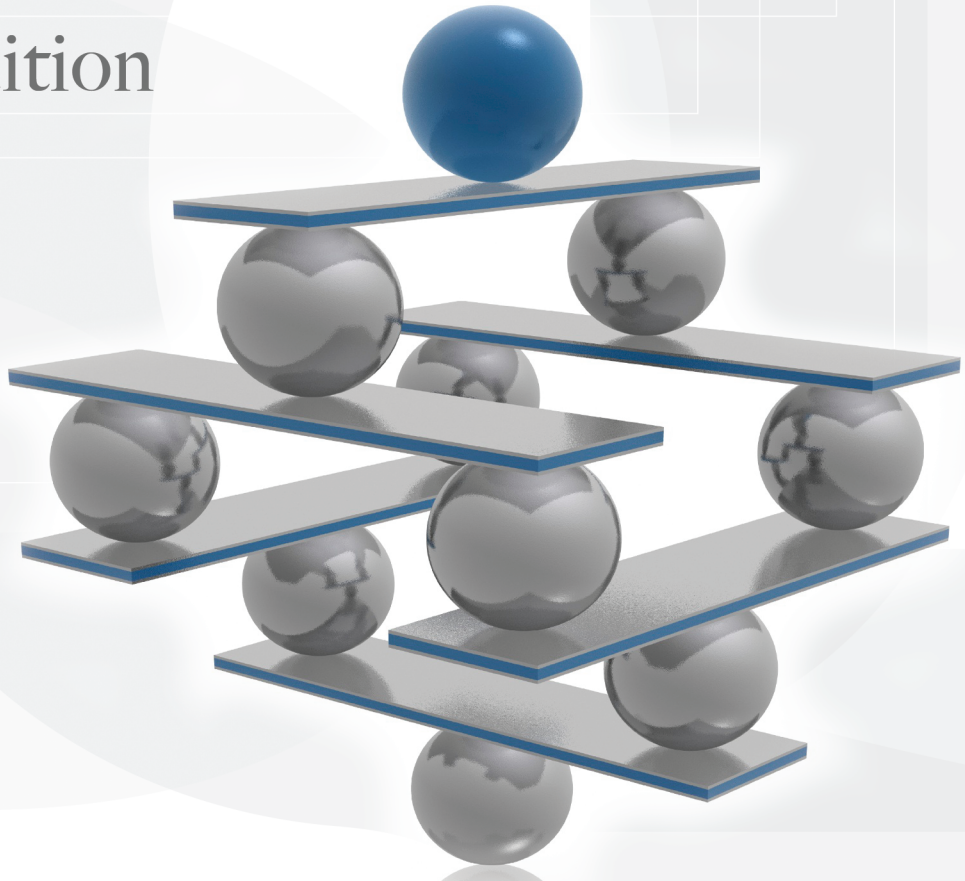
Microsoft® SQL Server® 2012

A BEGINNER'S GUIDE

Fifth Edition

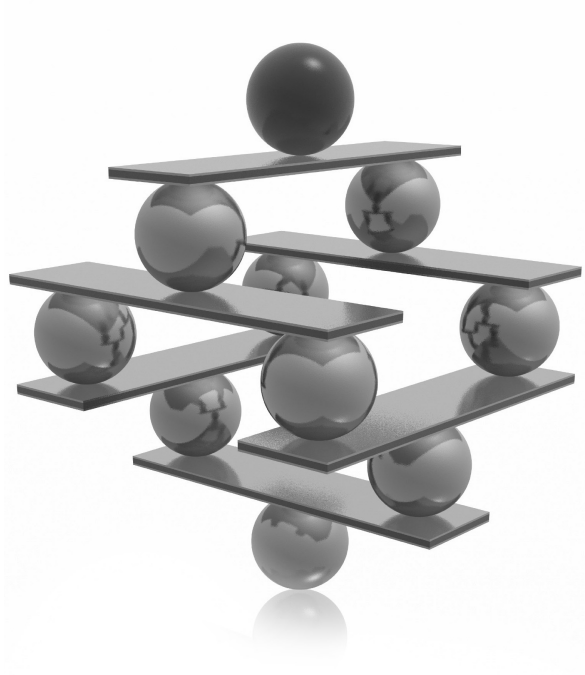


DUŠAN PETKOVIĆ

Mc Graw Hill

Part I

Basic Concepts and Installation



Chapter 1

Relational Database Systems: An Introduction

In This Chapter

- Database Systems: An Overview
- Relational Database Systems
- Database Design
- Syntax Conventions

This chapter describes database systems in general. First, it discusses what a database system is, and which components it contains. Each component is described briefly, with a reference to the chapter in which it is described in detail. The second major section of the chapter is dedicated to relational database systems. It discusses the properties of relational database systems and the corresponding language used in such systems—Structured Query Language (SQL).

Generally, before you implement a database, you have to design it, with all its objects. The third major section of the chapter explains how you can use normal forms to enhance the design of your database, and also introduces the entity-relationship model, which you can use to conceptualize all entities and their relationships. The final section presents the syntax conventions used throughout the book.

Database Systems: An Overview

A database system is an overall collection of different database software components and databases containing the following parts:

- Database application programs
- Client components
- Database server(s)
- Databases

A database application program is special-purpose software that is designed and implemented by users or by third-party software companies. In contrast, client components are general-purpose database software designed and implemented by a database company. By using client components, users can access data stored on the same or a remote computer.

The task of a database server is to manage data stored in a database. Each client communicates with a database server by sending user queries to it. The server processes each query and sends the result back to the client.

In general, a database can be viewed from two perspectives, the users' and the database system's. Users view a database as a collection of data that logically belong together. For a database system, a database is simply a series of bytes, usually stored on a disk. Although these two views of a database are totally different, they do have something in common: the database system needs to provide not only interfaces that enable users to create databases and retrieve or modify data, but also system components to manage the stored data. Hence, a database system must provide the following features:

- Variety of user interfaces
- Physical data independence
- Logical data independence
- Query optimization
- Data integrity
- Concurrency control
- Backup and recovery
- Database security

The following sections briefly describe these features.

Variety of User Interfaces

Most databases are designed and implemented for use by many different types of users with varied levels of knowledge. For this reason, a database system should offer many distinct user interfaces. A user interface can be either graphical or textual. Graphical user interfaces (GUIs) accept user's input via the keyboard or mouse and create graphical output on the monitor. A form of textual interface, which is often used by database systems, is the command-line interface (CLI), where the user provides the input by typing a command with the keyboard and the system provides output by printing text on the computer monitor.

Physical Data Independence

Physical data independence means that the database application programs do not depend on the physical structure of the stored data in a database. This important feature enables you to make changes to the stored data without having to make any changes to database application programs. For example, if the stored data is previously ordered using one criterion, and this order is changed using another criterion, the modification of the physical data should not affect the existing database applications or the existing database *schema* (a description of a database generated by the data definition language of the database system).

Logical Data Independence

In file processing (using traditional programming languages), the declaration of a file is done in application programs, so any change to the structure of that file usually requires the modification of all programs using it. Database systems provide logical data

independence—in other words, it is possible to make changes to the logical structure of the database without having to make any changes to the database application programs. For example, if the structure of an object named PERSON exists in the database system and you want to add an attribute to PERSON (say the address), you have to modify only the logical structure of the database, and not the existing application programs. (The application would have to be modified to utilize the newly added column.)

Query Optimization

Most database systems contain a subcomponent called *optimizer* that considers a variety of possible execution strategies for querying the data and then selects the most efficient one. The selected strategy is called the *execution plan* of the query. The optimizer makes its decisions using considerations such as how big the tables are that are involved in the query, what indices exist, and what Boolean operator (AND, OR, or NOT) is used in the WHERE clause. (This topic is discussed in detail in Chapter 19.)

Data Integrity

One of the tasks of a database system is to identify logically inconsistent data and reject its storage in a database. (The date February 30 and the time 5:77:00 P.M. are two examples of such data.) Additionally, most real-life problems that are implemented using database systems have *integrity constraints* that must hold true for the data. (One example of an integrity constraint might be the company's employee number, which must be a five-digit integer.) The task of maintaining integrity can be handled by the user in application programs or by the DBMS. As much as possible, this task should be handled by the DBMS. (Data integrity is discussed in two chapters of this book: declarative integrity in Chapter 5 and procedural integrity in Chapter 14.)

Concurrency Control

A database system is a multiuser software system, meaning that many user applications access a database at the same time. Therefore, each database system must have some kind of control mechanism to ensure that several applications that are trying to update the same data do so in some controlled way. The following is an example of a problem that can arise if a database system does not contain such control mechanisms:

1. The owners of bank account 4711 at bank X have an account balance of $2000.
2. The two joint owners of this bank account, Mrs. A and Mr. B, go to two different bank tellers, and each withdraws $1000 *at the same time*.
3. After these transactions, the amount of money in bank account 4711 should be $0 and not $1000.

All database systems have the necessary mechanisms to handle cases like this example. Concurrency control is discussed in detail in Chapter 13.

Backup and Recovery

A database system must have a subsystem that is responsible for recovery from hardware or software errors. For example, if a failure occurs while a database application updates 100 rows of a table, the recovery subsystem must roll back all previously executed updates to ensure that the corresponding data is consistent after the error occurs. (See Chapter 16 for further discussion on backup and recovery.)

Database Security

The most important database security concepts are authentication and authorization. *Authentication* is the process of validating user credentials to prevent unauthorized users from using a system. Authentication is most commonly enforced by requiring the user to enter a (user) name and a password. This information is evaluated by the system to determine whether the user is allowed to access the system. This process can be strengthened by using encryption.

Authorization is the process that is applied after the identity of a user is authenticated. During this process, the system determines what resources the particular user can use. In other words, structural and system catalog information about a particular entity is now available only to principals that have permission to access that entity. (Chapter 12 discusses these concepts in detail.)

Relational Database Systems

The component of Microsoft SQL Server called the Database Engine is a relational database system. The notion of relational database systems was first introduced by E. F. Codd in his article "A Relational Model of Data for Large Shared Data Banks" in 1970. In contrast to earlier database systems (network and hierarchical), *relational database systems* are based upon the relational data model, which has a strong mathematical background.

NOTE

A data model is a collection of concepts, their relationships, and their constraints that are used to represent data of a real-world problem.

The central concept of the relational data model is a relation—that is, a table. Therefore, from the user's point of view, a relational database contains tables and

nothing but tables. In a table, there are one or more columns and zero or more rows. At every row and column position in a table there is always exactly one data value.

Working with the Book's Sample Database

The sample database used in this book represents a company with departments and employees. Each employee in the example belongs to exactly one department, which itself has one or more employees. Jobs of employees center on projects: each employee works at the same time on one or more projects, and each project engages one or more employees.

The data of the **sample** database can be represented using four tables:

- department
- employee
- project
- works_on

Tables 1-1 through 1-4 show all the tables of the **sample** database.

The **department** table represents all departments of the company. Each department has the following attributes:

department (dept_no, dept_name, location)

dept_no represents the unique number of each department. **dept_name** is its name, and **location** is the location of the corresponding department.

The **employee** table represents all employees working for a company. Each employee has the following attributes:

employee (emp_no, emp_fname, emp_lname, dept_no)

dept_no	dept_name	location
d1	Research	Dallas
d2	Accounting	Seattle
d3	Marketing	Dallas

Table 1-1 *The Department Table*

emp_no	emp_fname	emp_lname	dept_no
25348	Matthew	Smith	d3
10102	Ann	Jones	d3
18316	John	Barrimore	d1
29346	James	James	d2
9031	Elke	Hansel	d2
2581	Elsa	Bertoni	d2
28559	Sybill	Moser	d1

Table 1-2 *The Employee Table*

project_no	project_name	budget
p1	Apollo	120000
p2	Gemini	95000
p3	Mercury	186500

Table 1-3 *The Project Table*

emp_no	project_no	job	enter_date
10102	p1	Analyst	2006.10.1
10102	p3	Manager	2008.1.1
25348	p2	Clerk	2007.2.15
18316	p2	NULL	2007.6.1
29346	p2	NULL	2006.12.15
2581	p3	Analyst	2007.10.15
9031	p1	Manager	2007.4.15
28559	p1	NULL	2007.8.1
28559	p2	Clerk	2008.2.1
9031	p3	Clerk	2006.11.15
29346	p1	Clerk	2007.1.4

Table 1-4 *The works_on Table*

emp_no represents the unique number of each employee. **emp_fname** and **emp_lname** are the first and last name of each employee, respectively. Finally, **dept_no** is the number of the department to which the employee belongs.

Each project of a company is represented in the **project** table. This table has the following columns:

project (project_no, project_name, budget)

project_no represents the unique number of each project. **project_name** and **budget** specify the name and the budget of each project, respectively.

The **works_on** table specifies the relationship between employees and projects. It has the following columns:

works_on (emp_no, project_no, job, enter_date)

emp_no specifies the employee number and **project_no** specifies the number of the project on which the employee works. The combination of data values belonging to these two columns is always unique. **job** and **enter_date** specify the task and the starting date of an employee in the corresponding project, respectively.

Using the **sample** database, it is possible to describe some general properties of relational database systems:

- Rows in a table do not have any particular order.
- Columns in a table do not have any particular order.
- Every column must have a unique name within a table. On the other hand, columns from different tables may have the same name. (For example, the **sample** database has a **dept_no** column in the **department** table and a column with the same name in the **employee** table.)
- Every single data item in the table must be single valued. This means that in every row and column position of a table there is never a set of multiple data values.
- For every table, there is at least one column with the property that no two rows have the same combination of data values for all table columns. In the relational data model, such an identifier is called a *candidate key*. If there is more than one candidate key within a table, the database designer designates one of them as the *primary key* of the table. For example, the column **dept_no** is the primary key of the **department** table; the columns **emp_no** and **project_no** are the primary keys of the tables **employee** and **project**, respectively. Finally, the primary key for the **works_on** table is the combination of the columns **emp_no**, **project_no**.

- In a table, there are never two identical rows. (This property is only theoretical; the Database Engine and all other relational database systems generally allow the existence of identical rows within a table.)

SQL: A Relational Database Language

The SQL Server relational database language is called Transact-SQL. It is a dialect of the most important database language today: Structured Query Language (SQL). The origin of SQL is closely connected with the project called System R, which was designed and implemented by IBM in the early 1980s. This project showed that it is possible, using the theoretical foundations of the work of E. F. Codd, to build a relational database system.

In contrast to traditional languages like C, C++, and Java, SQL is a set-oriented language. (The former are also called record-oriented languages.) This means that SQL can query many rows from one or more tables using just one statement. This feature is one of the most important advantages of SQL, allowing the use of this language at a logically higher level than the level at which traditional languages can be used.

Another important property of SQL is its nonprocedurality. Every program written in a procedural language (C, C++, Java) describes *how* a task is accomplished, step by step. In contrast to this, SQL, as any other nonprocedural language, describes *what* it is that the user wants. Thus, the system is responsible for finding the appropriate way to solve users' requests.

SQL contains two sublanguages: a data definition language (DDL) and a data manipulation language (DML). DDL statements are used to describe the schema of database tables. The DDL contains three generic SQL statements: CREATE object, ALTER object, and DROP object. These statements create, alter, and remove database objects, such as databases, tables, columns, and indexes. (These statements are discussed in detail in Chapter 5.)

In contrast to the DDL, the DML encompasses all operations that manipulate the data. There are always four generic operations for manipulating the database: retrieval, insertion, deletion, and modification. The retrieval statement SELECT is described in Chapter 6, while the INSERT, DELETE, and UPDATE statements are discussed in detail in Chapter 7.

Database Design

Designing a database is a very important phase in the database life cycle, which precedes all other phases except the requirements collection and the analysis. If the database design is created merely intuitively and without any plan, the resulting database will most likely not meet the user requirements concerning performance.

Another consequence of a bad database design is superfluous data redundancy, which in itself has two disadvantages: the existence of data anomalies and the use of an unnecessary amount of disk space.

Normalization of data is a process during which the existing tables of a database are tested to find certain dependencies between the columns of a table. If such dependencies exist, the table is restructured into multiple (usually two) tables, which eliminates any column dependencies. If one of these generated tables still contains data dependencies, the process of normalization must be repeated until all dependencies are resolved.

The process of eliminating data redundancy in a table is based upon the theory of functional dependencies. A *functional dependency* means that by using the known value of one column, the corresponding value of another column can always be uniquely determined. (The same is true for column groups.) The functional dependencies between columns A and B is denoted by A ⇒ B, specifying that a value of column A can always be used to determine the corresponding value of column B. ("B is functionally dependent on A.")

Example 1.1 shows the functional dependency between two attributes of the table **employee** in the **sample** database.

EXAMPLE 1.1

emp_no ⇒ emp_lname

By having a unique value for the employee number, the corresponding last name of the employee (and all other corresponding attributes) can be determined. This kind of functional dependency, where a column is dependent upon the primary key of a table, is called *trivial* functional dependency.

Another kind of functional dependency is called *multivalued dependency*. In contrast to the functional dependency just described, the multivalued dependency is specified for multivalued attributes. This means that by using the known value of one attribute (column), the corresponding *set of values* of another multivalued attribute can be uniquely determined. The multivalued dependency is denoted by ⇒ ⇒.

Example 1.2 shows the multivalued dependency that holds for two attributes of the object BOOK.

EXAMPLE 1.2

ISBN ⇒ ⇒ Authors

The ISBN of a book always determines all of its authors. Therefore, the **Authors** attribute is multivalued dependent on the **ISBN** attribute.

Normal Forms

Normal forms are used for the process of normalization of data and therefore for the database design. In theory, there are at least five different normal forms, of which the first three are the most important for practical use. The third normal form for a table can be achieved by testing the first and second normal forms at the intermediate states, and as such, the goal of good database design can usually be fulfilled if all tables of a database are in the third normal form.

NOTE

The multivalued dependency is used to test the fourth normal form of a table. Therefore, this kind of dependency will not be used further in this book.

First Normal Form

First normal form (1NF) means that a table has no multivalued attributes or composite attributes. (A composite attribute contains other attributes and can therefore be divided into smaller parts.) All relational tables are by definition in 1NF, because the value of any column in a row must be *atomic*—that is, single valued.

Table 1-5 demonstrates 1NF using part of the **works_on** table from the **sample** database. The rows of the **works_on** table could be grouped together, using the employee number. The resulting Table 1-6 is not in 1NF because the column **project_no** contains a set of values (p1, p3).

Second Normal Form

A table is in second normal form (2NF) if it is in 1NF and there is no nonkey column dependent on a partial primary key of that table. This means if (A,B) is a combination

emp_no	project_no	.....
10102	p1	.....
10102	p3	.....
.....	.....	….....

Table 1-5 *Part of the works_on Table*

emp_no	project_no	.....
10102	(p1, p3)	.....
.....	.....	.....

Table 1-6 *This "Table" Is Not in 1NF*

of two table columns building the key, then there is no column of the table depending either on only A or only B.

For example, Table 1-7 shows the **works_on1** table, which is identical to the **works_on** table except for the additional column, **dept_no**. The primary key of this table is the combination of columns **emp_no** and **project_no**. The column **dept_no** is dependent on the partial key **emp_no** (and is independent of **project_no**), so this table is not in 2NF. (The original table, **works_on**, is in 2NF.)

NOTE

Every table with a one-column primary key is always in 2NF.

Third Normal Form

A table is in third normal form (3NF) if it is in 2NF and there are no functional dependencies between nonkey columns. For example, the **employee1** table (see Table 1-8), which is identical to the **employee** table except for the additional column, **dept_name**, is not in 3NF, because for every known value of the column **dept_no** the corresponding value of the column **dept_name** can be uniquely determined. (The original table, **employee**, as well as all other tables of the **sample** database are in 3NF.)

emp_no	project_no	job	enter_date	dept_no
10102	p1	Analyst	2006.10.1	d3
10102	p3	Manager	2008.1.1	d3
25348	p2	Clerk	2007.2.15	d3
18316	p2	NULL	2007.6.1	d1
.....	.....	.....		.....

Table 1-7 *The works_on1 Table*

emp_no	emp_fname	emp_lname	dept_no	dept_name
25348	Matthew	Smith	d3	Marketing
10102	Ann	Jones	d3	Marketing
18316	John	Barrimore	d1	Research
29346	James	James	d2	Accounting
.....	.....	.....	.....	.....

Table 1-8 *The employee1 Table*

Entity-Relationship Model

The data in a database could easily be designed using only one table that contains all data. The main disadvantage of such a database design is its high redundancy of data. For example, if your database contains data concerning employees and their projects (assuming each employee works at the same time on one or more projects, and each project engages one or more employees), the data stored in a single table contains many columns and rows. The main disadvantage of such a table is that data is difficult to keep consistent because of its redundancy.

The *entity-relationship (ER) model* is used to design relational databases by removing all existing redundancy in the data. The basic object of the ER model is an *entity*—that is, a real-world object. Each entity has several *attributes*, which are properties of the entity and therefore describe it. Based on its type, an attribute can be

- **Atomic (or single valued)** An atomic attribute is always represented by a single value for a particular entity. For example, a person's marital status is always an atomic attribute. Most attributes are atomic attributes.
- **Multivalued** A multivalued attribute may have one or more values for a particular entity. For example, **Location** as the attribute of an entity called ENTERPRISE is multivalued, because each enterprise can have one or more locations.
- **Composite** Composite attributes are not atomic because they are assembled using some other atomic attributes. A typical example of a composite attribute is a person's address, which is composed of atomic attributes, such as **City**, **Zip**, and **Street**.

The entity PERSON in Example 1.3 has several atomic attributes, one composite attribute, **Address**, and a multivalued attribute, **College_degree**.

EXAMPLE 1.3

PERSON (Personal_no, F_name, L_name, Address(City,Zip,Street),{College_degree})

Each entity has one or more key attributes that are attributes (or a combination of two or more attributes) whose values are unique for each particular entity. In Example 1.3, the attribute **Personal_no** is the key attribute of the entity PERSON.

Besides entity and attribute, *relationship* is another basic concept of the ER model. A relationship exists when an entity refers to one (or more) other entities. The number of participating entities defines the degree of a relationship. For example, the relationship **works_on** between entities EMPLOYEE and PROJECT has degree two.

Every existing relationship between two entities must be one of the following three types: 1:1, 1:N, or M:N. (This property of a relationship is also called *cardinality ratio*.) For example, the relationship between the entities DEPARTMENT and EMPLOYEE is 1:N, because each employee belongs to exactly one department, which itself has one or more employees. Also, the relationship between the entities PROJECT and EMPLOYEE is M:N, because each project engages one or more employees and each employee works at the same time on one or more projects.

A relationship can also have its own attributes. Figure 1-1 shows an example of an ER diagram. (The ER diagram is the graphical notation used to describe the ER model.)

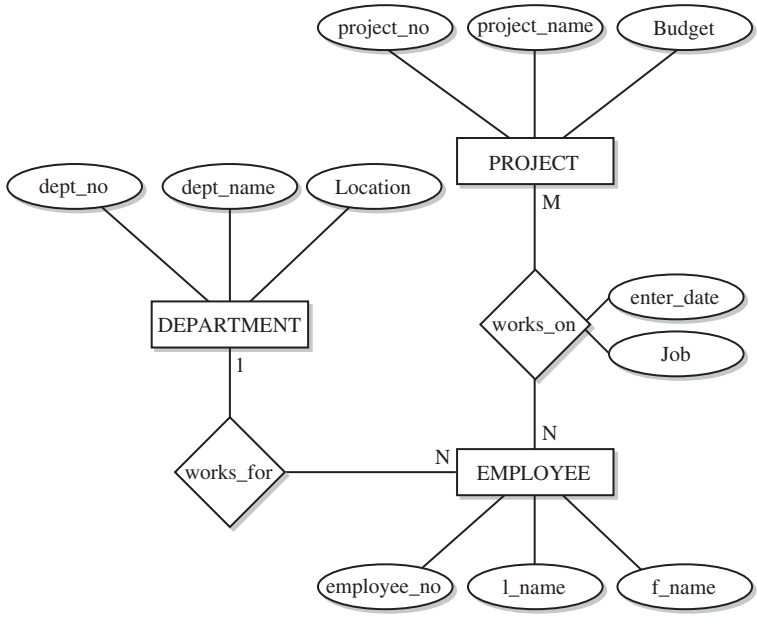


Figure 1-1 *Example of an ER diagram*

Using this notation, entities are modeled using rectangular boxes, with the entity name written inside the box. Attributes are shown in ovals, and each attribute is attached to a particular entity (or relationship) using a straight line. Finally, relationships are modeled using diamonds, and entities participating in the relationship are attached to it using straight lines. The cardinality ratio of each entity is written on the corresponding line.

Syntax Conventions

This book uses the conventions shown in Table 1-9 for the syntax of the Transact-SQL statements and for the indication of the text.

NOTE

In contrast to brackets and braces, which belong to syntax conventions, parentheses, **( )***, belong to the syntax of a statement and must always be typed!*

Convention	Indication
Italics	New terms or items of emphasis.
UPPERCASE	Transact-SQL keywords—for example, CREATE TABLE. Additional information about the keywords of the Transact-SQL language can be found in Chapter 5.
lowercase	Variables in Transact-SQL statements—for example, CREATE TABLE tablename. (The user must replace "tablename" with the actual name of the table.)
var1 | var2	Alternative use of the items var1 and var2. (You may choose only one of the items separated by the vertical bar.)
{ }	Alternative use of more items. Example: { expression | USER | NULL }
[ ]	Optional item(s). Example: [FOR LOAD]
{ } ...	Item(s) that can be repeated any number of times. Example: {, @param1 typ1} . . .
bold	Name of database object (database itself, tables, columns) in the text.
Default	The default value is always underlined. Example: ALL | DISTINCT

Table 1-9 *Syntax Conventions*

Summary

All database systems provide the following features:

- Variety of user interfaces
- Physical data independence
- Logical data independence
- Query optimization
- Data integrity
- Concurrency control
- Backup and recovery
- Database security

The next chapter shows you how to install SQL Server 2012.

Exercises

E.1.1

What does "data independence" mean and which two forms of data independence exist?

E.1.2

Which is the main concept of the relational model?

E.1.3

What does the **employee** table represent in the real world? And what does the row in this table with the data for Ann Jones represent?

E.1.4

What does the **works_on** table represent in the real world (and in relation to the other tables of the **sample** database)?

E.1.5

Let **book** be a table with two columns: **isbn** and **title**. Assuming that **isbn** is unique and there are no identical titles, answer the following questions:

a. Is **title** a key of the table?
b. Does **isbn** functionally depend on **title**?
c. Is the **book** table in 3NF?

E.1.6

Let **order** be a table with the following columns: **order_no**, **customer_no**, **discount**. If the column **customer_no** is functionally dependent on **order_no** and the column **discount** is functionally dependent on **customer_no**, answer the following questions and explain in detail your answers:

a. Is **order_no** a key of the table?
b. Is **customer_no** a key of the table?

E.1.7

Let **company** be a table with the following columns: **company_no**, **location**. Each company has one or more locations. In which normal form is the **company** table?

E.1.8

Let **supplier** be a table with the following columns: **supplier_no**, **article**, **city**. The key of the table is the combination of the first two columns. Each supplier delivers several articles, and each article is delivered by several suppliers. There is only one supplier in each city. Answer the following questions:

a. In which normal form is the **supplier** table?
b. How can you resolve the existing functional dependencies?

E.1.9

Let R(A, B, C) be a relation with the functional dependency B $\Rightarrow$ C. (The underlined attributes A and B build the composite key, and the attribute C is functionally dependent on B.) In which normal form is the relation R?

E.1.10

Let R(A, B, C) be a relation with the functional dependency C $\Rightarrow$ B. (The underlined attributes A and B build the composite key, and the attribute B is functionally dependent on C.) In which normal form is the relation R?