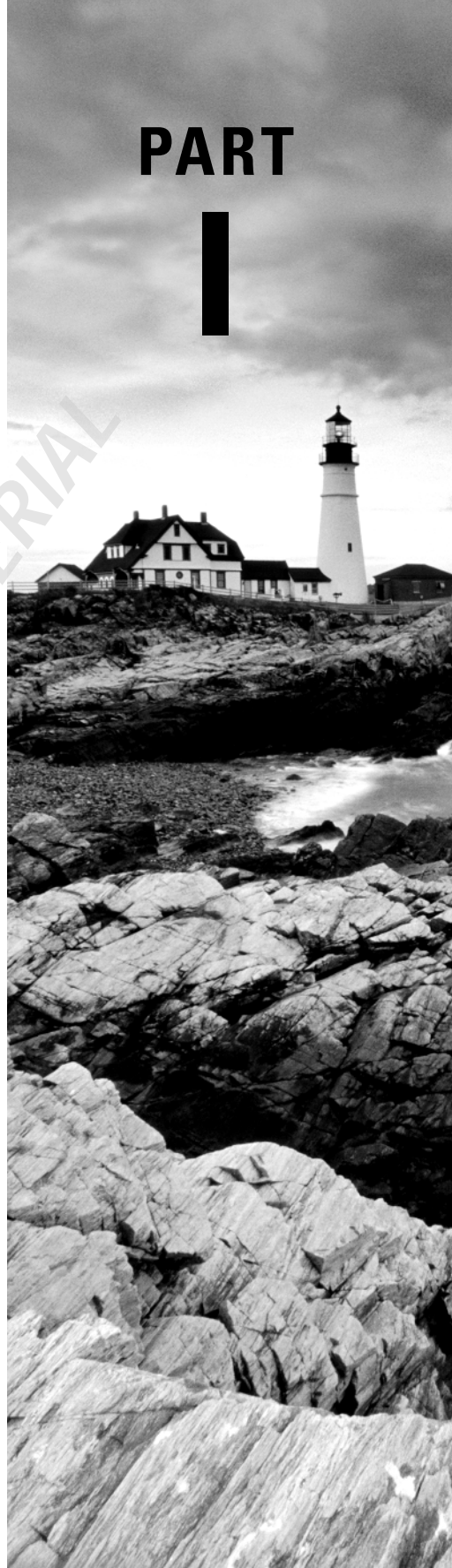


# Introducing SQL Server 2012

## PART

# I

- CHAPTER 1 ■** Understanding SQL Server's Role
- CHAPTER 2 ■** Installing SQL Server 2012
- CHAPTER 3 ■** Working with the Administration Tools
- CHAPTER 4 ■** SQL Server Command-Line Administration
- CHAPTER 5 ■** Querying SQL Server





# Chapter 1

## Understanding SQL Server's Role

---

### TOPICS COVERED IN THIS CHAPTER:

- ✓ What Is Information Technology?
- ✓ Introduction to Databases
- ✓ Database Servers and Applications
- ✓ SQL Server's Role





Microsoft SQL Server 2012 is a database management system that provides enterprise-class features for organizations of all sizes. If you are tasked with administering a SQL Server, you need to understand the various roles it can play within an organization. This understanding comes best by studying from the foundation up, and this chapter provides that foundation. From this foundation, you will move through this book to learn how to administer the essential aspects of SQL Server 2012. In addition, the contents of exams 70-461 (Querying Microsoft SQL Server 2012), 70-462 (Administering a Microsoft SQL Server 2012 Database), and 70-463 (Implementing Data Warehouses with Microsoft SQL Server 2012) are covered throughout the book.

The first major topics you'll tackle in this chapter are the concept of *information technology* and the role a database or database system plays within this concept. Next, you'll look at databases in more detail and gain an understanding of fundamental concepts that apply to *all* databases, not just SQL Server databases. Once you've sufficiently covered the general database concepts, you'll investigate database servers and applications. Finally, you'll explore SQL Server's features and the roles SQL Server can play in modern organizations.

## What Is Information Technology?

Many organizations differentiate between information systems (IS) and information technology (IT). In general, IS deals with software and system development, and IT is concerned with technology management. Certainly, IT is the collection of technologies and resources used to manage information. Organizations place great value on their information, as they should, and they expect the IT group to manage this information well. It is essential that those of us who work in IT remember the *I* stands for *information* and that our primary responsibilities are to collect, retain, distribute, protect, and when appropriate destroy that information. When a single group is responsible for these tasks, consistency is accomplished and security can be achieved.

### The Importance of IT

Consider an organization that manufactures and sells the components used to make fishing lures. These components are used by many different fabricators and distributors. What would happen if a competing company stole the customer database of the world's

top fishing-lure company? The results could be catastrophic. However, if the company's IT department creates and uses the proper information-protection mechanisms, the event could be mitigated or the theft itself could be prevented.



Throughout this book, the term *SQL Server* will refer to Microsoft's database server product in general. When a discussion is applicable only to a specific version of SQL Server, the appropriate version number, such as SQL Server 2012, will be specified.

In addition, I pronounce SQL Server as "sequel server," and I pronounce the SQL language as "ess-cue-el." You'll notice this based on the articles ("a" versus "an") that I use. I have reasons for my pronunciations, but I'll reserve those for a later chapter.

Although losing a database to a competitor is an extreme example of why an IT department is needed, there are many day-to-day problems and issues that arise within a company that are best handled by the IT department. For instance, customer service professionals aren't as productive or effective when they cannot access data (information distribution) when they need it to answer customers' questions. Customers may become impatient if their questions aren't sufficiently addressed, and they could very well choose a different provider. An effective IT department helps everyone within a company manage information so each team can be successful.

Effective IT solutions enable the five key responsibilities of information management to be accomplished.

**Information Collection** Database systems and applications are used to collect information from users. Well-coded applications validate data integrity and ensure that only valid users can enter or modify information.

**Information Retention** A good information storage system provides effective storage and backup mechanisms. You'll learn about SQL Server's backup solutions in Chapter 17, "Backup and Restoration."

**Information Distribution** The right people need the right information at the right time, and information distribution solutions allow for this. Examples include replication, mirroring, Integration Services packages, and more.

**Information Protection** There are many different types of information with varying degrees of priority and confidentiality. In most organizations, only certain users should have access to certain information. Security solutions from authentication to storage encryption should be used to protect valuable data. Additionally, coding best practices should be followed in order to prevent the opening of accidental back doors into your information stores.

**Information Destruction** Sometimes information needs to be destroyed. Your IT solutions should account for this and ensure that a nonrecoverable method is used to destroy the data when it is required.

These five facets of information management must be included in any IT plan. SQL Server databases can assist with these processes. Although SQL Server features and capabilities can be integrated with client solutions and network infrastructure solutions to do so, SQL Server cannot provide all of the best solutions alone. An authentication system, such as Microsoft's Active Directory, will be needed to provide secure authentication. Additionally, although SQL Server integrates with Windows Server Active Directory domains to provide stronger authentication, if the SQL Server is not integrated with a Windows domain and the client computers are running non-Windows operating systems, you may be required to implement a virtual private network (VPN) or Internet Protocol Security (IPSec) association with the SQL Server before the users can authenticate. This VPN solution can be implemented using Microsoft's Routing and Remote Access Services (RRAS) service or a third-party product.

## The Components of IT

In today's computing environments, IT is responsible for three core components:

**Client Solutions** These include desk-top computers, laptops or notebooks, portable devices, and even telephones in Voice over IP implementations.

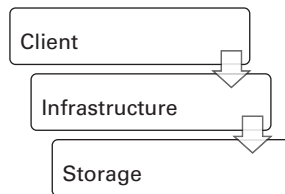
**Network Infrastructure Solutions** These include switches, routers, and network communications services. Network communications services allow communications to take place on the network, such as DNS, DHCP, authentication services, and so on.

**Information Storage Solutions** These include databases, file servers, and networked storage such as Network Attached Storage (NAS) and storage area networks (SANs).

These core components will be discussed further throughout this book as you learn about SQL Server and how to deploy and administer it in any environment.

Figure 1.1 shows the core components of IT.

**FIGURE 1.1** The core components of IT



Understanding how SQL Server operates within these three areas is crucial for the modern database administrator (DBA). Unlike DBAs in the past, today's DBAs must understand the basics of the operating system on which the database solution runs, the fundamentals of network communications, and the clients that talk to the database server. Gone are the days of simply replacing a dumb terminal if a user cannot communicate with the database (or at least those days are far less common for most of us).

When you implement advanced SQL Server features, such as database mirroring, you need to understand how to determine whether a communication problem is caused by an internal configuration error or a problem in the network infrastructure between the two SQL Servers involved. Even if you're not responsible for repairing the network infrastructure, you'll need to know when to contact the network administrator at the very least.

Many support professionals work in small organizations (or small groups within larger organizations), and they must be able to support practically everything that has a wire in their buildings. Of course, this means they need to understand everything in the communication chain from the database server to the client and back again. For this reason, this book will teach you more than just how to work with SQL Server. It will explain how SQL Server works with your other systems, including Windows clients, non-Windows clients, and other servers.

## Introduction to Databases

The word *data* is defined as meaningful information, and it can include words, numbers, letters, and binary information such as images. The word *base* means foundation or place. Simply put, a database is a place to put your data. If you're looking for a more technical definition of a database, it would go something like this: a computer database is a (usually) structured collection of information stored according to a defined model and accessible through standard or proprietary database communications languages.



---

If you've been working with databases for many years, you may choose to skip this section and move on to, "SQL Server's Role." However, if you do read this section, you may be surprised and learn a few things. This choice is yours.

Make sure you don't confuse the database with the database management system. The, "Database Servers and Applications," section of this chapter will cover this difference in more detail. For now, just remember that the database is separate from the database management system, and it can usually be transferred from one computer running the compatible database management system to another computer running the same system.

## Types of Databases

The *database model* defines the way in which the data is stored. Most modern databases use the relational model, but other models also exist. In general terms, the database model is the type of database. Two primary types are still in use today: flat-file and relational databases.

## Flat-File Databases

All of the information in a *flat-file database* is stored in a single storage container. When stored in a database, information regarding customer orders might look something like Figure 1.2.

**FIGURE 1.2** A table of flat-file databases

OrderID	CustomerNum	CustomerName	Phone	Email
23	413	DaleThomas	937-555-0135	DaleThomas4532@company.net
27	413	DaleThomas	937-555-0135	DaleThomas4532@company.net
36	413	DaleThomas	937-555-0135	DaleThomas4532@company.net
42	413	DaleThomas	937-555-0135	DaleThomas4532@company.net

Here are a few key points to consider regarding flat-file databases:

**Flat-file databases result in high levels of data redundancy.** If you examine Figure 1.2, you can see redundancy in action. Note that the name Dale Thomas is repeated for each line item, as well as the customer number, phone number, and email address. If a separate table were used to store the customer information, this redundancy could be avoided.

**Flat-file databases cost more when data is added.** Because flat-file databases result in more redundancy, the system simply must write more information when data is added. When referring to an information system, the term *cost* can mean dollars and cents, or it can mean resource costs (CPU, memory, and so on). In this case, the costs are resource costs. You cannot ask a system to do more without consuming more resources within that system.

**Working with flat-file databases may be easier for some users.** This point is actually a positive characteristic of flat-file databases, and it is one of the many reasons you create views in relational databases. Flat-file databases are often easier for users to work with because all of the data is in one location. Consider the two SQL statements in Listing 1.1. (Don't worry if you don't fully understand SQL yet; you will learn more about it in Chapter 5, "Querying SQL Server.") Although the increased complexity of the relational database query may seem trivial, consider what it might look like if you have to join five or more tables together to retrieve the needed information. Because all of the data is in a container in the flat-file format, no join statements are needed, and all of the data is easily accessed by decision-support professionals or business managers who may not understand the complexities of relational queries.

### Listing 1.1: SQL Statement Examples

```
--This first query is on a relational database
```

```
SELECT dbo.Products.ProductID, dbo.Products.ProductName,
       dbo.Sales.OrderID, dbo.Sales.Quantity, dbo.Sales.Price
```



```

FROM dbo.Products
INNER JOIN dbo.Sales ON dbo.Products.ProductID = dbo.Sales.ProductID;

--This second query retrieves the same information from a flat-file database
SELECT dbo.Sales.ProductID, dbo.Sales.ProductName,
       dbo.Sales.OrderID, dbo.Sales.Quantity, dbo.Sales.Price
FROM dbo.Products;

```

This simplification is one of the driving factors behind many views that are created and behind many of the decisions that are made when online analytical processing (OLAP) databases are implemented. OLAP databases are usually read from (far more read-operations are performed as opposed to write-operations), and they may benefit from a flattened model; however, even with OLAP databases, it is still common to have multiple tables. The tables may simply be less *normalized* (understood as more redundant) than those for an online transaction processing (OLTP) database that processes large numbers of writes to the data.



*Normalization* is the process used to ensure that relational data is stored in a manner that removes or reduces anomalies in data modifications. The process also results in a reduction in redundancy within the data store. Normalization will be covered in more detail in Chapter 8, “Normalization and Other Design Issues.”

## Relational Databases

*Relational databases* store information in separate containers called *tables*. Each table represents a single entity, although *denormalized relational databases* may not always do so. You’ll learn about normalization in Chapter 8; for now, you just need to know that a relational database is a collection of entity containers (tables) that are related to one another in various ways.

When you convert the data in Figure 1.2 to a relational database model, the results should be similar to those shown in Figure 1.3. Notice that the Customers table is related to the Sales table so that the customer information is entered only once. In each order, the customer ID is used to reference everything about the customer. You could further optimize this database by breaking the Sales table into two tables: Sales and Items. The Sales table would contain the header information for the sale (sale date, sale ID, customer ID, and so on), and the Items table would list the details for each item purchased (product ID, price, quantity, and so on).

**FIGURE 1.3** The Sales and Items tables interact in a relational structure.

OrderID	CustomerNum	ProductID	Quantity	UnitPrice
23	413	45	12	12.45
27	413	32	6	14.97
36	413	78	53	3.78
42	413	98	13	12.17

CustomerNum	CustomerName	Phone	Email	City
413	Dale Thomas	937-555-0135	DaleThomas4532@company.net	Marysville
414	Amie Freeman	405-555-9090	Amie_F@company.net	Urbana
415	Tracy Mathys	417-555-0078	Tracy@thenet.com	Austin
416	Jose Ramos	913-555-1616	JoseRamos@company.net	Elk City

The relational model provides several benefits.

**Relational databases can be indexed and optimized more efficiently.** Relational databases can be indexed and optimized more efficiently because you are dealing with smaller units of information in each data store (each table). For example, you can index the Customers table uniquely for retrieving common columns of information, and you can index the Sales table uniquely for retrieving common columns of information retrieved from the Sales table. If the two tables were crammed together into a single flat structure, you would have to ask which is more important: customer columns or sales columns. You can create only so many indexes before you start hurting more than you help.

**Relational databases consume less space to store the same information than flat-file databases.** Because the redundancies have been removed, a relational database requires less space to store the same information as a flat-file database. For example, consider Figure 1.2 again. The customer ID, customer name, phone number, and email address must be added every time Dale Thomas places an order; however, with the structure in Figure 1.3, only the customer ID must be added with each order. You are, therefore, dealing with one column instead of four. You can see how the relational structure saves on storage space.

**Relational databases can handle more concurrent users more easily.** Because data is broken into logical chunks, relational databases can handle more concurrent users more easily. With the data store represented in Figure 1.2, even if the user wants only the sales-specific information with no information about the customer, all of the data must be locked in some way while the user retrieves the information. This behavior prevents other users from accessing the data, and everyone else must wait in line (what a database system usually calls a *queue*). The relational model is better because one user can be in the Sales table while another is in the Customers table. Of course, modern database systems go even further and usually allow locking at the data page or even the row (record) level.

**Relational databases are more scalable.** Because they allow for more granular tweaking and tuning, relational databases *scale* better. They store more information in less space.

They allow more users to access the data more quickly. These benefits are all realized in SQL Server 2012 databases.

Of course, the fact remains that a relational database that is heavily normalized (with extreme reductions in redundancy) may be much more difficult for users to utilize. For example, it is not uncommon to see the typical customer record build from four or more underlying tables in modern relational databases. This structure means that the users have to join the four or more tables together to retrieve that typical customer record. One of the key decisions a DBA makes is determining just how normalized a database needs to be. That question is addressed in Chapter 8.

## **Weighing the Benefits of Using a Local or Server-Based Database**

In addition to the flat-file versus relational database debate, the value of local databases versus server-based databases needs to be considered. Developers must continually decide which to use, and IT support professionals in general must also make this decision frequently. For example, when a vendor tells you that you can run their application with a locally installed database for a single user or with a SQL Server server-based database for several users, you must choose between the two.

Additionally, you may have to choose between using a database intended for local use (i.e., Access) and a database intended for server-based access (i.e., SQL Server) when just a few users need access to the data. Some organizations have successfully implemented Microsoft Access databases for 5 to 10 people, and others have faced tremendous difficulties allowing just 2 or 3 users to share a Microsoft Access database. Databases that are designed primarily for local access simply do not scale well, and when multiple users need access to the data, implementing a server-based database system is usually a better multiuser solution.

## **Understanding Local Databases**

A local database, such as Microsoft Access or FileMaker Pro, is completely stored on the user's machine or a network share the user can access. When using local file storage, the application that accesses the database uses a local data access engine to talk to the database file. No network communications occur. When it is stored on a network share, the database file is still treated as a local file from the perspective of the database application. The networking functionality in Windows is handled in a different part of the operating system called Kernel mode.

Truly local databases are good from one perspective: they do not consume network bandwidth. If only one user needs access to the data, local databases are often the way to go. The good news is that Microsoft provides a free version of SQL Server for this scenario, called SQL Server 2012 Express. In addition, Microsoft provides the SQL Server Compact edition for use on mobile devices such as PDAs. The features of these free editions are similar to those of the SQL Server 2012 Standard edition as long as you are using small databases, and you can use a solution you are familiar with for both your local databases and your server-based databases.



Three versions of Express edition are available: Express, Express with Tools, and Express with Advanced Services. Express edition comes with no GUI management tools, but both Express with Tools and Express with Advanced Services come with SQL Server Management Studio. Express with Advanced Services also adds more features such as full-text search and Reporting Services Express.

So, why use Microsoft Access or any other single-user database system today? For many organizations, the built-in forms engine in Microsoft Access is enough to justify continued use of the tool, while other IT departments simply don't have any use for it. Of course, you can use Microsoft Access to build forms, queries, and reports against a backend SQL Server database as well. The latter option is probably the best use of Microsoft Access today. And, yes, Microsoft Access can be used as a frontend for local SQL Server 2012 Express databases, although you will probably have to design the database in SQL Server Management Studio Express 2012.

## Understanding Server-Based Databases

The benefits of server-based databases can be grouped into three primary categories:

- Data availability
- Data integrity
- Data security

### Data Availability

Users need access to data when they need it. Although this point may seem obvious, it is often overlooked when developers build database solutions. Data availability can be considered from two viewpoints:

- Data persistence or existence
- Data access efficiency

From the perspective of data persistence, you need to ensure that your data is stored safely, is backed up properly, and is accessible to the appropriate users. To accomplish this, data that must be accessed by multiple users should be stored in a network location. Of course, Microsoft Access databases can be stored in a network location; however, depending on the database in question, fewer than five users may be able to access that data concurrently. The power of server-based databases really shines in this area; many server-based databases can handle hundreds or even thousands of users accessing the data concurrently. Local databases simply cannot match this scale.

Although network storage ensures that the data is accessible, the storage engine used by the server-based database will ensure that the data is stored safely. SQL Server uses transaction logs to help in this area. Active transaction logs are used to recover from minor failures, and backed-up transaction logs may be used to recover from major mistakes or failures. Either way, the server system establishes solid data storage processes to make sure the data gets into the database properly.

The last element of data existence is backup. The backup features of a server-based database system are usually far more extensive than those of local databases. In fact, most local

databases are backed up at the file level only. The entire file is copied to a backup location, and the data is backed up in this simple way. This simple method may seem beneficial, but it is missing an important feature: the ability to back up the database while a user is connected to it. Server-based systems usually provide this feature. For example, SQL Server allows online backups of the data that is in the database. This feature allows backups to occur even in 24/7 businesses, and it is essential to modern database systems.

For the data to exist or persist, regardless of the calamity, all three of these factors must be in place:

- The data must be appropriately stored when it is initially entered.
- The data must be backed up to protect against catastrophic failures.
- The data must be available when users want it.

SQL Server provides for all three factors.

The next element of data availability is access efficiency. It's one thing to say that users can get to the data they need. It is quite another to say that they can get to it in a timely fashion. Server-based database systems have much more complex locking algorithms, which allow them to handle many more users more quickly than a local or single-user database system. SQL Server can lock an entire table, a single data page (which may contain one or more rows), or a single row (record). In addition, SQL Server can use different lock types. For example, a shared lock can be acquired for data reads. This type of lock allows other users to read the same data without waiting for the first user's shared lock to release. Of course, exclusive locks can also be used when modifying data to ensure data integrity.

From the perspective of data availability for multiuser applications, there is just no comparison between a proper server-based database system like SQL Server and an intended single-user database system like Microsoft Access. When you need the data to be available to the right users at the right time and multiple users must access the same data, server-based systems win every time.

## Data Integrity

For the purposes of this book, *data integrity* is defined in a slightly different way than in most resources. Data integrity means that the data could be what it should be. Notice that the definition reads *could be* what it should be and not that it *is* what it should be. There is a simple reason for this definition: it is impossible to guarantee that all data is what it is supposed to be even with excellent data integrity policies and procedures. Why? It's because of the human element.

Most of the time, data is entered by humans and not by machines. As long as the programming is accurate, you can predict with certainty what a machine will do or generate in relation to data output; however, humans are not so predictable.

For example, imagine a company has a website form that a user must fill out in order to retrieve a white paper from the company. In that form, they ask the user to enter his or her email address, and they require that the email address field include data that is formatted like an email address (i.e., it has some characters followed by the @ sign, followed by more characters, and then a period and at least two more characters). Will every user enter their valid email address? Of course not! Users will often use completely fabricated addresses to avoid receiving spam from the company.

The company may decide to send a link to the email address in order to download the white paper. Will this force users to enter email addresses where the company can actually reach them? Not really. They could simply use something like, <http://10MinuteMail.com> or any of the dozens of free email servers. Yes, users really hate spam that much.

In the end, website applications usually settle for something that looks like an email address. They may try emailing the link just to see whether it is a valid email address, but there is no way to know if it is the user's real email address. So, the outcome is simple. The email address could be what it should be, but you don't know that it is what it should be.

For some data elements, there may be methods to guarantee that the data is accurate. For email addresses and many other similar data elements, you have to accept reality. However, this acquiescence does not mean you give up on data integrity. It simply means you employ data integrity measures that are worth the effort and stop there.

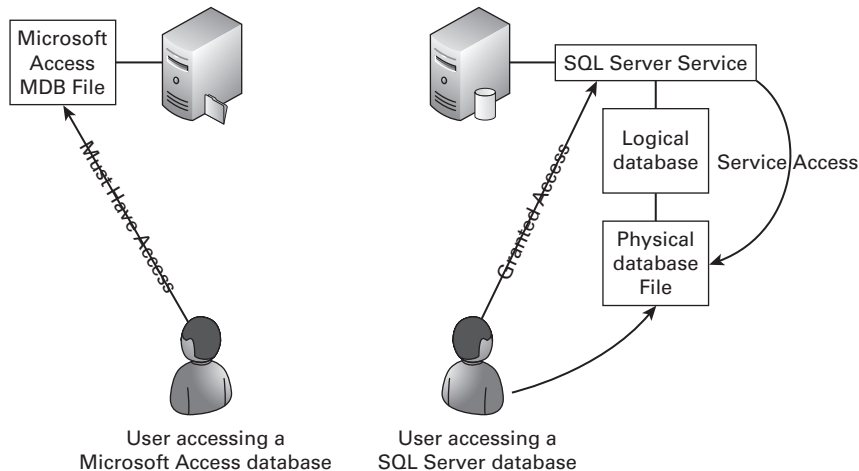
In the area of data integrity, there is not a tremendous difference between local database systems and server-based systems. For example, SQL Server offers triggers, and Access offers macros. SQL Server offers stored procedures, and, again, Access offers macros. SQL Server offers data types (to ensure that numbers are numbers, for example) and so does Access. The line is not as clear-cut here, but you will find that SQL Server triggers and stored procedures offer much more power than Access macros, thanks to the ability to run .NET code. Earlier versions of SQL Server used extended stored procedures, which were basically DLL files called by the SQL Server. This ability to run code developed in advanced languages is one of the separating factors between SQL Server and Microsoft Access in the area of data integrity. In addition, SQL Server has the Transact-SQL language, which is more powerful than the SQL version used in Microsoft Access.



In this context, data integrity is viewed from the perspective of accuracy. Data integrity can also be considered from a security or storage consistency perspective. From a security perspective, data integrity ensures that no malicious changes are made to the data. From a consistency perspective, it ensures that the data is not corrupted under normal data processing or storage operations. In Chapters 18 through 20, you'll learn about SQL Server security solutions. In Chapter 14, you'll learn how to analyze the integrity of the stored data.

## Data Security

Information is valuable, and for most organizations this information is stored primarily in two types of locations. The first type is a data file such as a spreadsheet, presentation, or typed document. The second is a server-based database. While databases are ultimately stored in files, the access methods for spreadsheets, presentations, and word processor documents differ. Server-based databases provide enhanced security for these databases. Figure 1.4 illustrates the difference between local or single-user database security and server-based database security.

**FIGURE 1.4** Comparing Microsoft Access and SQL Server database security

In the example in Figure 1.4, notice that the Access database requires users to have permissions on the database file itself. A user who wants to open an Access database from a network location must have at least Read permissions on the MDB file that holds the database. This presents a security concern in that many network operating systems allow a user with read access to a file to copy that file to their own computer or removable media such as a USB thumb drive.

Notice the difference in Figure 1.4 in the represented access to a SQL Server database. The user is given access to talk to the SQL Server service but is given no access to the data files themselves. This configuration means the user can access the data only through provided applications. If a user with read access wanted to copy all of the data to a thumb drive, the user would have to export the data. Such behavior could be easily logged and prevented through the use of security features built into SQL Server.

For example, access to data could be designed to occur only through stored procedures. With such a configuration, users are not given direct access to the tables. They are given access only to execute stored procedures. The stored procedures execute as a different user than the calling user, so they can access the data on the user's behalf. A data access model that relies solely on stored procedures could ultimately make it impossible for nonadministrative users to make a copy of the entire data set. Not only would the stored procedures limit the data returned with each execution, but they may further look for nonstandard use and disallow a given account access to the data until further investigation has been done or some acceptable time has passed.

A more passive security method would be the simple logging of any SELECT statements (basic database statements used mostly to read information) that read all of the data in a given table. For example, the system could watch for nonfiltered SELECT statements (statements without a WHERE clause) and log the username, the time of execution, and the actual statement. This log could be sent to security personnel who audit data access. Additionally,

the system could disallow more than one nonfiltered SELECT statement in a given window of time against multiple tables.

These actions do not need to be taken for every database. In fact, they should not be taken for most. However, these brief examples illustrate the power derived from an intermediary data access method that could be used if a very sensitive database must be placed online. The SQL Server service acts as the intermediary between the client and the database. As the man in the middle, SQL Server can provide many different data protection mechanisms. In Chapters 18 through 20, you'll learn about the most important security techniques at your disposal.

## Important Database Terms

As you learn about programming and SQL Server, you will encounter many terms related to SQL Server implementation and management. It is important that you understand the definitions for these terms as used in this book. Many terms have more than one definition, and it is important that you understand the meaning poured into the words in context. Some of these terms are basic, and some are more complex, but you will see them appearing again and again throughout this book and as you read articles, white papers, and websites related to the work of a DBA. The following list will define these common terms used in the world of databases and specifically SQL Server:

**Table/Record Set/Relation** In relational database design, a table is not something at which you sit down to eat. Rather, a *table* is a container for data describing a particular entity. Tables are sometimes called record sets, but the term *record set* usually references a result set acquired by a SELECT statement that may include all or a portion of the table data. The formal name for a table is a *relation*. All of the entries in the table are related in that they describe the same kind of thing. For example, a table used to track LCD projectors describes projectors. All entries are related to projectors.

**Column/Field/Domain** To describe the entity represented in a table, you must store information about that entity's properties or attributes. This information is stored in *columns* or *fields* depending on the database system you're using. SQL Server calls them columns, and Microsoft Access calls them fields, but they are the same thing. For example, the LCD Projectors table would include columns such as Brand, Model, SerialNum, and Lumens. Note that these properties all describe the projector. The term *domain* is used to reference a type of property or attribute that may be used throughout the database. For example, you may consider City, LastName, and eMail to be domains. To ensure domain integrity, you would enforce the same data type, constraints, and data entry rules throughout the database for these domains.

**Record/Row/Tuple** A collection of columns describing or documenting a specific instance of an entity is called a *record*. Stated simply, one entry for a specific unit in the LCD Projectors table is a record. Records are also called rows in many database systems and by many DBAs. The formal term for a record is a *tuple* (usually pronounced "too-pel," but some argue for "tyoo-pel").



**Index** An *index* is a collection of data and reference information used to locate records more quickly in a table. SQL Server supports two primary index types: clustered and non-clustered. Clustered indexes are similar to a dictionary or telephone book. Nonclustered indexes are similar to those found at the back of a book. For now, it's enough to know that they can be used to increase database performance and that they can equally decrease database performance when used improperly. You will learn about them in detail in Chapter 11, "Indexes and Views."

**View** One of the most over-explained objects in databases is the view. Here's the simple definition: a *view* is a stored SQL `SELECT` statement. That's really all it is. Views are used to make data access simple, to abstract security management, and to improve the performance of some operations. The most common use of views is the simplification of data access.

**SQL** *SQL* is the database communications language managed by the ANSI organization. It is a vendor-neutral standard language that is supported at some level by nearly every database product on the planet. SQL Server implements a customized version of SQL called Transact-SQL, or T-SQL for short.

**Stored Procedure** When you want to process logical operations at the server instead of the client, stored procedures can be used. A *stored procedure* is either a collection of T-SQL statements or a compiled .NET stored procedure in SQL Server 2008 and newer. Earlier versions of SQL Server supported and recommended extended stored procedures, which were really just DLLs called by the SQL Server. Stored procedures are used to centralize business rules or logic, to abstract security management, or to improve performance. Other reasons exist, but these are the three big motivators.

**Trigger** A trigger is like a dynamic stored procedure. A *trigger* is a group of T-SQL statements that is executed automatically when specified events occur. For example, you may want to launch a special procedure anytime someone attempts to execute a `DROP TABLE` (delete a table) statement. The trigger could either back up the table before deleting it or simply refuse to delete the table.

**Concurrency** *Concurrency* is defined as acting together. In the database world, either a system supports multiple concurrent users or it does not. *Concurrency* is a single word that says a database system supports multiple users reading and writing data without the loss of data integrity.

**DBA** A *DBA* is a database administrator. A DBA is the person who installs the routers and switches, implements the network operating system, builds the user databases, configures the client computers, programs the telephone system, troubleshoots production and security problems, and, oh yeah, works with databases on occasion. But seriously, you live in a new world of IT. Today, most IT professionals must wear multiple hats. This reality means that DBAs usually have to know about the database server service, the server operating system, and even a bit about the network infrastructure across which users communicate with the database system. It's a brave new world.



Remember, these are the basic terms that will appear throughout your experiences with databases, regardless of the database system with which you are working. Be sure you know what these terms mean. You'll learn about many more database terms as you read the rest of this book.

## Database Servers and Applications

Now that you've learned the fundamental concepts of a database, it's time to investigate server-side databases and database applications in a bit more detail. Let's immediately clear up one thing:

The database is not the database server, and the database server is not the database.

It's not uncommon for a DBA to say, "I have to restart the SQL Server database." What he really means is that he needs to restart the SQL Server service, which manages access to the database. The database is separate from the database management system. SQL Server is the database management system. Databases may be detached from one SQL Server instance and then attached to another. In fact, you can attach Excel spreadsheets, Access databases, and virtually any data source that you can connect to with Open Database Connectivity (ODBC) to a SQL Server as a linked server object. Once the link is made, the SQL Server service can manage access to that data source (via the ODBC or other connection type) for your users. ODBC is a standard database access method used by many database management systems.

To help you better understand the relationship that applications have with a database server, the following section will explain the three kinds of database applications:

- Localized
- Client-server (single tier)
- N-tier (multiple client-server relationships)

### Database Application Types

The three primary kinds of applications are localized, client-server, and n-tier applications. Localized applications will not be covered in detail here because our primary focus is on running SQL "servers" and not SQL Server on the clients. However, you should know that a localized application usually talks to a local install of SQL Server using a protocol called Shared Memory. The name says it all: the local application talks to the local SQL Server installation (usually SQL Server Express) without using the network interface card.

#### Client-Server (Single Tier)

Client-server implementations, also called *single tier*, involve a client application communicating directly with the database in most cases. An example of a client-server application is a Microsoft Access frontend that communicates with a SQL Server backend database. The SQL Server database is the server, and Microsoft Access is the client. Technically, an Excel

data import from a SQL Server is a client-server application. Figure 1.5 shows an example of this model.

**FIGURE 1.5** A simple implementation of client-server technology with a client accessing a single server directly

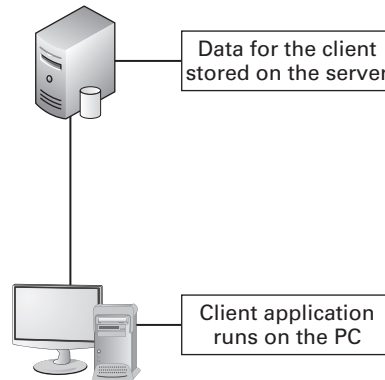


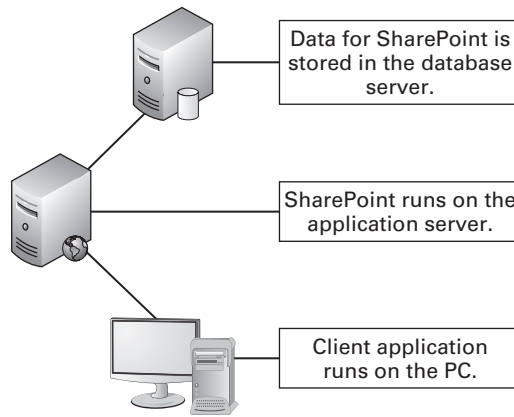
Figure 1.5 shows an application communicating with a SQL Server. Notice that the user interacts with the application as if everything is installed on her local machine. In fact, as long as the network is working and the database server is available, the user will usually feel as if the data is indeed in her computer. Of course, as you add more users—without increasing servers or the single server's capacity—she will likely notice a drop in performance; however, this drop should be minimal as long as the database server is well maintained and upgraded as needed.

## N-Tier (Multiple Client-Server Relationships)

An *n*-tier application is an application that requires multiple levels (tiers) of communication in order to accomplish meaningful work. For example, a SharePoint server farm that includes one server for the database and another server for the website is an *n*-tier application or, more specifically in this case, a two-tier application. The user communicates with the web server (tier 1), and the web server communicates with the database on the user's behalf (tier 2). The *n* in *n*-tier is simply replaced with the number of links in the communication chain.

Figure 1.6 shows the SharePoint implementation visually. You can see the links or tiers in the application. Such an implementation provides several benefits. First, developers can change the database without necessarily rewriting all of the code at the web server. This benefit assumes that a standard data access method was used between the web server and the database. Second, the developers can completely change the look and feel of the application without changing any of the data. In three-, four-, and more-tier implementations, the solution is even more componentized, and the result is greater flexibility in the solution over time.

**FIGURE 1.6** An n-tier application using a SharePoint server to access a backend database server



Finally, n-tier applications are easier to scale. Single-tier applications are notoriously difficult to scale. Everything is resting on a single server. If the performance of the database becomes too slow, you are very limited in what you can do. With an n-tier application, you can distribute the data across several servers on the backend and absolutely nothing changes from the users' perspectives. No wonder developers love to use this model. It's not without its faults, but it certainly has its benefits.

## SQL Server's Role

You are finally ready to explore how SQL Server fits into all of this discussion of database technologies. To help you understand the roles SQL Server can play in your organization, this section will begin by explaining the product's major new features and its evolution. First, you'll explore the new features introduced in SQL Server 2012. Next, you'll look at the features SQL Server 2008 introduced (and that, of course, are still in SQL Server 2012). The final new features section focuses on those features introduced in SQL Server 2005. This coverage of the two previous editions is very important. Some organizations are moving directly from SQL Server 2000 and skipping SQL Server 2005 and 2008 altogether. That decision is certainly acceptable as long as you can ensure compatibility with your applications. But compatibility was also an important consideration for those who upgraded from SQL Server 7.0 or 2000 to SQL Server 2005 a few years ago. This section provides you with a quick overview of the evolution of SQL Server for a little more than the last decade.

Finally, this section covers the roles SQL Server can play based on these new features. You'll learn about enterprise databases, departmental databases, reporting servers, ETL servers, analysis servers, and more. Let's jump into these exciting new features.

## New Features Introduced in SQL Server 2012

This section introduces the newest features found only in SQL Server 2012 and not in previous editions. To make the coverage simpler, the features are separated into two categories: management features and development. This book is primarily focused on management and administration of SQL Server 2012, so more information about the management enhancement features is provided; however, it is practically impossible to manage a SQL Server without doing some development or without understanding the components, languages, and processes used by developers. For this reason, the development features will be covered here and throughout the rest of the book, although less exhaustively.

### New Management Features

SQL Server 2012 introduces several features that enhance the manageability of the database engine (the core of SQL Server). These features and enhancements include the following:

- SQL Server Management Studio
- Contained databases
- Windows PowerShell
- New and improved dynamic management views (DMVs)

### SQL Server Management Studio

The first and most significant change to SQL Server Management Studio (SSMS) is that it is now built in the Visual Studio environment. This change makes the SSMS interface similar to other Visual Studio–based solutions, like the traditional Business Intelligence Development Studio (BIDS) used in previous versions of SQL Server. It also adds the powerful capabilities of the Visual Studio IDE, including the following:

- New Visual Studio default keyboard shortcuts
- Transact-SQL IntelliSense enhancements such as breakpoints, code snippets, and templates
- Watching Transact-SQL expressions during execution
- Quick Info pop-ups when you move the mouse cursor over a Transact-SQL identifier

In addition to this overall change to SSMS, the new database engine Query Editor adds many new debugging features for SQL code. This enhancement may seem more appropriate for the database developer than the database administrator at first glance; however, experienced administrators spend a tremendous amount of time working in SQL code with performance analysis and troubleshooting endeavors. The new debugging features benefit you, too.

SSMS also introduces enhanced database restore capabilities. The Restore feature is more efficient, and this improves restore times. A new visual timeline is available for point-in-time restores, making them much easier. Select corrupt pages can also be restored with the new Page Restore dialog using a GUI interface.

## Contained Databases

Contained databases allow users to exist in a user database without having associations with logins in the instance of SQL Server. You create a user in the database without creating a login for the SQL Server itself, which is where the phrase *contained database* comes from. This can make database moves from one instance to another much easier. SQL Server 2012 actually implemented partially contained databases because some items still depend on the instance. These items can be viewed using the `sys.dm_db_uncontained_entities` and `sys.sql_modules` views. The partially contained databases allow for improved database movement, improved failover when using the new AlwaysOn feature, and improved development processes for testing and troubleshooting.

## Windows PowerShell

Windows PowerShell 2.0 is a prerequisite for installing SQL Server 2012. PowerShell modules are used to load the SQL Server components into the PowerShell environment. The `sqlps` module is imported into PowerShell to load the SQL Server snap-ins. Two new cmdlets are `backup-sqldatabase` and `restore-sqldatabase`.

## New and Improved Dynamic Management Views

SQL Server provides DMVs and dynamic management functions (DMFs) for insight into the SQL Server operations. The following system views have been added or changed in SQL Server 2012:

- `sys.dm_exec_query_stats` (modified)
- `sys.dm_os_colume_stats` (added)
- `sys.dm_os_windows_info` (added)
- `sys.dm_server_memory_dumps` (added)
- `sys.dm_server_services` (added)
- `sys.dm_server_revistry` (added)

## New Development Features

The database engine of SQL Server 2012 offers the following development or programmability features:

- FileTables
- Statistical semantic search
- Full-text search enhancements
- New functions
- SQL Server Express LocalDB

### FileTables

The FileTable is an enhancement or extension to the filestream technology introduced in earlier versions of SQL Server. The FileTable feature allows you to store files and documents in SQL Server tables and access them as if they were stored in the Windows file

system. The FileTable is also called a *table of files*. Each row in a FileTable represents a file or a directory and contains a file\_id, path\_locator, and parent\_path\_locator. Traditional file attributes are also stored, such as date modified and last access time.



Files can be bulk-loaded into FileTables to convert traditional file storage (such as image asset libraries) into SQL Server data tables. Use the BCP, BULK INSERT, and INSERT INTO commands to accomplish this.

## Statistical Semantic Search

Full-text search has been in SQL Server for several versions. Statistical semantic search adds the element of meaning to full-text search. Where full-text searching allows you to locate data based on included words, semantic search allows you to locate data based on meaning. The SemanticSimilarityTable function can be used to locate documents or data similar to a specified document. Statistical semantic search is most commonly used with documents stored in the SQL Server. These documents are likely to be stored in FileTables moving forward.

## Full-Text Search Enhancements

In addition to semantic searches, full-text indexes now support property-scoped searching for documents stored in FileTables. For example, you can search by author, title, and other document properties. Only documents with appropriate filters installed can be searched in this way.

You can also now use a NEAR statement within a CONTAINS or CONTAINSTABLE function to perform proximity searches. You can determine that multiple words must exist in a data column or document and that they must exist in a specific order. You can also specify the number of words within which the defined words must occur. For example, NEAR( (Tom, Carpenter), 2, TRUE) would require that Tom is before Carpenter (this is what the keyword TRUE means) and that it is within two words of Carpenter. Therefore, Tom Dale Carpenter would match.

## New Commands and Functions

Fourteen new commands and functions were introduced to Transact-SQL in SQL Server 2012:

**PARSE** The PARSE command is used to translate an input value to a new data type. For example, you can translate or convert a string that contains a date into an actual datetime2 data type. It is used only with string source values. The PARSE command uses the following syntax:

```
PARSE ( string_value AS data_type [ USING culture ] )
```

**TRY\_PARSE** TRY\_PARSE is like PARSE except that it can return a NULL value if the command fails. The TRY\_PARSE command uses the following syntax:

```
TRY_PARSE ( string_value AS data_type [ USING culture ] )
```

**TRY\_CONVERT** The TRY\_CONVERT command is similar to the CONVERT command except that it can return a NULL value if the command fails. The TRY\_CONVERT command uses the following syntax:

```
TRY_CONVERT ( data_type [ ( length ) ], expression [, style ] )
```

**DATEFROMPARTS** The DATEFROMPARTS function takes individual values and converts them to a single date value. For example, it can take 2000, 10, and 24 as separate values and convert them to 200–10–24 as an actual date value. The following syntax is used:

```
DATEFROMPARTS ( year, month, day )
```

**DATETIMEFROMPARTS** This function is the same as DATEFROMPARTS except that it returns the data as a datetime data type and receives more inputs. The following syntax is used:

```
DATETIMEFROMPARTS ( year, month, day, hour, minute, seconds, milliseconds )
```

**DATETIME2FROMPARTS** This function is the same as DATEFROMPARTS except that it returns the data as a datetime2 data type and receives more inputs. The following syntax is used:

```
DATETIME2FROMPARTS ( year, month, day, hour, minute, seconds,  
                          fractions, precision )
```

**DATETIMEOFFSETFROMPARTS** This function is the same as DATEFROMPARTS except it returns the data as a datetimeoffset data type and receives more inputs. The following syntax is used:

```
DATETIMEOFFSETFROMPARTS ( year, month, day, hour, minute, seconds,  
                              fractions, hour_offset, minute_offset, precision )
```

**SMALLDATETIMEFROMPARTS** This function is the same as DATEFROMPARTS except it returns the data as a smalldatetime data type and receives more inputs. The following syntax is used:

```
SMALLDATETIMEFROMPARTS ( year, month, day, hour, minute )
```

**TIMEFROMPARTS** The TIMEFROMPARTS function returns a time value and data type when provided the hour, minute, seconds, fractions, and precision as input. The following syntax is used:

```
TIMEFROMPARTS ( hour, minute, seconds, fractions, precision )
```

**EOMONTH** The EOMONTH function is used to determine the end-of-month date for the month in which a provided date exists. For example, given the input of 2/12/2016, it would return 29, because 2016 is a leap year. The following syntax is used:

```
EOMONTH ( start_date [, month_to_add ] )
```

**CHOOSE** The CHOOSE function returns an item in a list based on the provided index value. For example, in the list horse, cow, pig, the value of 2 for the index returns cow. The following syntax is used:

```
CHOOSE ( index, val_1, val_2 [, val_n ] )
```

**IIF** The IIF function returns one of two values depending on whether the input expression equates to true or false. It is a short way for writing a CASE statement, which was available in previous versions of SQL Server. The following syntax is used:



```
IIF ( boolean_expression, true_value, false_value )
```

**CONCAT** The CONCAT function takes two or more string values and combines them into a single string. You simply provide it with the string values you want to concatenate. For example, CONCAT ('Tom', ' ', 'Carpenter') would return Tom Carpenter. The following syntax is used:

```
CONCAT ( string_value1, string_value2 [, string_valueN ] )
```

**FORMAT** The final new function in SQL Server 2012 is the FORMAT function. It can be used to convert data from one format to another, for example from U.S.-formatted dates to European-formatted dates. The following syntax is used:

```
FORMAT ( value, format [, culture ] )
```

## SQL Server Express

The SQL Server Express LocalDB is a lightweight version of SQL Server Express. It includes the same programmability features, but it runs in user mode and is installed quickly with no configuration. It is used for those projects where the developers desire to have a local database similar to a Microsoft Access database but in the format of SQL Server databases. The SQL Server Express LocalDB edition is managed using a utility called SqlLocalDB.exe. The database files (with an .mdf extension) can be attached to full SQL Server installations later, if desired.

## Features Introduced in SQL Server 2008

If you are upgrading from SQL Server 2005 to SQL Server 2012, the features covered in this section will be of great interest to you. The features were either new or greatly enhanced in SQL Server 2008. Like the new 2012 features, they are grouped here into management and development features.

All of the features mentioned here will be discussed in more detail in later chapters. The intent here is to help you understand the roles SQL Server 2012 can play in an organization based on the enterprise-class feature set it provides.

## Management Features Added in SQL Server 2008

The management features added in SQL Server 2008 were among the most talked-about new features. From policy-based management to the Resource Governor, SQL Server 2008 definitely provided the major capabilities needed for large multiserver enterprise implementations. These features included the following:

- Policy-based management
- Configuration servers
- The Resource Governor
- Transparent data encryption
- Performance data collectors

## Policy-Based Management

Policy-based management (PBM) allows for the configuration of SQL Server services through policies. This functionality means that DBAs can configure pools of SQL Servers together rather than having to configure each server individually. Of course, PBM is most useful for environments with 10 or more servers, but it may provide benefits to smaller organizations as well.

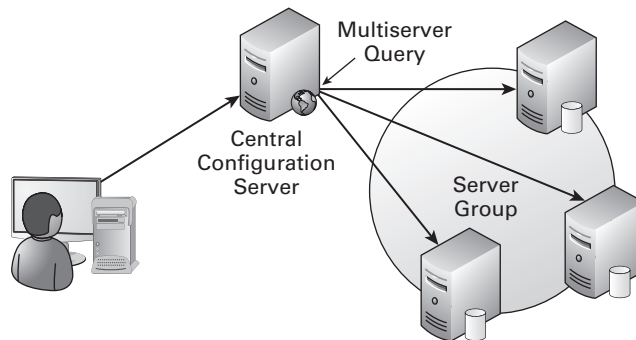
To use PBM, policies are grouped into facets that are configured as conditions and applied to targets. For example, the Surface Area facet can be used to disable the `xp_cmdshell` extended system stored procedure, and then this policy can be applied to every server or a selection of servers. A *policy*, in the PBM world, is defined as a condition enforced on one or more targets.

## Configuration Servers

Configuration servers are special SQL Servers that are used to centrally configure other servers. Any SQL Server instance can be converted to a configuration server. Once the configuration server is implemented, two primary tasks can be performed: centralized management of PBM and multiserver queries. The centralized management of PBM with a configuration server causes PBM to provide functionality similar to group policies in a Windows domain. From one central server, you can configure all of your SQL Servers based on configuration groups.

The multiserver query feature is exceptional. With this feature, you can execute a query from the configuration server to be run against all of the servers in a configuration group. For example, if you need to create a table in a database that has been replicated or simply duplicated to seven different servers, a configuration server would allow you to execute the code to create that table on all seven servers at the same time. Figure 1.7 illustrates the concept of the multiserver query.

**FIGURE 1.7** An example of multiserver queries with the user querying one server that queries three other servers in turn



## The Resource Governor

The Resource Governor is used to impose limits on workloads based on the user requesting the work, the application requesting the work, or the database against which the work is performed. Workloads can be assigned priorities so that, for example, a single user's actions do not prevent other users from completing necessary work. With previous versions of SQL Server, DBAs could use the Windows System Resource Manager (WSRM) on Enterprise editions of Windows Server to perform similar operations. Now, the feature is built into SQL Server and has more granular control over the resources in relation to SQL Server.

## Transparent Data Encryption

SQL Server 2005 first introduced encryption into the SQL Server Database Engine. The only problem was that existing applications could not use it because the application had to call both the encrypting and decrypting routines. SQL Server 2008 solved this problem with transparent data encryption (TDE). To use TDE, you must still generate the appropriate encryption keys and enable encryption for the database; however, these steps are taken at the server by the DBA, and the developers will not have to change anything in their applications. The encryption and decryption happen automatically, and the data is accessed in the same way as unencrypted data.

The TDE feature provides storage encryption. The data is decrypted by the SQL Server and then transferred to the client. Do not confuse this with transit encryption or communications encryption. To encrypt the communications between the SQL Server and the client, you will still usually need to implement IPsec or a VPN protocol.

## Performance Data Collectors

The next major management feature is the performance data collectors. *Data collectors* are simply the tools used to collect performance information about your server. Historical performance data can be automatically stored in a management data warehouse, allowing the DBA to review historical performance data at any time. The process of collecting the data is as follows:

1. SQL Server Agent schedules and launches the Data Collector component.
2. The Data Collector component launches the needed SSIS package.
3. The SSIS package collects the performance data and stores it in the management data warehouse.

As you can see, Microsoft has taken advantage of existing technologies from earlier versions of SQL Server to build the Data Collector engine. You could have accomplished something similar in earlier versions of SQL Server by collecting performance data using the System Monitor and configuring it to automatically store the data in a SQL Server table; however, the built-in tools to accomplish this are much more integrated in SQL Server 2008 and newer versions.

## Development Features Added in SQL Server 2008

The development enhancements in SQL Server 2008 were also important. As a DBA, you may never write a single line of code that gets compiled into an application, or you may

be a “programming DBA.” Many times programmers/developers must manage their own SQL Servers. However, even if you do not write the code, it is useful to understand its basic structure so that you can better troubleshoot problems in SQL Server–based applications. The development features new to SQL Server 2008 are included here.

**Developer Tool Improvements**

The SQL Server 2008 Query Editor, which was still built into SQL Server Management Studio, supported IntelliSense capabilities. This meant that the Query Editor could complete entire words for you representing functions, keywords, variables, and more. If you found yourself testing scripts in the Query Editor, this feature proved priceless. Additionally, an error list feature similar to that in Visual Studio had been incorporated into the Query Editor. When the editor detected errors, they would appear (see Figure 1.8), and you could click the instance to see the error and find help to repair it. The Query Editor is even better now in SQL Server 2012 because the entire SQL Server Management Studio has been reworked to operate within Visual Studio’s development environment.

**FIGURE 1.8** Errors listed in the Error List dialog

Error List						
		Description	File	Line	Column	Project
❗	1	Could not locate entry in sysdatabases for database 'AdventureWork'. No entry found with that name. Make sure that the name is entered correctly.	SQLQuery1.sql	1	5	Miscellaneous
❗	2	Invalid object name 'AdventureWorks.Production.Products'.	SQLQuery1.sql	4	15	Miscellaneous
❗	3	Incorrect syntax near 'End Of File'.	SQLQuery1.sql	8	1	Miscellaneous

**Change Data Capture**

Developers have been writing triggers and stored procedures for years in order to capture data changes. When a user modifies a record, for example, the trigger fires and saves to a History table a copy of what the data looked like before the modification. SQL Server 2008 Enterprise and Developer editions support a feature called Change Data Capture. This feature is still supported in SQL Server 2012. It is easily enabled for the entire database or a specific set of tables. Once enabled, historical states of the data can be queried.

**Data Type Changes**

SQL Server 2008 provided several data type enhancements and changes. The date and time data types were upgraded with a new datetime2 data type. The datetime2 data type supports a broader range of dates and greater accuracy. The new hierarchyid data type is used to reference the position of items in a hierarchy, such as an employee’s position in an organizational chart. Finally, the new filestream data type allows data to be stored on the NTFS files system outside of the database data files but managed by SQL Server like other data types.

## **New Report Designer**

SQL Server 2005 introduced the Report Builder, but SQL Server 2008 took this to the next level with the Report Designer. The Report Designer took on the look and feel of Microsoft Office 2007, including the Ribbon bar. Charting was enhanced, and there was a new *tablix* data region that looked oddly similar to a pivot table, although that name never seems to appear in Microsoft's documentation related to the tablix.

## **Sparse Columns**

When Windows 2000 was released in 1999, Microsoft implemented sparse files in the NTFS file system. These files consumed 0 literal bytes on the drive, although they appeared to consume from 1 byte to terabytes of space. Now, sparse columns have been added to the feature set of SQL Server 2008 and are still supported in SQL Server 2012. Sparse columns are most useful for columns that may have excessive records with NULL values. When the value is NULL, the column will consume 0 bytes in the data pages. Sparse columns can help you fit a few more bytes of data into that 8,060-byte limit imposed by SQL Server.

## **LINQ to SQL Provider**

Microsoft developed the Language Integrated Query (LINQ) feature for .NET development some time ago; however, there was no direct support for it in SQL Server 2005 and older. SQL Server 2008 implemented a LINQ-to-SQL provider, which meant that developers could write queries in standard .NET code (instead of embedded SQL variables), and SQL Server would take care of translating the request into T-SQL that the server could process.

# **Features Introduced in SQL Server 2005**

If you are upgrading from SQL Server 2000 to SQL Server 2012, you will get all of the new features covered in the previous sections, but you will also acquire all of the features that were first introduced in SQL Server 2005. Moving from SQL Server 2005 to SQL Server 2012 is like climbing a three- or four-rung stepladder; significant changes have occurred, but they are not massive. However, moving from SQL Server 2000 to 2012 is like climbing 35 to 40 rungs on a ladder. As you will see, SQL Server 2005 introduced drastically different administration tools, an entirely new way of thinking about custom stored procedures, and the ability to mirror databases, just to name a few changes. Like SQL Server 2012's and 2008's features, the features can be divided into management and development enhancements.

## **Management Features Added in SQL Server 2005**

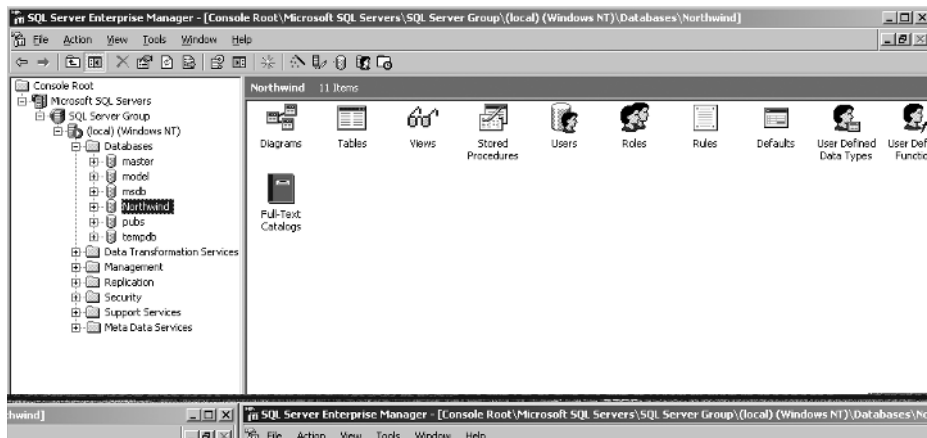
The management features introduced in SQL Server 2005 were many. This section focuses on a few key features:

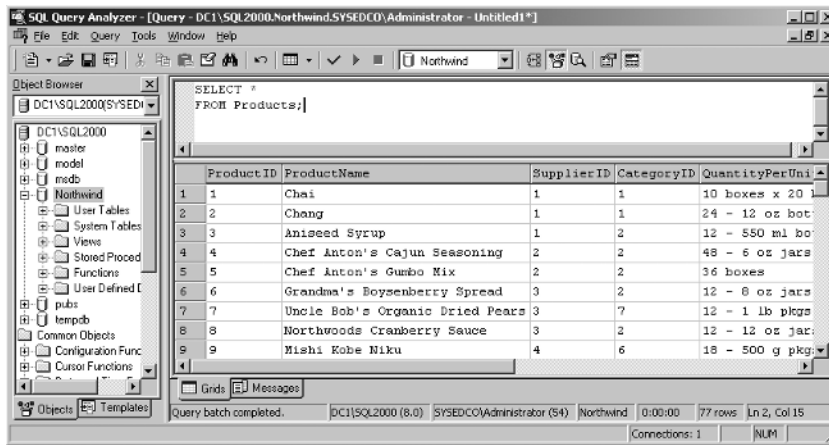
- New management tools
- Database Mail
- Dedicated administrator connection
- SQL Server Integration Services
- Database snapshots
- Database mirroring
- Failover clustering for Analysis Services
- Online indexing
- Security enhancements
- Reporting services

### New Management Tools

The new management tools introduced in SQL Server 2005 and enhanced in SQL Server 2008 were very welcome additions. As described earlier, they have been further enhanced in SQL Server 2012. The SQL Server 2005 tools did not simply upgrade what was available in SQL Server 2000; they completely replaced them. The Enterprise Manager (shown in Figure 1.9) and Query Analyzer (shown in Figure 1.10) were both replaced with SQL Server Management Studio.

**FIGURE 1.9** The SQL Server 2000 Enterprise Manager used in earlier versions of SQL Server



**FIGURE 1.10** The Query Analyzer from SQL Server 2000

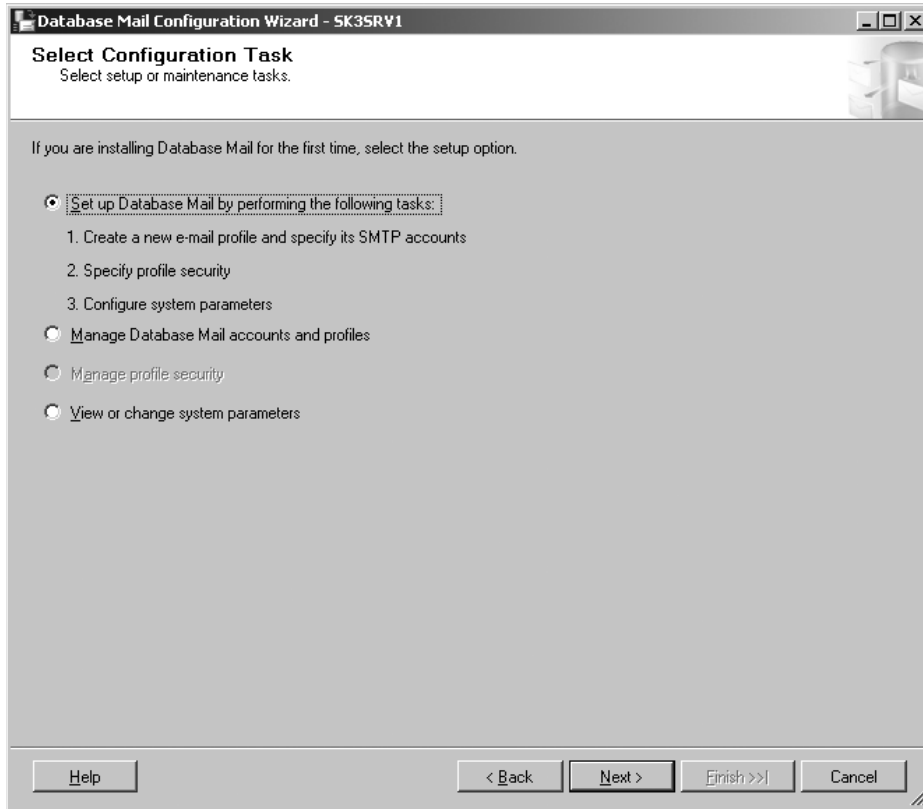
SSMS incorporated a query window into the management environment. Now, instead of switching between the Enterprise Manager and the Query Analyzer during testing and optimization efforts, you can use one tool to get the job done.

Of course, everything is not perfect in the new management tools—at least for those of us who used the earlier tools extensively. One example is the Object Browser in the Query Analyzer that shipped with SQL Server 2000. The Object Browser allowed you to easily browse through T-SQL functions to locate the one you needed by category. SSMS does not support this same Object Browser; although it does offer dynamic help, this feature requires that you remember the function name. You can always create a favorite link in Books Online (the SQL Server help system) to the T-SQL functions page, but you might miss the Object Browser just the same.

## Database Mail

In versions of SQL Server before SQL Server 2005, sending mail from the server was an arduous task. Oddly, you had to install Outlook on the server (although standard documentation suggested only the need to install “a MAPI client”). This was not because SQL Server used Outlook to send email but because you had to install Outlook to get a needed Control Panel applet for the configuration of SMTP accounts. By putting Outlook on the server, you were creating yet one more component to update and secure. The old way can be summarized by saying it was kludgy at best.

SQL Server 2005 solved the problem by introducing Database Mail, which provides direct sending of SMTP mail messages. This component can be used to send email from your jobs or applications. The component that actually does the sending runs outside of the memory space of SQL Server, and it simply checks in with the SQL Server periodically to see whether messages are waiting to be sent. The Service Broker component is used to queue the mail messages. Configuring Database Mail is as simple as stepping through a wizard (see Figure 1.11) and entering the parameters for your available SMTP servers. For your reference, the executable that sends the mail is DatabaseMail90.exe in SQL Server 2005. The new Database Mail solution is both more efficient and less annoying than the older SQL Mail alternative.

**FIGURE 1.11** The SQL Server 2005 Database Mail Configuration Wizard

### Dedicated Administrator Connection

The dedicated administrator connection (DAC) allows you to connect to a SQL Server system that is not responding to normal connections. The DAC was made available through the new SQLCMD command-prompt tool and could be initiated only by the members of the sysadmin server role. Once connected, you could execute standard diagnostic commands, such as basic DBCC commands and potentially data-query commands.

SQL Server listens for DAC connections on a different TCP port than used for normal connections. The default instance of SQL Server usually listens on TCP port 1433, but the DAC listens on TCP port 1434 by default.

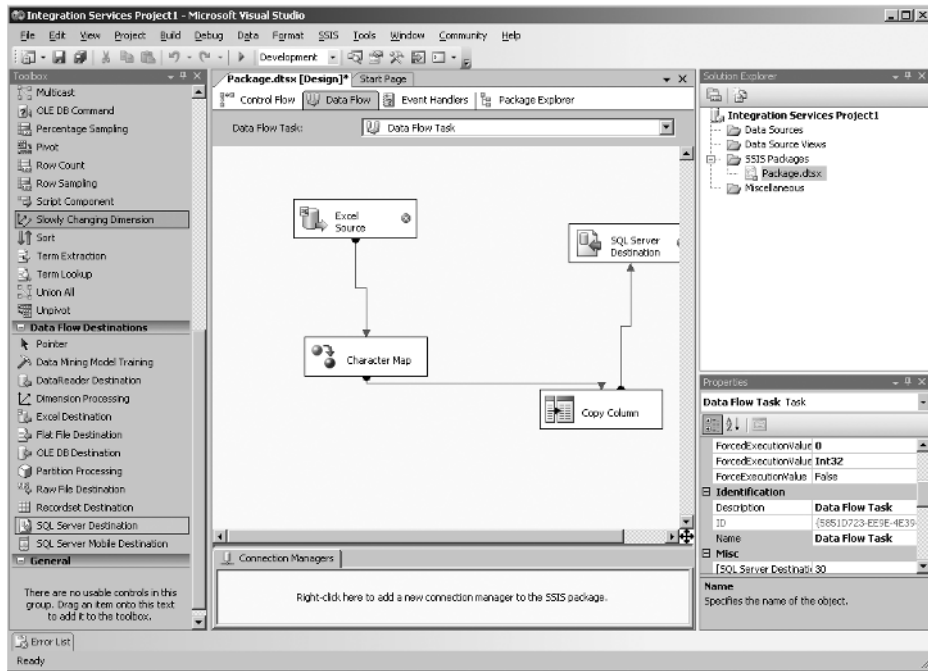
### SQL Server Integration Services

Data Transformation Services (DTS) provided an extraction, transformation, and loading (ETL) tool in SQL Server 2000. The tool was simple and to the point. This simplicity, however, also meant that it lacked the power for more advanced ETL procedures. With SQL Server Integration Services (SSIS), you have one of the most powerful ETL tool sets



in existence and certainly one of the best bundled ETL tool sets available with any database management system. With SSIS, you can do everything you did in DTS (you can even run an old DTS package if necessary) and a whole lot more. Figure 1.12 shows the SSIS interface.

**FIGURE 1.12** SQL Server Integration Services showing a sample project



## Database Snapshots

Database *snapshots* allow you to capture a point-in-time view of your entire database. The snapshots are created using sparse files on NTFS volumes, so they are created very quickly. In fact, the initial snapshot file contains 0 bytes of actual data. Before information changes in the database, the old information is copied into the snapshot file. This implementation allows several snapshots to exist at the same time without a tremendous burden on the server. Snapshots are useful for several practical functions, including these:

- Viewing and reporting on data as it existed at a point in the past
- Selectively restoring data to a specific point in time, such as restoring the Customers table to the state it was in before a user accidentally set everyone's email address to the same value
- Reverting the entire database to the state in the snapshot

## Database Mirroring

If you want to configure a warm or hot standby server, database mirroring is a potential solution. Standby servers are covered in detail in Chapter 23, “Database Mirrors and Snapshots.” Database mirroring allows you to mirror one database or several databases from one SQL Server onto another. The primary server will receive all changes, and those changes are then immediately transferred to the mirror database server transactionally (based on changes and not simply accesses). The latency is very low with database mirroring, which means that in the event of a failure, very little data should be lost.

Database mirroring can be implemented in two ways: warm standby and hot standby. In warm standby implementations, the mirror database cannot automatically be promoted to become the primary database. Some DBA intervention will be required. In hot standby implementations, the mirror database can automatically be promoted to become the primary database. However, to implement a hot standby mirroring solution, a third server is required. It is known as the *witness server* and ensures that the mirror server does not promote itself unless the primary server is really down.

## Failover Clustering for Analysis Services

*Failover clustering* provides fault tolerance for SQL Server. Earlier versions of SQL Server supported failover clustering for the database engine, but it was not supported for the other services. With the release of SQL Server 2005, failover clustering was supported for SQL Server Analysis Services (SSAS) as well. SSAS is used for data warehousing, data analysis, and various data management and storage operations. Failover clustering allows one server to act as the primary server and another server to automatically take over should the primary server fail. In addition to SSAS, failover clustering is also supported for Notification Services and Replication servers.

## Online Indexing

SQL Server 2005 first introduced online indexing. The concept is simple: you can create an index on a table while users are accessing that table for reads and writes. This feature means you do not have to wait for a nonbusy window to create indexes. In previous versions of SQL Server, it was common to schedule index creation during downtimes, such as 12 a.m. to 5 a.m. The only problem was that it was very difficult to find an inactive time window in 24/7 shops.

Consultants often spend a large portion of their consulting time optimizing existing databases. One key optimization strategy is the creation of indexes (and the deletion of unneeded indexes). The ability to create indexes on the fly without the need for downtime is priceless—not just to consultants but to their clients as well. They no longer have to carefully schedule consulting windows or prepare a mirror server that matches their production server for performance testing. A performance analysis can usually be scheduled to run on the server while users are accessing the data, and then that information can be analyzed offline. Next, proper indexes can be created, and the performance analysis can be run again to determine whether the desired outcome was accomplished. You'll learn all about this process in Chapter 15, “Performance Monitoring and Tuning.”

## Security Enhancements

Many security enhancements were introduced in SQL Server 2005, including these:

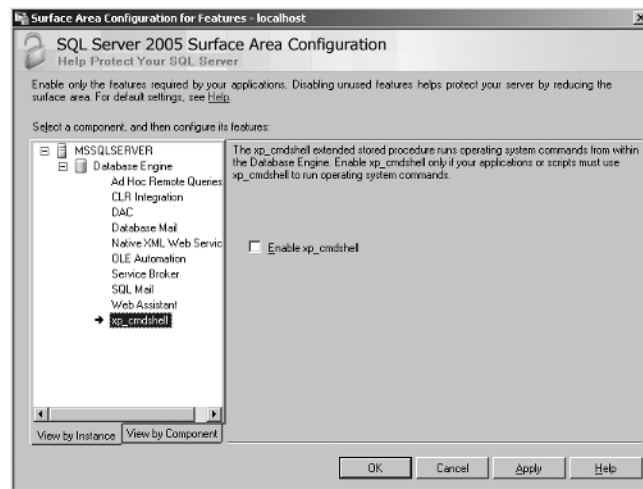
**Storage Encryption** Data can be encrypted in the database. SQL Server 2005 requires that the application be aware of the encryption and implement code to both encrypt and decrypt the data. Storage encryption helps protect your data if the backup media is stolen or an attacker otherwise steals the entire database file.

**Password Policies for SQL Logins** SQL logins have been a negative issue for SQL Server for years. The authentication process is not the most secure, and it is hampered even more by the use of weak passwords. Password policy association allows you to require that users' passwords meet the policy requirements established in the Windows domain or on the local server. Users can be required to select passwords greater than a specified length and requiring complexity in character types.

**Separated Owners and Schemas** Before SQL Server 2005, if you made a user the owner of a table, that table was placed in the user's schema. You would end up with table names such as fred.sales and jose.marketing. Needless to say, this structure was less than ideal. Because of this functionality, most DBAs chose to use the dbo schema for everything and, therefore, ensured that the dbo owned everything. With the release of SQL Server 2005, schemas became usable—in a practical way—for the first time. A user can own a table, and that table can remain in the assigned schema.

**Surface Area Configuration Tool** The Surface Area Configuration tool is used to lock down a SQL Server installation. The tool allows you to control the SQL Server services and enable or disable features as needed. Figure 1.13 shows the Surface Area Configuration for Features interface.

**FIGURE 1.13** The Surface Area Configuration for Features dialog box



## Reporting Services

Reporting Services was first introduced as an add-on for SQL Server 2000. The product was not supplied on the SQL Server 2000 distribution CDs but was an after-the-fact download or could be purchased on a CD. As the name implies, Reporting Services provides reporting features for your database deployments. Be careful not to be fooled by the fact that Reporting Services “comes with” SQL Server 2005 and newer. You must still license the product separately from your SQL Server deployment. Each Reporting Services server requires SQL Server licenses—either per processor or a server license, plus client access license (CAL) for each user who accesses the reports.

SQL Server 2005's implementation of Reporting Services added a new reporting tool. The Report Builder application provides simplified report creation for users. Because the Report Builder uses concepts familiar to those who use Microsoft Office 2003 and older, the learning curve is reduced. You can format, preview, print, and publish reports from the Report Builder.

Of course, Reporting Services not only comes with SQL Server 2012 today, but Express Edition with Advanced Services even includes a limited version of Reporting Services.

## New Development Features in SQL Server 2005

The enhancements for those developing applications for SQL Server were also important in the release of SQL Server 2005. They are briefly listed here and are covered in more detail in the appropriate locations throughout the book.

### .NET Integration

The integration of .NET and CLR capabilities into SQL Server means that developers can code stored procedures, triggers, and functions in the many modern languages supported by .NET. Additionally, the integration of .NET into the memory space of SQL Server improves performance and provides improved security. Because .NET integration is more of a programming feature than an administration feature, it is not covered in detail in this book.

### Transact-SQL Changes

Every version of SQL Server has introduced changes to the T-SQL language. Sometimes these changes add support for new commands and capabilities, and sometimes they remove support for older commands that are no longer needed or for which Microsoft simply chose to end support. SQL Server 2005 made improvements in the area of error handling, support for new database engine features (such as database snapshots), and recursive query capabilities. Chapter 5 covers T-SQL in depth.

### Enhanced Development Tools

The development tools that shipped with SQL Server 2005 were a huge leap over what was available in SQL Server 2000. Most developers used third-party tools or extra Microsoft tools to get the job done. With SQL Server 2005, many developers found that the built-in tools provided sufficient capabilities to accomplish their objectives. The Business Intelligence

Development Studio is a key component among these development tools. Figure 1.13 earlier in the chapter shows the BIDS environment while developing an SSIS solution. The enhancements to the administrative and development tools are covered in Chapter 3.

### **HTTP Endpoints**

SOAP is a communications protocol used to allow applications to interact based on XML data. If you wanted to implement an HTTP endpoint for use with SOAP development in SQL Server 2000, you had to install Internet Information Services (IIS) on the SQL Server. SQL Server 2005 introduced HTTP endpoints. These endpoints can be created without requiring IIS on the server, and they are more secure by default. For example, you can control which stored procedures are permitted for execution through the endpoint. HTTP endpoints are not commonly created by administrators, but they are important for some development projects using older versions of SQL Server before SQL Server 2012. SQL Server 2012 no longer supports XML web services endpoints.

### **XML Support Improvements**

The XML support in SQL Server 2000 was minimal. You could store XML data in columns with a text data type, but your application had to retrieve the data as text and then parse it as XML. No support existed for direct querying of the XML data within the SQL Server. However, you were able to return SELECT statement result sets as XML using the FOR XML clause. SQL Server 2005 takes XML support to the next level. You can store the data in an XML data-typed column, and you can use the XQuery subset of the SQL/T-SQL language in order to retrieve only the XML values you require. Chapter 10, “Data Types and Table Types,” will cover the XML data type in more detail.

### **Service Broker**

You're probably used to databases working in a synchronous manner. The client submits a request to the SQL Server and waits for the server to respond. The server receives the request from the client and processes it as quickly as the current workload allows and then responds to the client. This traditional database communications model does not scale well when a customer submits an order at a website and that order must update an inventory system, a shipment system, a billing system, and a website tracking system. Service Broker provides asynchronous communications without the need for building the entire core communications engine. It provides queuing, queue processing, and other services that allow a more scalable application. Service Broker is primarily an application development architecture and is not covered in detail in this book. To learn more, search: <http://msdn.microsoft.com> for Service Broker SQL Server 2012.

### **Notification Services**

If you've used SQL Server for any amount of time, you've probably used operators and had notifications sent to you when a backup completes or when a job fails. Notification Services provides similar notifications to your users. For example, Notification Services provides the framework to allow a sales representative to subscribe to a change in pricing. Imagine a customer who is ready to buy 50,000 units of item number 2043978 as soon as the price

drops below \$0.70 per unit. The salesperson can configure a notification so that she is notified immediately when the price-drop takes place. Notification Services is no longer in SQL Server 2012. Instead, most developers use Reporting Services to provide the same notification capabilities.

## Core Features of SQL Server

In addition to the special features introduced with SQL Server 2005, 2008, and 2012, core features have existed in the product going all the way back to SQL Server 6.5 and earlier. These important features include the following:

**Support for Concurrent Users** Support for concurrent users is provided using worker threads and connections. Each connection receives its own process ID and can be managed individually (for example, a single connection can be killed). The number of concurrent users that can be supported will be determined by the resources available in the server—for example, as memory, processors, network cards, and hard drives.

**Transactional Processing** Transactional processing ensures that the database maintains consistency. For example, in a banking application, you would not want to allow a transfer from savings to checking to take place in such a way that the money is removed from savings but doesn't make it into checking. Transactional processing ensures that the entire transaction is successful or none of the transaction components are allowed. Transactions can be implicit or explicit, and all changes are treated as transactions.

**Large Database Support** SQL Servers support large databases. SQL Server 2000 allowed database sizes as large as 1,048,516 terabytes, which is equivalent to 1 exabyte in size, which is very large. Of course, finding hardware that can handle a database that size is a different story. Interestingly, according to Microsoft's documentation, the maximum allowed database size was reduced in SQL Server 2005 and 2008 to 524,272 terabytes. This size constraint is still very large at 524 petabytes, so it will not likely be a problem soon. Very few databases exceed 5 terabytes in size today. The database size constraints imposed in SQL Server 2005 and 2008 remain the same in SQL Server 2012.

**Advanced Storage Mechanisms** The storage mechanisms provided by SQL Server allow databases to be stored in single files or multiple files. The database can be spread across multiple files located on multiple storage volumes. By using filegroups, the DBA can control on which file which tables will be placed. The storage mechanisms are far more advanced than those available in a simple database system such as Microsoft Access.

**Large Object Support** Large objects, up to 2GB, can be stored in SQL Server databases. Depending on the application, however, it may be better to store the large objects (LOBs) outside of the database and simply reference them in the database; however, internal storage is supported. You can store large amounts of text (up to 2GB) in the text data type. You can store any binary data in the image data type, which also allows up to 2GB of data to be stored in a single record.

**Replication** Sometimes you need to distribute your data to multiple locations. You may need to provide localized reporting servers at branch offices, or you may need to aggregate new data from several remote offices into a central reporting server. Whatever the motivation behind data distribution, SQL Server offers replication as a solution. SQL Server 6.5 supported basic replication features. With each version since then, more capabilities have been added. For example, SQL Server 2005 added support for replication over the HTTP protocol, and SQL Server 2008 adds a new graphical tool for creating peer-to-peer replication maps and an enhanced version of the Replication Monitor tool.

These core features, and more, have been with SQL Server for well over 10 years, and they continue to evolve and improve. They have a tremendous impact on the roles that SQL Server can play within your organization and can help you decide between it and other database systems—particularly single-user database systems.



Many people have been waiting for SQL Server 2012 to upgrade their SQL Server 2000 installations. If you're one of those people, don't forget about the features that were deprecated in SQL Server 2005 and 2008; they may be gone now in SQL Server 2012. To locate such features, search for *SQL Server 2005 deprecated features* or *SQL Server 2008 deprecated features* in your favorite search engine. Several sites provide information about these removed features.

## SQL Server Roles

Now that you've explored the many features and capabilities of databases in general and SQL Server specifically, let's quickly explore the roles that SQL Servers can play in your organization. This section will cover the following roles:

- Enterprise database servers
- Departmental database servers
- Web database servers
- Reporting servers
- ETL servers
- Analysis and decision support servers
- Intermediary servers
- Standby servers
- Local databases

This list may be longer than you expected, but believe it or not, it's not exhaustive. More roles exist, but these are the most common roles played by SQL Servers.

## Enterprise Database Servers

Enterprise database servers provide data access to the entire organization. These servers usually house enterprise resource planning (ERP) applications, customer resource management (CRM) applications, and other applications that are accessed by individuals from practically every department in the organization. The databases tend to be very large and must usually be distributed across multiple servers.

As an example, consider a SharePoint implementation that is used as a company portal. Each department may have a section on the SharePoint server, but the portal is there for the entire company. With an implementation this large, the SharePoint installation would most likely be a farm-based installation with one or more backend SQL Servers. This implementation qualifies as an enterprise database server implementation.

Common SQL Server features and concepts used on enterprise database servers include the following:

- Failover clustering
- Log shipping or database mirroring
- 64-bit implementations for increased memory support
- Replication
- Encryption
- Third-party backup software
- ETL packages
- Reporting Services
- Windows domain membership
- RAID or SAN data storage

## Departmental Database Servers

Many times an application is needed only for a single department. For example, the engineering group may need a database server for the management of their drawings, or the marketing group may need a database server to track their marketing efforts. While this information could be stored in an enterprise server, if the server will be heavily used by the department, it may be more efficient to use a dedicated server. The hardware may be identical to that which is commonly implemented for enterprise servers. The only difference is usually the number of users who access the server.

Departmental servers tend to utilize features and concepts such as the following:

- Built-in backup software
- Log shipping or database mirroring
- Reporting Services
- Windows domain membership
- RAID data storage



## Web Database Servers

Web database servers are used to provide data for websites. The data is frequently replicated or otherwise merged into the web database server from other internal servers. Web database servers are usually less powerful than enterprise servers, with the obvious exception of web-based companies such as eBay or Amazon.com.

Web database servers tend to utilize features and concepts such as the following:

- Built-in backup software
- Reporting Services
- RAID data storage
- Explicitly no Windows domain membership

## Reporting Servers

Dedicated reporting servers are used to gather report information from enterprise, departmental, and other database servers. A dedicated reporting server usually houses the reporting databases locally but accesses all other data from remote database servers.

Reporting servers tend to utilize features and concepts such as the following:

- Built-in backup software
- Reporting Services
- RAID data storage
- Windows domain membership

## ETL Servers

Dedicated ETL servers are used to perform nightly or periodic data moves, data transformations, and even data destruction. Because of licensing costs, dedicated ETL servers are usually found only on very large-scale deployments. Many smaller deployments simply use SSIS on existing database servers.

## Analysis and Decision Support Servers

Analysis, or decision support, servers are used by business analysts to determine the state of operations within an organization. Analysis servers may run Microsoft's Analysis Services, or they may run third-party tools as well. These servers get their data from other servers in the environment. Depending on the size of deployment, the transaction processing servers may send data to a warehouse from which the analysis servers retrieve it, or the data may be housed entirely within the analysis server.

## Intermediary Servers

Intermediary servers are becoming more common. These servers exist between two or more servers. An example of an intermediary server is a replication distributor server. The distributor sits between the publisher and the subscriber. The subscriber pulls information from the distributor that was first pulled by the distributor from the publisher. A model that deploys intermediary servers can often scale to be much larger than a single-server model.

## Standby Servers

Standby servers include log-shipping receiver servers, database mirror servers, and any server that is not actively used unless the primary server fails. Standby servers fall into three primary categories: cold standby, warm standby, and hot standby. These standby solutions are categorized by two factors.

The first factor is the latency of updates. A hot standby server receives updates at the same time as (or within seconds of) the primary server. A warm standby server may receive updates within minutes, and a cold standby server may receive updates only every few hours.

The second factor is the amount of time it takes to bring the standby server online. A hot standby server comes online immediately when the primary server fails. A cold standby server will never come online without manual intervention. A warm standby server may require manual intervention, or it may simply delay before coming online.

## Local Databases

The final role that SQL Server can play is the local database role. Many assume that only SQL Server Express edition is used as a local database; however, because of the size constraints of SQL Server Express (the maximum database size is 4GB), some applications require SQL Server Standard edition to be installed for a local database. The good news is that no client access licenses will be required; the bad news is that there is some expense, which is the price of the SQL Server Express edition.



### Real World Scenario

#### Enterprise Databases and 64-Bit Computing

When choosing the role of your SQL Server implementation, remember to also choose the proper edition. I was working with an organization that used SQL Server for a very large database application. Both performance and stability problems were prevalent, and we had to find a solution. I went through the normal performance and configuration analysis procedures and found no significant changes that could be made with the existing hardware.

In the end, we decided to upgrade the system to a 64-bit server running Windows Server 2003 Enterprise edition 64-bit. We also installed the 64-bit version of SQL Server 2005. What do you think happened? If you guessed that both the performance problems and the stability problems disappeared, you are correct. The culprit was, as it often is, memory—and the 64-bit version provided us with much more memory to utilize. Don't forget this when considering your implementation.

At the same time, it is important to ensure that your applications will run on a 64-bit installation of SQL Server. Many application vendors still develop extended stored procedures, which are basically DLL files called by the SQL Server. If these DLLs are provided only in 32-bit versions, they may or may not work with a 64-bit edition of SQL Server. Yes, there is much involved, but selecting the right edition and bit level is very important. Chapter 2, "Installing SQL Server 2012," explores this choice in more detail.

# Summary

The goal of this chapter was to lay a solid foundation on which to build the information contained in the rest of this book. You learned about the role played by databases in general and the specific roles that Microsoft recommends for SQL Server. You also learned the definitions of several important terms and concepts and found out what features were introduced in SQL Server 2005, 2008, and 2012. Now you're ready to begin installing and working with SQL Server.

## Chapter Essentials

**Understanding Information Technology's Many Components** Database servers make up one part of information technology (IT); however, they are a very important part. After all, they are the permanent locations where most of your data is stored.

**Understanding Databases' Many Flavors** Both server-based and local databases are common. Server-based databases provide advanced features such as support for concurrent users, improved security, and improved availability.

**Understanding SQL Server's Evolution** Over the years the SQL Server product line has matured into a very stable and feature-rich database solution. SQL Server 2005 was a big upgrade over SQL Server 2000, and SQL Server 2008 took the product to even greater heights. SQL Server 2012 adds to these capabilities in the areas of availability, management, and programmability.

**Understanding SQL Server's Many Roles** From enterprise servers to local databases, the SQL Server product can be utilized throughout an organization. Choosing the appropriate edition for the intended role is essential for performance and stability.

