Database: Database: 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.

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.

[](javascript:PopImage('IMG_3','http://images.books24x7.com/bookimages/id_52806/fig1-2_0.jpg','795','155'))  
Figure 1.2: A table of flat-file databases

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."](http://skillport.books24x7.com/assetviewer.aspx?bkid=52806&destid=484#484)) 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.OrderiID, 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;

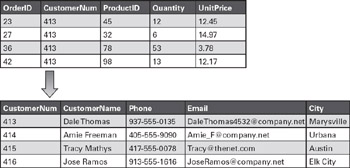
This simplification is one of the driving factors behind many views that are created and behind many of the decisions that are made when online analytical processing (OLAP) databases are implemented. OLAP databases are usually read from (far more read-operations are performed as opposed to write-operations), 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.

|  |  |  |
| --- | --- | --- |
|  | Note | *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."](http://skillport.books24x7.com/assetviewer.aspx?bkid=52806&destid=745#745) |

#### Relational Databases

*Relational databases* store information in separate containers called [*tables.*](http://skillport.books24x7.com/assetviewer.aspx?bkid=52806&destid=2113#2113) Each table represents a single entity, although *denormalized relational databases* may not always do so. You'll learn about normalization in [Chapter 8](http://skillport.books24x7.com/assetviewer.aspx?bkid=52806&destid=745#745); 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).

[](javascript:PopImage('IMG_4','http://images.books24x7.com/bookimages/id_52806/fig1-3_0.jpg','802','385'))  
Figure 1.3: The Sales and Items tables interact in a relational structure

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](http://skillport.books24x7.com/assetviewer.aspx?bkid=52806&destid=745#745).

### 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.

|  |  |  |
| --- | --- | --- |
|  | Note | 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.

|  |  |  |
| --- | --- | --- |
|  | Note | 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](http://skillport.books24x7.com/assetviewer.aspx?bkid=52806&destid=1465#1465) through [20](http://skillport.books24x7.com/assetviewer.aspx?bkid=52806&destid=1638#1638), you'll learn about SQL Server security solutions. In [Chapter 14](http://skillport.books24x7.com/assetviewer.aspx?bkid=52806&destid=1116#1116), you'll learn how to analyze the integrity of the stored data. |