Structured Query Language (SQL)

Outlines

- SQL Data Definition and Data Types
- Specifying Constraints in SQL
- Basic Retrieval Queries in SQL
- INSERT, DELETE, and UPDATE Statements in SQL
- Additional Features of SQL

Database System Environment

SQL language may be considered one of the major reasons for the commercial success of relational databases. Because it became a standard for relational databases, users were less concerned about migrating their database applications from other types of database systems—for example, older network or hierarchical systems—to relational systems. This is because even if the users became dissatisfied with the particular relational DBMS product they were using, converting to another relational DBMS product was not expected to be too expensive and time-consuming because both systems followed the same language standards. In practice, of course, there are differences among various commercial relational DBMS packages. However, if the user is diligent in using only those features that are part of the standard, and if two relational DBMSs faithfully support the standard, then conversion between two systems should be simplified. Another advantage of having such a standard is that users may write statements in a database application program that can access data stored in two or more relational DBMSs without having to change the database sublanguage (SQL), as long as both/all of the relational DBMSs support standard SQL.

This chapter presents the practical relational model, which is based on the SQL standard for commercial relational DBMSs, whereas Chapter 5 presented the most important concepts underlying the formal relational data model. In Chapter 8 (Sections 8.1 through 8.5), we shall discuss the relational algebra operations, which are very important for understanding the types of requests that may be specified on a relational database. They are also important for query processing and optimization in a relational DBMS, as we shall see in Chapters 18 and 19. However, the relational algebra operations are too low-level for most commercial DBMS users because a query in relational algebra is written as a sequence of operations that, when executed, produces the required result. Hence, the user must specify how—that is, in what order—to execute the query operations. On the other hand, the SQL language provides a higher-level *declarative* language interface, so the user only specifies *what* the result is to be, leaving the actual optimization and decisions on how to execute the query to the DBMS. Although SQL includes some features from relational algebra, it is based to a greater extent on the *tuple relational calculus*, which we describe in Section 8.6. However, the SQL syntax is more user-friendly than either of the two formal languages.

The name **SQL** is presently expanded as Structured Query Language. Originally, SQL was called SEQUEL (Structured English QUEry Language) and was designed and implemented at IBM Research as the interface for an experimental relational database system called SYSTEM R. SQL is now the standard language for commercial relational DBMSs. The standardization of SQL is a joint effort by the American National Standards Institute (ANSI) and the International Standards Organization (ISO), and the first SQL standard is called SQL-86 or SQL1.

A revised and much expanded standard called SQL-92 (also referred to as SQL2) was subsequently developed. The next standard that is well-recognized is SQL:1999, which started out as SQL3. Additional updates to the standard are SQL:2003 and SQL:2006, which added XML features (see Chapter 13) among other updates to the language. Another update in 2008 incorporated more object database features into SQL (see Chapter 12), and a further update is SQL:2011. We will try to cover the latest version of SQL as much as possible, but some of the newer features are discussed in later chapters. It is also not possible to cover the language in its entirety in this text. It is important to note that when new features are added to SQL, it usually takes a few years for some of these features to make it into the commercial SQL DBMSs.

SQL is a comprehensive database language: It has statements for data definitions, queries, and updates. Hence, it is both a DDL *and* a DML. In addition, it has facilities for defining views on the database, for specifying security and authorization, for defining integrity constraints, and for specifying transaction controls. It also has rules for embedding SQL statements into a general-purpose programming language such as Java or C/C++. The later SQL standards (starting with **SQL:1999**) are divided into a **core** specification plus specialized **extensions**. The core is supposed to be implemented by all RDBMS vendors that are SQL compliant. The extensions can be implemented as optional modules to be purchased independently for specific database applications such as data mining, spatial data, temporal data, data warehousing, online analytical processing (OLAP), multimedia data, and so on.

Because the subject of SQL is both important and extensive, we devote two chapters to its basic features. In this chapter, Section 6.1 describes the SQL DDL commands for creating schemas and tables, and gives an overview of the basic data types in SQL. Section 6.2 presents how basic constraints such as key and referential integrity are specified. Section 6.3 describes the basic SQL constructs for specifying retrieval queries, and Section 6.4 describes the SQL commands for insertion, deletion, and update. In Chapter 7, we will describe more complex SQL retrieval queries, as well as the ALTER commands for changing the schema. We will also describe the CREATE ASSERTION statement, which allows the specification of more general constraints on the database, and the concept of triggers, which is presented in more detail in Chapter 26. We discuss the SQL facility for defining views on the database in Chapter 7. Views are also called *virtual* or *derived tables* because they present the user with what appear to be tables; however, the information in those tables is derived from previously defined tables.

SQL Data Definition and Data Types

SQL uses the terms **table**, **row**, and **column** for the formal relational model terms *relation*, *tuple*, and *attribute*, respectively. We will use the corresponding terms interchangeably. The main SQL command for data definition is the CREATE statement, which can be used to create schemas, tables (relations), types, and domains, as well as other constructs such as views, assertions, and triggers. Before we describe the relevant CREATE statements, we discuss schema and catalog concepts in Section 6.1.1 to place our discussion in perspective. Section 6.1.2 describes how tables are created, and Section 6.1.3 describes the most important data types available for attribute specification. Because the SQL specification is very large, we give a description of the most important features. Further details can be found in the various SQL standards documents (see end-of-chapter bibliographic notes).

Schema and Catalog Concepts in SQL

Early versions of SQL did not include the concept of a relational database schema; all tables (relations) were considered part of the same schema. The concept of an SQL schema was incorporated starting with SQL2 in order to group together tables and other constructs that belong to the same database application

(in some systems, a *schema* is called a *database*). An **SQL schema** is identified by a **schema name** and includes an **authorization identifier** to indicate the user or account who owns the schema, as well as **descriptors** for *each element* in the schema. Schema **elements** include tables, types, constraints, views, domains, and other constructs (such as authorization grants) that describe the schema. A schema is created via the CREATE SCHEMA statement, which can include all the schema elements' definitions. Alternatively, the schema can be assigned a name and authorization identifier, and the elements can be defined later. For example, the following statement creates a schema called COMPANY owned by the user with authorization identifier 'Jsmith'. Note that each statement in SQL ends with a semicolon.

CREATE SCHEMA COMPANY **AUTHORIZATION** 'Jsmith';

In general, not all users are authorized to create schemas and schema elements. The privilege to create schemas, tables, and other constructs must be explicitly granted to the relevant user accounts by the system administrator or DBA.

In addition to the concept of a schema, SQL uses the concept of a **catalog**—a named collection of schemas.[2] Database installations typically have a default environment and schema, so when a user connects and logs in to that database installation, the user can refer directly to tables and other constructs within that schema without having to specify a particular schema name. A catalog always contains a special schema called INFORMATION_SCHEMA, which provides information on all the schemas in the catalog and all the element descriptors in these schemas. Integrity constraints such as referential integrity can be defined between relations only if they exist in schemas within the same catalog. Schemas within the same catalog can also share certain elements, such as type and domain definitions.

The CREATE TABLE Command in SQL

The **CREATE TABLE** command is used to specify a new relation by giving it a name and specifying its attributes and initial constraints. The attributes are specified first, and each attribute is given a name, a data type to specify its domain of values, and possibly attribute constraints, such as NOT NULL. The key, entity integrity, and referential integrity constraints can be specified within the CREATE TABLE statement after the attributes are declared, or they can be added later using the ALTER TABLE command (see Chapter 7). Figure 6.1 shows sample data definition statements in SQL for the COMPANY relational database schema shown in Figure 3.7. Typically, the SQL schema in which the relations are declared is implicitly specified in the environment in which the CREATE TABLE statements are executed. Alternatively, we can explicitly attach the schema name to the relation name, separated by a period. For example, by writing

CREATE TABLE COMPANY.EMPLOYEE

rather than

CREATE TABLE EMPLOYEE

as in Figure 6.1, we can explicitly (rather than implicitly) make the EMPLOYEE table part of the COMPANY schema. The relations declared through CREATE TABLE statements are called base tables (or

base relations); this means that the table and its rows are actually created

```
CREATE TABLE EMPLOYEE
        ( Fname                 VARCHAR(15)             NOT NULL,
          Minit                 CHAR,
          Lname                 VARCHAR(15)             NOT NULL,
          Ssn                   CHAR(9)                 NOT NULL,
          Bdate                 DATE,
          Address               VARCHAR(30),
          Sex                   CHAR,
          Salary                DECIMAL(10,2),
          Super_ssn             CHAR(9),
          Dno                   INT                     NOT NULL,
        PRIMARY KEY (Ssn),
CREATE TABLE DEPARTMENT
        ( Dname                 VARCHAR(15)             NOT NULL,
          Dnumber               INT                     NOT NULL,
          Mgr_ssn               CHAR(9)                 NOT NULL,
          Mgr_start_date        DATE,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
CREATE TABLE DEPT_LOCATIONS
        ( Dnumber               INT                     NOT NULL,
          Dlocation             VARCHAR(15)             NOT NULL,
        PRIMARY KEY (Dnumber, Dlocation),
        FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE PROJECT
        ( Pname                 VARCHAR(15)             NOT NULL,
          Pnumber               INT                     NOT NULL,
          Plocation             VARCHAR(15),
          Dnum                  INT                     NOT NULL,
        PRIMARY KEY (Pnumber),
        UNIQUE (Pname),
        FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE WORKS_ON
        ( Essn                  CHAR(9)                 NOT NULL,
          Pno                   INT                     NOT NULL,
          Hours                 DECIMAL(3,1)            NOT NULL,
        PRIMARY KEY (Essn, Pno),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );
CREATE TABLE DEPENDENT
        ( Essn                  CHAR(9)                 NOT NULL,
          Dependent_name        VARCHAR(15)             NOT NULL,
          Sex                   CHAR,
          Bdate                 DATE,
          Relationship          VARCHAR(8),
        PRIMARY KEY (Essn, Dependent_name),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```

Figure 6.1
SQL CREATE TABLE data definition statements for defining the COMPANY schema from Figure 5.7.

and stored as a file by the DBMS. Base relations are distinguished from **virtual relations**, created through the CREATE VIEW statement (see Chapter 7), which may or may not correspond to an actual physical file. In SQL, the attributes in a base table are considered to be *ordered in the sequence in which they are*

specified in the CREATE TABLE statement. However, rows (tuples) are not considered to be ordered within a table (relation).

It is important to note that in Figure 6.1, there are some *foreign keys that may cause errors* because they are specified either via circular references or because they refer to a table that has not yet been created. For example, the foreign key Super_ssn in the EMPLOYEE table is a circular reference because it refers to the EMPLOYEE table itself. The foreign key Dno in the EMPLOYEE table refers to the DEPARTMENT table, which has not been created yet. To deal with this type of problem, these constraints can be left out of the initial CREATE TABLE statement, and then added later using the ALTER TABLE statement (see Chapter 7). We displayed all the foreign keys in Figure 6.1 to show the complete COMPANY schema in one place.

Attribute Data Types and Domains in SQL

The basic **data types** available for attributes include numeric, character string, bit string, Boolean, date, and time.

- **Numeric** data types include integer numbers of various sizes (INTEGER or INT, and SMALLINT) and floating-point (real) numbers of various precision (FLOAT or REAL, and DOUBLE PRECISION). Formatted numbers can be
- declared by using DECIMAL(*i*, *j*)—or DEC(*i*, *j*) or NUMERIC(*i*, *j*)—where *i*, the *precision*, is the total number of decimal digits and *j*, the *scale*, is the number of digits after the decimal point. The default for scale is zero, and the default for precision is implementation-defined.
- **Character-string** data types are either fixed length—CHAR(*n*) or CHARACTER(*n*), where *n* is the number of characters—or varying length—VARCHAR(*n*) or CHAR VARYING(*n*) or CHARACTER VARYING(*n*), where *n* is the maximum number of characters. When specifying a literal string value, it is placed between single quotation marks (apostrophes), and it is *case sensitive* (a distinction is made between uppercase and lowercase).[3] For fixedlength strings, a shorter string is padded with blank characters to the right. For example, if the value 'Smith' is for an attribute of type CHAR(10), it is padded with five blank characters to become 'Smith' if needed. Padded blanks are generally ignored when strings are compared. For comparison purposes, strings are considered ordered in alphabetic (or lexicographic) order; if a string *str1* appears before another string *str2* in alphabetic order, then *str1* is considered to be less than *str2*.[4] There is also a concatenation operator denoted by || (double vertical bar) that can concatenate two strings in SQL. For example, 'abc' || 'XYZ' results in a single string 'abcXYZ'. Another variable-length string data type called CHARACTER LARGE OBJECT or CLOB is also available to specify columns that have large text values, such as documents. The CLOB maximum length can be specified in kilobytes (K), megabytes (M), or gigabytes (G). For example, CLOB(20M) specifies a maximum length of 20 megabytes.
- **Bit-string** data types are either of fixed length *n*—BIT(*n*)—or varying length—BIT VARYING(*n*), where *n* is the maximum number of bits. The default for *n*, the length of a character string or bit string, is 1. Literal bit strings are placed between single quotes but preceded by a B to distinguish them from character strings; for example, B'10101'.[5] Another variable-length bitstring data type called BINARY LARGE OBJECT or BLOB is also available to specify columns that have large binary values, such as images. As for CLOB, the maximum ength of a BLOB can be specified in kilobits (K), megabits (M), or gigabits (G). For example, BLOB(30G) specifies a maximum length of 30 gigabits.

- A **Boolean** data type has the traditional values of TRUE or FALSE. In SQL, because of the presence of NULL values, a three-valued logic is used, so a third possible value for a Boolean data type is UNKNOWN. We discuss the need for UNKNOWN and the three-valued logic in Chapter 7.
- The **DATE** data type has ten positions, and its components are YEAR, MONTH, and DAY in the form YYYY-MM-DD. The TIME data type has at least eight positions, with the components HOUR, MINUTE, and SECOND in the form HH:MM:SS. Only valid dates and times should be allowed by the SQL implementation.
- This implies that months should be between 1 and 12 and days must be between 01 and 31; furthermore, a day should be a valid day for the corresponding month. The < (less than) comparison can be used with dates or times—an *earlier* date is considered to be smaller than a later date, and similarly with time. Literal values are represented by single-quoted strings preceded by the keyword DATE or TIME; for example, DATE '2014-09-27' or TIME '09:12:47'. In addition, a data type TIME(*i*), where *i* is called *time fractional seconds precision*, specifies *i* + 1 additional positions for TIME—one position for an additional period (.) separator character, and *i* positions for specifying decimal fractions of a second. A TIME WITH TIME ZONE data type includes an additional six positions for specifying the *displacement* from the standard universal time zone, which is in the range +13:00 to –12:59 in units of HOURS:MINUTES. If WITH TIME ZONE is not included, the default is the local time zone for the SQL session.
- Some additional data types are discussed below. The list of types discussed here is not exhaustive; different implementations have added more data types to SQL.
- A timestamp data type (TIMESTAMP) includes the DATE and TIME fields, plus a minimum of six positions for decimal fractions of seconds and an optional WITH TIME ZONE qualifier. Literal values are represented by single-quoted strings preceded by the keyword TIMESTAMP, with a blank space between data and time; for example, TIMESTAMP '2014-09-27 09:12:47.648302'. ■ Another data type related to DATE, TIME, and TIMESTAMP is the INTERVAL data type. This specifies an interval—a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp. Intervals are qualified to be either YEAR/MONTH intervals or DAY/TIME intervals. The format of DATE, TIME, and TIMESTAMP can be considered as a special type of string. Hence, they can generally be used in string comparisons by being cast (or coerced or converted) into the equivalent strings. It is possible to specify the data type of each attribute directly, as in Figure 6.1; alternatively, a domain can be declared, and the domain name can be used with the attribute specification. This makes it easier to change the data type for a domain that is used by numerous attributes in a schema, and improves schema readability. For example, we can create a domain SSN_TYPE by the following statement:
    - CREATE DOMAIN SSN_TYPE AS CHAR(9); We can use SSN_TYPE in place of CHAR(9) in Figure 6.1 for the attributes Ssn and Super_ssn of EMPLOYEE, Mgr_ssn of DEPARTMENT, Essn of WORKS_ON, and Essn of DEPENDENT. A domain can also have an optional default specification via a DEFAULT clause, as we discuss later for attributes. Notice that domains may not be available in some implementations of SQL.
    - In SQL, there is also a CREATE TYPE command, which can be used to create user defined types or UDTs. These can then be used either as data types for attributes, or as the basis for creating tables. We shall discuss CREATE TYPE in detail in Chapter 12, because it is often used in conjunction with specifying object database features that have been incorporated into more recent versions of SQL.

Specifying Constraints in SQL

This section describes the basic constraints that can be specified in SQL as part of table creation. These include key and referential integrity constraints, restrictions on attribute domains and NULLs, and constraints on individual tuples within a relation using the CHECK clause. We discuss the specification of more general constraints, called assertions, in Chapter 7.

Specifying Attribute Constraints and Attribute Defaults

Because SQL allows NULLs as attribute values, a *constraint* NOT NULL may be specified if NULL is not permitted for a particular attribute. This is always implicitly specified for the attributes that are part of the *primary key* of each relation, but it can be specified for any other attributes whose values are required not to be NULL, as shown in Figure 6.1. It is also possible to define a *default value* for an attribute by appending the clause **DEFAULT** <value> to an attribute definition. The default value is included in any

```
CREATE TABLE EMPLOYEE
    ( ... ,
      Dno          INT          NOT NULL       DEFAULT 1,
    CONSTRAINT EMPPK
      PRIMARY KEY (Ssn),
    CONSTRAINT EMPSUPERFK
      FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn)
                  ON DELETE SET NULL          ON UPDATE CASCADE,
    CONSTRAINT EMPDEPTFK
      FOREIGN KEY(Dno) REFERENCES DEPARTMENT(Dnumber)
                  ON DELETE SET DEFAULT       ON UPDATE CASCADE);
CREATE TABLE DEPARTMENT
    ( ... ,
      Mgr_ssn CHAR(9)          NOT NULL       DEFAULT '888665555',
      ... ,
    CONSTRAINT DEPTPK
      PRIMARY KEY(Dnumber),
    CONSTRAINT DEPTSK
      UNIQUE (Dname),
    CONSTRAINT DEPTMGRFK
      FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn)
                  ON DELETE SET DEFAULT       ON UPDATE CASCADE);
CREATE TABLE DEPT_LOCATIONS
    ( ... ,
    PRIMARY KEY (Dnumber, Dlocation),
    FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber)
                  ON DELETE CASCADE           ON UPDATE CASCADE);
```

Figure 6.2
Example illustrating how default attribute values and referential integrity triggered actions are specified in SQL.

new tuple if an explicit value is not provided for that attribute. Figure 6.2 illustrates an example of specifying a default manager for a new department and a default department for a new employee. If no default clause is specified, the default *default value* is NULL for attributes *that do not have* the NOT NULL constraint. Another type of constraint can restrict attribute or domain values using the **CHECK** clause following an attribute or domain definition.[6] For example, suppose that department numbers are restricted to integer numbers between 1 and 20; then, we can change the attribute declaration of Dnumber in the DEPARTMENT table (see Figure 6.1) to the following:

Dnumber INT **NOT NULL CHECK** (Dnumber > 0 **AND** Dnumber < 21);

The CHECK clause can also be used in conjunction with the CREATE DOMAIN statement.

For example, we can write the following statement:

CREATE DOMAIN D_NUM **AS** INTEGER

CHECK (D_NUM > 0 **AND** D_NUM < 21);

We can then use the created domain D_NUM as the attribute type for all attributes that refer to department numbers in Figure 6.1, such as Dnumber of DEPARTMENT, Dnum of PROJECT, Dno of EMPLOYEE, and so on.

Specifying Key and Referential Integrity Constraints

Because keys and referential integrity constraints are very important, there are special clauses within the CREATE TABLE statement to specify them. Some examples to illustrate the specification of keys and referential integrity are shown in Figure 6.1.7 The **PRIMARY KEY** clause specifies one or more attributes that make up the primary key of a relation. If a primary key has a *single* attribute, the clause can follow the attribute directly. For example, the primary key of DEPARTMENT can be specified as follows (instead of the way it is specified in Figure 6.1):

Dnumber INT **PRIMARY KEY**,

The **UNIQUE** clause specifies alternate (unique) keys, also known as candidate keys as illustrated in the DEPARTMENT and PROJECT table declarations in Figure 6.1. The **UNIQUE** clause can also be specified directly for a unique key if it is a single attribute, as in the following example:

Dname VARCHAR(15) **UNIQUE**, Referential integrity is specified via the **FOREIGN KEY** clause, as shown in Figure

6.1. As we discussed in Section 5.2.4, a referential integrity constraint can be violated when tuples are inserted or deleted, or when a foreign key or primary key attribute value is updated. The default action that SQL takes for an integrity violation is to **reject** the update operation that will cause a violation, which is known as the RESTRICT option. However, the schema designer can specify an alternative action to be taken by attaching a **referential triggered action** clause to any foreign key constraint. The options include SET NULL, CASCADE, and SET DEFAULT. An option must be qualified with either ON DELETE or ON UPDATE. We illustrate this with the examples shown in Figure 6.2. Here, the database designer chooses ON DELETE SET NULL and ON UPDATE CASCADE for the foreign key Super_ssn of EMPLOYEE. This means that if the tuple for a *supervising employee* is *deleted*, the value of Super_ssn is automatically set to NULL for all employee tuples that were referencing the deleted employee tuple. On the other hand, if the Ssn value for a supervising employee is *updated* (say, because it was entered incorrectly), the new value is *cascaded* to Super_ssn for all employee tuples referencing the updated employee tuple.8 In general, the action taken by the DBMS for SET NULL or SET DEFAULT is the same for both ON DELETE and ON UPDATE: The value of the affected referencing attributes is changed to NULL for SET NULL and to the specified default value of the referencing attribute for SET DEFAULT. The action for CASCADE ON DELETE is to delete all the referencing tuples, whereas the action for CASCADE ON UPDATE is to change the value of the referencing foreign key attribute(s) to the updated (new) primary key value for all the referencing tuples. It is the responsibility of the database designer to choose the appropriate action and to specify it in the database schema. As a general rule, the CASCADE option is suitable for "relationship" relations (see Section 9.1) , such as WORKS_ON; for relations that represent multivalued attributes, such as DEPT_LOCATIONS; and for relations that represent weak entity types, such as DEPENDENT.

Giving Names to Constraints

Figure 6.2 also illustrates how a constraint may be given a constraint name, following the keyword CONSTRAINT. The names of all constraints within a particular schema must be unique. A constraint name is used to identify a particular constraint in case the constraint must be dropped later and

replaced with another constraint, as we discuss in Chapter 7. Giving names to constraints is optional. It is also possible to temporarily *defer* a constraint until the end of a transaction, as we shall discuss in Chapter 20 when we present transaction concepts.

Specifying Constraints on Tuples Using CHECK

In addition to key and referential integrity constraints, which are specified by special keywords, other *table constraints* can be specified through additional CHECK clauses at the end of a CREATE TABLE statement. These can be called **row-based** constraints because they apply to each row *individually* and are checked whenever a row is inserted or modified. For example, suppose that the DEPARTMENT table in Figure 6.1 had an additional attribute Dept_create_date, which stores the date when the department was created. Then we could add the following CHECK clause at the end of the CREATE TABLE statement for the DEPARTMENT table to make sure that a manager's start date is later than the department creation date.

```
CHECK (Dept_create_date <= Mgr_start_date);
```

The CHECK clause can also be used to specify more general constraints using the CREATE ASSERTION statement of SQL. We discuss this in Chapter 7 because it requires the full power of queries, which are discussed in Sections 6.3 and 7.1.

Summary

In this chapter, we introduced the SQL database language. This language and its variations have been implemented as interfaces to many commercial relational DBMSs, including Oracle's Oracle; ibm's DB2; Microsoft's SQL Server; and many other systems including Sybase and INGRES. Some open source systems also provide SQL, such as MySQL and PostgreSQL. The original version of SQL was implemented in the experimental DBMS called SYSTEM R, which was developed at IBM Research. SQL is designed to be a comprehensive language that includes statements for data definition, queries, updates, constraint specification, and view definition.

We discussed the following features of SQL in this chapter: the data definition commands for creating tables, SQL basic data types, commands for constraint specification, simple retrieval queries, and database update commands. In the next chapter, we will present the following features of SQL: complex retrieval queries; views; triggers and assertions; and schema modification commands.