OBJECTIVES

Create and alter databases. Considerations include file groups, file placement, growth strategy, and space requirements.

- **Specify space management parameters. Parameters include autoshrink, growth increment, initial size, and maxsize.**
- **Specify file group and file placement. Considerations include logical and physical file placement.**
- **Specify transaction log placement. Considerations include bulk load operations and performance.**

▶ The placement of the files related to a SQL Server 2000 database environment helps to ensure optimum performance while minimizing administration. Recoverability can also be improved in the event of data corruption or hardware failures if appropriate measures are taken. On the exam, you must be prepared to respond to these requirements and properly configure the interactions with the file system.

Create and alter database objects. Objects include constraints, indexes, stored procedures, tables, triggers, user-defined functions, and views.

- **Specify table characteristics. Characteristics include cascading actions, `CHECK` constraints, clustered, defaults, `FILLFACTOR`, foreign keys, nonclustered, primary key, and `UNIQUE` constraints.**
- **Specify schema binding and encryption for stored procedures, triggers, user-defined functions, and views.**
- **Specify recompile settings for stored procedures.**
- **Specify index characteristics. Characteristics include clustered, `FILLFACTOR`, nonclustered, and uniqueness.**

CHAPTER 3

Physical Database Design and Implementation

OBJECTIVES

▶ An absolute necessity of building databases that interact with SQL Server 2000 is using the appropriate database objects to obtain a usable database system while improving response times and maintaining data integrity. There are considerations and trade-offs for choosing one technique over the other. The selection of the most appropriate method to obtain the desired result requires that you know where each is best implemented. The exam will test on the appropriate application of each of these objects.

Alter database objects to support replication and partitioned views.

- **Support merge, snapshot, and transactional replication models.**
- **Design a partitioning strategy.**
- **Design and create constraints and views.**
- **Resolve replication conflicts.**

▶ A variety of technologies exist in handling multiple server environments, knowing what each technology offers—as well its restrictions—helps you adapt a database system applied across multiple machines. Considerations for controlling data alterations, having the data available when needed, and responding to queries in a timely manner will be the aim of questions within this area of the exam.

Troubleshoot failed object creation.

▶ Troubleshooting is certainly a broad topic. In this chapter, the focus for troubleshooting is on the interactions with objects and the server as well as application settings that are required for an object to be created and used. On the exam, troubleshooting will be approached from a wide variety of angles. In the “real world,” it is good practice to always view a procedure from a problem-solving perspective. Always be ready to ask yourself, “What could go wrong?” and “What can be done to resolve the problem?”

OUTLINE

Study Strategies

- Ensure that you have a thorough understanding of the variety of objects and technologies available within the realm of physical design. Know what each technique accomplishes (advantages) and also watch out for associated pitfalls (disadvantages).
- Understand the basics of the file system and its use by SQL Server. Know when to split off a portion of the database structure and storage to a separate physical disk drive.
- Know the interaction between SQL Server and the OS (operating system). Some of the physical design concepts that are discussed point out the role that the OS performs and the reason for its participation.
- Recognize the changes to the actual data structure and other areas of the database definition that might occur. Some technologies impact the database schema by applying their own objects.
- Watch out for “What’s new in SQL Server 2000.” Typically the exam tests on new features within the software, and this exam is certainly no different. The discussion of physical design topics reviews many important design and exam criteria, many of which are new features.

INTRODUCTION

Chapter 1, "Introduction to SQL Server 2000," looked at the modeling considerations and the logical structure of a database design. In moving from an idea to a logical structure to the actual physical elements, you must remember to consider elements that contribute to performance, reliability, and data integrity. Having a model of the system is one thing, but it must be able to meet the demands of an environment where inevitably the system must meet the intended goals of the company and add to the bottom line.

This chapter explores the aspects of the physical database design. It compares and contrasts the physical design and logical data modeling of Chapter 1 and then proceeds towards the implementation of a database system. Included are discussions of the file system, data structure, index structure, SQL Server objects, hardware, and finally, troubleshooting of the database design. Of particular interest to anyone preparing for the exam will be the discussion of schema binding and data partitioning. Because these two topics represent functionality that has been provided over and above the previous version, you can certainly expect questions on these areas when you take the exam.

In any physical design, the goal is to provide an efficient and responsive database system that also lends itself to appropriate maintenance tasks without becoming a database administrator's burden. At this stage of a database implementation, care is taken to provide a system structure that is usable, provides for optimum user response time, can be readily maintained, and above all meets the needs of the business for which it was designed.

As each of the physical design topics are discussed, pay close attention to the options that are available. Many different database technologies can be applied. Selecting the most appropriate technique to apply, based on what the problem warrants, is a very imperfect science. To become proficient in all these technologies, a developer must have a diverse background covering many data solutions. It would be beneficial, therefore, to try to experiment with as many different types of applications as possible. You will find that a manufacturing solution is considerably different from an online sales-oriented solution, which differs as well from a scientific application's solution, and so on.

The exam covers physical elements in numerous areas. The material contained in this chapter hits many of the exam topics. This chapter covers the exam objectives for the physical database design section of the Database Development SQL Server 2000 exam, including the hardware, the operating system, SQL Server, the database, all database objects, and the application. Many of these features are overviewed in this chapter but are further defined in other chapters throughout the book.

CREATING AND ALTERING DATABASES

- **Create and alter databases. Considerations include file groups, file placement, growth strategy, and space requirements.**

The file format in SQL Server 2000 has not significantly changed from the previous version (SQL Server 7). SQL Server uses a set of files to store the data, indexes, and log information for a database. A primary file also has some header information in it, providing SQL Server with necessary information about a database. Each database has a minimum of two files associated with it, one for the data and a second for the log. It is also possible to create multiple files for each of these purposes as described in the following paragraphs. File placement, and object placement within these files, plays an important role in the responsiveness of SQL Server. A database consists of two or more files with each file used for only a single database. A single file cannot be shared by multiple databases.

Each database has one or more files used to store indexes and data. The first file created for this purpose is referred to as the *primary* file. The primary file contains the information needed to start up a database and is also used to store some or all of the data. If desired, secondary files might be created to hold some of the data and other objects. Some databases might be large enough or complex enough in their design to have multiple secondary files used for storage.

Normally the *log* is maintained in a single file. The log file is used to store changes to the database before these changes are recorded in the data files themselves. The storage of information into log files in this manner enables SQL Server to use these files as an important part of its recovery process. Every time the SQL Server is started, it uses the log files for each of the databases to determine what units of work were still being handled at the time the server was stopped.

The file names given to all data and log files can be any desired name, although it is recommended that you select a name that gives some indication as to the content of the file. The file extensions for the primary data file, secondary data file(s), and log files can also be any chosen set of characters. It is recommended for consistency and standardization that the extensions be .mdf, .ndf, and .ldf for the primary, secondary, and log files, respectively.

Creating Database Files and Filegroups

All files needed for a database can be created through a single activity using SQL Server's Enterprise Manager, or with a single `CREATE DATABASE` Transact SQL statement. Either of these methods can be used to initialize all files and create the database and logs in a single step. The number of files you create and the placement of the files are addressed a little later in this chapter. (See the sections, "Using Filegroups" and "File Placement.")

The default names for the primary database and transaction log files are created using the database name you specified as the prefix—for example, *`NewDatabase_Data`*`.mdf` and *`NewDatabase_Log`*`.ldf`. These names and locations can be changed if desired from the default values provided for the new database file. The Transact SQL (T-SQL) syntax for creating a database is as follows:

```
CREATE DATABASE databasename
  [ON[<filespec>[,...n]][,<filegroup>[,...n ]]][LOG
➥ON{<filespec>[,...n ]}]
[COLLATE collationname][FOR LOAD|FOR ATTACH]
 <filespec>::=
  [PRIMARY]
  ([NAME=logicalfilename,][FILENAME='physicalfilename']
  ➥[,SIZE=size][,MAXSIZE={size|UNLIMITED}]
   [,FILEGROWTH=growthincrement])
  [,...n ]<filegroup>::=FILEGROUP filegroupname
  ➥<filespec>[,...n ]]
```

In the procedures that follow, you have the opportunity to create a database one step at a time. There are two techniques that can be used for these procedures. The first option uses the Create Database Wizard tool and Enterprise Manager in Step by Steps 3.1 and 3.2. The second option available provides for the syntax for the creation of a database using T-SQL code.

STEP BY STEP

3.1 Creating a Database Using the Create Database Wizard

1. Expand a server group, and then select the server in which to create a database.
2. On the Tools menu, click Wizards.
3. Expand Database.
4. Double-click Create Database Wizard.
5. Complete the steps in the wizard.

Or, if you prefer to use the Enterprise Manager:

STEP BY STEP

3.2 Creating a Database Using the Enterprise Manager

1. Expand a server group and then the server where the database is to be placed.
2. Right-click Databases, and then click New Database.
3. Enter a name for the database.
4. To specify how any specific file should grow, switch to the Data Files or Transaction Log tabs, highlight the file, select Automatically Grow File, then choose In Megabytes or By Percent and specify a value. To specify the file size limit, select Unrestricted Filegrowth or Restrict Filegrowth (MB) and specify a value.

To use T-SQL to create a database, use this code:

```
CREATE DATABASE MyDatabase
ON
(NAME = 'DataStore',
   FILENAME = 'd:\data directory\DataStore_MyDatabase.mdf',
   SIZE = 1MB, MAXSIZE = 5MB, FILEGROWTH = 1MB)
LOG ON
```

continues

continued

```
  (NAME ='LogStore',
     FILENAME = 'e:\log directory\LogStore_MyDatabase.ldf',
     SIZE = 1MB, MAXSIZE = 5MB, FILEGROWTH = 1MB)
```

You can use the `CREATE DATABASE` statement to create a database from script. Saving the script enables you to re-create a similar database on another server in the future. Any SQL Server object can have its creation script saved. The advantages of saving these scripts are discussed later in this chapter. Using the `CREATE DATABASE` statement to create a database using multiple files and log files would look similar to this:

```
CREATE DATABASE Example
ON
PRIMARY ( NAME = ExampleData,
FILENAME = 'c:\mssql\data\sampdat.mdf',
          SIZE = 10MB,
          MAXSIZE = 20MB,
          FILEGROWTH = 2MB),
        ( NAME = ExampleIndexes,
FILENAME = 'c:\mssql\data\sampind2.ndf',
SIZE = 10MB,
          MAXSIZE = 20MB,
          FILEGROWTH = 2MB),
        ( NAME = ExampleArchive,
FILENAME = 'c:\mssql\data\samparch.ndf',
SIZE = 10MB,
          MAXSIZE = 20MB,
          FILEGROWTH = 2MB)
LOG ON  ( NAME = ExampleLog1,
FILENAME = 'd:\mssql\log\samplog1.ldf',
SIZE = 10MB,
          MAXSIZE = 20MB,
          FILEGROWTH = 2MB),
        ( NAME = ExampleLog2,
FILENAME = 'd:\mssql\log\samplog2.ldf',
SIZE = 10MB,
          MAXSIZE = 20MB,
          FILEGROWTH = 2MB)
```

Important issues with regard to appropriate use of the `CREATE DATABASE` statement are as follows:

- The default growth increment measure is MB, but can also be specified with a KB or a % suffix. When % is specified, the growth increment size is the specified percentage of the size of the file at the time the increment occurs.
- A maximum of 32,767 databases can be defined on a server.
- The minimum size for a log file is 512K.

- Each database has an owner. The owner is the user who creates the database. The database owner can be changed through `sp_changedbowner`.
- The `Master` database should be backed up after a user database is created.
- The default unit of measure for the size and maxsize settings is MB if you supply a number, but no measure is provided. If no options are supplied, maxsize defaults to unlimited and the filegrowth is 10%.

At the time that you create the database and its associated files, you provide values to determine the initial file sizes, whether and how the files will grow, as well as some other basic database and file properties. The initial settings are used as a basis for future file system activities. If at a later date the initial settings are in need of alteration, you can perform this activity through the Enterprise Manager or by using the `ALTER DATBASE` T-SQL statement.

NOTE

Selecting a Secure Partition When interacting with a Windows 2000 or Windows NT Server operating system, ensure that all data is stored on an NTFS partition with appropriate security measures. NTFS provides for a flexible file system while maintaining a complete permission set for files and folders stored on disk. Using NTFS partitions helps prevent file tampering and allows for more flexible disk administration.

Using the Model Database

When you create a database for the first time, that database initially takes most of its attributes from the `Model` database. The `Model` database is a system database that SQL Server uses as a kind of template for database creations. It is a good and common practice to set the properties and contents of the `Model` database based on the majority of new databases that are to be created.

IN THE FIELD

OBJECT AND CODE REUSE

In practice, many objects are stored in the `Model` database to minimize the need to re-create these objects every time a database is created. Common elements placed in the `Model` often include specialized user-defined functions and data types that are present and frequently used by the development staff in their coding. In theory, objects are created for use in a single database, but all developers realize that object and code reuse is an important facet of easing the development process.

Often an object, such as a user-defined function, standard security role, or corporate information table, can be found in most if not all databases within a company. A property value, such as recovery level, might also have a standard implementation across all servers in the enterprise. If an object or property value is going to be present in most of the user databases, placing the object into the `Model` database or setting a property accordingly can save you the work of performing the activity as a post-creation task.

Using a Collation Sequence

A *collation sequence* is a set of rules governing over the characters that are used within a database and the means by which characters are sorted and compared. In SQL Server 2000 this sequence can be set on a database-by-database basis. In previous versions of SQL Server, the collation sequence was a server-wide setting. You therefore had to either perform a whole series of rebuilding actions to create a database that did not use the server collation, or install the database on a separate server altogether.

In SQL 2000 you can specify a non-default collation for any database on the server. This means that one database does not have to have the same characters or sorting rules as the rest of the databases on the server. If all but one or two of your databases have the same set of characters, then a single server can now implement the functionality that would have previously taken two separate machines.

To create a database with a non-default collating sequence, provide the `COLLATE` clause on the `CREATE DATABASE` command. You might also select the collation name from the drop-down box in the Enterprise Manager when you create the database from the GUI.

Be careful in the use of multiple collating sequences because it makes the transfer and entry of data more complex. It might also limit the application development environment and techniques normally used for data entry and editing.

Altering Database Properties

A number of the database properties affect the way in which some SQL Server commands operate. You can use the Enterprise Manager to make appropriate adjustments to some of the database properties. Alternatively you can use the `ALTER DATABASE` T-SQL statement to script these changes.

In altering a database, you can add or remove files and filegroups and/or modify attributes of the files and filegroups. `ALTER DATABASE` also enables you to set database properties, whereas in previous versions these properties could only be changed using the `sp_dboption` stored procedure.

WARNING

Be Sure of Your Collation Sequence After the collation sequence is set, it can be changed only through rebuilding of the database. If possible, collation decisions should be made during the logical design of the system so that you don't have to rebuild. Although collations can be different, if you want to change the sequence post creation, you will have to rebuild the database.

NOTE

Setting Options Using T-SQL The system-stored procedure `sp_dboption` can still be used to set database options, but Microsoft has stated that in future versions of SQL Server this functionality might not be supported.

Using Filegroups

In a lot of database scenarios, you will not implement more than one data file and one log file. In a number of instances, however, you might want to implement a *filegroup*. Filegroups enable a group of files to be handled as a single unit, and thus make implementations that require multiple files easier to accommodate. With filegroups, SQL Server provides an administrative mechanism of grouping files within a database. You might want to implement filegroups to spread data across more than one logical disk partition or physical disk drive. In some cases, this provides for increased performance as long as the hardware is sufficient to optimize reading and writing to multiple drives concurrently (see the section on "File Placement"). You might also have a performance gain through the appropriate placement of objects within these groups.

You can create a filegroup when a database is created, or you might add them in later when more files are needed or desired. After a filegroup has been assigned to a database, you cannot move its files to a different filegroup. Therefore, a file cannot be a member of more than one filegroup. SQL Server provides for a lot of flexibility in the implementation of filegroups. Tables, indexes, text, ntext, and image data can be associated with a specific filegroup, allocating all pages to one specific group. Filegroups can contain only data files; log files cannot be part of a filegroup.

Objects can easily be moved from one filegroup to another. Using the appropriate property page, you just select the new filegroup into which you wish to move the object.

Placement of Objects Within Filegroups

Placement of individual objects can aid in organizing data and at the same time provide for improved performance and recoverability. Many different objects can be assigned to separate files or filegroups.

For reasons given in the next few paragraphs, you might want to place the following objects into separate filegroups:

- Indexes
- A single table
- Text, ntext, or image columns

If you place indexes into their own filegroup, the index and data pages can be handled as separate physical read elements. If the associated filegroups are placed onto separate physical devices, then each can be read without interfering with the reading of the other. This is to say that while reading through an index in a sequential manner, the data can be accessed randomly without the need for manipulating the physical arm of a hard drive back and forth from the index and the data. This can improve performance and at the same time save on hardware wear and tear.

Placing an entire table onto its own filegroup offers many benefits. If you do so, you can back up a table without having to perform a much larger backup operation. Archived or seldom-used data can be separated from the data that is more readily needed. Of course the reverse is true: A table that needs to be more readily available within a database can be placed into its own filegroup to enable quicker access. In many instances, planned denormalization (the purposeful creation of redundant data) can be combined with this feature to obtain the best response.

Placing text, ntext, and image data in their own filegroup can improve application performance. Consider an application design that allows the data for these column types to be fetched only upon user request. Frequently, it is not necessary for a user to view pictures and extensive notes within a standard query. Not only does this accommodate better-performing hardware, but it can also provide faster query responses and less bandwidth saturation, because data that is not required is not sent across the network.

Considerations for Backup and Restore

Filegroups can provide for a more effective backup strategy for larger database environments. If a large database is placed across multiple filegroups, then the database can be backed up in smaller pieces.

This is an important aspect if the time to perform a full backup of the entire database is too lengthy.

To perform a backup in this manner, you would create a schedule to back up the individual filegroups (after an initial full database backup). In between each of the filegroup backups you then schedule log backups. Using this strategy enables you to break up an exceedingly large and long backup into more manageable increments.

After a determination has been made to use a filegroup strategy for storing data, always ensure that when a backup is performed against a filegroup that the indexes are also backed up at the same time. This is easily accomplished if the data and indexes are stored in the same filegroup. If they are located on separate filegroups, ensure that both the data and index filegroups are included in a single backup operation.

File Placement

After the decision has been made to go with filegroups, then comes the next major decision in the physical design: where to put the filegroups. Also, although logs are not stored into filegroups, they are stored in files and the placement of these files is very important.

WARNING

SQL Server Does Not Enforce Backup Be aware that SQL Server does not enforce backup of data and index filegroups in a single operation. You must ensure that the files associated with the indexes tied to a particular data set are backed up with the data during a filegroup backup.

Considerations in the placement within the file system depend on a number of variables. The first consideration is sequential versus random access. When a file is being read sequentially, the moving parts of the physical data device do less work (assuming no fragmentation). A large read/write process can use multiple physical devices at one time if they are placed on appropriate RAID hardware. Of course, there is also a software implementation of RAID that might not outperform the hardware one but is still beneficial.

Another consideration for file placement is system recoverability. When files are spread amongst multiple physical volumes, a fuller and faster recovery becomes possible in the event of hardware failure. Also, many other operations can benefit from appropriate file placement. The next four topics look at these considerations and discuss some of the instances where they each might be implemented.

NOTE

Log Files Logs are not stored in filegroups. You can, however, use multiple log files and place them in different locations to obtain better and more varied maintenance and allow more storage space for log content.

Sequential/Random Access Considerations

Many processes performed within SQL Server can be classified as sequential or random. In a sequential process, the data or file can be read in a forward progression without having to locate the next data to be read. In a random process, the data is typically more spread out, and getting at the actual physical data requires multiple accesses.

Where possible, it is desirable to keep sequential processes running without physical interruption caused by other processes contending for the device. Using file placement strategies to keep random processes separate from sequential ones enables the configuration to minimize the competition over the placement of the read/write heads.

In an ideal configuration (somewhat tongue in cheek), you might want to separate the operating system from its page file. You would then place the log onto its one drive, separate from the data, with the data configured over a RAID volume as described in the following section. Take the seldom-used data (column or table data) and separate it from data that will be accessed more frequently. Place the indexes off on their own volume as well, and for about $150.00–$200,000.00, you have the optimum performance in a database server. In fact, while you're at it, why not throw in a couple of extra network cards and a few processors?

Obviously, in most production environments the database team must balance an ideal configuration with the company bottom line. For many of these volume placements, a definitive cost must be budgeted.

As a minimum requirement for almost any implementation, you should separate the normal sequential processing of the log files from the random processing of the data. You also improve recoverability by separating the data from the log and placing them on separate physical volumes. If the volume where the data is stored is damaged and must be restored from backup, you will still have access to the last log entries. The final log can be backed up and restored against the database, which gives something very close to 100% recoverability right to the point of failure.

An interesting and flexible strategy is to provide a separate drive solely for the log. This single volume does not have to participate in RAID architecture, but RAID might be desired for full recoverability. If you give the log the space of an entire volume, you give the log more room to grow and accumulate more of the log over time without the need for periodic emptying. Less frequent log backups are needed and the best possible log performance is achieved.

RAID Considerations

RAID (Redundant Array of Independent/Inexpensive Disks) is a technology where two or more disk drives can be configured in such a manner as to provide

- ◆ Larger volumes, because space on multiple disks is combined to form a single volume.
- ◆ Improved performance, by interacting with more than one physical disk at a time (disk striping).
- ◆ Safeguarding data, by providing mechanisms (mirror or parity) for redundant data storage.

RAID is classified under many categories, each category assigned as a number. For more information about RAID hardware or other interesting information, visit the web site of the RAID advisory board: `http://www.raid-advisory.com/CIC.html`. This book is concerned with only three RAID levels, 0 (zero), 1, and 5, although there are many different other qualifications for RAID.

RAID 0 (stripe set) provides for multiple simultaneous read/write access across two or more disks. There is no data redundancy and thus no fault tolerance. A striped implementation is valid when strong backups exist and recovery time is not relevant. It might also be considered if the data is considered somewhat trivial and loss of data is unimportant. Parity sets provide optimum performance with no waste space allocated to data redundancy. Microsoft recommends a 64K stripe size, which should be considered if you are using RAID 0.

RAID 1 (mirror) provides the exact duplication of one volume onto another. This solution offers quick recoverability but has a performance cost. Everything written to one volume is then written a second time to the alternate volume. A mirror implementation is valid for operating system drives or any other system where speed is not as important as recovery time in the event of a failure. In most implementations, if the first drive fails, the system has little or no downtime because it can operate fully on the mirror drive. Mirrors are a more costly form of fault tolerance than parity sets, losing a full 50% of available space to data redundancy.

An alternative form of mirroring, duplexing, involves not only the duplication of hard drives but also redundant drive controllers. In using duplexing, you achieve fault tolerance over the loss of the controller as well as hard drive failure. To achieve up-to-the minute recovery in any failure, you might want to place your log files on a mirror or duplexed volume.

RAID 5 (parity set) provides the best read performance while still giving the recoverability through data redundancy. In a parity set, the data is written across all the available drives in segments referred to as *stripes*. In each stripe, all but one drive will contain data, with the remaining drive containing the parity check information. Each time data is written, a checksum is calculated and written to the parity segment. If a failure causes the loss of a disk drive, the parity segment can be used to enable the stripe set to be regenerated. RAID 5 is usually referred to as a poor man's mirror because the more drives that are included in the set, the more cost-effective this solution. For example, if three drives are used, a third of the available space is lost to redundancy. If ten drives are used there is only a 10% loss of usable space.

IN THE FIELD

RAID: SOFTWARE VERSUS HARDWARE

Even though software implementations of RAID must be known to pass certification exams and will be found in production systems, they are not nearly regarded as reliable as hardware RAID. For any high-volume, mission-critical application, it is therefore preferred to set up data redundancy mechanisms at the hardware level.

Recoverability in the Event of Failure

Two primary concerns in most data environments are data recoverability in the event of the inevitable failures and considerations for minimal downtime. In the industry, one of the optimum ratings to strive for is the elusive "five nines" (99.999). This rating means that over any given period of time (generally accepted standard of 365 days minimum), the server remained online and servicing the end user 99.999 percent of the time. In other words, the total downtime for an entire year is a little over 5 minutes.

In an attempt to achieve as little downtime as possible, it is essential to consider a strategy that involves multiple servers and redundant other hardware, as well as other issues on each machine. Data redundancy, adequate backups, and some form of disaster recovery plan must all be a part of a complete solution. Most of the topics surrounding server clustering fall out of the scope of this book, although the partitioned views will be discussed at length within the section "Multiple Server Implementations," later in this chapter. Other multi-server functionality, such as data replication, is addressed in Chapter 11, "Implementing and Understanding Replication Methodologies."

Though most of the topics related to recoverability fall into the realm of administration, you need to give some consideration to these processes when you put together a physical design. The following three sections explain these considerations as they pertain to database design.

System and Data Recovery

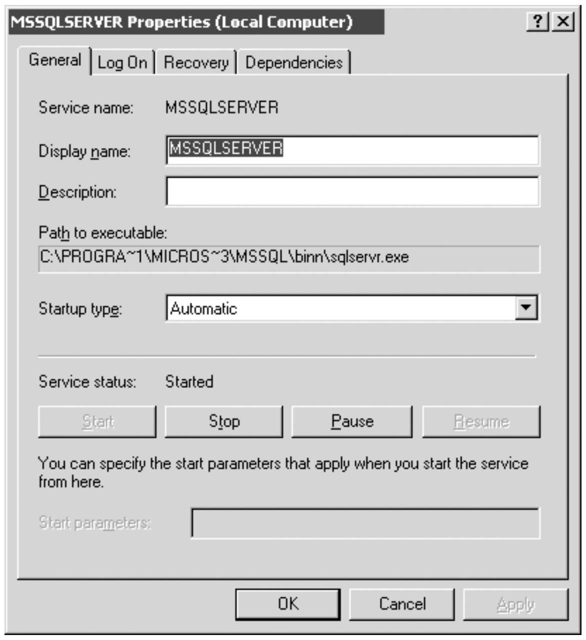
Recovering from outages and minimizing data loss in the event of hardware failures involves prior planning, adequate backups, and the setting of appropriate database and server options. On the server, the recovery interval and service startup options can be adjusted to lessen the time it takes for a SQL Server to be online and operational after a power failure or other serious service interruption. In each database, the recovery model can be set to determine the log usage and amount of lost data activity upon failure. Backups are one of the most important aspects of recovery. Backups must be maintained in a diligent and thorough manner. Finally, a plan of action that is regularly practiced must be part of a workable solution.

Operating System Service Properties

In SQL Server 2000, two aspects of the server allow for a successful server database restart in the event of failure. The operating system's services can be configured to automatically start upon computer startup and can also be set up to respond to service interruptions. To set service properties, you must locate the MSSQLSERVER service. This service can be found in your administrative tools, Services for Windows 2000, or Control Panel Services for NT. For the configuration options as displayed using the Windows 2000 services properties, see Figures 3.1 and 3.2. The database recovery interval can be set for the number of minutes each database takes to start up after an outage or controlled server startup. You can find the Recovery

Interval option in the Enterprise Manager by right-clicking the server, selecting Properties from the pop-up menu, and navigating to the Database Settings tab.

FIGURE 3.1
General properties for operating system services.



MSSQLSERVER Properties (Local Computer)

General | Log On | Recovery | Dependencies

Service name: MSSQLSERVER

Display name: MSSQLSERVER

Description:

Path to executable:
C:\PROGRA~1\MICROS~3\MSSQL\binn\sqlservr.exe

Startup type: Automatic

Service status: Started

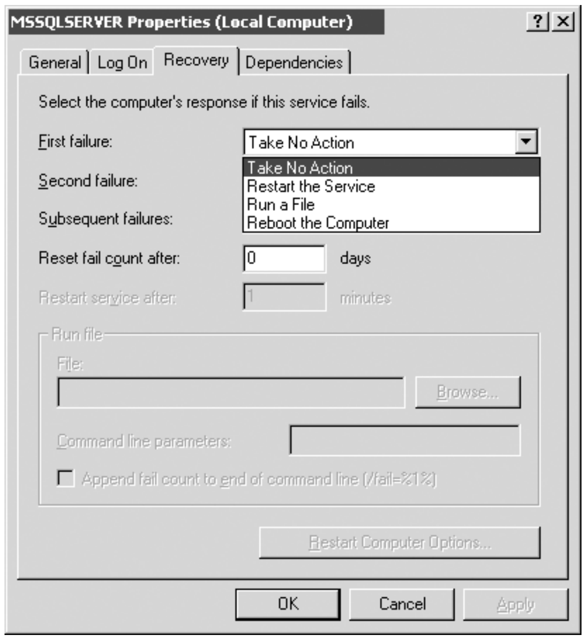
Start | Stop | Pause | Resume

You can specify the start parameters that apply when you start the service from here.

Start parameters:

OK | Cancel | Apply

FIGURE 3.2
Recovery properties for operating system services.



MSSQLSERVER Properties (Local Computer)

General | Log On | Recovery | Dependencies

Select the computer's response if this service fails.

First failure: Take No Action

Take No Action
Restart the Service
Run a File
Reboot the Computer

Second failure:

Subsequent failures:

Reset fail count after: 0 days

Restart service after: 1 minutes

Run file

File:

Browse...

Command line parameters:

Append fail count to end of command line (/fail=%1%)

Restart Computer Options...

OK | Cancel | Apply

Use of Recovery Models

Some of the database properties that were available in previous releases of SQL Server have been combined to form what is referred to in SQL Server 2000 as *recovery models*. Setting the appropriate model can enable most if not all data activity to be recovered in the event of system failures. Three models are supported: Simple, Bulk-Logged, and Full.

The Simple recovery model ensures higher performance during bulk copy operations and maintains a smaller database log. However, the model does not support transaction log backups and, therefore, there will be data loss in the event of a failure because the database can be restored only to the point of the last full or differential database backup.

Bulk-Logged recovery also allows for high-performance bulk procedures that use minimal log space. Some data might be lost, but because you can perform log backups, usually the only loss will be bulk operations since the last backup.

If recoverability to a specific point in time with as little data loss as possible is the goal, then the Full recovery model should be set. The Full recovery model makes the heaviest use of the database logs.

NOTE

User Accounts for Services
Separate user accounts can be identified for each of the SQL Server services. Conversely, the same account can be used for all services and several servers. As a standard implementation, it is usually best to use the same account. You might want to use separate accounts for each server, particularly if you want each server to send and receive email as a separate identity.

Backup and Disaster Recovery

Usually backup and disaster recovery falls into the realm of the database and network administrators. In a total systems design strategy, a complete plan of recovery activity that includes a variety of scheduled backups and other tasks is documented and tested. This disaster recovery plan will be updated as needed, because test recovery and practicing the plan is sure to point out anything that might be otherwise missed. Though not a specific detail of implementation, the topic of recoverability would not be complete without at least the mention of a system-wide plan. Consider a regular exercise of simulating failures to test the plan.

Standby Servers and Log Shipping

A warm backup server or standby server is a lower-cost implementation that is often selected as an alternative to replication or clustering. The premise is to back up the production server on a regular basis, restoring it to a second machine that can be put into production in the event of failure in the first computer. A standby server can also assist in taking some of the workload from the production machine if it is used as a read-only query server.

In SQL Server 2000 you can use the Maintenance Plan Wizard to implement a standby server configuration. The wizard prompts you through the configuration of backups and regularly scheduled log shipments to the standby machine.

Space Requirements

Over time, the size of the database will need to be adjusted to accommodate new data or data removal. The configuration of the ideal server in any given situation will vary greatly. The applications that a server is intended to handle usually point toward the size of machine needed and its associated peripherals.

For a general guideline or minimum starting point, consider the following:

- **Multiple processors**. Preferred in most database environments (keep in mind that licensing models change).
- **RAM**. Can you ever have enough RAM? Start out with 1GB and don't be afraid to work your way up.
- **OS drive mirror**. Two physical disks set up in a physical or software mirror. In some cases, the use of two physical controllers provides for complete disk duplexing.
- **Data parity array**. A number of separate physical drives. A number of 4 to 6 usually provides an adequate size volume, but this might vary in larger systems.
- **Log volume**. One disk used for log storage. In some cases, this volume also stores data files for implementations that include archived data storage. You might also want to mirror this volume to ensure up-to-the-minute data recovery.

NOTE

Licensing Models With the release of SQL Server 2000, the licensing models available included Per Seat or Per Processor. The Per Server model has been discontinued.

File Growth Strategies

SQL Server 2000 enables you to set database files so that they expand and shrink automatically as needed, eliminating the need for additional administration. By default, SQL Server enables data files to increase in size as needed for data storage. Therefore, a file can grow to the point where all disk space is exhausted. You can specify that a file is not to grow beyond its creation size or implement a maximum size for file growth. Ensure that disk space is not exhausted by using the `MAXSIZE` option of the `CREATE DATABASE` or `ALTER DATABASE` statements to indicate the largest size to which a file can grow.

In a volatile environment, the database and its related files might frequently increase and decrease in size and this activity might be the desired operation of the server. In most instances, an implementation providing for more stability in the file system is the desired end result. A determination has to be made as to whether the database stays at about the same size or grows or shrinks over time. In most scenarios, a database grows over time and needs to be reduced only when data is archived.

When creating the files, you should set the `SIZE`, `MAXSIZE`, and `FILEGROWTH` parameters so that the database can increase in volume over time. The `FILEGROWTH` configuration should be implemented in larger increments so that growth within the file system isn't occupying too much of the server's resources. Growth of files occurs in the background and can be minimized by using a larger growth increment. Always provide a `MAXSIZE` entry even if the entry itself is close to the capacity of the volume.

Shrinking Files

File "shrinking" might be required as an application ages. In most operations, the older the data is, the less valuable its presence is among the mainstream data. As data ages, it is less likely to be queried and thus is passed over by most reads. It might become "wasted space" in the database and unnecessarily consume system resources. A system design usually includes means by which data is aged out into archive tables. After the archival process has completed, there might be a high percentage of empty space in the data files.

You can shrink each file within a database to remove unused pages. This applies to both data and log files. It is possible to shrink a database file manually or as a group. You use the `DBCC` statement with the `SHRINKDATABASE` or `SHRINKFILE` parameters (`DBCC` parameters are shown in the Fast Facts section in Part II "Final Review"). Use `DBCC SHRINKDATABASE` to shrink the size of the data files in the specified database, or you can selectively choose a specific file and shrink its size using `DBCC SHRINKFILE`.

You can set the database to automatically shrink at periodic intervals by right-clicking the database and selecting the database Properties page from within the Enterprise Manager.

Ongoing System Maintenance

After a database and associated files have been created and the implementation is complete, it's necessary to maintain the system using the periodic application of several commands. It might be necessary to adjust the database and file properties as the database system matures. In addition, DBCC (Database Consistency Checker) has a number of parameters to assist in regular maintenance activities.

As a starting point, use the SQL Server Maintenance Wizard to perform the necessary maintenance tasks. Adjust the scheduling of these tasks as needed to maintain a healthy server. Watch and adjust indexing and data structures because over time they will become fragmented. Indexing and data structures as well as other database objects are discussed more fully after the Review Break.

REVIEW BREAK

Physical Storage

Creating and altering databases involves selecting the physical volume type for each database file, setting the appropriate file properties, placing the objects into the files/filegroups, and ensuring appropriate adjustments are made as the database matures. The type of business needs that the database is being designed to meet helps to indicate the measures needed to ensure adequate performance.

Try to place onto separate volumes any files that might tend to compete with each other for read cycles during a single operation. Place log files away from the data to ensure adequate recovery and make sure that database properties have been set in such a way as to ensure that maintenance tasks can be performed.

Creating and Altering Database Objects

- ▶ **Create and alter database objects. Objects include constraints, indexes, stored procedures, tables, triggers, user-defined functions, and views.**

The next thing to consider is the creation of objects within the database. Database objects include constraints, indexes, stored procedures, tables, triggers, user-defined functions, views, and more. Each object is discussed in detail, paying particular attention to the impact on the system as a whole. In many implementations, there are several different approaches to meeting a particular need. Selecting the appropriate technique for a task requires trade-offs between functionality, performance, and resource utilization.

Each database contains a number of tables other than those used to store data. These tables store information that enables SQL Server to keep track of objects and procedures within the database. The sysobjects and syscomments system tables maintain entries containing the object definitions and other tracking information for each object. A number of other tables also exist to maintain information about specific objects. For more information regarding system tables, refer to SQL Server Books Online. These tables are used whenever SQL Server needs object information. You should never alter system tables directly but instead allow SQL Server to manipulate the entries as needed.

To help you secure the server, you might choose not to display system objects to the user from the Enterprise Manager interface. Also, hiding these objects from the user presents a cleaner interface to objects with which the user normally interacts. Step by Step 3.3 describes how to hide system objects:

STEP BY STEP

3.3 Setting Registration Options

1. Select the server from the Enterprise Manager interface.
2. Right-click to access the server menu.

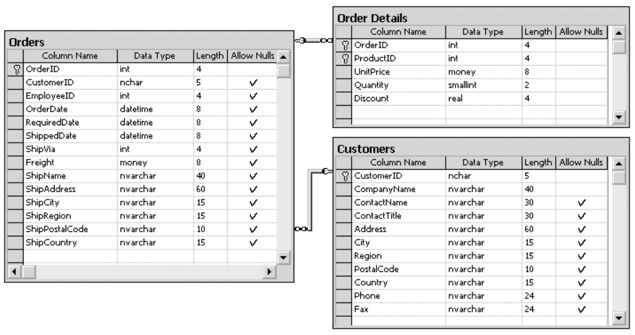
3. Select the option to Edit SQL Server Registration Properties.
4. Clear the Show System Databases and System Objects check box.

Table Characteristics

The makeup of a table in SQL Server is more than just simply data definition. A complete table definition includes column descriptions, storage location, constraints, relationships with other tables, indexes, and keys, as well as table-level permissions and text indexing columns.

When defining tables, it is a good idea to have some form of data dictionary prepared to help you make appropriate choices for individual properties. A data dictionary defines data usage and is an extension of the logical data modeling discussed in Chapter 1. In SQL Server, the term "database diagram" is usually used rather than "dictionary," although a database diagram is not a complete data dictionary in the sense of documentation.

A *data dictionary* is a form of documentation generally considered a complete reference for the data it describes. The dictionary is usually a lot more than just a collection of data element definitions. A complete dictionary should include schema with reference keys and an entity-relationship model of the data elements or objects. A pseudo data dictionary can be represented using the database diagram tool provided with SQL Server. A partial dictionary for the `Northwind` database is illustrated in Figure 3.3.



Orders

Column Name	Data Type	Length	Allow Nulls
OrderID	int	4	
CustomerID	nchar	5	✓
EmployeeID	int	4	✓
OrderDate	datetime	8	✓
RequiredDate	datetime	8	✓
ShippedDate	datetime	8	✓
ShipVia	int	4	✓
Freight	money	8	✓
ShipName	nvarchar	40	✓
ShipAddress	nvarchar	60	✓
ShipCity	nvarchar	15	✓
ShipRegion	nvarchar	15	✓
ShipPostalCode	nvarchar	10	✓
ShipCountry	nvarchar	15	✓

Order Details

Column Name	Data Type	Length	Allow Nulls
OrderID	int	4	
ProductID	int	4	
UnitPrice	money	8	
Quantity	smallint	2	
Discount	real	4	

Customers

Column Name	Data Type	Length	Allow Nulls
CustomerID	nchar	5	
CompanyName	nvarchar	40	
ContactName	nvarchar	30	✓
ContactTitle	nvarchar	30	✓
Address	nvarchar	60	✓
City	nvarchar	15	✓
Region	nvarchar	15	✓
PostalCode	nvarchar	10	✓
Country	nvarchar	15	✓
Phone	nvarchar	24	✓
Fax	nvarchar	24	✓

FIGURE 3.3
Database diagram showing column properties and table relationships.

Column Definition

After the file structure and content of each file has been determined, the tables themselves can be created and assigned to the files. If the purpose of the table is to hold data that is frequently accessed, then the file placement of the table should take that into consideration. Tables that hold archive data and other less frequently accessed data require less maintenance and don't have to be as responsive to user queries.

The initial definition of each column within a table consists of a name for the column, the type and length of data for the column, and an indicator as to whether the column must have data or allow `NULL` content. A number of additional column descriptors can be included to define characteristics of how the column obtains its value and how the column is treated within the table. A complete list of potential column descriptors is as follows:

- **Column Name.** Should be meaningful so as to describe the column content.
- **Data Type.** Any one of 25 possible definitions provides the basis for the data a column will contain. Choices include several possibilities for each data type. (Data types are discussed more fully later in this book.)
- **Length.** For many of the data types, the length is predetermined. You must, however, specify a length for character, Unicode (`nCHAR`), and binary data. A length must also be specified for variable-length data columns. If a `char` or `nCHAR` data type is only a single character, then no length has to be defined.
- **Allow Nulls.** You can provide an indicator for allowing `NULL` content for any variable except those assigned as primary keys.
- **Primary Key.** Enforces unique content for a column and can be used to relate other tables. Must contain a unique non-`NULL` value.
- **Description.** Provides an explanation of the column for documentation purposes. (This is an extended table property.)
- **Default Value.** Provides a value for a column when one is not explicitly given during data entry. A default object must be created and then bound to a column, but the preferred technique is to provide the default definition, directly attached to the column in the `CREATE/ALTER` table definition. It is defined at the database level and can be utilized by any number of columns in a database.

NOTE

Object Placement Keep in mind when assigning objects to files that some objects can be placed away from the mainstream data through the use of filegroups. You can select the object placement from Table Design Properties in the Enterprise Manager or through the use of an `ON` clause in a `CREATE/ALTER` statement. SQL Server enables you to place the following table objects:

- Tables
- Indexes
- Text, nText, or Image data

- **Precision.** The number of digits in a numeric column.
- **Scale.** The number of digits to the right of a decimal point in a numeric column.
- **Identity.** Inserts a value automatically into a column, based on seed and increment definitions.
- **Identity Seed.** Provides the starting value for an Identity column.
- **Identity Increment.** Defines how an Identity will increase or decrease with each new row added to a table.
- **Is RowGuid.** Identifies a column that has been defined with the Unique Identifier data type as being the column to be used in conjunction with the `ROWGUIDCOL` function in a `SELECT` list.
- **Formula.** Provides a means of obtaining the column content through the use of a function or calculation.
- **Collation.** Can provide for a different character set or sort order than other data. (Use with extreme caution if at all because it impairs front-end development, capability and hampers data input and alteration processes.)

Many characteristics of column definitions affect other columns, tables, and databases. For a more complete definition of any of these properties, consult SQL Server Books Online.

Using *CHECK* Constraints

A `CHECK` constraint is one of several mechanisms that can be used to prevent incorrect data from entering the system. Restrictions on data entry can be applied at the table or column level through the use of a `CHECK` constraint. You might also apply more than a single check to any one column, in which case, the checks are evaluated in the order in which they were created.

A `CHECK` constraint represents any Boolean expression that is applied to the data to determine whether the data meets the criteria of the check. The advantage of using a check is that it is applied to the data before it enters the system. However, `CHECK` constraints do have less functionality than mechanisms, such as stored procedures or triggers. You can find a comparison of features for a number of these mechanisms with provisions for where each one is applied at the close of this section, just before the "Review Break."

One use for a `CHECK` constraint is to ensure that a value entered meets given criteria based on another value entered. A table-level `CHECK` constraint is defined at the bottom of the `ALTER/CREATE TABLE` statement, unlike a `COLUMN CHECK` constraint, which is defined as part of the column definition. For example, when a due date entered must be at least 30 days beyond an invoice date, a table-level constraint would be defined as:

```
(DueDate – InvoiceDate) >= 30
```

A column-level check might be used to ensure that data is within acceptable ranges, such as in the following:

```
InvoiceAmount >= 1 AND InvoiceAmount <= 25000
```

A check can also define the pattern or format in which data values are entered. You might, for example, want an invoice number to have an alphabetic character in the first position, followed by five numeric values, in which case, the check might look similar to the following:

```
InvoiceNumber LIKE '[A-Z][0-9][0-9][0-9][0-9][0-9]'
```

Finally, you might want to apply a check where an entry must be from a range of number choices within a list. An inventory item that must be one of a series of category choices might look similar to this:

```
ProductCategory IN ('HARDWARE', 'SOFTWARE', 'SERVICE')
```

A `COLUMN CHECK` (or other constraint) is stated as a portion of the column definition itself and applies only to the column where it is defined. A `TABLE CHECK` (or other constraint), on the other hand, is defined independently of any column, can be applied to more than one column, and must be used if more than one column is included in the constraint.

A table definition that is to define restrictions to a single column (minimum quantity ordered is 50), as well as a table constraint (date on which part is required must be later than when ordered), would be as follows:

```
CREATE TABLE ProductOrderLine
      (ProductLineKey  BigInt,
       OrderMatchKey   BigInt,
```

continues

continued

```
     ProductOrdered  Char(6),
     QtyOrdered      BigInt
       CONSTRAINT Over50 CHECK (QtyOrdered > 50),
     OrderDate       DateTime,
     RequiredDate    DateTime,
       CONSTRAINT CK_Date CHECK (RequiredDate >
       ➥OrderDate))
```

Usually a single table definition would provide clauses for key definition, indexing, and other elements that have been left out of the previous definition to focus in more closely on the use of CHECK constraints.

Index Organization

Putting the data into sequence to accommodate quick retrieval, and at the same time provide meaningful and usable output to an application, usually requires that a variety of indexes be defined. A *clustered index* provides the physical order of the data storage, whereas a *nonclustered index* provides an ordered list with pointers to the physical location of the data.

Indexing is most easily defined and understood if you compare the data and index storage of a database to that of a book. In a book, the data itself is placed onto the pages in a sequence that is meaningful if you read the book sequentially from cover to cover. An index at the back of the book enables you to read the data randomly. You can locate a topic by looking through a list of topics that is accompanied by a physical page reference to the place where the topic can be found. To read a single topic you need not skim through the entire book.

In a similar manner, data in a database can be handled randomly or in sequence. The location of a single record can be found in the database by looking it up in the index, rather than reading through all the rest of the data. Conversely, if a report is to be generated from all the data in a database, the data itself can be read sequentially in its entirety.

Index storage in SQL Server has a B-tree structured storage. The indexes are maintained in 8KB pages qualified as *root*, *intermediate*, and *leaf-level* pages. In a clustered index, the leaf level is the data itself, and all other levels represent index pages. In a nonclustered index, all pages contain indexes (see Figure 3.4).

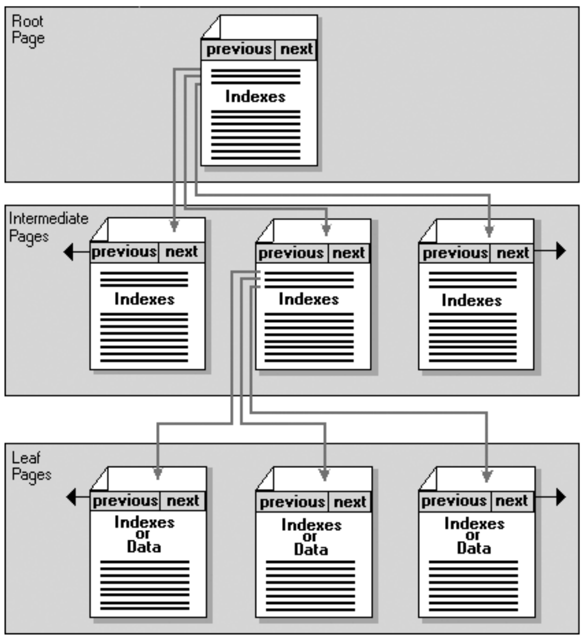


FIGURE 3.4
Illustration of the B-tree structure used for index storage.

If a clustered index has not been defined for a given table, then the data is stored in a "heap." A data heap does not maintain data in any particular order; it simply stores the data in the order in which it is entered. In some applications, where data is never retrieved in any particular order on a regular basis, this might actually be advantageous.

Indexes can be created using the T-SQL `CREATE INDEX` command. When you use the Enterprise Manager to create an index, you must access the table design and then reveal the table properties.

Step by Step 3.4 shows you how to use the Enterprise Manager to create an index.

STEP BY STEP

3.4 Selection of Index Creation from the Enterprise Manager

1. From the Enterprise Manager, select the table with which the index will be associated.

continues

continued

2. Right-click on the table name and select "Design Table" from the pop-up menu.
3. Select Table and Index Properties.
4. Click on the Indexes/Keys tab.
5. Click on the New button.
6. From the drop-down list boxes, select the column on which to base the index and whether the index is to be ascending or descending. If you are creating a compound index based on a number of columns, then add additional columns as needed.
7. Select a filegroup for the storage of the index.
8. If desired, you can make the index unique, supply fill factor and pad index values, make the index clustered (default is nonclustered), and choose to not recalculate statistics for the index.
9. Close the dialog box to save the index and exit table design properties and select OK to save the changes.

Create an index using the T-SQL `CREATE INDEX`. The following example creates a compound, nonclustered index that is 75% full:

```
CREATE INDEX IXProductItem
ON ProductOrderLine (OrderMateKey, ProductLineKey)
WITH FILLFACTOR = 75
```

Clustered Indexing

The selection of the appropriate column(s) on which to base a clustered index is important for a number of reasons. As previously mentioned, a clustered index represents the order in which the data is physically stored on the disk. For this reason, you can define only a single clustered index for any table. If you choose not to use a clustered index in a table, the data on disk will be stored in a heap. A clustered index, if present, has clustering keys that are used by all nonclustered indexes to determine the physical location of the data.

The basis for the index usually is determined by the order in which the majority of applications and queries want their output. The clustered index values are also present in other indexes and the size of the defined index should be kept as small as possible. When you select a clustering key, try to utilize a numeric data type because character types cause index storage to occupy much more space.

Always define a clustered index first before you define any of the nonclustered indexes. If you do these tasks in reverse order, then all nonclustered indexes rebuild themselves upon creation of the clustered index.

Nonclustered Indexing

Nonclustered indexes provide a means of retrieving the data from the database in an order other than that in which the data is physically stored. The only alternative to the use of these indexes would be provisions for a sort operation that would place undue overhead on the client system and might not produce the desired response times. A data sort implementation is usually performed only for one-time operations or for applications that will have very limited usage.

Although the creation of indexes saves time and resources in a lot of cases, avoid the creation of indexes that will rarely be utilized. Each time a record is added to a table, all indexes in the table must be updated, and this might also cause undue system overhead. For that reason, careful planning of index usage is necessary.

Unique Indexing

At times when indexes are created, it is important to guarantee that each value is distinctive. This is particularly important for a primary key. SQL Server automatically applies a unique index to a primary key to ensure that each key value uniquely defines a row in the table. You might want to create additional unique indexes for columns that are not going to be defined as the primary key.

Leaving Space for Inserts

Fill factor is the percent at which SQL Server fills leaf-level pages upon creation of indexes. Provision for empty pages enables the server to insert additional rows without performing a page-split operation. A page split occurs when a new row is inserted into a table that has no empty space for its placement. As the storage pages fill, page splits occur, which can hamper performance and increase fragmentation.

IN THE FIELD

MORE ON THE FILL FACTOR

You will normally find that queries (the reading of existing data) outweigh data updates by a substantial margin. Providing the extra room slows down the query process. Therefore, you might not want to adjust the fill factor value at all.

Equally, setting the fill factor too low hampers read performance because the server must negotiate a series of empty pages to actually fetch the desired data. It is beneficial to specify a fill factor when you create an index on a table that already has data and will have a high volume of inserts. If you do not specify this setting when creating an index, the server default fill factor setting is chosen. The fill factor for the server is a configuration option set through the Enterprise Manager or the `sp_configure` stored procedure.

The percentage value for the fill factor is not maintained over time; it applies only at the time of creation. Therefore, if inserts into a table occur frequently, it's important to take maintenance measures for rebuilding the indexes to ensure that the empty space is put back in place. A specific index can be rebuilt using the `CREATE INDEX` T-SQL command with the `DROP EXISTING` option. Indexes can also be de-fragmented using the `DBCC INDEXDEFRAG` command, which also reapplies the fill factor.

The Pad Index setting is closely related to the setting for fill factor to allow space to be left in non-leaf levels. Pad Index cannot be specified by itself and can be used only if you supply a fill factor. You do not provide a value to this setting; it matches the setting given for the fill factor.

Maintaining Referential Integrity

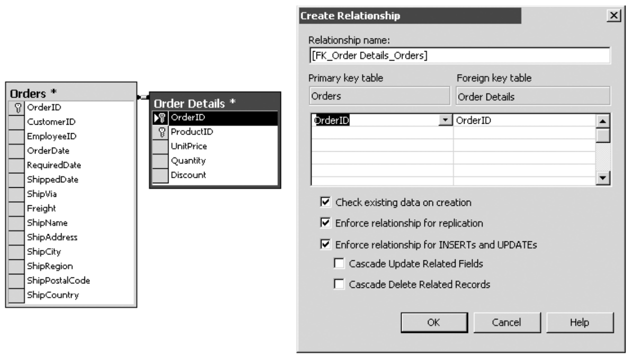
When multiple tables maintained in a database are related to each other, some measures should be taken to ensure that the reliability of these relationships stays intact. To enforce referential integrity, you create a relationship between two tables. This can be done through the database diagram feature of the Enterprise Manager or through the `CREATE` and `ALTER TABLE` T-SQL statements. Normally, you relate the referencing or Foreign Key of one table to the Primary Key or other unique value of a second table.

Step by Step 3.5 shows you how to use a database diagram to define a relationship:

STEP BY STEP

3.5 Using the Wizard to Create a Relationship

1. From the Enterprise Manager, expand the tree view of the database you want to use and select Diagrams.
2. Right-click Diagrams and select New Database Diagram from the pop-up menu.
3. A wizard asks you for the tables to use in the diagram. Select the tables to be related and follow the wizard to completion.
4. Select the column to be related from the subsidiary table, and drag the column to the primary table.
5. Complete the desired options from the dialog, and press OK to establish the relationship (see Figure 3.5).
6. Exit the diagram, and select Yes to save the changes to the respective tables.



Relationship name: [FK_Order Details_Orders]

Primary key table	Foreign key table
Orders	Order Details
OrderID	OrderID

- [✓] Check existing data on creation
- [✓] Enforce relationship for replication
- [✓] Enforce relationship for INSERTs and UPDATEs
  - [ ] Cascade Update Related Fields
  - [ ] Cascade Delete Related Records

FIGURE 3.5
The Create Relationship dialog box as seen from the database diagram window.

Step by Step 3.6 shows you how to define a relationship from the Table Design Properties box:

STEP BY STEP

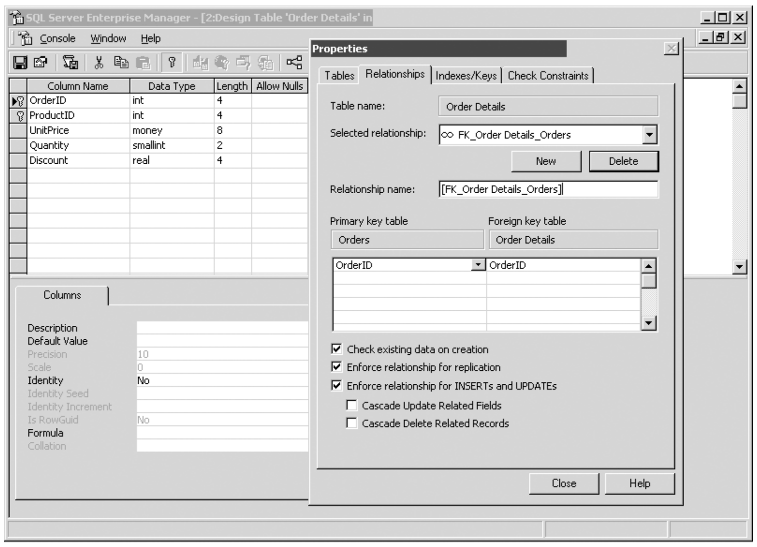
3.6 Using the Table Designer to Create a Relationship

1. From the Enterprise Manager, select the table that contains the Foreign Key element of the relationship.
2. Right-click on the table name that will represent the subsidiary table to be referenced, and select Design Table from the pop-up menu.
3. Select Table and Index Properties.
4. Click on the Relationships tab and click New.
5. Select the desired options from the dialog, and click Close to establish the relationship (see Figure 3.6).
6. Exit and save changes to the table.

You can define a relationship when creating or altering a table definition. The following example defines a relationship using T-SQL:

```
CREATE TABLE OrderDetails
     ( DetailsID          smallint,
       OrderID            smallint
          FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
       QtyOrdered bigint,
       WarehouseLocation  smallint
      )
```

The most common relationships are one-to-many, in which the unique value in one table has many subsidiary records in the second table. Another form of relationship, which is usually used to split a table with an extraordinary number of columns, is a one-to-one relationship. The use of one-to-one splits a table and associates a single unique value in one table with the same unique value in a second table. A many-to-many relationship can also be defined, but this form of referencing requires three tables and is really two separate one-to-many relationships.



Column Name	Data Type	Length	Allow Nulls
OrderID	int	4	
ProductID	int	4	
UnitPrice	money	8	
Quantity	smallint	2	
Discount	real	4	

Description	
Default Value	
Precision	10
Scale	0
Identity	No
Identity Seed	
Identity Increment	
Is RowGuid	No
Formula	
Collation	

Table name: Order Details

Selected relationship: FK_Order Details_Orders

Relationship name: [FK_Order Details_Orders]

Primary key table	Foreign key table
Orders	Order Details
OrderID	OrderID

- ✓ Check existing data on creation
- ✓ Enforce relationship for replication
- ✓ Enforce relationship for INSERTs and UPDATEs
  - ☐ Cascade Update Related Fields
  - ☐ Cascade Delete Related Records

FIGURE 3.6
Create Relationship options dialog as seen from the table diagram window.

Utilizing referential integrity guidelines helps maintain the accuracy of data entered into the system. A database system uses referential integrity rules to prohibit subsidiary elements from being entered into the system unless a matching unique element is in the referenced table. The system also protects the data from changes and deletions, assuming that cascading actions (defined later in this chapter) have been carefully and properly implemented.

Primary and Foreign Keys

The definition of a Primary Key for each table, though not a requirement of the SQL Server database environment, is recommended. A Primary Key helps records maintain their identities as unique rows of a table and also provides a means of relating tables to other tables in the database to maintain normal forms. (For further information on normalization and normal forms, see Chapter 2, "Data Modeling.") A Foreign Key is defined in a subsidiary table as a pointer to the Primary Key or other unique value in the primary table.

Both Primary and Foreign Keys are defined in the form of a constraint. The pair of keys work together to accommodate table relationships. A Foreign Key refers back to the Primary Key in the parent table, forming a one-to-one or one-to-many relationship. (To see more about relationships, refer to Chapter 2.)

Primary Key Constraint

A Primary Key constraint enforces entity integrity in that it does not permit any two rows in a table to have the same key value. This enables each row to be uniquely defined in its own right. Although a Primary Key should be created when a table is initially created, it can be added or changed at any time after creation.

A Primary Key cannot have `NULL` content nor can there be any duplicate values. SQL Server automatically creates a unique index to enforce the exclusiveness of each value. If a Primary Key is referenced by a Foreign Key in another table, the Primary Key cannot be removed unless the Foreign Key relationship is removed first. A Primary Key is easily assigned in the table design window by either of the following actions:

- Right-click on the desired column name and select the Set Primary Key option. To select a compound key based on more than one column, hold down the Ctrl key while selecting multiple columns before right-clicking.
- Select the desired field and click the Set Primary Key button on the toolbar. To select a compound key based on more than one column, hold down the Ctrl key while selecting the appropriate columns.

WARNING

Documentation Discrepancy The capability to set a relationship to any unique column is not noted in most SQL Server documentation. SQL Server Books Online reports that a Foreign Key must be set to a Primary Key or a `UNIQUE` constraint. In SQL Server, you can create a relationship against a Primary Key, `unique index`, or a `UNIQUE` constraint. You need not have a Primary Key or constraint. You can even set a unique index to ignore duplicates and the operation will still be permitted.

EXAM TIP

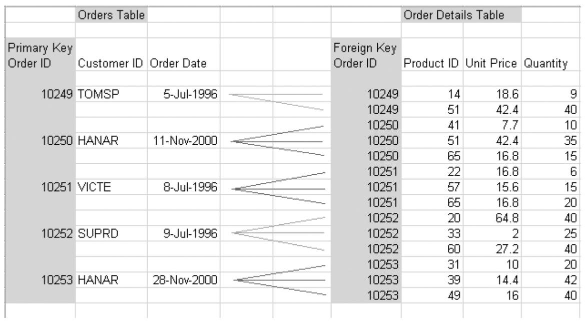
Go with the Documentation If you run into this on the exam, the correct answer is likely to be one chosen based on the documentation and not on actual functionality. The capability to set a relationship to any unique column is not noted in documentation. The correct technique to use when answering an exam question would be one that involves a Foreign Key set to a Primary Key or Unique constraint.

Foreign Key Constraint

A Foreign Key constraint is defined so that a primary and subsidiary table can be linked together by a common value. A Foreign Key can be linked to any unique column in the main table; it does not necessarily have to be linked to the Primary Key. It can be linked to any column that is associated with a unique index.

With a Foreign Key defined, you cannot add a value to the Foreign Key column if a matching value is not present in the primary table. For instructions on setting a Foreign Key constraint, see the section on referential integrity, earlier in this chapter.

Note in the example shown in Figure 3.7 that there are matching Order IDs in the child `Order Details` table for only those Order IDs included in the parent `Orders` table. An Order ID must match from a child to a parent. If a child entry with an ID were not found in the parent table, then that is known as an *orphan child* and would be a breach of referential integrity rules.



	Orders Table					Order Details Table		
Primary Key Order ID	Customer ID	Order Date			Foreign Key Order ID	Product ID	Unit Price	Quantity
10249	TOMSP	5-Jul-1996			10249	14	18.6	9
					10249	51	42.4	40
					10250	41	7.7	10
10250	HANAR	11-Nov-2000			10250	51	42.4	35
					10250	65	16.8	15
					10251	22	16.8	6
10251	VICTE	8-Jul-1996			10251	57	15.6	15
					10251	65	16.8	20
					10252	20	64.8	40
10252	SUPRD	9-Jul-1996			10252	33	2	25
					10252	60	27.2	40
					10253	31	10	20
10253	HANAR	28-Nov-2000			10253	39	14.4	42
					10253	49	16	40

FIGURE 3.7
Primary Key/Foreign Key referential integrity.

Using Cascade Action to Maintain Integrity

New to SQL Server with the 2000 release is a cascading action feature that many other database environments have been enjoying for quite some time. Cascading actions affect update and delete activity where an existing Foreign Key value is changed or removed. Cascade action is controlled through the `CREATE` and `ALTER TABLE` statements, with clauses for `ON DELETE` and `ON UPDATE`. You can also select these features using the Enterprise Manager.

In a cascading update, when you change the value of a key in a situation where a Foreign Key in another table references the key value, those changed values are reflected back to the other tables. A similar thing happens with a delete operation: if a record is deleted, then all subsidiary records in other tables are also deleted. For example, if an invoice record is deleted from an invoice table that has invoice details stored in another table and referenced by a Foreign Key, then the details would also be removed.

A series of cascading actions could easily result from the update or deletion of important keys. For example, the deletion of a customer could cause the deletion of all that customer's orders, which could cause the deletion of all its invoices, which in turn could cause the deletion of all the customer's invoice details. For this reason, careful system design is important and the potential archival of data through the use of triggers should be considered.

In the case of multiple cascading actions, all the triggers to be fired by the effects of the original deletion fire first. `AFTER` triggers then fire on the original table and then the `AFTER` triggers in the table chain subsequently fire.

EXAM TIP

Cascading Actions Is a New Feature You can expect that something about it will be asked on the exam. Also be prepared for the exam by knowing all the results and implications of cascading actions. For example, you might be asked what occurs when a record contained in the parent table is deleted, or has its key value changed.

Stored Procedures

A *stored procedure* is a set of T-SQL statements that can be saved as a database object for future and repeated executions. With stored procedures, you can enable a lot of the development and processing to be performed on the server, producing much more efficient and lightweight front-end applications. Any commands that can be entered via SQL Query tools can be included in a stored procedure.

Using stored procedures is a powerful and flexible technique for performing tasks within an application. A stored procedure, when it is first used, is compiled into an execution plan that remains in the procedure cache. This provides for some of the performance over ad-hoc operations. The performance improvements in SQL 7 and 2000 are not as drastic as in previous versions because changes in the way that other operations now execute provides them with some of the same benefits as stored procedures. A stored procedure can accept parameters, process operations against any number of databases, and return results to the calling process. Performance will be discussed in more detail in Chapter 12, "Monitoring SQL Server 2000."

The SQL Server 2000 implementation has many other capabilities that speed processing, secure data, reduce bandwidth usage, and enable advanced operations to be performed. Procedures that are repeatedly used will be held in memory in the SQL Server procedure cache for faster execution. A stored procedure, like other operations, can be encrypted to protect the details of the operation (the following section covers encryption). An application might need to send several operations across a network and respond conditionally to the results. This can be handled with a single call if the logic is contained in a single stored procedure. The use of local and global cursors can expose information to the application or other applications as needed, giving provisions for complex development processes with conversations between separate processes.

Temporary stored procedures used frequently in earlier versions are still supported by SQL Server, although improvements in other areas should eliminate or reduce the need for their use. The most significant improvement is the capability to compile and maintain most SQL operations in cache for prolonged periods.

Many system-stored procedures have already been created and are available upon installation of SQL Server. Extended stored procedures, which enable DLL files to be accessed from the operating system, are pre-established and present in the `Master` database.

The T-SQL `CREATE PROCEDURE` statement is used to create a stored procedure. This statement can be executed from the Query Analyzer or it is available through the Enterprise Manager by right-clicking on Stored Procedures under the database and choosing the New Stored Procedure option. The procedure is then saved within the current database as an object.

Encryption Can Secure Definitions

Data encryption is a mechanism that can be used to secure data, communications, procedures, and other sensitive information. When encryption techniques are applied, sensitive information is transformed into a non-readable form that must be decrypted to be viewed. Encryption slows performance, regardless of the method implemented, because extra processing cycles are required whenever encryption or decryption occurs. SQL Server can use data encryption at several levels:

- Login information
- Application role passwords
- Stored procedures
- Views
- User-defined functions
- Triggers
- Defaults
- Rules
- Data sent over the network

A variety of encryption procedures can be performed by a developer or administrator depending on what level of encryption is desired. SQL Server always encrypts login and role passwords within the system tables stored on the server. This automatic encryption of the login information stored on the server can be overridden using `sp_addlogin`, but this is not recommended. By default, however, application role passwords are not encrypted if they are provided across the network to invoke a role. The encryption of these passwords must be coded into the invoking application by utilizing the encryption capabilities of the `sp_setapprole` procedure as follows:

```
sp_setapprole 'SampleRole', (ENCRYPT N 'password'), 'odbc'
```

SQL Server can use SSL (Secure Sockets Layer) encryption across all network libraries, although multiprotocol encryption is still supported for backward compatibility reasons. A consideration in any SQL Server installation that uses multiple instances installed on the same server is that multiprotocol encryption is not supported by named instances of SQL Server.

Process definition encryption applied to stored procedures, defaults, rules, user-defined functions, triggers, and view definitions are all implemented in a similar fashion. The definition stored on the server is encrypted to prevent someone from viewing the details of the process. To encrypt these definitions, use the applicable `CREATE` statement, providing the `WITH ENCRYPTION` option as illustrated in the following `VIEW` definition:

```
CREATE VIEW SampleEncryptedView WITH ENCRYPTION AS
     SELECT FirstName, LastName, Wage FROM PayTable
```

Encryption can also serve the purpose of protecting the copyright that a developer might have over the processes created. In any case, before you encrypt a procedure, make sure you save a copy of the procedure to a file server or other backup mechanism, because future changes are difficult to implement if you do not have the original definition. To update any definition or remove encryption, simply supply the `CREATE` statement without the `WITH ENCRYPTION` option. This overwrites the encrypted process with a new version that is not encrypted.

Schema Binding

Schema binding involves attaching an underlying table definition to a view or user-defined function. Normally, if this process is not used, a function or view definition does not hold any data or other defining characteristics of a table. The definition is stored as a set of T-SQL statements and handled as a query or procedure. With binding, a view or function is connected to the underlying objects. Any attempt to change or remove the objects fails unless the binding has first been removed. Normally, you can create a view, but the underlying table might be changed so that the view no longer works. To prevent the underlying table from being changed, the view can be "schema-bound" to the table. Any table changes, which would break the view, are not allowed.

Indexed views require that a view be defined with the binding option and also that any user-defined functions referenced in the view must also be bound. In previous versions of SQL Server, it was not possible to define an index on a view. With the advent of binding, however, meaningful indexes can now be defined over a view that has been bound to the underlying objects. Other system options must be set to define an indexed view. These options are discussed later in the chapter in the "Indexed Views" section. More information on the use of all types of views can be found in Chapter 7, "Working With Views." The following example uses T-SQL of the creation of a schema-bound view:

```
CREATE VIEW SampleBoundView WITH SCHEMABINDING AS
          SELECT ProductID, Description, PurchPrice,
  PurchPrice * Markup AS SalesPrice
          FROM dbo.ProductTable
```

EXAM TIP

The Many Meanings of "Schema" The word *schema* has several different uses and definitions within SQL Server; the exam will leverage this and attempt to confuse the separate definitions. Make sure you are aware of how the term is used with relation to XML, Indexed Views, and maintaining metadata. For more information about these particulars, you can consult Chapter 5, "Advanced Data Retrieval and Modification," in the section on XML schema; Chapter 7, "Working With Views," in the section on indexed views; and Chapter 12, "Monitoring SQL Server 2000," in the section on metadata.

Recompilation of Procedures

Adding or altering indexes or changing a stored procedure causes SQL Server to automatically recompile the procedure. This optimization occurs the next time the stored procedure is run, but only after SQL Server is restarted. In instances where you want to force a recompilation, you can use the `sp_recompile` system-stored procedure. Alternatively, you can use the `WITH RECOMPILE` option when you create or execute a stored procedure. Stored procedures are dealt with in depth in Chapter 9, "Stored Procedures and User-Defined Functions."

Extended Stored Procedures

These procedures, like many of the system-stored procedures, are loaded automatically when you install SQL Server. Extended stored procedures access DLL files stored on the machine to enable the calling of the functions contained in the DLLs from within a SQL Server application. You might add to this set of procedures stored in the `Master` database using the `sp_addextendedproc` procedure as follows:

```
sp_addextendedproc 'MyFunction', 'MyFunctionSet.DLL'
```

Trigger Utilization

Triggers are like stored procedures in that they contain a set of T-SQL statements saved for future execution. The big difference is that unlike stored procedures, triggers are executed automatically based on data activity in a table. A trigger may fire based on an `UPDATE`, `INSERT`, or `DELETE` operation.

In SQL Server 2000, triggers can be fired `AFTER` an operation completes (SQL Server default) or `INSTEAD OF` the triggering operation. An `AFTER` trigger can be used to archive data when it is deleted, send a notification that the new data has been added or changed, or to initiate any other process that you might want to automate based on data activity. An `INSTEAD OF` trigger can be used to perform more advanced activities (such as advanced data checking), to enable updates in a view to occur across multiple tables, and to perform many other functions that might be necessary in place of a triggering activity.

Many `AFTER` triggers can be specified for each `INSERT`, `UPDATE`, or `DELETE` action. If multiple triggers exist, you can specify the first and last trigger to fire. The others are fired in no particular order, and you cannot control that order. An `AFTER` trigger can be defined only on a table. Only one `INSTEAD OF` trigger can be defined for each of the triggering actions; however, an `INSTEAD OF` trigger can be defined on a view as well as a table.

In previous releases, you could use triggers to help enforce referential integrity constraints. This was difficult and required that you eliminate other elements, such as Foreign Key constraints. In SQL Server 2000, it is far more efficient to use cascading actions, discussed earlier in this chapter, for the purpose of cascading changes and deletions.

To define a trigger, you can select the Manage Triggers option from the Table Design window. You can also go to the table to which you want to attach a trigger, and you can find the option as an extension of the pop-up menu off the All Tasks option.

You can use the T-SQL `CREATE TRIGGER` statement to create triggers for all applicable operations. You can access this command from the Enterprise Manager by using the Managing Triggers option. Managing Triggers in the Enterprise Manager (and in other objects as well) provides you with the shell of a `CREATE` statement even if you are changing an existing trigger. The Enterprise Manager enables you to change an existing trigger and the trigger will be first dropped prior to re-creation. The `ALTER TRIGGER` statement is used to change the definition of a trigger without dropping it first, and is used only through T-SQL. An example of the creation of a trigger using T-SQL is as follows:

```
CREATE TRIGGER UpdatedCustomer ON CustomerTable
FOR INSERT, UPDATE AS
            declare @phone nvarchar(20)
            declare @Contact nvarchar(100)
            select @phone = phoneno,
                   @contact = contactname from inserted
            RAISERROR(50100, 1, 1, @Contact, @Phone)
```

This procedure is one of my favorite implementations for use in customer applications. In the case of customer information, an automated alert that sends an email message to the salesperson could be defined around the error being raised. On an `INSERT`, a clerk or salesperson may make an initial client contact call based on an email that the alert may send. In the event of an `UPDATE`, the clerk could call the client to ensure the new information is accurate. The benefit is that the trigger automatically fires when new rows are added to the table or changes are made to the customer information.

Consider some other possible implementations of triggers. A government database is present, which can be replicated into a local copy of the database. Revenues are based on the capability to ferret out new clients ahead of the competition. An `INSERT` trigger can fire an email directly to the client with a promotional sales package attached. The end result is that action occurs as quickly as possible, which might provide an edge over the competition. For a more complete guide to the use of triggers and other facets of this technology, see Chapter 8, “Triggers.”

User-Defined Functions

In some applications, the functions available from the SQL Server installation do not suit all needs. It is for these instances that user-defined functions were intended. The functions can contain any combination of T-SQL statements. These functions act similarly to stored procedures with the exception that any errors occurring inside the function cause the entire function to fail.

SQL Server supports three varieties of user-defined functions:

- Scalar functions
- Inline table-valued functions
- Multi-statement table-valued functions

The functions defined can accept parameters if needed and return either a scalar value or a table. A function cannot change any information outside the scope of the function and therefore maintains no information when processing has been completed. Other activities that are not permitted include returning information to the user and sending email. The `CREATE FUNCTION` statement is used to define a user-defined function similar to the following:

```
CREATE FUNCTION MyFunction (@Num1 smallint, @Num2 smallint)
        RETURNS real AS
   BEGIN
   Declare @ReturnValue real
             If (@Num1 > @Num2)
              Set @ReturnValue = @Num1 * 2 + 30
             If (@Num1 = @Num2)
              Set @ReturnValue = @Num1 * 1.5 + 30
             If (@Num1 < @Num2)
              Set @ReturnValue = @Num1 * 1.25 + 30
             If (@Num1 < 0)
              Set @ReturnValue = @Num2 * 1.15 + 30
             Return(@ReturnValue)
            End
```

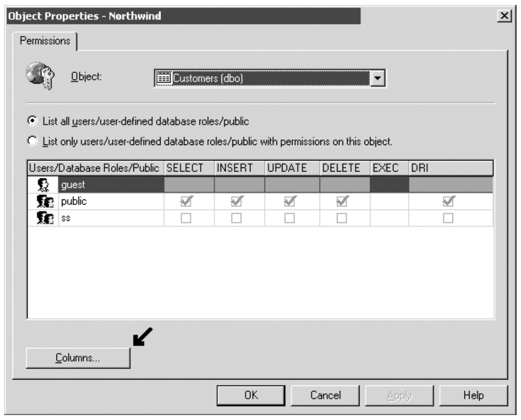
User-defined functions (UDFs) represent powerful functionality that has a wide variety of uses within the SQL Server environment. For more complete information on how to use UDFs see Chapter 9, “Stored Procedures and User-Defined Functions.”

Focusing Interaction with Views

A view is a `SELECT` statement that is saved and given a name. In most respects, a view acts as a table. A `VIEW` name can be used in `SELECT INSERT`, `UPDATE`, and `DELETE` statements as if it were a table. No data is stored within views (except indexed views).

Often you would like to design an application that gives the user a list of specific columns out of a table but does not grant the user access to all data. A view can be used to limit what the user sees and the actions the user can perform over a portion of the data in a table.

An alternative to creating a view would be to handle column-level permissions over a table, which can be a true nightmare to administer. A new interface feature in SQL 2000 does enable you to use the GUI to set column-level permissions. However, this feature should be used as little as possible—if ever. (See Figures 3.8, 3.9, and 3.10 for column permission availability.) In previous releases, you could set column permissions, but only through the use of T-SQL commands. These illustrations shows that the GUI representations for column permissions are significantly different from standard permissions and thus stand out if they are set.



Object Properties - Northwind

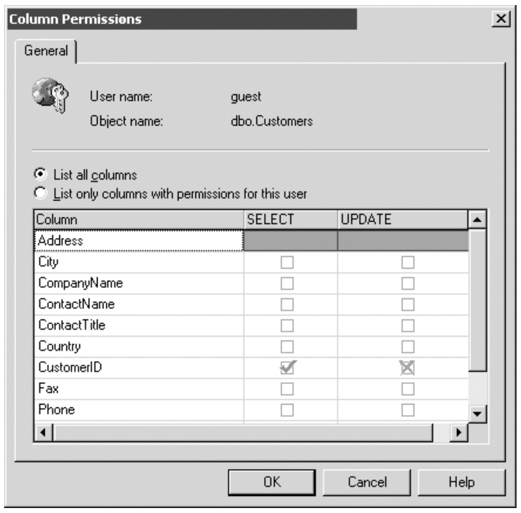
Permissions

Object: Customers (dbo)

(●) List all users/user-defined database roles/public
( ) List only users/user-defined database roles/public with permissions on this object.

Users/Database Roles/Public	SELECT	INSERT	UPDATE	DELETE	EXEC	DRI
guest						
public	✓	✓	✓	✓		✓
ss	☐	☐	☐	☐		☐

FIGURE 3.8
Command button now available on the interface to set column-level permissions.



Column Permissions

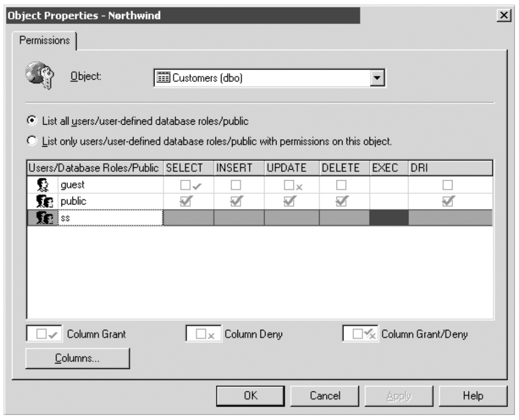
General

User name: guest
Object name: dbo.Customers

(●) List all columns
( ) List only columns with permissions for this user

Column	SELECT	UPDATE
Address		
City	☐	☐
CompanyName	☐	☐
ContactName	☐	☐
ContactTitle	☐	☐
Country	☐	☐
CustomerID	✓	☒
Fax	☐	☐
Phone	☐	☐

FIGURE 3.9
Column Permissions dialog box, available from the GUI.



Object Properties - Northwind

Permissions

Object: Customers (dbo)

(●) List all users/user-defined database roles/public
( ) List only users/user-defined database roles/public with permissions on this object.

Users/Database Roles/Public	SELECT	INSERT	UPDATE	DELETE	EXEC	DRI
guest	☐✓	☐	☐✗	☐		☐
public	☑	☑	☑	☑		☑
ss						

☐✓ Column Grant ☐✗ Column Deny ☐✓✗ Column Grant/Deny

FIGURE 3.10
Permissions dialog box, showing that column permissions have been set.

The problem with column-level permissions is the initial creation process of the permission is time-consuming, and the granularity of maintenance of the permissions requires extremely careful documentation. Imagine a table with 100 columns and 1000 or more users, groups, and roles. Trying to document and keep track of all the permissions set is an immense task that will overwhelm even the best administrator.

Use a view to simplify administration and provide a more meaningful perspective on the data for the user. The following example shows the creation of a view:

```
CREATE VIEW InStock AS
SELECT ProductID, Description, QTYOnHand FROM Products
WHERE QTYOnHand > 0
```

Indexed Views

If you want to use indexed views, a number of session-level options must be set On when you create the index. You need to set `NUMERIC_ROUNDABORT` Off. The options that need to be set On are as follows:

- `ANSI_NULLS`
- `ANSI_PADDING`
- `ANSI_WARNINGS`
- `ARITHABORT`
- `CONCAT_NULL_YIELDS_NULL`
- `QUOTED_IDENTIFIERS`

Other than setting the specific set of options, nothing more needs to be done for the optimizer to utilize an index with a query on a view. Essentially, the SQL SERVER Optimizer handles the view query in the same manner that it would a standard query against a table. The view cannot reference another view; only underlying tables are permitted and you must create the view with the `SCEMABINDING` option. Only the Enterprise and Developer editions support the creation of an indexed view.

EXAM TIP

Know Your Options A lot of specific options need to be in place to allow for Indexed Views. Make sure you are confident with the set of configuration features that are needed. Make sure that you read up on this topic as presented in Chapter 7, "Working with Views."

There are limitations to the content of the `SELECT` statement for the view definition. They are as follows:

- No use of `*`.
- A column name used as a simple expression cannot be specified in more than one view column.
- No derived tables.
- Rowset functions are not permitted.
- `UNION`, Outer Joins, Subqueries, or Self-joins cannot be used only simple Joins.
- No `TOP`, `ORDER BY`, `COMPUTE`, or `COMPUTE BY` clause.
- `DISTINCT` is not permitted.
- `COUNT(*)` cannot be used, but `COUNT_BIG(*)` is allowed.
- Aggregate functions: `AVG`, `MAX`, `MIN`, `STDEV`, `STDEVP`, `VAR`, or `VARP` are not permitted.
- A `SUM` function cannot reference a nullable expression.
- No use of full-text predicates `CONTAINS` or `FREETEXT`.

Partitioned Views

A partitioned view enables the creation of a view that spans a number of physical machines. These views can fall into one of two categories: local and distributed. A distinction is also made between views that are updateable and those that are read-only. The use of partitioned views can aid in the implementation of *federated database servers*, which are multiple machines set up to share the processing load. For more information on federated server implementations, see SQL Server Books Online, "Designing Federated Database Servers."

To use partitioned views, you horizontally split a single table into several smaller tables, each having the same column definitions. Set up the smaller tables to accept data in ranges and enforce the ranges using `CHECK` constraints. Then you can define the distributed view on each of the participating servers. Add linked server definitions on each of the member servers. An example of a distributed view definition is as follows:

```
CREATE VIEW AllProducts AS
           Select * FROM Server1.dbo.Products9999
UNION ALL
           Select * FROM Server2.dbo.Products19999
UNION ALL
           Select * FROM Server3.dbo.Products29999
```

Easing Data Entry with Defaults

A default is used to provide a value for a column so as to minimize data entry efforts or to provide an entry when the data is not known. A default provides a value for the column as a basis for initial input. Any data that is entered for the column overrides the default entry. You can apply a default definition to a column directly using the `CREATE` or `ALTER TABLE` statement or through the Design Table option from within the Enterprise Manager. You can also create a default as its own object and then bind it to one or more columns.

A default definition provided as part of a table definition is a standard and preferred method of implementing default entries. The advantages of this technique are that the default is dropped when the table is dropped and that the definition is stored within the table itself. A default object must be created and bound to the column in a two–step operation.

To create and bind a default object, use the following code:

```
CREATE DEFAULT StateDefault AS 'IN'
     sp_bindefault StateDefault, 'customers.state'
```

To create a default within a table definition, use the following:

```
CREATE TABLE SampleDefault
       ( SampleID        smallint NOT NULL
           CONSTRAINT UPKCL_SampleID PRIMARY KEY CLUSTERED,
         City            varchar(50)
           DEFAULT       ('Woodstock'),
         State           char(2)
           DEFAULT       ('NY')
        )
```

When an `INSERT` operation is performed on a table, you must supply values for all columns that do not have a default entry defined or that allow `NULL` content.

Application of Integrity Options

As discussed throughout this chapter, a number of techniques are available to maintain the integrity of the database. Each of these techniques will in part provide a usable, responsive system that prevents inappropriate data from getting into the system. Table 3.1 summarizes these techniques and provides some further detail as to what can be expected in their use.

TABLE 3.1

INTEGRITY MAINTENANCE OBJECTS

Technique	*Integrity Achieved*	*Usage*	*Timing (Log)*
Primary Key	Entity	Identify each row	Before
Foreign Key	Referential/Domain	Ensure no orphan child elements	Before
Unique Index	Entity	Ensure entries are exclusive	Before
Unique Constraint	Entity	No duplicate column values	Before
Identity	Entity	Auto-incremented values	Before
Check Constraint	Domain	Ensure correct column entry	Before
Not NULL	Domain	A value must be present	Before
Default	Domain	Provides initial value	Before
Rule	Domain	Ensure correct column entry	Before
Trigger	Referential/Domain	Respond to add, change, delete	After
Stored Procedures	Referential/Domain/Entity	Process-controlled operations	Before

Note from Table 3.1 that most integrity techniques are applied before the data is actually allowed into the system, and therefore operate faster with a much lower overhead. Triggers offer the most functionality, but at a cost: the data is allowed into the system and then reacted upon. If a trigger determines that data should not be permitted, it must be rolled back. Use triggers sparingly and only when the application requires the additional functionality.

When you define the table structure from the GUI, fill in the upper portion of the Design Table dialog box with column information by supplying the column name, selecting the appropriate data type from the drop-down list box, completing the Length field if the data type allows this to be adjusted, and setting whether the column is to allow `NULL` content. In the lower portion of the dialog box, you can optionally complete a description for the column, provide a default value, and, for numeric data types that allow for scale and precision, supply appropriate entries. If the column is to be an Identity column, select one of the integer data types, select Identity, and provide a starting value in the seed identification and an increment for the Identity. If the column is to be a row guid, select this option. If the column value is derived from a formula, specify the formula.

WARNING

Collation for an Individual Column A collation can be selected for an individual column but is not recommended because it causes great difficulty in the development of front-end applications.

STEP BY STEP

3.7 Creating Tables, Indexes, and Constraints

1. Expand a server group, then the server, and then the database where the table is to be located.
2. Right-click Tables, and then select New Table.
3. Provide column definitions until all columns are defined.
4. Right-click on the column to be used as the Primary Key and select Set Primary Key.
5. In the top-left corner of the dialog box is a Table and Index Properties button that provides access to an area where you can provide further information to define the table.
6. Select the Relationships tab to define a Foreign Key.
7. Select the Indexes/Keys tab to define more indexing options.

8. Select the Check Constraints tab to define any constraint options for the table.
9. Close the Properties dialog box, and then save and exit to finish defining the table.

Using T-SQL code, you create the tables as follows:

```
CREATE TABLE projects
 ( project_id  smallint not Null
IDENTITY(1,1)
      PRIMARY KEY CLUSTERED,
   project_description       varchar(25)     NOT NULL
      DEFAULT '*Unknown*',
   start_date                datetime        NOT NULL
      CHECK (start_date >= getdate()),
   completion_date           datetime        NOT NULL
      CHECK (completion_date >= getdate())
  )
```

REVIEW BREAK

Database Design, an Involved Process

Consider the functionality that is needed for any procedure that maintains the integrity of the data. In many cases, there are a number of choices that can be made between different techniques. The highest degree of functionality is achieved from stored procedures, followed closely by triggers. Also consider the basics of database design, regardless of application. Primary Keys, for example, serve a number of purposes and it is a good practice to implement them in almost any structure.

To achieve availability and performance, redundancy is the starting point. Data redundancy can be obtained at the table and database level through denormalization, and at the server level through the use of log shipping, replication, and/or partitioning. Each technique has a variety of requirements that must be closely adhered to.

There is no single, basic solution for any plan. Knowledge of the environment and how the data is going to be used will answer a number of the initial questions. There is no replacement for proper system development; you can't throw a technology at a problem and just hope it will go away. Research and plan the implementation and document each step of the way.

MULTIPLE SERVER IMPLEMENTATIONS

- ▶ **Alter database objects to support replication and partitioned views.**

Many high-demand implementations require the use of multiple servers to handle the workload. A variety of options exist for using SQL Server on multiple servers. Partitioning, log shipping, replication, federated servers, and clustering are all potential implementations for using multiple servers and obtaining load balancing.

Using log shipping is a way to have two separate servers that contain the same database split the query load away from the updating of data. The log information is periodically copied to a read-only, standby server that can then be used for query purposes, thereby offloading some of the work from the main production machine. For further information on log shipping implementations, see SQL Server Books Online, "Maintenance Planning Wizard." Microsoft failover clustering provides for availability by enabling a secondary server to take over activities if a primary server fails. For further information on failover clustering, see SQL Server Books Online, "Creating a Failover Cluster."

Partitioning and federated servers provide load balancing for a single application and work together to provide users with better response. This type of implementation is in place of a multiple server cluster where all machines load balance to share the workload. In a federation, each server is completely independent and it is the application and database design that implement the load balancing.

Replication places potentially updateable copies of the same data on multiple servers so that applications that allow for site independence can, at the same time, keep all copies of the data synchronized. Most models have a degree of latency or delay between the initial updates and the moment when all data is in agreement.

Use of Replication

Placing the same data on multiple servers, with each server closer to the user's location, can reduce the use of bandwidth and provide the user with faster update operations and retrieval of data. Replication is discussed in full in Chapter 11, “Implementing and Understanding Replication Methodologies,” so this section focuses more or the specifics of designing a system to support the replication models.

These replication techniques can be applied to three replication models, as well as several different physical models. The physical aspects and models have no direct correlation. The replication model supplies the functionality, whereas the physical aspects lay out the placement and roles of individual servers.

Merge, snapshot, and transactional replication all involve essentially the same basic elements to begin with. However, each model has idiosyncrasies of its own that require some thought during the design of the implementation. For further information on replication, consult Chapter 11.

Partitioning to Achieve a Balance

Data partitioning as defined previously involves the horizontal division of a singular table into a number of smaller tables, each dealing with a range of data from the original and split off onto separate servers. Some configuration options can help gain performance when operating against partitions. Setting the Lazy Schema Validation option using `sp_serveroption` can optimize performance. Attempting to ensure that the correct query goes to the appropriate server also helps to improve performance while minimizing bandwidth use.

A partitioned view is considered to be updateable if a set of `SELECT` statements is combined into one resultset using `UNION ALL` operations, as was shown in the section “Partitioned Views.” Indexes based on calculated columns are not permitted within any table definitions and all tables must have a Primary key and `ANSI_PADDING` set.

When you use a partitioned view to insert data, all columns must be included in the `INSERT` statement, even if the table definitions provide `DEFAULTS` or allow for `NULL` content. Also, `IDENTITY` columns cannot be referenced; therefore, no underlying tables can have `IDENTITY` columns nor are they permitted to have timestamp columns.

Remote partitioned views require that you keep a few additional considerations in mind. A distributed transaction is automatically initiated to ensure that integrity is maintained throughout all operations, and the `XACT_ABORT` option must be set to `ON` `Smallmoney` and `smalldatetime` data types in the remote tables are mapped to money and `datetime` types locally.

Partition Creation Strategy

Partitions can be designed in a symmetric or asymmetric fashion, and although it is most useful to design symmetrically, the access requirements of a lot of systems necessitate an asymmetric design.

A symmetrical design is one in which all related data is placed on the same server so that most queries do not have to cross network boundaries to access the data. It is also easier to manage the data if the division of tables can be performed in such a manner that each server has the same amount of data.

In most real-world applications, data is accessed in a random fashion that can make the designer lean toward an asymmetric implementation. The design can be configured so that one server has a larger role and/or contains more data than the others. Performance can be improved if you weigh each server's use and make one server work harder on the partitioned applications because the other servers perform larger tasks that deal with other unrelated processes.

EXAM TIP

Be Comfortable with Configuration You are likely to find exam questions that will ask you to balance the work load and properly configure partitioned views. Ensure you are comfortable with the configuration required. To find out more information on these and other styles of views you can expect to see, refer to Chapter 7.

Designing for distributed partitioned views requires appropriate planning of front-end applications to ensure that, whenever possible, data queries are sent to the appropriate server. Middleware, such as Microsoft Transaction Server or Application Server or other third-party equivalents, should attempt to match queries against data storage.

Constraint Considerations

Constraints need to be defined on each participating server so that only the data pertaining to the table(s) stored on that server is handled. Although `CHECK` constraints are not needed to return the correct results, they enable the query optimizer to more appropriately select the correct server to find the requested data.

REVIEW BREAK

Spreading It Out

Multiple server operations balance the load so that updates are potentially separated from queries and query load can be spread across multiple machines.

Partitioned views drastically restrict the underlying table designs and require a number of options to be set when using indexes.

The application of the initial snapshot that begins the entire replication process can be compressed and/or saved to a CD so that some of the necessary communications can be offloaded. Doing so makes more efficient use of network bandwidth in slow-link or dial-up environments in particular.

TROUBLESHOOTING SQL SERVER OBJECTS

▶ **Troubleshoot failed object creation.**

Most problems associated with creating and/or accessing objects can be resolved through setting appropriate object access permissions. However, other elements that can hamper the creation or use of objects include (but are not limited to) the following:

- Backup and restore operations
- Other users' operations locking parts of the system
- Metadata corruption
- Hardware or resource problems
- Network connectivity
- Configuration settings
- Operating system
- Metadata corruption

A good starting point from which to resolve most problems is the wealth of feedback SQL Server gives in the form of OS Application

Event Log, SQL Server Logs, and the Current Activity Window, as well as the permission properties of the users, roles, and objects.

To create a database, you need to be a member of System Administrators or Database Creators server roles or have the Create Database permission. To create objects within a database, you must be a member of `db_owner` or `db_ddladmin` database roles or have the specific permission to create the object as given by statement-level permissions. Statement-level permissions can be found on the Permissions tab of the database Properties dialog box.

As databases and their objects are created, the system uses the default filegroup for the physical storage of the element. It is a good practice to create a storage group for user objects and make that the default filegroup. This way, as the user creates objects, those objects don't compete for storage with other data.

If a user lacks the permission to create or alter an object, an alternative is available that grants the user creation permission without giving the user too much control over the environment. An Application role that has permission to work with objects in this manner can be assigned to a stored procedure that creates the objects for the user. When the user executes the procedure, objects can be created or altered in a controlled manner.

Setting Server Configuration Options

Standard configuration settings are available through the Server Properties dialog box in the Enterprise Manager or can be accessed using the `sp_configure` stored procedure. Some of the more advanced options require that you enable Show Advanced Options.

```
sp_configure 'show advanced options', 1
reconfigure
```

Affinity Mask (Advanced)

Use this option in systems that have four or more processors. It increases performance when the system is under a heavy workload. You can specify which processors Microsoft SQL Server is to use. You can exclude SQL Server activity from processors that have been given specific workload assignments by the Windows NT 4.0 or Windows 2000 operating system.

Allow Updates

This option is used to allow the direct alteration of system tables. When Allow Updates is set to `1`, any user with appropriate permissions can either update system tables directly with ad-hoc updates or create stored procedures that update system tables.

AWE Enabled (Advanced)

Address Windowing Extension (AWE) is an advanced option used to support up to a maximum of 64 gigabytes (GB) of physical memory.

C2 Audit Mode

Use C2 audit mode to review both successful and unsuccessful attempts to access statements and objects. Allowing for the documentation of system activity and observance of security policy violations, C2 auditing tracks C2 audit events and records them to a file in the \mssql\data directory or the \mssql$instancename\data directory for named instances of SQL Server. If the file reaches a size limit of 200 megabytes (MB), C2 auditing starts a new file.

Cost Threshold for Parallelism (Advanced)

Use this option to specify the threshold where SQL Server creates and executes parallel query plans. Parallel query plans are executed only when the estimated cost to execute a serial plan for the same query is higher than the value set. The cost refers to an estimated elapsed time in seconds that is required to execute a standard plan. Only set cost threshold for parallelism on symmetric multiprocessors.

Cursor Threshold (Advanced)

Use this option to indicate the number of rows in the cursor set at which cursor keysets are generated asynchronously. If you set Cursor Threshold to `-1`, all keysets are generated synchronously, which benefits small cursor sets. If you set Cursor Threshold to `0`, all cursor keysets are generated asynchronously. With other values, the query optimizer compares the number of expected rows in the cursor set and builds the keyset asynchronously if it exceeds the number set in Cursor Threshold. Do not set Cursor Threshold too low because small result sets are better built synchronously.

Default Language

Use this option to specify the default language for all newly created logins.

Fill Factor (Advanced)

Use this option to specify how full the server should make each page when it creates a new index using existing data. The Fill Factor percentage affects performance because SQL Server must take time to split pages when they fill up. The default for Fill Factor of 0 (zero) does not mean that pages are 0% full. It is treated similarly to a fill factor value of 100 in that indexes are created with full data pages and nonclustered indexes with full leaf pages. The default setting is different from 100 in that SQL Server leaves some space within the upper level of the index tree.

Index Create Memory (Advanced)

Use this option to control the amount of memory used by index creation sorts. The Index Create Memory option is self-configuring and should operate without requiring adjustment. If difficulties are experienced creating indexes, consider increasing the value. Query sorts are controlled through the Min Memory Per Query option. The default value for this option is `0` (self-configuring).

Default Full-Text Language (Advanced)

Use the default full-text language option to specify a default language value for full-text indexed columns. The default value of this option is the language of the server.

Lightweight Pooling (Advanced)

This option provides a means of reducing the overhead associated with the excessive context switching sometimes seen in multiprocessor environments. When excessive context switching is present, lightweight pooling might provide better throughput.

Locks (Advanced)

The Locks option sets the maximum number of available locks, limiting the amount of memory the server uses. The default setting is `0`, which enables SQL Server to allocate and deallocate locks dynamically based on changing system requirements.

Max Degree of Parallelism (Advanced)

This option limits the number of processors to use in parallel plan execution. The default value is `0` (actual number of CPUs) and the maximum is `32`.

Max Server Memory/Min Server Memory

These two settings establish upper and lower limits to the amount of memory the database engine uses. The database engine starts with only the memory required to initialize. As the workload increases, it acquires additional memory. The database engine frees any of the acquired memory until it reaches the amount specified in Min Server Memory.

Max Text Repl Size

Specifies the maximum size (in bytes) of text and image data that can be added to a replicated column in a single `INSERT`, `UPDATE`, `WRITETEXT`, or `UPDATETEXT` statement.

Max Worker Threads (Advanced)

Configures the number of worker threads available to the server and its processes. SQL Server uses the threads so that one or more threads simultaneously support each network that SQL Server supports; another thread handles database checkpoints; and a pool of threads handles user connections.

Media Retention (Advanced)

Use the Media Retention option to provide a default for the length of time each backup should be retained. Overridden by the `RETAINDAYS` clause of the `BACKUP` statement, Media Retention helps protect backups from being overwritten until the specified number of days has elapsed.

Min Memory Per Query (Advanced)

Use this option to specify the minimum amount of memory that will be allocated for the execution of a query.

Nested Triggers

The Nested Triggers option enables actions that initiate another trigger to be performed. When the Nested Triggers option is set to `0`, triggers cannot cascade. When the Nested Triggers option is set to the default setting of `1`, triggers can cascade to as many as 32 levels.

Network Packet Size (Advanced)

Use this option to set the packet size used across the entire network. The default packet size is 4096 bytes. If an application does bulk copy operations, or sends or receives large amounts of text or image data, a packet size larger than the default can improve efficiency because it results in fewer network reads and writes. If an application sends and receives small amounts of information, you can set the packet size to 512 bytes, which is sufficient for most data transfers.

Open Objects (Advanced)

Use this option to set the maximum number of database objects that can be open at one time. Database objects are those objects defined in the sysobjects table: tables, views, rules, stored procedures, defaults, and triggers.

Priority Boost (Advanced)

This option specifies the processor scheduling priority. If you set this option to `1`, SQL Server runs at a priority base of 13 in the Windows NT 4.0 or Windows 2000 Scheduler. The default is `0`, which is a priority base of 7.

Query Governor Cost Limit (Advanced)

Specifies an upper limit for the time in which a query can run. Query cost refers to the estimated elapsed time, in seconds, required to execute a query.

Query Wait (Advanced)

Memory-intensive queries, such as those involving sorting and hashing, are queued when not enough memory is available to run the query. The query times out after a set amount of time that SQL Server calculates (25 times the estimated cost of the query) or the time amount specified by the non-negative value of the query wait.

Recovery Interval (Advanced)

Use this option to set the maximum number of minutes per database that the server needs to recover the database activity. The recovery process is initiated each time SQL Server starts or as the basis for completing a restore operation. The recovery process rolls back transactions that did not commit and rolls forward transactions that did commit. This configuration option sets an upper limit on the time it should take to recover each database. The default is `0`, indicating automatic configuration by SQL Server. In practice, this means a recovery time of less than one minute and a checkpoint marker is placed into the transaction log approximately every one minute for active databases.

Remote Access

Use the Remote Access option to control logins from remote servers running SQL Server. Set Remote Access to `1` (default) to enable logins from remote servers. Set the option to `0` to secure a local server and prevent access from a remote server.

Remote Login Timeout

Use this option to specify the number of seconds to wait before returning from a failed remote login attempt.

Remote Proc Trans

This option protects the activities of a server-to-server process through the use of the Distributed Transaction Coordinator. Set Remote Proc Trans to `1` to provide an MS DTC-coordinated distributed transaction that protects the ACID properties of transactions. Sessions begun after setting this option to `1`, inherit the configuration setting as their default.

Remote Query Timeout

This option is used to indicate the number of seconds that must elapse when processing a remote operation before the operation times out. The default of `600` sets a ten-minute wait.

Scan for Startup Process (Advanced)

Use this option to scan for automatic execution of stored procedures at startup time. If it is set to `1`, SQL Server scans for and executes all automatically executed stored procedures defined on the server. The default value is `0` (do not scan).

Set Working Set Size (Advanced)

Reserves physical memory space for SQL Server that is equal to the server memory setting. SQL Server, based on workload and available resources, configures the server memory setting automatically. It varies dynamically between the Min Server Memory and Max Server Memory settings.

Two Digit Year Cutoff

Use the Two Digit Year Cutoff option to specify an integer from 1753 to 9999 that represents the last year for interpreting two-digit years as four-digit years.

User Connections (Advanced)

Use this option to specify the maximum number of simultaneous user connections. The actual number of user connections allowed also depends on the version of SQL Server you are using and the limits of your application(s) and hardware. SQL Server enables a maximum of 32,767 user connections.

User Options

The User Options option is used to specify global defaults for all users. A list of default query processing options is established for the duration of a user's work session. A user can override these defaults by using the `SET` statement. You can configure user options dynamically for new logins. After you change the setting of user options, new logins use the new setting; current logins are not affected.

Reconfigure

This option updates the server configuration. It is used after the application of `sp_configure` to change server settings and make the new settings take effect. Because some configuration options require that a server stop and restart before the currently running value can be updated, `Reconfigure` does not always update the currently running value. Use the `With Override` option of this command to force a value that might or might not meet ranges of allowed values or recommended settings.

Changing configuration options can easily be performed using T-SQL operations:

```
TSQL

USE master

EXEC sp_configure 'show advanced option', '1'

RECONFIGURE

GO

EXEC sp_configure 'recovery interval', '3'

RECONFIGURE WITH OVERRIDE
```

Step by Step 3.8 takes you through the setting of some common server options.

STEP BY STEP

3.8 Setting Server Configuration Options

1. Expand a server group, then right-click the server, and then select Properties.
2. Select the Database Settings tab.
3. Set the Recovery Interval to 3.
4. Select OK to save the setting.

Configuration Exam Considerations

The SQL Server configuration options are used to fine-tune the database environment. Many options provide a mechanism for an administrator or developer to obtain optimum performance and achieve a more secure and stable server. A total approach to an optimum environment also involves the proper use of database configuration options. Server property adjustments affect all the databases stored on the server where database configuration options are used to control a database and not affect other databases.

Setting Database Configuration Options

Standard database configuration settings are available through the Database Properties in the Enterprise Manager or can be accessed using the `sp_dboption` stored procedure. Some of the more advanced settings cannot be set singly; they must be set in combination with other settings.

There are five categories of database options:

- **Auto options**. AUTO_CLOSE, AUTO_CREATE_STATISTICS, AUTO_UPDATE_STATISTICS, and AUTO_SHRINK
- **Cursor options**. CURSOR_CLOSE_ON_COMMIT and CURSOR_DEFAULT LOCAL or GLOBAL
- **Recovery options**. RECOVERY FULL or BULK_LOGGED or SIMPLE and TORN_PAGE_DETECTION
- **SQL options**. ANSI_NULL_DEFAULT, ANSI_NULLS, ANSI_PADDING, ANSI_WARNINGS, ARITHABORT, NUMERIC_ROUNDABORT, CONCAT_NULL_YIELDS_NULL, QUOTED_IDENTIFIER, and RECURSIVE_TRIGGERS
- **State options**. OFFLINE or ONLINE, READ_ONLY or READ_WRITE, SINGLE_USER or RESTRICTED_USER or MULTI_USER and WITH ROLLBACK AFTER or WITH ROLLBACK IMMEDIATE or NO_WAIT.

These options are described in the following sections, and are covered in alphabetical order.

ANSI_NULL_DEFAULT

This option enables the user to control the default nullability. When `NULL` or `NOT NULL` is not specified, a user-defined data type or a column definition uses the default setting for nullability. When this option is set to `ON`, all user-defined data types or columns that are not explicitly defined as `NOT NULL` during a `CREATE TABLE` or `ALTER TABLE` statement default to allowing null values. Columns that are defined with constraints follow constraint rules regardless of this setting.

ANSI_NULLS

When set to `ON`, all comparisons to a null value evaluate to `NULL` (unknown). When set to `OFF`, comparisons of non-Unicode values to a null value evaluate to `TRUE` if both values are `NULL`. By default, the ANSI_NULLS database option is `OFF`.

ANSI_PADDING

When set to `ON`, trailing blanks in character values inserted into varchar columns and trailing zeros in binary values inserted into varbinary columns are not trimmed. Values are not padded to the length of the column. When set to `OFF`, the trailing blanks and zeros are trimmed. This setting affects only the definition of new columns. It is recommended that ANSI_PADDING always be set to `ON`. SET ANSI_PADDING must be `ON` when creating or manipulating indexes on computed columns or indexed views.

ANSI_WARNINGS

When set to `ON`, errors or warnings are issued when conditions such as “divide by zero” occur or null values appear in aggregate functions. When set to `OFF`, no warnings are raised when null values appear in aggregate functions, and null values are returned when conditions such as “divide by zero” occur. By default, ANSI_WARNINGS is `OFF`.

ARITHABORT

When set to `ON`, an overflow or divide-by-zero error causes the query or batch to terminate. If the error occurs in a transaction, the transaction is rolled back. When set to `OFF`, a warning message is displayed if one of these errors occurs, but the query, batch, or transaction continues to process as if no error occurred.

AUTO_CLOSE

When set to `ON`, server resources are freed up as soon as the database is closed and shut down cleanly when the last user of the database exits. By default, this option is set to `ON` for all databases in the Desktop Engine, and `OFF` for all other editions. The database reopens automatically when a user tries to use the database again. When set to `OFF`, the database remains open even if no users are currently using it.

AUTO_CREATE_STATISTICS

When set to `ON`, statistics are automatically created on columns used in a predicate. Adding statistics improves query performance because the optimizer can better determine how to evaluate queries. If the statistics are not used, SQL Server automatically deletes them. When set to `OFF`, SQL Server does not automatically create statistics; instead, statistics can be manually created.

AUTO_SHRINK

When set to `ON`, the database files are set up for periodic shrinking. Any database-associated file, data, or log can be shrunk automatically. When set to `OFF`, the database files are not automatically shrunk during periodic checks for unused space. By default, this option is set to `ON` for all databases in SQL Server Desktop Edition, and `OFF` for all other editions, regardless of operating system.

AUTO_UPDATE_STATISTICS

When set to `ON`, existing statistics are automatically updated when the statistics become out-of-date because the data in the tables has changed. When set to `OFF`, existing statistics are not automatically updated; instead, statistics can be manually updated.

CONCAT_NULL_YIELDS_NULL

When set to `ON`, if one of the operands in a concatenation operation is `NULL`, the result of the operation is `NULL`. When set to `OFF`, concatenating a null value with a character string yields the character string as the result.

CURSOR_CLOSE_ON_COMMIT

When set to `ON`, any open cursors are closed automatically when a transaction using the cursor is committed. By default, this setting is `OFF` and cursors remain open across transaction boundaries, closing only when the connection is closed or when they are explicitly closed, which is usually when a procedure finishes.

CURSOR_DEFAULT LOCAL or GLOBAL

When CURSOR_DEFAULT LOCAL is set, and a cursor is not defined as GLOBAL when it is created, the scope of the cursor is local to the batch, stored procedure, or trigger. The cursor name is valid only within this scope. When CURSOR_DEFAULT GLOBAL is set, and a cursor is not defined as LOCAL when created, the scope of the cursor is global to the connection. The cursor name can be referenced in any stored procedure or batch the connection executes.

NUMERIC_ROUNDABORT

If set to `ON`, an error is generated when the loss of precision occurs in an expression. When set to `OFF`, losses of precision do not generate error messages and the result is rounded to the precision of the column or variable storing the result.

OFFLINE or ONLINE

When OFFLINE is specified, the database is closed and shut down cleanly and marked offline. The database cannot be modified while it is offline. When ONLINE is specified, the database is open and available for use.

QUOTED_IDENTIFIER

When set to `ON`, identifiers can be delimited by double quotation marks and literals must be delimited by single quotation marks. All strings delimited by double quotation marks are interpreted as object identifiers. Quoted identifiers do not have to follow the T-SQL rules for identifiers. They can be keywords and can include characters not generally allowed in T-SQL identifiers. When set to `OFF` (default), identifiers cannot be in quotation marks and must follow all T-SQL rules for identifiers. Literals can be delimited by either single or double quotation marks. Identifiers must be enclosed in square brackets ([ ]) if they contain spaces or other characters or key words.

READ_ONLY or READ_WRITE

When READ_ONLY is specified, users can retrieve data from the database but cannot modify the data. Automatic recovery is skipped at system startup and shrinking the database is not possible. No locking takes place in read-only databases, which can result in faster query performance. When READ_WRITE is specified, users can retrieve and modify data.

RECOVERY FULL or BULK_LOGGED or SIMPLE

When FULL is specified, database backups and transaction log backups are used to provide full recoverability from media failure. All operations, including bulk operations, such as SELECT INTO, CREATE INDEX, and bulk loading data, are fully logged. When BULK_LOGGED is specified, logging for all SELECT INTO, CREATE INDEX, and bulk loading data operations is minimal and therefore requires less log space. In exchange for better performance and less log space usage, the risk of exposure to loss is greater than with full recovery. When SIMPLE is specified, the database can be recovered only to the last full database backup or last differential backup.

RECURSIVE_TRIGGERS

When set to `ON`, triggers are enabled to fire recursively. When set to `OFF` (default), triggers cannot be fired recursively.

SINGLE_USER or RESTRICTED_USER or MULTI_USER

SINGLE_USER enables only one user at a time to connect to the database. All other user connections are broken. The timeframe for breaking the connection is controlled by the termination clause of the `ALTER DATABASE` statement. New connection attempts are refused. RESTRICTED_USER enables only members of the `db_owner` fixed database role and `dbcreator` and `sysadmin` fixed server roles to connect to the database, but it does not limit their number. MULTI_USER enables all users with the appropriate permissions to connect to the database.

TORN_PAGE_DETECTION

This recovery option enables SQL Server to detect incomplete I/O operations caused by power failures or other system outages. When set to `ON`, this option causes a bit to be reversed for each 512-byte sector in an 8-kilobyte (KB) database page as the page is written to disk. If a bit is in the wrong state when the page is later read by SQL Server, the page was written incorrectly; a torn page is therefore detected.

WITH <termination>

The termination clause of the `ALTER DATABASE` statement specifies how to terminate incomplete transactions. Breaking their connections to the database terminates transactions. If the termination clause is omitted, the `ALTER DATABASE` statement waits indefinitely, until the transactions commit or roll back on their own. ROLLBACK AFTER *'integer'* SECONDS waits for the specified number of seconds. ROLLBACK IMMEDIATE breaks unqualified connections immediately. NO_WAIT checks for connections before attempting to change the database state and causes the `ALTER DATABASE` statement to fail if certain connections exist. When the transition is to SINGLE_USER mode, the `ALTER DATABASE` statement fails if any other connections exist. When the transition is to RESTRICTED_USER mode, the `ALTER DATABASE` statement fails if any unqualified connections exist.

Similar to setting server configuration options, the process of setting database options is described in Step by Step 3.9.

STEP BY STEP

3.9 Setting Database Options

1. Expand a server group, and then expand the server where the database is to be placed.
2. Right-click Databases, and then click Properties.
3. Select the Options tab.
4. Change the appropriate desired settings and select OK to save.

T-SQL database options can be set programmatically, although this is not normally recommended. The following procedure illustrates how this can be done in those rare instances where it is desired:

TSQL

View settable database options:

```
Sp_dboption
```

View which options have been set on `Northwind` database:

```
Sp_dboption Northwind
```

Turn off an option:

```
Sp_dboption Northwind, 'autoclose', False
```

Turn on an option:

```
Sp_dboption Northwind, 'autoclose', True
```

Setting the Database Options

Database options are set to allow for application or procedural requirements and to provide for administrative configuration. You will interact with these settings to set up backups, allow for specific procedures, and provide appropriate access levels, depending on what is needed for a given process. Learn the settings that are required for each process and know the resulting effect on the system under different operating configurations.

CASE STUDY: ACME DISTRIBUTORS

ESSENCE OF THE CASE

Here are the essential points being addressed in this case:

- Budget is limited.
- High volume of updates.
- Downtime must be minimized.
- Data loss must be minimized if a failure occurs.
- Referential integrity must be configured for replication.

SCENARIO

ACME Distributors is a mail-order company with warehouse operations located in strategic locations throughout the United States. You are in charge of planning the physical server environment to be used in the implementation of a multi-server warehousing system. Each of the warehouses needs information on what the other warehouses have in stock and transactional replication has been chosen as the technique to deliver information among the warehouses.

Each warehouse is expected to handle about 1000 transactions of its own each day, as well as receive 9000 replicated transactions from other warehouses. You need to supply an environment that will function 24 hours a day, 7 days a week. You must pay particular attention to achieving an environment that has a minimum amount of downtime and data loss in the event of a failure.

With a limited budget, you have been asked to establish priorities within the system. A list should be prepared providing a range of choices from the minimum requirements to fulfill the needs of the system, and on the upper range, some alternatives that would still fall into a low-budget scenario.

ANALYSIS

This seems to be a far-too-typical situation, in which a company desires to have the optimum environment but is hesitant to invest the necessary funds to achieve the desired results.

To eliminate downtime, there are two separate possibilities. First, you can provide for failover clustering, which can incur a significant cost.

continues

Case Study: ACME Distributors

continued

Second, you can include in the solution a mechanism where a middle-tier procedure can select the local server by default, but if the server is unavailable, the procedure can obtain the necessary information from one of the other warehouses. Although the later solution can appear less expensive at first, because hardware costs are lower, other factors, such as development, update ability, bandwidth, and licensing costs have to be considered.

Also worth considering is the use of a standby server or replication as a means of minimizing—though not eliminating—downtime. This would be a less expensive solution than failover clustering, but in the event of failure, you will have a small amount of downtime while adjusting the actual machine being used for production purposes. An additional advantage of this solution is the near elimination of data loss if a failure were to occur.

To minimize data loss in the event of failure, a number of options are available. On the less-expensive side, setting the database recovery mode to Full Recovery and adding a second disk volume can provide for the storage of the log files and allow for almost full data recovery. Other possibilities from less to more expensive would include RAID parity storage, RAID mirror storage, log shipping, replication, and failover clustering. As long as replication is already being planned for, this might be part of the desired solution.

To enable multiple warehouses to participate in replication and to share information, you must create a Primary Key constraint that combines both the location of the warehouse and the identifier for the data itself.

Here is the desired list:

- Minimum Requirements:

  Compound Primary Key

- Recommended Requirements:

  Second volume for transaction log and Full Recovery mode

  Frequent log backups and the development of a disaster recovery plan

  Data storage set up in RAID parity with OS mirrored

- Also Possible (Recommended Order):

  Replication to additional subscriber at each location

  Standby server with log shipping configured (achieves similar results as replication)

  Cost Restrictive

- Failover Clustering

CHAPTER SUMMARY

This chapter has considered a number of different subjects, all pertaining to the physical aspects of a database system. Although the exam concentrates on the new features in SQL Server 2000, you will find that almost all topics are addressed in some manner. In a real-world scenario, you will find that the best solution often involves a compromise, rather than the ideal. Many considerations, such as budget, time constraints, user knowledge, and technology bias might hamper your ability to achieve an optimum environment.

When in doubt, select the options that provide for the best performance in the system. Next to performance, a provision to minimize administrative effort would probably come in a close second. Performance is gained through the use of the optimum levels of hardware, especially options pertaining to the disk, controller, and volume configuration. Performance is gained in interactions between the database and file system, so appropriate placement of files, file-groups, and database objects becomes very important. Within the databases, selection of appropriate indexing, constraints, triggers, and other related processing all help as well.

To achieve minimal levels of administration, look to set up and utilize existing features that SQL Server can perform automatically. Be careful: Using too many of the features that are automated might detract from the system performance.

KEY TERMS

- constraint
- collation Sequence
- identity
- indexes
- stored procedures
- triggers
- user-defined functions (UDFs)
- views
- cascading actions
- `CHECK` constraints
- clustered index
- defaults
- `FILLFACTOR`
- Foreign Keys
- non-clustered index

CHAPTER SUMMARY

- Primary Key
- UNIQUE constraints
- UNIQUE index
- schema binding
- encryption
- recompile
- partitioned views
- merge replication

- RAID
- snapshot replication
- filegroups
- log
- rules
- roles
- UNIQUE constraint
- transactional replication

APPLY YOUR KNOWLEDGE

Exercises

The following set of exercises runs through the set of steps needed to create a basic physical environment. Each step requires that you have completed all subsequent steps and are thus better done in sequential order.

3.1 Creating a Database

In Exercise 3.1, you create a database with two data files and a single log file. The exercise makes the assumption that SQL Server 2000 has been installed into the default locations and uses this location for all files. You might want to alter the actual file locations.

Estimated Time: 5 minutes

1. If the SQL Query Analyzer is not already open, load it to enable you to create a database using T-SQL commands. Supply the logon connection information if requested.

2. Select the `Master` database from the database drop-down list box from the toolbar.

3. Type the following command to create the database and accompanying files:

```
CREATE DATABASE Sample ON
PRIMARY (NAME = 'SampleData', FILENAME =
'c:\Program Files\Microsoft SQL Server
➥\MSSQL\Data\SampData.mdf',
  SIZE = 10, MAXSIZE = 50, FILEGROWTH = 5),
FILEGROUP Archive (Name = 'ArchiveData',
➥FILENAME =
'c:\Program Files\Microsoft SQL Server
➥\MSSQL\Data\ArchData.ndf',
  SIZE = 10, MAXSIZE = 50, FILEGROWTH = 5)
LOG ON
  (NAME = 'LogStore', FILENAME =
'c:\Program Files\Microsoft SQL Server
➥\MSSQL\Data\SampLog.ldf',
    SIZE = 1MB, MAXSIZE = 5MB,
    ➥FILEGROWTH = 1MB)
Go
```

4. Execute the query. Check the Windows environment using the Explorer to ensure the files were actually created. You can check for the existence of the database and ensure the associated database properties were set up by running the `sp_helpdb` stored procedure.

Note that when the database is created that there are two resulting filegroups. The Primary filegroup is used for data storage and the Archive filegroup is used for the storage of noncurrent data.

3.2 Creating a Table

In Exercise 3.2, you create three tables within the database. One table contains contact information; the second is a table to list events in the upcoming year; the third table is for holding archive data for events that have already passed.

Estimated Time: 10 minutes

1. If the SQL Query Analyzer is not already open, load it to allow for the creation of the tables. Supply the logon connection information if requested.

2. Select the `Sample` database created in Exercise 3.1 from the database drop-down list box from the toolbar.

3. Type the following command to create the database and accompanying files:

```
CREATE TABLE Contacts
          ( ContactID     smallint    NOT
            ➥NULL
              CONSTRAINT  PKContact
              ➥PRIMARY KEY CLUSTERED,
            FirstName     varchar(25) NOT
            ➥NULL,
            LastName      varchar(25) NOT
            ➥NULL,
            PhoneNo       varchar(15) NULL,
            StreetAddress varchar(25) NULL,
            City          varchar(25) NULL,
            ZipCode       varchar(15) NULL )
```

continues

continued

```
CREATE TABLE Events
           ( EventID      smallint    NOT
             ➥NULL
               CONSTRAINT  PKEvent
               ➥PRIMARY KEY CLUSTERED,
             EventName     varchar(50) NOT
             ➥NULL,
             EventLocation varchar(50) NOT
             ➥NULL,
             ContactID     smallint    NOT
             ➥NULL,
             EventAddress  varchar(25) NULL,
             City          varchar(25) NULL,
             ZipCode       varchar(15) NULL )

CREATE TABLE EventArchive
           ( EventID      smallint    NOT
             ➥NULL
               CONSTRAINT  PKEventArch
               ➥PRIMARY KEY CLUSTERED,
             EventName     varchar(50) NOT
             ➥NULL,
             EventLocation varchar(50) NOT
             ➥NULL,
             ContactID     smallint    NOT
             ➥NULL,
             EventAddress  varchar(25) NULL,
             City          varchar(25) NULL,
             ZipCode       varchar(15) NULL )
             ON Archive
```

4. Execute the query. Check the Object Browser to ensure the files were actually created. Execute `sp_help` with the table name to ensure that each table was appropriately created as follows:

```
sp_help Contacts
Go
sp_help Events
Go
sp_help EventArchive
Go
```

Note that by supplying the `ON` clause for the `EventArchive` table that the table was placed onto a separate filegroup whereas the other tables were placed onto the `PRIMARY` file group. Also notice the existence of the Primary Key and an associated clustered index.

3.3 Setting Up Referential Integrity

In Exercise 3.3, you alter the tables so that the Contact IDs of the `Event` table become Foreign Keys pointing to the Contact ID in the `Contacts` table.

Estimated Time: 5 minutes

1. If the SQL Query Analyzer is not already open, load it to allow for the creation of the tables. Supply the logon connection information if requested.

2. Select the `Sample` database created in the previous exercise from the database drop-down list box from the toolbar.

3. Type the following command to create the database and accompanying files:

```
ALTER TABLE Events
           ADD CONSTRAINT  FKEventContactID
               FOREIGN KEY (ContactID)
               REFERENCES Contacts(ContactID)

ALTER TABLE EventArchive
           ADD CONSTRAINT  FKArchContactID
               FOREIGN KEY (ContactID)
               REFERENCES Contacts(ContactID)
```

4. Execute the query. Check the Object Browser to ensure the constraints were actually created. Execute `sp_help` with the table name to ensure each table was modified correctly:

```
sp_help Events
Go
sp_help EventArchive
Go
```

Note that new constraints were added that reference the appropriate column in the `Contacts` table.

APPLY YOUR KNOWLEDGE

Review Questions

1. How would you design a set of tables in a circumstance where the deletion of one record should cause the deletion of records in related tables?
2. What are the storage considerations in an extremely limited budget? What basic configuration requirements would be set up to ensure optimum data recoverability in the event of data corruption?
3. How do you use SQL Server 2000 technologies to maintain data integrity?
4. If data and indexes are stored in two different filegroups, what considerations are there for performing backups?
5. For what purposes does the term "schema" apply?

Exam Questions

1. You are working for a large international organization that supplies packaging materials for companies that require custom commercial designs. The number of products is becoming too large for the current computer system to handle and you need to provide a solution that will spread the load over the current server and a new machine coming into the system. Queries need to be performed over a wide variety of products and there is no predictable pattern to the queries. What is an appropriate technique to implement the changes?

   A. Configure replication using the new machine as a subscriber and the original machine as the publisher/distributor to balance the workload.

   B. Separate the table into two smaller tables and place one table on each server. Configure a partitioned view and appropriate constraints on each of the machines.

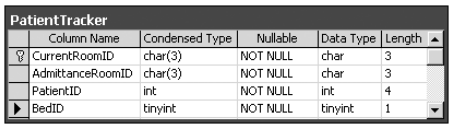
   C. Implement multi-server clustering so that each of the two servers can respond to data activities, thus achieving a balanced workload.

   D. Configure log shipping on both servers to have a copy of the data on each of the servers and propagate all changes to the alternate machine.

2. As a developer for a large healthcare provider, you are assigned the task of developing a process for updating a patient database. When a patient is transferred from one floor to another, an internal identifier, `CurrentRoomID`, which is used as the Primary Key, needs to be altered while the original key, `AdmittanceRoomID`, is still maintained. If a patient is moved more than once, only the original key and the current key need to be maintained. Several underlying tables have been configured for referential integrity against the patient table. These underlying tables must change in an appropriate manner to match with one or the other of the room keys in the patient table. These relationships will be altered based upon different situations in other tables. Figure 3.11 illustrates the `PatientTracker` table design exhibit. What method would you use to accommodate the update?

   A. Use the Cascade Update Related Fields option to have changes in the Primary Key automatically update the keys in all referenced tables.

   B. Use an indexed view to enable the user to make changes to multiple tables concurrently.



PatientTracker

Column Name	Condensed Type	Nullable	Data Type	Length
CurrentRoomID	char(3)	NOT NULL	char	3
AdmittanceRoomID	char(3)	NOT NULL	char	3
PatientID	int	NOT NULL	int	4
BedID	tinyint	NOT NULL	tinyint	1

FIGURE 3.11
`PatientTracker` table design exhibit.

APPLY YOUR KNOWLEDGE

C. Disable the Enforce Relationship for `INSERTs` and `DELETEs` option to enable an `AFTER TRIGGER` to handle the necessary changes.

D. Define an `INSTEAD OF UPDATE TRIGGER` to perform the necessary updates to all related tables.

3. A large organization needs to maintain `IMAGE` data on a database server. The data is scanned in from documents received from the federal government. Updates to the images are infrequent. When a change occurs, usually the old row of data is archived out of the system and the new document takes its place. Other column information that contains key identifiers about the nature of the document is frequently queried by an OLAP system. Statistical information on how the data was queried is also stored in additional columns. The actual document itself is rarely needed except in processes that print the image. Which of the following represents an appropriate storage configuration?

   A. Place the `IMAGE` data into a filegroup of its own, but on the same volume as the remainder of the data. Place the log onto a volume of its own.

   B. Place all the data onto one volume in a single file. Configure the volume as a RAID parity set and place the log into a volume of its own.

   C. Place the `IMAGE` onto one volume in a file of its own and place the data and log files together on a second volume.

   D. Place the `IMAGE` into a separate filegroup with the log on one volume and the remainder of the data on a second volume.

4. You are the administrator of a SQL Server 2000 computer. The server contains your company's `Accounts` database. Hundreds of users access the database each day. You have been experiencing power interruptions, and you want to protect the physical integrity of the `Accounts` database. You do not want to slow down server operations. What should you do?

   A. Enable the torn page detection database option for each database.

   B. Disable write caching on all disk controllers.

   C. Create a database maintenance plan to check database integrity and make repairs each night.

   D. Ensure that the write caching disk controllers have battery backups.

5. An Internet company sells outdoor hardware online to over 100,000 clients in various areas of the globe. Servicing the web site is a SQL Server whose performance is barely adequate to meet the needs of the site. You would like to apply a business rule to the existing system that will limit the outstanding balance of each customer. The outstanding balance is maintained as a denormalized column within the customer table. Orders are collected in a second table containing a trigger that updates the customer balance based on `INSERT`, `UPDATE`, and `DELETE` activity. Up to this point, care has been taken to remove any data from the table if the client balance is too high, so all data should meet the requirements of your new process. How would you apply the new data check?

   A. Modify the existing trigger so that an order that allows the balance to exceed the limit is not permitted.

APPLY YOUR KNOWLEDGE

B. Create a check constraint with the No Check option enabled on the `customer` table, so that any inappropriate order is refused.

C. Create a rule that doesn't permit an order that exceeds the limit and bind the rule to the `Orders` table.

D. Create a new trigger on the `Orders` table that refuses an order that causes the balance to exceed the maximum. Apply the new trigger to only INSERT and UPDATE operations.

6. An existing sales catalog database structure exists on a system within your company. The company sells inventory from a single warehouse location that is across town from where the computer systems are located. The product table has been created with a non-clustered index based on the product ID, which is also the Primary Key. Non-clustered indexes exist on the product category column and also the storage location column. Most of the reporting done is ordered by product category. How would you change the existing index structure?

   A. Change the definition of the Primary Key so that it is a clustered index.

   B. Create a new clustered index based on the combination of storage location and product category.

   C. Change the definition of the product category so that it is a clustered index.

   D. Change the definition of the storage location so that it is a clustered index.

7. You are the sole IT person working in a small branch office for a non-profit organization that deals with natural resource conservation issues. A non-critical database is maintained on the database server. You have been given the task of configuring appropriate database properties that would allow for a minimum use of execution time and storage resources. Which of the following set of properties is most appropriate?

   A. Full Recovery, Auto Shrink, Torn Page Detection

   B. Bulk Recovery, Auto Shrink, Single User

   C. Simple Recovery, Auto Close, Auto Shrink

   D. Simple Recovery, Auto Shrink, Single User

   E. Bulk Recovery, Auto Close, Auto Shrink

8. You are designing an application that will provide data entry clerks the capability of updating the data in several tables. You would like to ease entry and provide common input so the clerks need not enter data into all fields or enter redundant values. What types of technologies could you use to minimize the amount of input needed? Select all that apply.

   A. Foreign Key

   B. Cascading Update

   C. Identity Column

   D. Default

   E. `NULL`

   F. Primary Key

   G. Unique Index

9. A database that you are working on is experiencing reduced performance. The database is used almost exclusively for reporting, with a large number of inserts occurring on a regular basis. Data is cycled out of the system four times a year as part of quarter-ending procedures. It is always impor-

tant to be able to attain a point-in-time restoration process. You would like to minimize the maintenance needed to accommodate increases and decreases in file storage space. Which option would assist the most in accomplishing the task?

A. `SIMPLE RECOVERY`

B. `AUTOSHRINK`

C. `MAXSIZE`

D. `AUTOGROW`

E. `COLLATE`

10. You are the administrator of a SQL Server 2000 computer. The server contains a database named `Inventory`. Users report that several storage locations in the `UnitsStored` field contain negative numbers. You examine the database's table structure. You correct all the negative numbers in the table. You must prevent the database from storing negative numbers. You also want to minimize use of server resources and physical I/O. Which statement should you execute?

A.
```
ALTER TABLE dbo.StorageLocations ADD
➥CONSTRAINT
CK_StorageLocations_UnitsStored
CHECK (UnitsStored >= 0)
```

B.
```
CREATE TRIGGER CK_UnitsStored On
➥StorageLocations
FOR INSERT, UPDATE AS
IF INSERTED.UnitsStored < 0 ROLLBACK TRAN
```

C.
```
CREATE RULE CK_UnitsStored As @Units >= 0
GO
sp_bindrule 'CK_UnitsStored'
'StorageLocations.UnitsStored'
    GO
```

D.
```
CREATE PROC UpdateUnitsStored
(@StorageLocationID int, @UnitsStored
➥bigint) AS
IF @UnitsStored < 0
RAISERROR (50099, 17)
ELSE
UPDATE StorageLocations
SET UnitsStored = @UnitsStored
WHERE StorageLocationID =
➥@StorageLocationID
```

11. You are the administrator of a SQL Server 2000 computer. The server contains a database named `Inventory`. In this database, the `Parts` table has a Primary Key that is used to identify each part stored in the company's warehouse. Each part has a unique UPC code that your company's accounting department uses to identify it. You want to maintain the referential integrity between the `Parts` table and the `OrderDetails` table. You want to minimize the amount of physical I/O that is used within the database. Which two T-SQL statements should you execute? (Each correct answer represents part of the solution. Choose two.)

A. `CREATE UNIQUE INDEX IX_UPC On Parts(UPC)`

B.
```
CREATE UNIQUE INDEX IX_UPC On
➥OrderDetails(UPC)
```

C.
```
CREATE TRIGGER UPCRI On OrderDetails
FOR INSERT, UPDATE As
If Not Exists (Select UPC From Parts
Where Parts.UPC = inserted.UPC) BEGIN
    ROLLBACK TRAN
END
```

D.
```
CREATE TRIGGER UPCRI On Parts
FOR INSERT, UPDATE As
If Not Exists (Select UPC From Parts
Where OrderDetails.UPC = inserted.UPC)
➥BEGIN
    ROLLBACK TRAN
END
```

E.
```
ALTER TABLE dbo.OrderDetails ADD
➥CONSTRAINT
FK_OrderDetails_Parts FOREIGN KEY(UPC)
REFERENCES dbo.Parts(UPC)
```

F.
```
ALTER TABLE dbo.Parts ADD CONSTRAINT
FK_Parts_OrderDetails FOREIGN KEY (UPC)
REFERENCES dbo.Parts(UPC)
```

APPLY YOUR KNOWLEDGE

12. You are the database developer for a leasing company. Your database includes a table that is defined as follows:

```
CREATE TABLE Lease
(Id Int IDENTITY NOT NULL
  CONSTRAINT pk_lesse_id PRIMARY KEY
➥NONCLUSTERED,
Lastname varchar(50) NOT NULL,
FirstName varchar(50) NOT NULL,
SSNo char(9) NOT NULL,
Rating char(10) NULL,
Limit money NULL)
```

Each SSNo must be unique. You want the data to be physically stored in SSNo sequence. Which constraint should you add to the `SSNo` column on the `Lease` table?

A. `UNIQUE CLUSTERED` constraint

B. `UNIQUE UNCLUSTERED` constraint

C. `PRIMARY KEY CLUSTERED` constraint

D. `PRIMARY KEY UNCLUSTERED` constraint

13. You are building a database and you want to eliminate duplicate entry and minimize data storage wherever possible. You want to track the following information for employees and managers: First name, middle name, last name, employee identification number, address, date of hire, department, salary, and name of manager. Which table design should you use?

A. Table1: EmpID, MgrID, Firstname, Middlename, Lastname, Address, Hiredate, Dept, Salary. Table2: MgrID, Firstname, Middlename, Lastname.

B. Table1: EmpID, Firstname, Middlename, Lastname, Address, Hiredate, Dept, Salary. Table2: MgrID, Firstname, Middlename, Lastname. Table3: EmpID, MgrID.

C. Table1: EmpID, MgrID, Firstname, Middlename, Lastname, Address, Hiredate, Dept, Salary.

D. Table1: EmpID, Firstname, Middlename, Lastname, Address, Hiredate, Dept, Salary. Table2: EmpID, MgrID Table3: MgrID.

14. You are developing an application and need to create an inventory table on each of the databases located in New York, Detroit, Paris, London, Los Angeles, and Hong Kong. To accommodate a distributed environment, you must ensure that each row entered into the inventory table is unique across all locations. How can you create the inventory table?

A. Supply `Identity` columns using a different sequential starting value for each location and use an increment of 6.

B. Use the `identity` function. At first location, use `IDENTITY(1,1)`, at second location use `IDENTITY(100000,1)`, and so on.

C. Use a `Uniqueidentifier` as the key at each location.

D. Use `TIMESTAMP` column as the key at each location.

15. You are building a new database for a company with ten departments. Each department contains multiple employees. In addition, each employee might work for several departments. How should you logically model the relationship between the department entity and the employee entity?

A. A mandatory one-to-many relationship between department and employee.

B. An optional one-to-many relationship between department and employee.

C. Create a new entry; create a one-to-many relationship from the employee to the new entry; and create a one-to-many relationship from the department entry to the new entry.

D. Create a new entry; create a one-to-many relationship from the new entry to the employee entry; then create a one-to-many relationship from the entry to the department entry.

Answers to Review Questions

1. Configuring a Cascading Delete Action causes the deletion of a record to propagate deletions throughout the underlying related table. This option should be configured with caution, because the deletion of underlying data might not be desired. You need to set two tables up in a parent-child relationship, using appropriate Primary and Foreign Keys. Cascading Update Action performs a similar operation when key values are changed, propagating the new values to underlying child tables.
2. First, put the log files onto a volume other than where the data is stored to ensure optimum recoverability. If possible, use a mirror on the OS volume to minimize downtime in a system failure. If the data volume becomes corrupt, a restore can be performed to get that data back. Having the log on a separate volume means that you can recover additional data because the log volume should be unaffected by the damage to the data.
3. Referential integrity is used to create a link between two related tables. A Foreign Key in one table references a Primary Key or unique index in the other table. Any entry to the table in which the Foreign Key resides must have a matching record in the table containing the Primary Key. Rules, constraints, triggers, and defaults all participate in maintaining data integrity.
4. Both filegroups must be backed up within the same backup set whenever the indexes are separated from the data. Care must be taken so that the indexes always maintain pointers to the corresponding data.
5. In SQL Server, there are several uses of the term "schema." Information, Database, XML, and Warehouse all use schema to define the structure of elements, whether they be statistics, data dictionaries, data structures, or cube dimensions.

Answers to Exam Questions

1. **B.** This is a perfect example of where partitioning a table into two smaller objects enables you to use two machines to help reduce the load on the overall application. Remember that failover clustering is the only form of clustering supported by SQL and therefore does not actually reduce the load; it only assists in obtaining an around-the-clock operation. Log shipping assists in offloading query load, but does little to reduce update load because it leaves the second server in a read-only state. Merge replication may enable updates to span many servers, but the associated overhead and data latency makes it a less than desirable alternative. For more information, see "Partitioning to Achieve a Balance."
2. **D.** The `INSTEAD OF` trigger was designed specifically for this type of situation and also to handle complicated updates where columns are defined as Timestamp, Calculated, or Identity. Cascade operations are inappropriate because the updated key is not always stored. Indexed views by themselves do not allow for the type of alteration

APPLY YOUR KNOWLEDGE

desired and would have to be complemented with the actions of a trigger. Disabling referential integrity is a poor solution to any problem, especially considering the medical nature of this application and the possible ramifications. For more information, see "Trigger Utilization."

3. **D.** Because the `IMAGE` data will seldom be accessed, it makes sense to get the remainder of the data away from the images while moving the log away from the data. This will help to improve performance while providing optimum recoverability in the event of a failure. For more information, see "Using Filegroups."

4. **D.** Good controllers suitable for database use will have a battery backup. The battery should be regularly tested under controlled circumstances. Disabling caching if currently in place is likely to affect performance, as will enabling torn page detection. Torn page detection might help point out whether data is being corrupted because of failures. A maintenance plan is recommended, although it is not an entire solution in its own right.

5. **A.** Because a trigger is already in place, it can easily be altered to perform the additional data check. A rule cannot provide the required functionality because you cannot compare the data. The `CHECK` constraint may be a viable solution but you would have to alter the trigger to check for an error and provide for nested operations. The number of triggers firing should be kept to a minimum. To accommodate additional triggers, you would have to check the order in which they are being fired and again set properties of the server and database accordingly. For more information, see "Trigger Utilization."

6. **C.** Because the majority of the reporting is going to be performed using the storage location, it would be the likely candidate. The clustered index represents the physical order of the data and would minimize sorting operations when deriving the output. For more information, see "Index Organization."

7. **C.** Simple Recovery uses the least amount of log space for recording changes to the database. Full recovery uses the most space because it fully logs any bulk operations. Bulk recovery represents a mid-point between the two. Auto Close frees up resources at the earliest possible point during process execution, and Auto Shrink minimizes the space used in the file system by periodically reducing the files when there is too much unused space. For more information, see "Use of Recovery Models."

8. **B, C, D, E.** All these options have activities that provide or alter data so that it does not have to be performed as an entry operation. In the case of `NULL`, data need not be provided, possibly because the column contains non-critical information. For more information, see "Table Characteristics."

9. **D.** Use `AUTOGROW` to set the system so that the files will grow as needed for the addition of new data. You may want to perform a planned shrinkage of the database as part of the quarter-ending process and save on overhead by leaving the `AUTOSHRINK` option turned off. For more information, see "Creating Database Files and Filegroups."

10. **A.** You need to add a constraint to prevent negative data entry. The best method of implementing this functionality is a constraint. A trigger has too much overhead and the `RULE` is not accurately implemented. A procedure could handle the process but is normally only used for processes requiring more complex logic. For more information, see "Table Characteristics."

APPLY YOUR KNOWLEDGE

11. **A, E.** The `UNIQUE` constraint on the `Parts` table UPC column is required first, so that the `FOREIGN KEY` constraint can be applied from the OrderDetails.UPC column referencing Parts.UPC. This achieves the referential integrity requirement. It also reduces I/O required during joins between Parts and OrderDetails, which make use of the `FOREIGN KEY` constraint defined. For more information, see "Maintaining Referential Integrity."

12. **A.** To obtain the physical storage sequence of the data, you must use a clustered constraint or index. Although a Primary Key would also provide for the level of uniqueness, it is not the desired key for this table. For more information, see "Unique Indexing" in Chapter 10.

13. **C.** A single table could provide all the necessary information with no redundancy. The table could easily be represented using a self-join operation to provide the desired reporting. Join operations will be discussed in detail in the next chapter.

14. **A.** Using identities in this fashion enables records to be entered that have no overlap. One location would use entry values 1, 7, 13, 19; the next would have 2, 8, 14, 20; the third 3, 9, 15, 21, and so on. For more information, see "Application of Integrity Options."

15. **C.** This is a many-to-many relationship scenario, which in SQL Server is implemented using three tables. The center table, often referred to as the connecting or joining table, is on the many side of both of the relationships to the other base table. For more information, see "Maintaining Referential Integrity."

Suggested Readings and Resources

1. Inside SQL Server 2000 – Kalen Delaney (`www.insidesqlserver.com`)

Not a beginner book, but it fills in many of the gaps left out of the SQL Server Books Online documentation. Explains fully how SQL Server stores and processes data internally.

2. SQL Server 2000 Books Online
   - Creating and Maintaining Databases (Look particularly at the sections on indexes, views, and triggers.)
   - Transact-SQL Reference (Use this as a resource for the specific syntax requirements of each statement, as well as some code examples.)
   - Optimizing Database Performance (Focus on Database and Application Design.)
   - Troubleshooting: Server and Database Troubleshooting

3. MSDN Online Internet Reference (`http://msdn.microsoft.com`)
   - Transact SQL Overview (`/library/psdk/sql/ts_tsqlcon_6lyk.htm`)
   - Transact SQL Syntax Conventions (`/library/psdk/sql/ts_syntaxc_9kvn.htm`)
   - Transact SQL Tips (`/library/psdk/sql/ac_8_qd_14_2kc3.htm`)