(f) Get employee numbers of employees who do not work on project COMP453.
(g) Get employee numbers of employees who work on all projects.
(h) Get employee numbers of employees who work on at least one project that employee 107 works on.

**5.11** Repeat Exercise 5.10 using QUEL.

**5.12** Repeat Exercise 5.10 using QBE.

**5.13** Express the queries of Exercise 4.16 of Chapter 4 using SQL or QUEL.

**5.14** Using SQL, get the *Empl_No, Skill,* and average chef's pay rate for the EMPLOYEE relation shown in Figure 5.4.

**5.15** For the sample tuples given in Figures 5.2 and 5.4, evaluate the QBE queries given in Examples 5.53 through 5.59.

**5.16** Repeat Exercise 5.15 for Examples 5.60 through 5.63.

**5.17** Consider a database for the Universal Hockey League (UHL), discussed in Chapter 2, which records statistics on teams, players, and divisions of the league. Write the following queries in SQL and QUEL:

(a) Give the names of the players who played as forwards in 1987 in the franchise Blades.
(b) Find the names of all the goalies who played with the forward Ozzy Xavier over the span of his hockey career.
(c) List forwards and their franchises for those forwards who had at least 50 goals in the years 1985 and 1986. A player must have at least 50 goals in both the years but may have been with two different franchises.
(d) Give the complete details of players who played in the same franchises as Ozzy Xavier did over his career, but not necessarily in the same year or as a forward.
(e) Compile the list of goalies who played during their career for franchises in St. Louis, Edmonton, and Paris. A goalie should be listed if and only if he played in all three cities.

## Bibliographic Notes

The query language QUEL was defined by Stonebraker et al. (Ston 76). It was also described in the paper by Wong and Youssefi (Wong 76) that described a method for the decomposition of the queries for query processing. The precursor of SQL, SEQUEL, was described by Chamberlin et al. (Cham 76). Commercial versions are described in manufacturers' reference manuals. QBE was proposed by Zloof (Zloo 77). The three languages are also described in varying detail in textbooks by Date (Date 86a), Korth and Silberschatz (Kort 86), Maier (Maie 83), and Ullman (Ullm 82). Semantics of updates and their application to relational databases are discussed in Desai et al. (Desa 87).

## Bibliography

(Astr 75) M. M. Astrahan & D. D. Chamberlin, "Implementation of a Structured English Query Language," *CACM* 18(10), 1975, pp. 580–587.

(Astr 76) M. M. Astrahan, et al., "System R: A Relational Approach to Database Management," *ACM TODS* 1(2), 1976, pp. 97–137.

(Boyc 75) R. F. Boyce, D. D. Chamberlin, W. F. King, & M. M. Hammer, "Specifying Queries as Relational Expressions: The SQUARE Data Sublanguage," *CACM* 18(11), 1975, pp. 621–628.

(Cham 76) D. D. Chamberlin, et al., "SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control," *IBM J. of Res. and Dev.* 20(6), 1976, pp. 560–575.

(Codd 86) E. F. Codd, "An Evaluation Scheme for Database Management Systems That Are Claimed To Be Relational," Data Engineering Conf., 1986, pp. 720–729.

(Codd 88) E. F. Codd, "Fatal Flaws in SQL," *Datamation,* August 15, 1988, pp. 45–48, September 1, 1988, pp. 71–74.

(Date 86a) C. J. Date, "A Critique of the SQL Database Language," *ACM SIGMOD Record,* November, 1984, Vol 14-3, pp. 8–54.

(Date 86b) C. J. Date, *An Introduction to Database Systems* 4th ed. Reading, MA. Addison-Wesley, 1986.

(Date 87a) C. J. Date, *A Guide to the SQL Standard*. Reading, MA: Addison-Wesley, 1987.

(Date 87b) C. J. Date, "Where SQL Falls Short," *Datamation,* May 1, 1987, pp. 83–86.

(Desa 87) B. C. Desai, P. Goyal, F. Sadri, "Fact Structures and its Application to Updates in Relational Databases," *Information Systems* 12(2), 1987, pp. 215–221.

(Epst 77) R., Epstein, R., "A Tutorial on INGRES," ERL-M77-25, University of California, December 1977. Berkeley, CA:

(Held 75) C. D. Held, M. Stonebraker, & E. Wong, "INGRES: A Relational Database System," Proc. ACM Pacific 1975 Regional Conf., 1975, pp. 409–416.

(Kala 85) J. Kalash, et al., "INGRES Version 8 Reference Manual," in *UNIX 4.3*. Berkley, CA: University of California, December 1985.

(Kort 86) H. F. Korth & A. Silberschatz, *Database System Concepts,"* New York: McGraw-Hill, 1986.

(Lawr 88) A. Lawrence, "Living Up to the Hype," *Computing,* May 5, 1988, pp. 22–23.

(Maie 83) D. Maier, *The Theory of Relational Databases,* Rockville, MD: *Computer Science Press,* 1983.

(Moad 88) J. Moad, "DB2 Performance Gets Kick with Closer Ties to 3090S, ESA," *Datamation,* September 1988, pp. 19–20.

(ORAC 87) *SQL*Plus Reference Guide*. Belmont, CA: Oracle Corp., July 1987.

(Pasc 88) F. Pascal, "SQL Redundancy and DBMS Performance," *Database Programming and Design,* December 1988, pp. 22–28.

(RTI 88) *INGRES Reference Guide*. Alameda, CA: Relational Technology Inc., August 1988.

(Ston 76) M. Stonebraker, E. Wong, P. Kreps, & C. D. Held, "The Design and Implementation of INGRES," *ACM TODS* 1(3), 1976, pp. 189–222.

(Ullm 82) J. D. Ullman, *Principles of Database Systems* 2nd ed. Rockville, Md: Computer Science Press, 1982.

(Wong 76) E. Wong & K. Youssefi, "Decomposition—A Strategy for Query Processing," *ACM Transactions on Database Systems* 1, pp. 223–241.

(Wood 79) J. Woodfill, et al., "INGRES Version 6.2 Reference Manual," ERL Technical Memorandum M79-43. Berkeley, CA: University of California, 1979.

(Zloo 75) M. M. Zloof, "Query-By-Example: Operations on the Transitive Closure." Yorktown Heights, NY: IBM Research Report RC5526, 1975.

(Zloo 77) M. M. Zloof, "Query-By-Example: A Database Language," *IBM Systems Journal* 16(4), 1977, pp. 324–343.

Chapter

6

# Relational Database Design

**Contents**

A relation in a relational database is based on a relation scheme, which consists of a number of attributes. A relational database is made up of a number of relations and the relational database scheme, in turn, consists of a number of relation schemes. In this chapter, we focus on the issues involved in the design of a database scheme using the relational model. Section 6.2 discusses the importance of having a consistent database without repetition of data and points out the anomalies that could be introduced in a database with an undesirable design. Section 6.3 presents the universal relation assumption. In Section 6.4 we look at some of the theoretical results from the functional dependency theory and present basic algorithms for the design process. In Section 6.5 we present the relational database design process. This process uses the functional dependencies among attributes to arrive at their desirable groupings. We discuss the first, second, third, and Boyce Codd normal forms and give algorithms for converting a relation in the first normal form into higher order normal forms. The next chapter introduces the synthesis approach to relational database design and higher order normal forms.

## 6.1 Relation Scheme and Relational Design

A relation scheme **R** is a plan that indicates the attributes involved in one or more relations. The scheme consists of a set S of attributes $\{A_1, A_2, \ldots, A_n\}$, where attribute $A_i$ is defined on domain $\mathbf{D_i}$ for $1 \leq i \leq n$. We will use **R(S)**, or **R** if there is no confusion, to indicate both the logical construction of the relation (its scheme) as well the name of this set $\mathbf{S}$ of attributes. Relation R on the relation scheme **R** is a finite set of mappings or tuples $\{t_1, t_2, \ldots, t_p\}$ such that for each $t_j \in R$, each of the attribute value $t_j(A_i)$ must be in the corresponding domain $\mathbf{D_i}$.

**Example 6.1**

Consider the relation SCHEDULE shown in Figure A. It contains the attributes *Prof, Course, Room, Max_Enrollment* (enrollment limit), *Day, Time*. Thus, the relation scheme for the relation SCHEDULE, say **SCHEDULE,**

**Figure A** The SCHEDULE relation.

| *Prof* | *Course* | *Room* | *Max_Enrollment* | *Day* | *Time* |
|---|---|---|---|---|---|
| Smith | 353 | A532 | 40 | mon | 1145 |
| Smith | 353 | A532 | 40 | wed | 1145 |
| Smith | 351 | C320 | 60 | tue | 115 |
| Smith | 351 | C320 | 60 | thu | 115 |
| Clark | 355 | H940 | 300 | tue | 115 |
| Clark | 355 | H940 | 300 | thu | 115 |
| Turner | 456 | B278 | 45 | mon | 845 |
| Turner | 456 | B278 | 45 | wed | 845 |
| Jamieson | 459 | D110 | 45 | tue | 1015 |
| Jamieson | 459 | D110 | 45 | thu | 1015 |

is *(Prof, Course, Room, Max_Enrollment, Day, Time)*. The domain of the attribute *Prof* (professors) is all the faculty members of the university; the domain of the attribute *Course* is the courses offered by the university; that of *Room* is all the rooms in the buildings of the university; that of *Max_Enrollment* is an integer value and indicates the maximum enrollment in the course (which is related to the capacity of the room, i.e., it should be less than or equal to the capacity of the room in which the course is scheduled). The domain of *Day* is {MON, TUE, WED, THU, FRI, SAT, SUN} and that of *Time* is the possible times of day. ■

The relation SCHEDULE of Figure A has ten tuples, the first one being *Prof* = Smith, *Course* = 353, *Room* = A532, *Max_Enrollment* = 40, *Day* = mon, *Time* = 1145. As mentioned earlier, the tabular representation of a relation is only for the purpose of illustration. Explicitly naming the columns of the table to show the mapping or association of an attribute and its value for a particular tuple avoids the requirement of a particular ordering of the attributes in the relation scheme and hence in the representation of the time-varying tuples of the relation. We will continue to represent relations as tables. We will also write the attributes of the relation in a particular order and show the tuples of the relation with the list of values for the corresponding attributes in the same order. The attribute names will be attached to the columns of the table when the tuples of a relation are shown in a table.

Since a relation is an abstraction of some portion of the real world that is being modeled in the database, and since the real world changes with time, the tuples of a relation also vary over time. Thus, tuples may be added, deleted, or updated over a period of time. However, the relation scheme itself does not change. (at least until the database is reorganized).

## 6.2 Anomalies in a Database: A Consequence of Bad Design

Consider the following relation scheme pertaining to the information about a student maintained by an university:

**STDINF***(Name, Course, Phone_No, Major, Prof, Grade)*

Figure 6.1 shows some tuples of a relation on the relation scheme **STDINF** ***(Name, Course, Phone_No, Major, Prof, Grade)***. The functional dependencies[1] among its attributes are shown in Figure 6.2. The key of the relation is *Name Course* and the relation has, in addition, the following functional dependencies {*Name* → *Phone_No*, *Name* → *Major*, *Name Course* → *Grade*, *Course* → *Prof*}.

Here the attribute *Phone_No*, which is not in any key of the relation scheme **STDINF**, is not functionally dependent on the whole key but only on part of the

[1]Recall the definition of functional dependency from Chapter 2, repeated here: Given attributes X and Y (each of which may contain one or more attributes), Y is said to be functionally dependent on X if a given value for each attribute in X uniquely determines the value of the attributes in Y. X is called the determinant of the functional dependency (FD) and the FD is denoted as X → Y.

student takes the course only once, or if he or she has to repeat it to improve the grade, the TRANSCRIPT relation stores only the highest grade.)

The third relation scheme records the teacher of each course.

One of the disadvantages of replacing the original relation scheme **STDINF** with the three relation schemes is that the retrieval of certain information requires a natural join operation to be performed. For instance, to find the majors of student who obtained a grade of A in course 353 requires a join to be performed: (STUDENT_INFO $\bowtie$ TRANSCRIPT). The same information could be derived from the original relation STDINF by selection and projection.

When we replace the original relation scheme **STDINF** with the relation schemes **STUDENT_INFO, TRANSCRIPT,** and **TEACHER,** the consistency and referential integrity constraints have to be enforced. The referential integrity enforcement implies that if a tuple in the relation TRANSCRIPT exists, such as (Jones, 353, in prog), a tuple must exist in STUDENT_INFO with *Name* = Jones and, furthermore, a tuple must exist in TEACHER with *Course* = 353. The attribute *Name,* which forms part of the key of the relation TRANSCRIPT, is a key of the relation STUDENT_INFO. Such an attribute (or a group of attributes), which establishes a relationship between specific tuples (of the same or two distinct relations), is called a foreign key. Notice that the attribute *Course* in relation TRANSCRIPT is also a foreign key, since it is a key of the relation TEACHER.

Note that the decomposition of **STDINF** into the relation schemes **STUDENT***(Name, Phone_No, Major, Grade)* and **COURSE***(Course, Prof)* is a bad decomposition for the following reasons:

1. Redundancy and update anomaly, because the data for the attributes *Phone_No* and *Major* are repeated.
2. Loss of information, because we lose the fact that a student has a given grade in a particular course.

The rest of this chapter examines the problem of the design of the relational database and how to decide whether a given set of relations is better than another set.

## 6.3 Universal Relation

Let us consider the problem of designing a database. Such a design will be required to represent a finite number of entity sets. Each entity set will be represented by a number of its attributes. If we refer to the set of all attributes as the universal scheme U then a relation R(U) is called the **universal relation.** The universal relation is a single relation made up of all the attributes in the database. The term **universal relation assumption** is the assumption that all relations in a database are derived from the universal relation by appropriate projection. The attribute names in the universal relation scheme **U** have to be distinct to avoid obvious confusion. One reason for using the universal relation assumption is to allow the user to view the database using such a relation. Consequently, the user does not have to remember the relation schemes and which attributes are grouped together in each such scheme.

Consider the relation $R_1$ *(Course, Department)* in Figure 6.3. The attribute *Department* is used to indicate the department responsible for the course. For instance,

**Figure 6.3** Relation $R_1$.

| *Course* | *Department* |
|---|---|
| 353 | Comp Sci |
| 355 | Mathematics |
| 456 | Mathematics |
| 221 | Decision Sci |

course 353 is offered by and is under the jurisdiction of the Comp(uter) Sci(ence) department.

The relation $R_2$*(Professor, Department)* of Figure 6.4 shows another role or interpretation of the attribute *Department;* here it is used to signify that a given professor is assigned to a given department. Thus, Smith is a member of the Comp Sci department. Note from Figures A, 6.3, and 6.4 that we are allowing for the incidence of a professor teaching a course in a outside department. Professor Clark of the Comp Sci department is teaching course 355 of the Mathematics department, and Professor Turner of the Chemistry department is teaching course 456, also of the Mathematics department.

The domain of the attribute *Department* in the relations $R_1$ and $R_2$ is the same, that is, all the departments in the university. Let us consider the representation of the data in the limited database indicated in Figures 6.3 and 6.4 as a universal relation $U_1$, where $U_1$ is defined as $U_1$*(Course, Department, Professor)*. The problem of using the universal relation $U_1$ becomes obvious when we try to represent the data from the relations $R_1$ and $R_2$ as shown in Figures 6.3 and 6.4. Here we have to decide whether or not data from different relations could appear in the same tuple of the universal relation. In Figure 6.5 we do not allow the data from different relations to appear in the same tuple of $U_1$, giving rise to a large number of empty or null values ($\perp$). These null values could signify one of three things: (1) the values, are not known, but they exist, (2) the values do not exist, or (3) the attribute does not apply. In case (1) we have to distinguish the null values by indicating them as $\perp_i$, and thus the two null values $\perp_i$ and $\perp_j$ (for $i \neq j$) are not equal and indicate that the values are not known to be the same.

In Figure 6.6, we have combined the data from the relations $R_1$ and $R_2$ in the same tuple of the universal relation $U_2$ with the scheme *(Course, Department, Pro-*

**Figure 6.4** Relation $R_2$.

| *Professor* | *Department* |
|---|---|
| Smith | Comp Sci |
| Clark | Comp Sci |
| Turner | Chemistry |
| Jamieson | Mathematics |

same tuple and the value of the attributes in Y must be determined by the key value. Similarly, if **R** represents a many-to-one relationship between two entities, say from $E_1$ to $E_2$, and if **X** contains attributes that form a key of $E_1$ and Y contains attributes that contain a key of $E_2$, again the FD **X → Y** will hold. But if **R** represents a one-to-one relationship between entity $E_1$ and $E_2$, the FD **Y → X** will hold in addition to :he FD **X → Y.**

Let **R** be a ɪelation scheme where each of its attribute $A_i$ is defined on some domain $\mathbf{D_i}$ for $1 \leq i \leq n$. Let **X, Y, Z,** etc. be subsets of $\{A_1, A_2, \ldots, A_n\}$. We will write **X ∪ Y** as simply **XY.**

Let R be a relation on the relation scheme **R.** Then R satisfies the functional dependency **X → Y** if a given set of values for each attribute in **X** uniquely determines each of the values of the attributes in **Y. Y** is said to be functionally dependent on **X.** The functional dependency (FD) is denoted as **X → Y,** where **X** is the left-hand side or the determinant of the FD and **Y** is the right-hand side of the FD. We can say that the FD **X → Y** is satisfied on the relation R if the cardinality of $\pi_Y(\sigma_{X=x}(R))$ is at most one. In other words, if two tuples $t_i$ and $t_j$ of R have the same **X** value, the corresponding value of **Y** will be identical.

A functional dependency **X → Y** is said to be **trivial** if **Y ⊆ X.**

**Example 6.2** In the relation SCHEDULE*(Prof, Course, Room, Max_Enrollment, Day, Time)* of Figure A, the FD *Course → Prof* is satisfied. However, the FD *Prof → Course* is not satisfied. ■

In order to verify if a given FD **X → Y** is satisfied by a relation R on a relation scheme **R,** we find any two tuples with the same **X** value; if the FD **X → Y** is satisfied in R, then the **Y** values in these tuples must be the same. We repeat this procedure until we have examined all such pairs of tuples with the same **X** value. A simpler approach involves ordering the tuples of R on the **X** values so that all tuples with the same **X** values are together. Then it is easy to verify if the corresponding **Y** values are also the same and verify if R satisfies the FD **X → Y.**

**Figure 6.8** The SCHEDULE relation.

| *Prof* | *Course* | *Room* | *Max_Enrollment* | *Day* | *Time* |
|---|---|---|---|---|---|
| Smith | 353 | A532 | 40 | mon | 1145 |
| Smith | 353 | A532 | 40 | wed | 1145 |
| Clark | 355 | H940 | 300 | tue | 115 |
| Clark | 355 | H940 | 300 | thu | 115 |
| Turner | 456 | B278 | 45 | mon | 845 |
| Turner | 456 | B278 | 45 | wed | 845 |
| Jamieson | 459 | D110 | 45 | tue | 1015 |
| Jamieson | 459 | D110 | 45 | thu | 1015 |

The FD $\mathbf{X} \to \mathbf{Y}$ on a relation scheme must hold for all possible relations defined on the relation scheme **R**. Thus, we cannot look at a table representing a relation on the scheme **R** at a point in time and say, simply by inspection, that some FD $\mathbf{X} \to \mathbf{Y}$ holds. For example, if the relation SCHEDULE at some point in time contained the tuples as shown in Figure 6.8, we might erroneously conclude that the FD {*Prof* → *Course*} holds. The examination of the real world situation corresponding to the relation scheme **SCHEDULE** tells us that a particular professor may be teaching more than one course.

**Example 6.3** In the relation scheme **STDINF** *(Name, Course, Phone_No, Major, Prof Grade)*, the following functional dependencies are satisfied: {*Name* → *Phone _No, Name* → *Major, Name Course* → *Grade, Course* → *Prof*}. ■

## 6.4.1 Dependencies and Logical Implications

Given a relation scheme **R** and a set of functional dependencies **F**, let us consider a functional dependency $\mathbf{X} \to \mathbf{Y}$, which is not in **F**. **F** can be said to logically imply $\mathbf{X} \to \mathbf{Y}$ if for every relation R on the relation scheme **R** that satisfies the functional dependencies in **F**, R also satisfies $\mathbf{X} \to \mathbf{Y}$.

**F** logically implies $\mathbf{X} \to \mathbf{Y}$ is written as $\mathbf{F} \models \mathbf{X} \to \mathbf{Y}$.

**Example 6.4** $\mathbf{R} = (A, B, C, D)$ and $\mathbf{F} = \{A \to B, A \to C, BC \to D\}$, then $\mathbf{F} \models A \to D$. ■

### Inference Axioms

Suppose we have **F**, a set of functional dependencies. To determine whether a functional dependency $\mathbf{X} \to \mathbf{Y}$ is logically implied by **F** (i.e., $\mathbf{F} \models \mathbf{X} \to \mathbf{Y}$), we use a set of rules or axioms. The axioms are numbered **F1** through **F6** to indicate that they pertain to functional dependencies (as opposed to multivalued dependencies, which we examine in Chapter 7).

In the following discussions, we assume that we have a relation scheme $\mathbf{R}(A_1, A_2, A_3, \ldots, A_n)$; R is a relation on the relation scheme **R** and **W, X, Y, Z** are subsets of **R**. The symbol ⊧ used below is read as "logically implies."

- **F1: Reflexivity:** $(\mathbf{X} \to \mathbf{X}$ and $\mathbf{V} \subseteq \mathbf{Z})$
- **F2: Augmentation:** $(\mathbf{X} \to \mathbf{Y}) \models (\mathbf{XZ} \to \mathbf{Y}$, and $\mathbf{XZ} \to \mathbf{YZ})$
- **F3: Transitivity:** $(\mathbf{X} \to \mathbf{Y}$ and $\mathbf{Y} \to \mathbf{Z}) \models (\mathbf{X} \to \mathbf{Z})$
- **F4: Additivity:** $(\mathbf{X} \to \mathbf{Y}$ and $\mathbf{X} \to \mathbf{Z}) \models (\mathbf{X} \to \mathbf{YZ})$
- **F5: Projectivity:** $(\mathbf{X} \to \mathbf{YZ}) \models (\mathbf{X} \to \mathbf{Y}$ and $\mathbf{X} \to \mathbf{Z})$
- **F6: Pseudotransitivity:** $(\mathbf{X} \to \mathbf{Y}$ and $\mathbf{YZ} \to \mathbf{W}) \models (\mathbf{XZ} \to \mathbf{W})$

**Example 6.6** Let **R** = $(A, B, C, D)$ and **F** = $\{A \rightarrow B, A \rightarrow C, BC \rightarrow D\}$. Since $A \rightarrow B$ and $A \rightarrow C$, then by **F4** $A \rightarrow BC$. Now since $BC \rightarrow D$, then by **F3** $A \rightarrow D$, i.e., **F** $\models A \rightarrow D$ and thus $A \rightarrow D$ is in $\mathbf{F}^+$. ■

An example of an FD not implied by a given set of FDs is illustrated below.

**Example 6.7** Let **F** = {**W** → **X**, **X** → **Y**, **W** → **XY**}. Then $\mathbf{F}^+$ includes the set {**W** → **W**, **X** → **X**, **Y** → **Y**, **W** → **X**, **X** → **Y**, **W** → **XY**, **W** → **Y**}. The first three FDs follow from axiom **F1**; the next three FDs are in **F** and hence in $\mathbf{F}^+$. Since **W** → **XY**, then by axiom **F5**, **W** → **X** and **W** → **Y**. However, $\mathbf{F}^+$ does not contain an FD, e.g., **W** → **Z**, because **Z** is not contained in the set of attributes that appear in **F**. ■

## 6.4.3 Testing if F ⊨ X → Y: Algorithm to Compute a Closure

To compute the closure $\mathbf{F}^+$ for a set of FD **F** is a lengthy process because the number of dependencies in $\mathbf{F}^+$, though finite, can be very large. The reason for computing $\mathbf{F}^+$ is to determine if the set of FDs **F** ⊨ **X** → **Y**; this would be the case if and only if **X** → **Y** ∈ $\mathbf{F}^+$. However, there is an alternative method to test if **F** ⊨ **X** → **Y** without generating $\mathbf{F}^+$. The method depends on generating $\mathbf{X}^+$, the closure of **X** under **F**.

**Definition:** **The closure of X under a set of functional dependencies F, written as $\mathbf{X}^+$, is the set of attributes $\{A_1, A_2, \ldots, A_m\}$ such that the FD X → $A_i$ for $A_i \in \mathbf{X}^+$ follows from F by the inference axioms for functional dependencies.**

$\mathbf{X}^+$, the closure of **X** with respect to the set of functional dependencies **F**, is the set of attributes $\{A_1, A_2, A_3, \ldots, A_m\}$ such that each of the FDs **X** → $A_i$, $1 \leq i \leq m$ can be derived from **F** by the inference axioms. Also, by the additivity axiom for functional dependency, **F** ⊨ **X** → **Y** if **Y** ⊆ $\mathbf{X}^+$. (By the completeness of the axiom system, if **F** ⊨ **X** → **Y**, then **Y** ⊆ $\mathbf{X}^+$—see lemma below.)

Having found $\mathbf{X}^+$, we can test if **F** ⊨ **X** → **Y** by checking if **Y** ⊆ $\mathbf{X}^+$: **X** → **Y** is logically implied by **F** if and only if **Y** ⊆ $\mathbf{X}^+$.

We now present the algorithm to compute the closure $\mathbf{X}^+$ given a set of FDs **F** and a set of attributes **X**. The importance of computing the closure $\mathbf{X}^+$ is that it can be used to decide if any FD **X** → **Y** can be deduced from **F**. The following lemma establishes that if **Y** ⊆ $\mathbf{X}^+$ then **F** ⊨ **X** → **Y**.

**Lemma:** **F** ⊨ **X** → **Y** if and only if **Y** ⊆ $\mathbf{X}^+$.

**Proof:** Suppose that **Y** ⊆ $\mathbf{X}^+$. Then by the definition of $\mathbf{X}^+$, **X** → $A$ can be derived from **F** using the inference rules for each $A$ ∈ **Y**. By the soundness of these rules, **F** ⊨ **X** → $A$ for each $A$ ∈ **Y** and by the additivity rule, **F** ⊨ **X** → **Y**. Now suppose that **F** ⊨ **X** →

Algorithm **6.1** **Algorithm to Compute $X^+$**

*Input:* A set of functional dependencies **F** and a set of attributes **X**.

*Output:* The closure $X^+$ of **X** under the FDs in **F**.

```
X⁺ := X; (* initialize X⁺ to X *)
change := true;
while change do
    begin
    change := false;
    for each FD W → Z in F do
        begin
        if W ⊆ X⁺ then do
                    begin
                    X⁺ := X⁺ ∪ Z;
                    change := true;
                    end
        end
    end
(* X⁺ now contains the closure of X under F *)
```

**Y**. Then by completeness of the inference rules, $\mathbf{X} \rightarrow \mathbf{Y}$ can be derived from **F** using them. By projectivity, $\mathbf{X} \rightarrow A$ can be derived for each $A \in \mathbf{Y}$. This clearly implies that $\mathbf{Y} \subseteq \mathbf{X}^+$ by the definition of $\mathbf{X}^+$.

Algorithm 6.1 to compute $\mathbf{X}^+$ follows. It starts with the set $\mathbf{X}^+$ initialized to **X**, the left-hand side of the FD $\mathbf{X} \rightarrow \mathbf{Y}$, which is to be tested for logical implication under **F**. For each FD $\mathbf{W} \rightarrow \mathbf{Z}$ in **F**, if $\mathbf{W} \subseteq \mathbf{X}^+$, the algorithm modifies $\mathbf{X}^+$ by torming a union of $\mathbf{X}^+$ and **Z**. The algorithm terminates when there is no change in $\mathbf{X}^+$.

**Example 6.8** Let $\mathbf{X} = BCD$ and $\mathbf{F} = \{A \rightarrow BC, CD \rightarrow E, E \rightarrow C, D \rightarrow AEH, ABH \rightarrow BD, DH \rightarrow BC\}$. We want to compute the closure $\mathbf{X}^+$ of **X** under **F**. We initialize $\mathbf{X}^+$ to **X**, i.e., $\mathbf{X}^+ := BCD$. Now since the left-hand side of the FD $CD \rightarrow E$ is a subset of $\mathbf{X}^+$, i.e., $CD \subseteq \mathbf{X}^+$, $\mathbf{X}^+$ is augmented by the right-hand side of the FD, i.e., $E$; thus $\mathbf{X}^+$ now becomes equal to $BCDE$. Similarly, since $D \subseteq \mathbf{X}^+$, the right-hand side of the FD $D \rightarrow AEH$ is added to $\mathbf{X}^+$, which now becomes $ABCDEH$. $\mathbf{X}^+$ cannot be augmented any further and Algorithm 6.1 ends with $\mathbf{X}^+$ equal to $ABCDEH$. ■

The time complexity of the closure algorithm can be derived as follows. Suppose the number of attributes in **F** is **a** and the number of FDs in **F** is **f** where each FD in **F** involves only one attribute on the right-hand side. Then the inner for loop will be executed at most **f** times. one for each FD in **F**, and each such execution can take the time proportional to **a** to check if one set is contained in another set. Thus the order of execution of the for loop is **O(af)**. In the worst case each execution of the while loop can increase the closure by one element and since there are **f** FDs, the while loop can be repeated at most **f** times. Hence the time complexity of the algorithm is $\mathbf{O(af^2)}$. The algorithm can be modified to run in time proportional to the number of symbols needed to represent the FDs in **F**. The modification takes into account the fact that the FDs whose right-hand sides are already added to $\mathbf{X}^+$ need not be reconsidered in the for loop. Furthermore, the FDs whose left-hand side lengths are greater than the current length of $\mathbf{X}^+$ need not be tested in the for loop. See the bibliographic notes for reference to a closure algorithm with these modifications.

## 6.4.4 Testing if an FD is in a Closure

As mentioned earlier, to find out whether $\mathbf{F} \models \mathbf{X} \rightarrow \mathbf{Y}$ without computing $\mathbf{F}^+$ requires the computation of $\mathbf{X}^+$ under the set of FDs **F**, and if $\mathbf{Y} \subseteq \mathbf{X}^+$ then **F** logically implies the functional dependency $\mathbf{X} \rightarrow \mathbf{Y}$, otherwise it does not. Algorithm 6.2 tests the membership of $\mathbf{X} \rightarrow \mathbf{Y}$ in $\mathbf{F}^+$ by this indirect scheme. It uses Algorithm 6.1 to compute the closure of **X** under **F**.

**Example 6.9** Let $\mathbf{F} = \{A \rightarrow BC, CD \rightarrow E, E \rightarrow C, D \rightarrow AEH, ABH \rightarrow BD, DH \rightarrow BC\}$. We want to find if $\mathbf{F} \models BCD \rightarrow H$.
Having computed $BCD^+$ as being $ABCDEH$ we can clearly see that the FD $BCD \rightarrow H$ is implied by the FD **F** since $H \subseteq BCD^+$. ■

The time complexity of the membership algorithm is similar to the closure algorithm because the membership algorithm uses the closure algorithm.

## 6.4.5 Covers

Given a set of FDs **F**, $\mathbf{F}^+$ is the closure of **F** and contains all FDs that can be derived from **F**. As mentioned earlier, $\mathbf{F}^+$ can be very large; hence, we will look for a smaller set of FDs that are representative of the closure of **F**. Suppose we have another set of FDs **G**. We say that **F** and **G** are equivalent if the closure of **F** is identically equal to the closure of **G**, i.e., $\mathbf{F}^+ = \mathbf{G}^+$. If the sets of FDs **F** and **G** are equivalent, we can consider one to be representative of the other or one **covers** the other. Thus **F** covers **G** and **G** covers **F**.

***Definition:*** **Given two sets of FDs F and G over a relation scheme R, F and G are equivalent (i.e., F ≡ G) if the closure of F is identically equal to the closure of G (i.e., $\mathbf{F}^+ = \mathbf{G}^+$). If F and G are equivalent, then F covers G and G covers F.**