, you should monitor PhysicalDisk: Current Disk Queue Length to see how many system requests are waiting in line for disk access. This value should be no more than 1.5 or 2.0 times the spindles making up the physical disk, so check out the hardware before counting. If both of these counters are consistently high, you probably have a bottleneck in your system. To solve this problem, consider using a faster disk

drive or moving some files to an additional disk or even a different server. You could also add additional disks to your RAID system if you use one.

These counters are used if you only have one disk partition on your hard drives. If you have partitioned your disks, you should monitor the LogicalDisk counters instead of the PhysicalDisk counters (the names of the corresponding LogicalDisk counters are similar to those just mentioned for PhysicalDisk).

As you saw in Chapter 4, you should avoid paging on your server. To monitor this, use the Memory: Page faults/sec counter.

If you want to isolate the disk writes/reads that SQL Server is responsible for, monitor the following two counters (see Figure 8-16):

- SQLServer: Buffer manager page reads/sec
- SQLServer: Buffer manager page writes/sec

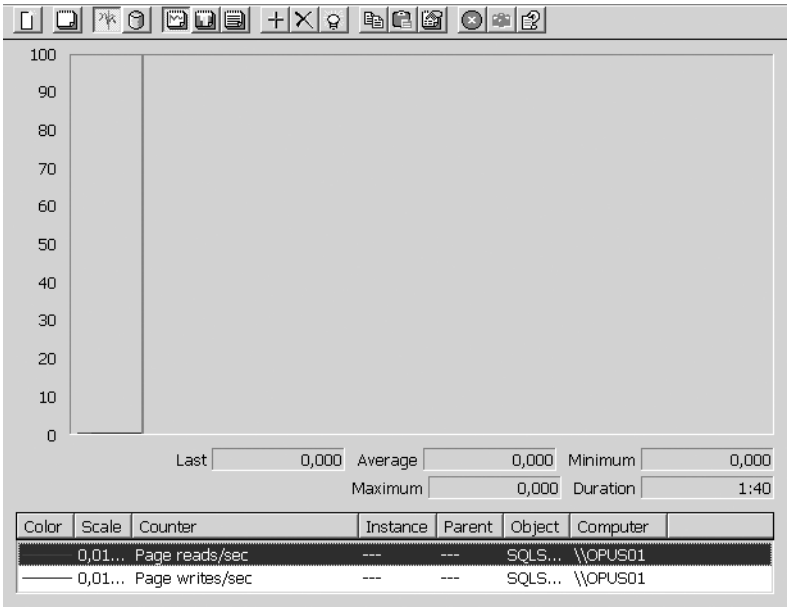


Figure 8-16. Two performance counters are used to separate SQL Server disk I/O from other I/O.

If the values for these two counters are near the capacity of the hardware, you should try to reduce them by tuning your databases or applications. You can do this by reducing I/O operations: Check your indexes so they are accurate or try normalizing your databases even more. If this still does not help, try increasing the I/O capacity of your hardware or adding more memory.

TIP *If you find your bottleneck involves inserts and updates, you could try to normalize your data more. If you instead find that queries cause the bottleneck, you could try to denormalize the database. The problem with such a solution is that you might need to rewrite your applications to handle a new database structure. This might not be a problem if you discover this early in development, but in production this can be costly. Sometimes this might be the only way to correct the problem, however, and then you just have to deal with it.*

You could monitor how much memory SQL Server is using by checking the value for SQL Server: Memory Manager Total Server Memory (KB). Compare this to how much memory the operating system has available by checking the counter Available Kbytes from the Memory object. This could give a clue to deciding if there is a need to restrict how much memory SQL Server is allowed to use.

Next, we will move on to clustering SQL Server.

Clustering

In previous chapters, you have seen the two ways you can cluster Windows servers: Network Load Balancing (NLB) and Microsoft Cluster Service (MSCS). You can only use MSCS with SQL Server, but as you will see, several servers can share the workload of a database server in a similar way as NLB. First, we will start with a look at using MSCS with SQL Server.

MSCS

SQL Server is built as a cluster-aware application and can therefore use MSCS to provide high availability. In Windows Server 2003, you can have up to eight nodes in a cluster, and you can also disperse the nodes across geographic locations. By using MSCS with SQL Server, you can build a back-end data store solution that provides your applications with a secure and highly available data feed. Figure 8-17 shows a typical four-node cluster exposed as a single virtual server to the clients.

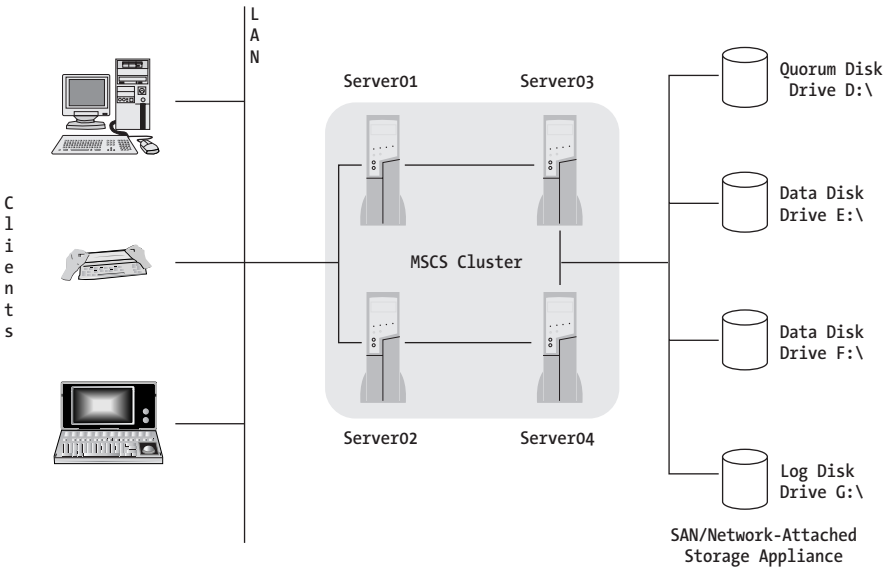


Figure 8-17. A MSCS cluster with four nodes

The disks used to store the database and its transaction log(s) are shared between the nodes in the cluster.

NOTE *To avoid a single point of failure, duplicate all fiber channels and other communications hardware your cluster uses.*

Remember that only one instance of SQL Server can access the database at a time, and when a node fails, it takes a little time before the failover node is online and processing requests. If the failure depends on a corrupt database, obviously the failover node will not work either. Database corruption like the kind a user can make, called *logical corruption,* and the type that occurs within a database, called *internal corruption,* is not prevented by clustering. This means you must always provide a disaster recovery plan, and also make certain that it works when you need it.

This sounds so logical and easy, but this is not always the case. One time, a client of ours experienced an erroneous update in their sales batches during the night, which corrupted 25 customer databases. All of the customers had regular database backups scheduled, but only 15 of them were able to perform a restore operation. We ended up writing scripts and doing tricks so the databases were rolled back to an earlier date, and all batches were then inserted again, this time

with correct data. That took us close to a week. Do not put yourself in this situation—we can guarantee it is not pleasant.

Index Tuning

Indexes are one of a database designer's best friends. Efficient index design is a crucial part of achieving good performance. An index reduces the amount of I/O necessary to retrieve requested data by helping SQL Server to quickly determine where data is stored, hence making retrieval quicker.

If you do not have an index in your table, all data must be read and compared to what you are looking for. Compare this to how hard and slow it would be to find a certain topic in a book without a table of content or index, and you soon understand the importance of good indexing. By using indexes, you speed up the process of finding what you are looking for.

There are two kinds of indexes that SQL Server uses:

- Clustered indexes
- Nonclustered indexes

A *clustered index* can be compared to a phone directory. All data is sorted alphabetically, in the phone directory by last name and first name, and in a database depending on what column(s) the index is created on. There can only exist one clustered index in a table, since the clustered index dictates the physical storage order of the data.

A *nonclustered index* can be compared to an index in a book. The data is stored in one place and the index, with pointers to the data, is stored in another. The index and the items it consists of are stored in the order of the index key values. The data in the table can be stored in a different order. You can, for instance, store it in the order of a clustered index, but if you do not have a clustered index in your table, it can be stored in any possible way. SQL Server searches the index to find the location of the data, and then retrieves it from that location. You can have multiple nonclustered indexes in a table, and use different ones depending on your queries.

When to Use Which Index

If you have a column in a table where data is unique for every row, you should consider having a clustered index on that column. An example of this is a column with Social Security numbers or employee numbers. Clustered indexes are very good at finding a specific row in this case. This can be effective for an OLTP application, where you look for a single row and need access to it quickly.

Another great opportunity for using clustered indexes is when you search a table for a range of values in a column(s). When SQL Server has found the first value in the range, all others are sure to follow after it. This improves performance of data retrieval.

If you have columns often used in queries that have a GROUP BY or ORDER BY clause, you can also consider using a clustered index. This eliminates the need for SQL Server to sort the data itself, since it is already sorted by the index.

TIP *We have learned from experience that it is not a good idea to have a clustered index on a GUID (or any other random value). The clustered index works well for columns you know will be sorted a particular way (according to last name, for example), because you do not need to perform a sort after retrieval. You will rarely (if ever) sort or seek data according to their GUIDs (or random values for that matter) because that would only increase the overhead during and after data retrieval, since you then would have to resort the data to be able to use it properly.*

Nonclustered indexes are used to great advantage if a column or columns contain a large number of distinct values. (Refer to a combination of last name and first name here to get the picture. There can be many John Smiths in a table, for example.)

You can also use the nonclustered indexes when queries do not return large data sets. The opposite is, of course, true for clustered indexes.

If you have queries that use search conditions (WHERE clauses) to return an exact match, you can also consider using nonclustered indexes on the columns in the WHERE clause.

NOTE *SQL Server comes with a tool called the* Index Tuning Wizard, *which can be used to analyze your queries and suggest the indexes that you should create. Check it out on a few queries and tables, and do some testing. We will cover this feature more later in this chapter.*

Query Optimizer

SQL Server 2000 comes with a built-in query optimizer that in most cases chooses the most effective index to use. When you design your indexes, you should provide the query optimizer with a carefully thought-out selection of indexes to choose from. It will in most cases select the best index from these.

The task of the query optimizer is to select an index when it attempts to optimize performance. It will not use an index that will degrade performance. The truth is that having an index does not necessarily mean you get the best performance, especially if an index is designed poorly. So you should be grateful that there is logic built into the query optimizer to help you out. The rule is to test your indexes as you test all other solutions to find the most effective ones.

Index Tuning Wizard

As you saw in the previous section, in SQL Server you do have available a tool that helps you analyze your queries and suggest indexes for your tables. If you were going to do this manually, you would have quite a task in front of you. First of all, you would need to have an expert understanding of the database. If you are the database designer, this would not be a problem, but that is not always the case. You would also need to have deep knowledge of how SQL Server works internally. If you are like most developers, you just do not want to dig that deep into the tools you use, and you should not need to either. That is why it is good to have tools that make your lives easier.

To use the Index Tuning Wizard, you need a sample of the normal workload your SQL Server has to deal with. You can use this as input on how you should design your indexes, so you know they will be effective. Only it will be the Index Tuning Wizard that makes the recommendations for you.

To create the sample, you use another tool that ships with SQL Server: the *SQL Profiler*. Once you are satisfied with the trace SQL Profiler creates, you can let the Index Tuning Wizard analyze the workload and a sample Transact-SQL statement. When done, the wizard will recommend an index configuration that will improve the performance of the database. At this moment, you can choose to let it implement the suggested changes immediately, schedule the implementation for later, or save it to a SQL script that you can run at any time.

NOTE *Run the Index Tuning Wizard against a test SQL Server. It consumes quite a lot of CPU from the server, which could create problems in a production environment. Also, try to use a separate computer to run the wizard from, so it is not running on the same server where the SQL Server instance is installed.*

Partitioned Views

You cannot use Network Load Balancing to distribute the workload for a SQL Server. Another method lets you accomplish the same thing, however, or at least something similar.

Take a look at an example. If you have a customer database that only stores information about your customers A–Z, you can partition the data. One partition could hold all customers in the range from A–E, another F–I, and so on. Once you have decided which way to partition the data, you must select one out of two following choices of storing it for best performance:

- Local partitioned view
- Distributed partitioned view

Local Partitioned Views

A *local partitioned view* stores all partitions in the same instance of your SQL Server. (See the section "Separating Read Data from Write Data" later in this chapter for a discussion on when this is a good solution.)

Distributed Partitioned Views

A *distributed partitioned view* uses a *federation of servers* to store data. Although this sounds like something out of *Star Wars,* it is really quite simple. A federation of servers consists of two or more servers, each holding one partition of the data (see Figure 8-18).

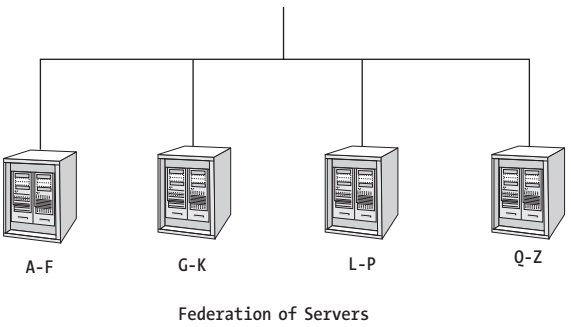


Figure 8-18. A federation of SQL Servers

Every server runs a separate instance of SQL Server. The servers are administered independently of each other, but they cooperate to share the processing load of the data. The horizontally partitioned table is exposed as a single table through the use of partitioned views.

On every server, the other SQL Servers are added as linked servers so you can execute distributed queries on them. You then create a distributed partitioned

view on each of the servers as the code listing that follows shows, so that clients can access this view on any of the servers and still get the view of the full table. As you see, the view is a set of SQL statements whose individual result sets are combined into one, using the UNION ALL statement in Transact-SQL.

```
create view CustomerView as
      select * from server01.MyDatabase.dbo.Customers_01
union all
      select * from server02.MyDatabase.dbo.Customers_02
union all
      select * from server03.MyDatabase.dbo.Customers_03
union all
      select * from server04.MyDatabase.dbo.Customers_04
```

You can also make the view updatable if you want to, which further adds to the flexibility of this solution.

Remember that you can use the servers for storing other data at the same time as they store a partition. The servers are not dedicated to this partition in any way. You might only need to partition one large table, in which case it would be a pity not to use the servers' capacity to their full extent.

By using this method, you can let several servers share the burden of handling a large table. Clients can access any of the servers and get results from all of them by using one single view, allowing this solution to scale even though you do not use NLB. Since you can use a MSCS cluster behind each of the participating servers, you can also achieve high availability with this technique.

NOTE *Most common way of implementing distributed partitioned views is by letting the client applications access the member server that has most of the data required by the SQL statement. This is called* collocating *the SQL statement with the data it requires.*

Separating Read Data from Write Data

There is a way to reduce the impact locking has on your database. As you may recall, locking can reduce performance, and if you have a lot of read activity, a write command should not lock these reads. By defining vertical partitions, you can separate read columns from write columns and place them in different tables. This can improve performance in your applications.

You could also use horizontal partitioning to separate read data from write data. Say you have a table that stores sales data for your company. Data is written on a daily basis, and all other data is maintained for reporting purposes. This kind of table could grow extremely large and unmanageable. By partitioning the

data horizontally, you can enhance performance on write and read operations. You could create a table for the current month, another for the current year, and a third for the rest of the data. How you do the partitioning is determined by the way the table is used. After that, you create a view that lets you display data from all tables.

By partitioning this way, you also enhance backup performance, since you do not need to back up all historical data every day.

During the design phase, you should carefully consider how your data is going to be accessed so you minimize locking.

Query Tuning

Even if you have top-of-the-line hardware and it is tuned extremely well, you might still experience performance troubles. It is as important to optimize your queries as it is to optimize your hardware. Although writing queries is quite easy, if you have no knowledge of query design issues, your queries could degrade performance. Resources are used intensively especially in two particular cases: queries returning large result sets and highly nonunique WHERE clauses. We have seen some horrible queries in our times (not to mention the embarrassing queries we ourselves have produced over the years), despite the fact that the developers who wrote them should have known better.

Sometimes you cannot avoid using very complex and resource-intensive queries; however, you could try to move the processing of these to a stored procedure, where such a query probably would execute better than if the application issued it. This would also reduce network traffic if the query returned a large result set.

If this still does not help, you could also add more CPUs to your server if your hardware allows this possibility. This way SQL Server can execute queries in parallel, and performance will increase.

What you should do first of all is to go over the design of the query one more time. Is there really no way you can rewrite it to make it perform better? If you use cursors in your query, you should definitely see if you can make them more effective. It might also be possible to exchange the cursor for a GROUP BY or CASE statement instead.

If your application uses a loop that executes a parameterized stored procedure, every time you travel down the loop, a round-trip is made to the SQL Server. This slows down performance. Instead, you should try to rewrite the application and the stored procedure to create a single, more complex query using a temporary table. This way the SQL Server query optimizer can better optimize the execution of the query. A temporary table causes a recompile of the stored procedure, however, which negatively affects performance. So if you find it causing problems, you could use a table variable instead. Try to avoid getting too large of a result set back, since

this also degrades performance. This can be like walking on the edge of a blade, so test, test, and test.

NOTE *Stored procedures are not all good, unfortunately. See the section "Stored Procedures" later in this chapter for specifics.*

A long-running query costs a lot since it might prevent others from getting their queries executed. It also consumes a lot of resources. You can stop these queries by setting the *query governor configuration* option. The default is that the query governor allows all queries to execute. You can set the query governor to not allow queries that will take more than a specified number of seconds to execute. It also allows you to control this type of query execution on each separate connection, or on all if you wish. The query governor estimates the cost of the queries and stops long-running queries before they have even started executing. Since it does not allow them to run, the overhead of this process is quite low.

Connecting to the Database

There are several ways you can connect to a SQL Server, *named pipes* and *TCP/IP* being the most common. In certain situations, one of these may be preferable to the other if you want to get the best performance out of your solution.

If you have a fast LAN, not much differs between these two approaches. On a slower network, like a WAN or dial-up network, the difference becomes apparent. Named pipes is a very interactive protocol—it sends a lot of messages before it actually begins reading any data. Named pipes communication starts when a peer requests data from another, using the read command. After they have exchanged pleasantries, data is read. On a fast network, this communication will not be noticed much, but on a slower connection this costs performance because the network traffic consists of many messages. Obviously, this affects clients on the network negatively.

TCP/IP sockets have more streamlined data transmissions with less overhead than named pipes. Windowing, delayed acknowledgements, and other features of TCP/IP sockets help you improve performance. This is very useful on a slow network for providing significant performance benefits for your client applications.

Our suggestion is to use TCP/IP on WANs and dial-up connections; it will save a lot of complaints on slow data retrieval.

If the application using the SQL Server is located on the SQL Server itself, you can use *local named pipes,* which can result in great performance. Local named pipes run in kernel mode, making them extremely fast. Use this option if

you are planning to implement the data access component on the same server as the SQL Server database.

To summarize what we have just covered:

- Use TCP/IP on slower network connections.
- Use local named pipes when data access components are running on the SQL Server itself.
- Use either TCP/IP or named pipes on a fast connection.

Stored Procedures

A good way of optimizing how your data access components talk to SQL Server is through stored procedures. Because they are compiled on the server, they execute much faster than if you had used a Transact-SQL statement to retrieve data. You do not want to move business logic to the database in most cases; you only want to use the enhanced performance you get from including parameterized stored procedures.

NOTE *All database interaction can benefit from using stored procedures. Your insert, select, and delete operations could, and should in most cases, be implemented as stored procedures.*

However, even stored procedures can negatively affect performance. Try to avoid stored procedure recompiles because they cost performance. One good thing to remember is to be careful with the use of temporary tables in stored procedures, since they cause a recompile of the stored procedure every time it is executed. In the case of temporary tables, you could try using a table variable instead.

If you want to learn more about troubleshooting stored procedure recompilations, you can find a good document on this topic at `http://support.microsoft.com/default.aspx?scid=kb;en-us;243586`.

Another benefit is that stored procedures utilize the network more efficiently than an ordinary SQL statement does. For example, say you have an INSERT statement that inserts a large binary data value into an Image data column. If you do not use a stored procedure, the application issuing the Insert statement must convert the value to a character string. This doubles the statement's size. Only after this conversion can it send the data to the server. The server in turn converts the statement back to binary format and stores it in the image column.

If you instead use a stored procedure, the image would stay in binary format all the way to the server. This would greatly reduce overhead on the server and the client, as well as cutting down on network traffic.

Sometimes it can be of great use to move the business rules from the business logic into the database server. You must carefully consider your use of this, however, so you do not lose control of your application. The benefits of moving processing to a stored procedure are purely performance related. Instead of moving a large set of data to where the processing usually occurs, you bring the processing functionality to the data, which increases performance. The complexity of your application increases, however, so you should think twice before using this technique. The best way to see if the performance gains outweigh the increased complexity is to test the solution in the lab.

SQL Server Security

So far we have covered some important topics when it comes to data storage and SQL Server. The last topic we will cover before taking you through our demo application in the next chapter is SQL Server security.

In Chapter 4, we discussed authentication and authorization. Authentication occurs when a user or application tries to log on to your SQL Server. Authorization determines what the user is allowed to do in the database server and its tables. In the following section, we will take a look at how authentication works in SQL Server.

Choosing Your Authentication

There are two ways SQL Server can authenticate a user or group. Early in the design process, you should decide which method you are going to use, because each affects the way you set up your user accounts for your application. The two methods are

- SQL Server authentication
- Windows authentication

SQL Server Authentication

If you use *SQL Server authentication,* SQL Server matches the account and password the user supplies to a list that is stored in the sysxlogins system table. This

table is found in the master database (see Figure 8-19). SQL Server authentication is quite easy to implement, and you use SQL Server Enterprise Manager to add, remove, or modify these logins (see Figure 8-20).

Column Name	Data Type	Length	Allow Nulls
srvid	smallint	2	✓
sid	varbinary	85	✓
xstatus	smallint	2	
xdate1	datetime	8	
xdate2	datetime	8	
name	sysname	128	✓
password	varbinary	256	✓
dbid	smallint	2	
[language]	sysname	128	✓
isrpcinmap	smallint	2	✓
ishqoutmap	smallint	2	✓
selfoutmap	smallint	2	✓

Figure 8-19. The sysxlogins table from the master database

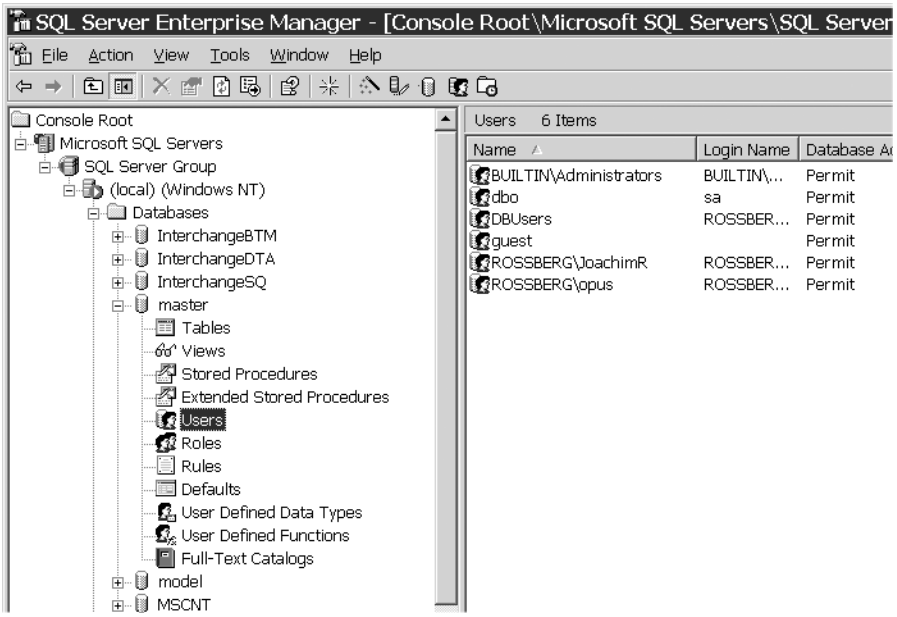


Figure 8-20. The Enterprise Manager gives you an overview of the users in a database.

The logins created are local to the server on which the SQL Server resides, which is not a good solution if you have a multiserver environment. In that case, you cannot manage them very easily, and you would have to implement the accounts on several servers. This solution could work, however, if the number of

users does not exceed 20 to 25. With more, you will soon discover how impractical this is, and how much administrative overhead this will cost. Imagine implementing a change to only one user on several servers.

Windows Authentication

What you should use instead in the preceding scenario is Windows authentication. When SQL Server authenticates using this method, it asks a domain controller to validate a user's credentials. A domain controller must be accessible for this to work, so you need to make sure one is available. When SQL Server authenticates a user or group against a domain controller, it receives an access token containing the user's SID and the SIDs of every group the user has membership in. SQL Server will then assign access to the database server based on these SIDs.

You can use Active Directory (AD) to store your users and groups for your applications, but in many cases you do not want to mix your internal corporate users with your application users. The SQL Server will perhaps be used only for a Web solution and not needed by anyone but those accessing the Web site. In that case, it could present a security risk to use the same AD as for your enterprise. To solve this, you could implement a separate AD infrastructure and set up a new server and domain, but this is not something you can do just like that. Luckily, Microsoft recently released a "light" version of Active Directory, called Active Directory Application Mode, or ADAM. ADAM is a stand-alone version of AD, intended for the use as a directory for Web-based applications and other types of applications. ADAM is deployed separately from the standard AD, and its directory data is not replicated throughout the enterprise core NOS directory. It also gives administrators the flexibility to deploy a directory without having to set up an entire Windows Server operating system environment on a domain controller. Nor do they have to activate Kerberos, DNS, or PKI.

ADAM is Microsoft's answer to Web-based directories from competitors like Sun's ONE Directory Server and Novell's eDirectory. (Given Microsoft's aggressive marketing in the past, ADAM will probably be seen on a server near you rather soon.)

Windows authentication gives you central management control regardless of whether you use AD or ADAM, which will give you lower TCO over time.

There are also ways of accessing SQL Server over the Internet using IIS, but this solution is not especially relevant when it comes to building an enterprise application, so we will not cover it here.

Determining Permissions

After you have decided which authentication method you should use, you must plan what permissions you will give to your users and groups. (We will in this

discussion consider only Windows authentication.) You should strive to give permissions to your database server only to groups. If you do, you can more easily manage who has access to the server. Say you have a group of users belonging to the accounting department. Often these users will need the same permissions in the SQL Server. You do not want to add every user from this department to the database server and give them each the same permissions. What you do want to do is create a domain global group, add the accounting users to this group, and give only the domain global group database access. So instead of granting 30 people access and permissions on the server, you only do this for the domain global group they belong to, which cuts down on administrative overhead.

When you want to give permissions in a database on your SQL Server, you can use something called *roles*. There are some built-in roles you can use (see Figure 8-21), but you can also create your own. You give permissions only to roles, and then make the domain global groups members of those roles.

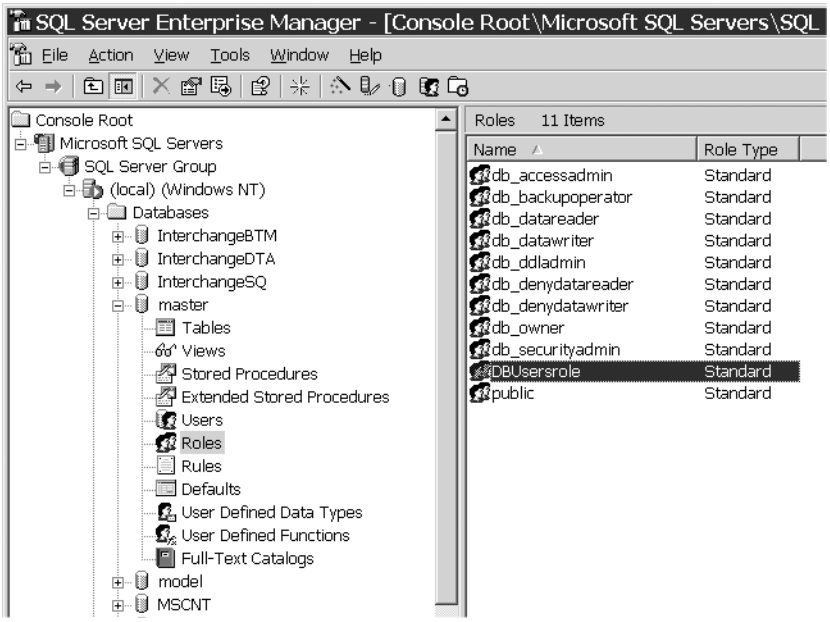


Figure 8-21. The roles in SQL Server

You create the roles you need, name them so you understand what they are, and then add the correct domain global group to the role. After that, you assign database permissions to it.

NOTE *A good practice is to name roles after the domain global groups, so as not to cause confusion.*

This procedure is the same as is often used to give users and groups access to data on a server. The global groups are members of local groups, and it is only the local groups that are given permissions to the data.

Summary

This chapter has covered some important issues when it comes to data storage and how SQL Server fits into a storage policy. You have also seen some tricks you can use to improve performance in your SQL Server systems.

Now it is time to put everything you have seen in this book to work. The next chapter will cover the development of a new application, following the guidelines we have discussed so far.