Database Systems for Management Third edition

James F. Courtney

David B. Paradice

Kristen L. Brewer

Julia C. Graham

For any questions about this text, please email: drexel@uga.edu

The Global Text Project is funded by the Jacobs Foundation, Zurich, Switzerland

This book is licensed under a Creative Commons Attribution 3.0 License

CHAPTER CONTENTS

CHAPTER 6
The Relational Database Model

In the previous chapter we presented an overview of several database models. Here and in Chapter 7 we examine one model, the relational database model, in detail.

BASIC CONCEPTS

The relational database model was originally proposed by E.F. Codd in the early 1970s. This model presents data in the form of tables, a way of picturing data with which almost everyone is comfortable. The relational model has its foundation in set theory, which provides the basis for many of the operations

performed on relations. This theoretical basis also allows one to distinguish relational from "relational-like" database systems.

Characteristics of Relations

Figure 6-1 presents an Entity-Relationship model of a part of a database for employees and projects in a hypothetical organization. The relations shown in Figure 6-2 are derived from this logical description.

For the purposes of this chapter, a relation is a two-dimensional table of data about an entity class or the relationships between entity classes. Figure 6-2 illustrates six relations: EMPLOYEE, PROJECT, ASSIGNMENT, VACATION TAKEN, SICK LEAVE TAKEN, and TIMESHEETS. These relations exhibit the following characteristics:

1. all entries at the intersection of a row and column of the relation are single-valued (there are no arrays or repeating groups);

2. the entries in any column of the relation are all of the same kind;

3. each column has a unique name (although columns of different relations may have the same name);

4. no two rows of the relation are identical.

The order of rows does not matter and, in general, the order of the columns does not matter. This data is typical organizational information. Obviously, one would expect an organization to keep much more detailed information than this, but this amount suffices to demonstrate how an organizational database in a relational database model can support decision-making processes.

The rows of a relation are called **tuples** (rhymes with "couples"). The row in the EMPLOYEE relation that contains the values 847478599, Clarke, 373 Ave A, Smyrna, 30364, 837-9383, Consultant, 2500, 4, and 5 is one tuple. The number of tuples in a relation determines its **cardinality.** The EMPLOYEE relation has cardinality 8.

The columns of a relation are called **attributes.** Name is an attribute of the EMPLOYEE relation, as is E-ID. The values an attribute can assume are said to come from that attribute's **domain.** The domain of the E-ID attribute in the EMPLOYEE relation is the set of nine-digit numbers. You may notice that the values for this attribute appear to be social security numbers.

The PROJECT relation has an attribute named P-ID, but it does not appear to be the same as E-ID in the EMPLOYEE relation. Indeed, it is not the same, as it is defined on a different domain. The domain for P-ID in the PROJECT relation is a set of four digit numbers, perhaps authorized by the company's controller.

The number of attributes in a relation determines the **degree** of the relation. A relation with six attributes, such as the PROJECT relation, is said to be of degree 6. Additionally, a tuple from this relation can be referred to as a "six-tuple." A relation with only one attribute is called a **unary relation,** one with only two attributes is called a **binary relation,** and one with only three attributes is called a **ternary relation.**

Two types of relations can be defined. A **base relation** exists independently of all other relations in the database. A **view** (or derived relation) is a relation that can be constructed from other relations in the database. Figure 6-3 illustrates these concepts. The PAID LEAVE relation is formed by combining the Vacation Taken and the SICK LEAVE TAKEN relations. The PAID LEAVE relation does not actually exist in the database. To have it in the database would be very inefficient and would cause the problems related to

data redundancy discussed in Chapter 1. Whether a relation is a base relation or a view, the relational structure is time-invariant. The relationships between the attributes do not vary with time. Of course the values assumed by the attributes for a specific tuple can vary with time.

The relational database model terminology is summarized in Table 6-1.

Null Values

There are occasions when knowing all of the data values for an entity is impossible. For example, a hospital emergency room patient may be incapable of giving needed data or a suspect in a crime may be unwilling to give needed data. Similarly, future values of data items may be impossible to determine and may only be estimated.

We noted above that the data value at the intersection of a relation row and column must be atomic, or single-valued. Throughout the text we have generally assumed that data values would exist and be known for attributes stored in a database. However, such is not always the case, and because the relational model puts some restrictions on when and where data values may be omitted, we shall consider this point further before continuing.

An unspecified value is a **null value**. Null values are different from zero values or blank values. A zero value represents the numeric quantity zero. A null value represents one of two cases: the actual value is unknown or the attribute is not applicable. We will represent a null value with the symbol -0-.

Consider the ASSIGNMENTS relation from Figure 6-4. It contains null entries for the Total Hours and Evaluation attributes. These values indicate the total number of hours worked on project 2651 by employee 383783993 is unknown. Similarly, the evaluations of employee 252928462 on project 6301 and employee 383783993 on project 2651 are also unknown.

We will see how null values can be manipulated in the section about data manipulation below.

Keys

Since (by definition) no two tuples in a relation are identical, there is always at least one way to identify a particular tuple--by specifying all attributes (data values) of the tuple. Usually a tuple can be uniquely identified by specifying fewer than all attributes. Attributes or collections of attributes that uniquely identify a tuple are called **candidate keys.** In the PROJECT relation, Name and P-ID are candidate keys. Each project has a unique name and a unique project number.

The VACATION TAKEN relation has two candidate keys. E-ID and Begin Date combined form one. E-ID and End Date combined form the other. Although the combination of E-ID, Begin Date, and End Date cold be considered another candidate key, we will always limit our consideration to the candidate keys with the smallest number of attributes. E-ID alone is not a candidate key if we make the reasonable assumption that the relation may contain a record of more than one vacation taken by an employee. The date fields alone are not reasonable choices for candidate keys if we assume more than one employee may begin or end a vacation on the same date. Similarly, the Authorization attribute is not a candidate key if a manager can authorize more than one employee's vacation.

When a candidate key is selected to be the key of a relation, it is called the **primary key.** As with candidate keys, we are only interested in primary keys constructed of the fewest number of attributes possible. In the illustrations in this chapter, the attribute names for primary keys are underlined, as the E-ID in the EMPLOYEE relation. When two attributes compose a primary key, we imply that the values for those two

attributes considered together form a combination that is unique to that tuple. The combination of E-ID and Begin Date in the VACATION TAKEN relation form a unique set of values for each tuple. We would expect only one entry for an employee on a given date. Logically speaking, an employee may begin only one vacation on a given date.

An attribute is frequently defined on the same domain as a primary key in a relation. If this attribute is also logically related to the attribute serving as primary key, then it is called a **foreign key.** The Authorization attribute in the VACATION TAKEN relation is a foreign key, since it is defined on the same domain as E-ID in the EMPLOYEE relation.

The two attributes that form this primary key / foreign key relationship may exist in the same relation. Consider the modified version of the EMPLOYEE relation shown in Figure 6-5. Here, Manager ID has been added as an attribute. Since a manager must also be an employee, the Manager ID attribute is a foreign key to the E-ID attribute in the same relation.

A foreign key establishes a relationship by default between a relation containing the foreign key and the relation in which the primary key related to the foreign key exists. This situation occurs because the existence of a value for the attribute implies that the same value should exist in another relation as a primary key. In the discussion of referential integrity we explore this situation in more detail.

For example, suppose the VACATION TAKEN relation contained the data shown in Figure 6-6. Here, the last tuple shown contains a value for authorization that does not correspond to an existing manager's identification number. Thus, one must question the validity of the authorization.

ENTITY RELATIONSHIPS

A relation provides data to a decision maker about an entity class. Usually a decision maker is also interested in relationships between data items. If you have a relation containing information about airlines (say airline name and telephone number) and another relation containing information about flights (say destination, departure, and arrival time), but no way of relating flights to the appropriate airlines, you will have a difficult time making a reservation. (Be careful to distinguish between "relation" and "relationship" here and throughout this chapter.)

Relationships Within Entities

Two types of relationships can be distinguished in the relational model. The first type is the obvious one that occurs within a tuple due to the existence of data for a particular attribute. The EMPLOYEE relation in Figure 6-2, for example, has been designed so that all of the relevant data about a single employee appears in a tuple. We know that a consultant named Smith who lives on 28 Alma Street in Smyrna, has five vacation days available because the EMPLOYEE relation contains a tuple for Smith with these values.

Relationships Between Entities

Trees The second type of relationship occurs between relations. In Chapter 3 we presented the concept of a tree, or a one-to-many relationship. An example of a one-to-many relationship can be seen in the EMPLOYEE and ASSIGNMENT relations (see Figure 6-2): one employee works on many projects.

This relationship is designed into the relations by repeating the primary key for an employee in the PROJECT relation. In fact, all relationships between relations are designed this way. (This aspect of the relational database model is discussed in more detail shortly.) A consequence of this method of building relationships is that the concept of a domain takes on great importance. The domains of the attributes must

be the same, not just similar.

Networks Designing a relational schema to model a more complicated relationship is not too difficult. Our previous example was really a subset of the relationship between projects and employees. This relationship is many-to-many: an employee may be assigned to many projects and a project may engage many employees.

The design reflects the decomposition of this complex network structure into a simple network. The ASSIGNMENTS relation is the intersection relationship that implements the decomposition. As we noted in Chapter 3, information specific to the intersection relationship can be modeled here. The ASSIGNMENT relation contains the total number of hours a particular employee has worked on a particular project, and the employee's evaluation on that project.

The relationships can be seen by focussing on the key values. As we noted above, the employee's identification number, which is the primary key in the EMPLOYEE relation, appears multiple times in the ASSIGNMENTS relation. The project's identification number, which is the primary key in the PROJECT relation, appears multiple times in the ASSIGNMENTS relation. The EMPLOYEE - ASSIGNMENTS one-to-many relationship implements the first one-to-many relationship needed in the decomposition. The PROJECT - ASSIGNMENTS relationship implements the second one-to-many relationship needed for the decomposition. Notice that the primary key in the intersection relation is the combination of the primary keys for the two relations in the complex network.

Thus far, we have discussed primarily superficial characteristics of the relational model. The theory behind this model is much more comprehensive and may be discussed in terms of three distinguishing features of the model. These features describe constraints on the data structures available, relationships that must be maintained, and operations that perform data manipulation in a relational database environment.

Essentiality

We are now in a position to address the first of three distinguishing features of the relational database model. The only data structure that has been discussed thus far is the relation. In fact, the relation is the only data structure provided in the relational database model. There are no links or pointers. To appreciate the significance of this aspect of the model, one must recognize the additional complexity associated with a linking data structure. The data structure in Figure 6-7 is similar to the EMPLOYEE and VACATION TAKEN relations in Figure 6-2, but the employee identification number attribute in the VACATION TAKEN relation has been replaced with links from the EMPLOYEE tuples to the proper VACATION TAKEN tuples. The links represent physical pointers from the employee records to the vacation records. To determine the vacation that has been taken by an employee, the database system must process the link in some manner. Hence the link is essential to determining the information. This example illustrates the concept of **essentiality.** A data structure is essential if its removal causes a loss of information from the database. A loss of information obviously includes a lost relationship as well as a lost attribute.

In our example, the EMPLOYEE and VACATION TAKEN relations and their attributes are all essential. Each logical unit of information in Figure 6-7 requires two rows and a link, all of which are essential. In the original data structures in Figure 6-2, however, the link is not essential. It adds only complexity to the processing; removing it does not cause any loss of information.

The essentiality concept appears in other aspects of the relational model. Recall that the order of the tuples in a relation does not matter. In fact, requiring tuples to be ordered violates the premise that the relation is the only essential data structure. A required ordering adds complexity to the processing of the model and implies a loss of information should the tuples become unordered.

The fact that the relation is the only essential data structure in the relational model separates this model from all others. All other models explicitly include an essential linking structure. As we have seen above, models that include inessential linking structures can be converted to equivalent relational models.

Integrity Rules

The second distinguishing feature of the relational database model involves integrity rules. Integrity concerns the overall assurance that the content of the database (taken as a whole) is accurate and consistent. Since the content of the database is defined by the base relations, the integrity rules need apply only to the base relations. Two types of integrity are defined in the relational database model: entity integrity and referential integrity.

Entity integrity gives one the ability to uniquely identify each entity in the relational database. This ability guarantees access to all data. Because each entity is described by a tuple, entity integrity is supported by the requirement that no primary key (or attribute composing a primary key) have a null value. In addition, once the primary key is established, no change may be made which corrupts the ability of the primary key to uniquely identify a tuple.

Entity integrity follows from the fact that the relation models entities that really exist. We have noted earlier in the book that we are only interested in modeling entities that are real and distinguishable. One would not expect any need to specify a null value for the attributes that distinguish one entity from another. A null value would imply either that the attribute does not exist or that it is unknown. It must exist, otherwise our assumption that the entities are distinguishable is erroneous. If it is unknown, we might be storing redundant data since the data may actually be for an entity already represented in the relation. Also, the primary key provides the only way to access a tuple, hence the primary key must have a value to support access.

Referential integrity gives one the ability to reference tuples by way of foreign keys. Referential integrity is supported by the requirement that the values assumed by a foreign key either match a primary key that exists in the database or be completely null. This rule ensures that if another tuple is referenced in the database, then that tuple exists.

Figure 6-6 illustrates the absence of referential integrity. Here we have introduced a new tuple in the VACATION TAKEN relation. Note that the last tuple in this relation has the value 837954503. This value should reference a valid employee identification number. Specifically, it should reference a manager's identification number. This value, however, does not appear in the EMPLOYEE relation. The VACATION TAKEN relation therefore refers to a nonexistent manager in the EMPLOYEE relation. The validity of this information is questionable, since the authorization appears suspect.

Referential integrity also follows naturally from the assumption that the database models entities that actually exist. We expect that any value for an attribute that can be used to reference a tuple in another relation should be a valid value. Unlike the case for primary keys in entity integrity, however, the value of a foreign key may be null and may not be unique. If, for example, we focus only on the vacations authorized by Wray, we see that the value for Authorization in VACATION TAKEN is 252022010 and it appears several times. We could also enter into this relation a tuple in which the authorization attribute contains a null value. This entry would imply that the Authorization was unknown. If it is unknown, it does not reference any employee. Thus, it cannot reference a nonexistent employee.

Data Manipulation

The third distinguishing feature of the relational model is data manipulation. A database is of relatively little value unless some means exists to manipulate the data. Data manipulation is a primary way to create information for decision making. Several types of data manipulation language (DML) have been developed for use with the relational model. Some data manipulation languages are very formal in the way they define mathematical operators for manipulating relations. Others are less formal (at least compared to the rigors of mathematics) and are more like high-level programming languages.

Relational algebra provides a set of operators for manipulating entire relations. Relational algebra is considered a procedural approach to data manipulation because the user must specify in a specific order how the relational algebra operators will be used. **Relational calculus** is a nonprocedural set of operators that provide great flexibility in manipulating the data in relations. These operators are said to be nonprocedural because the user only specifies what the desired data is, not how to get it. Relational algebra and relational calculus are presented in detail in the following sections.

Transformation languages are high-level languages that transform input relations into the desired output relations. One example of a transformation language is SQL, which is discussed in Chapter 7. **Graphic languages** display pictures of relations; the user fills in the appropriate attribute with an example of the desired data. A graphic language called QBE is covered in Appendix D.

RELATIONAL ALGEBRA

Relational algebra operations manipulate entire relations. Relational algebra has the characteristic of **closure,** which means an operation on a relation (or several relations) always produces another relation. Closure is desirable because it reduces the complexity of database processing by eliminating any "special cases." Operations in relational algebra always produce a valid relation, never anything else. Because the relational database model is based on set theory, many of the algebra operations closely resemble set operations. We will review each operation intuitively in the sections that follow. In each section, we will also provide a formal description of how the operation works. The relational algebra operations are summarized in Table 6-2.

Projection

Sometimes a database user needs only one or a few attributes in a relation. The payroll function of our illustrated organization, for example, probably needs only the name and employee number from the EMPLOYEE relation in order to print payroll checks. An operation for choosing specific attributes to be displayed is called **projection.** Because the payroll function requires data only on the names and numbers of employees, EMPLOYEE can be "projected over" E-ID and Name and the result stored in a relation named EMPLOYEE-NUMBER-NAME. The result of such a command is presented in Figure 6-8b. Note that the order of the attributes specified in the command corresponds to the order of the attributes in the new relation.

Closure ensures a projection always produces a relation. Thus, duplicate tuples are always deleted from the result. If PROJECT is projected over the Type attribute, the result contains only three tuples. If the specific values for project type were repeated in the result, the table would contain duplicate rows since the value E and C would occur twice. Valid relations, however, do not contain duplicate tuples.

We can formally specify the projection operation as follows:

Given a relation R with attributes $<A_1, A_2, ..., A_n>$, the projection of R over attributes $<A_i, A_j, ...>$ for any A_i, A_j, ... that appear in R produces a new relation S with attributes $<A_i, A_j, ...>$.

Selection

Occasionally a user desires only a subset of tuples from a relation. The executive vice-president of this organization, for example, may wish to have information on projects to answer a question like "What is the status of enhancement projects?" The vice-president could select from PROJECT all tuples where Type has the value "E" and store the result in ENHANCEMENTS. This yields the relation in Figure 6-9b. The **selection** operation is used to obtain this type of result.

The result of selection contains all attributes in the relation. But why include the Type attribute when it is already known that the value is "E" in every tuple? Most users do not want to process more information than they need. Suppose a manager needs only a list of names of all programmers. Selection and projection can be combined to specify exactly this information. The result of selecting from EMPLOYEE all tuples in which Title has the value "Programmer" can be projected over Name to produce the relation PROGRAMMERS as shown in Figure 6-9c.

We can formally specify the selection operation (involving one attribute) as follows:

Given a relation R with attributes $<A_1, A_2, ..., A_n>$, where A_1 has domain $D_1 = <d_{11}, d_{12}, ..., d_{1k}>$, A_2 has domain $D_2 = <d_{21}, d_{22}, ..., d_{2m}>$, ..., and A_n has domain $D_n = <d_{n1}, D_{n2}, ... d_{np}>$, and given a specific value v and an operation Θ valid on domain D_k, the selection from R where A_k has relationship Θ with v for any A_k that appears in R produces a new relation S with attributes $<A_1, A_2, ..., A_n>$ which contains only tuples whose value for A_k has relationship Θ with v.

Union

The preceding examples draw data from a single relation. Frequently one must combine data from two or more relations to get needed data. Suppose that the data on all paid absences of the organization's personnel is needed. In this case, data is needed from both the VACATION TAKEN and SICK LEAVE TAKEN relations.

Two relations can be combined vertically to form a third relation by the **union** operation. There are two restrictions: first, the two relations must have the same number of attributes (have the same degree); second, the attributes in the corresponding columns must assume values from the same domain. Two relations that satisfy these criteria are said to be **union compatible.** VACATION TAKEN and SICK LEAVE TAKEN are union compatible. The union of VACATION TAKEN and SICK LEAVE TAKEN produces the relation named PAID LEAVE TAKEN in Figure 6-10b.

You may recall that the PAID LEAVE TAKEN relation was introduced earlier in the chapter (see Figure 6-3). It illustrates a view, or derived relation. Now you know how the view is constructed.

We will formally define union compatibility first, then define the union operation.

Given a relation R with attributes $<A_1, A_2, ..., A_n>$ and a relation S with attributes $<B_1, B_2, ..., B_n>$, R and S are union compatible if the domain of A_i is the same as the domain of B_i for all attributes in R and S.

Given two union compatible relations R and S, the union of R and S produces a relation T which contains all of the tuples of R and all of the tuples of S subject to the constraint that T is a valid relation.

Intersection

Consider the revised versions of the VACATION TAKEN and SICK LEAVE TAKEN relations in Figure 6-11a. In particular, notice that the data for the last tuple in each relation is identical. Typical accounting principles dictate that the absence should be accounted for as either vacation time or sick leave, but not both. In a large corporate database, examining all entries in relations such as these to find possibly one or two mistakes like this would be very difficult, perhaps impossible.

The **intersection** relational algebra operation can perform this task. Intersection generates a relation which contains only tuples that appear in both relations. Only the tuple for employee 383783993 with a beginning date of 11/20 and an ending date of 11/21 and so forth occurs in both VACATION TAKEN and SICK LEAVE TAKEN. Thus, it is the only one in the resulting relation, shown in Figure 6-11b. As before, the relations must be union compatible to perform the intersection operation.

We can formally specify the intersection operation as follows:

Given two union compatible relations R and S, the intersection of R and S produces a relation T which contains tuples that occur in both R and S.

Difference

Consider the following problem. A list of all non-managers is required. In the small database from Figure 6-12, the problem appears trivial. One might use selection to obtain a relation containing tuples for programmers. Then one might use selection again to form a relation containing tuples for consultants. These relations could be combined using a union operation to obtain the desired result: a relation containing information on the non-managers.

Although this approach will work, it is not a feasible approach to use in a complex situation. A typical corporate database might have thousands of employees with dozens of job titles. An approach that required someone to identify each title, create a separate relation for each, then combine all of the relations would be very tedious. Fortunately, an alternative exists. The **difference** operation identifies tuples that occur in one relation but not in another. Once again, union compatibility is required to perform the difference operation.

To respond to the need for this data, one would first create a relation containing information for managers. This relation can be created using the selection operation on EMPLOYEE (see Figure 6-12a). Assume the result of this selection forms a relation called MANAGER, as shown in Figure 6-12b. Now, the difference of EMPLOYEE and MANAGER can be taken. This operation produces a relation with the data shown in Figure 6-12c. The difference operation produces a relation containing all tuples in one relation that do not occur in the second relation.

The order in which the relations are specified is significant in the difference operation. The difference of MANAGER and EMPLOYEE is different from the difference of EMPLOYEE and MANAGER. The difference of MANAGER and EMPLOYEE would produce a relation with no tuples. The first difference, EMPLOYEE - MANAGER, answers the question "Which employees are not managers?". The second difference, MANAGER - EMPLOYEE, answers the question "Which managers are not employees?". Since all managers are also employees, one would expect the result of this operation to be an empty relation.

We can formally specify the difference operation as follows:

Given two union compatible relations R and S, the difference of R and S (R - S) produces a relation T which contains tuples that occur in R that do not occur in S.

Product

The union operation described earlier combines the tuples in two relations vertically. It puts the tuples from one relation under the tuples from a second relation. The relational algebra operation known as **product** constructs a new relation that combines all of the tuples in two relations horizontally. It connects each tuple of one relation with all tuples of a second relation, combining all of the attributes into the resulting relation.

The product operation is illustrated in Figure 6-13. We have modified the ASSIGNMENTS relation so that it contains only the employee identification number and the project identification number. We have also reduced the number of tuples in ASSIGNMENTS, for a reason that will become evident very soon. The modified ASSIGNMENTS relation has been combined with the PROJECT relation. You can see that the result contains twenty tuples. This result occurs because each tuple in PROJECT is combined with every tuple in the modified ASSIGNMENTS relation. In general, if one relation contains m tuples and the second relation contains n tuples, the result of a product operation will contain m times n tuples.

The product operation is not too useful alone. However, it can be used to construct a join operation, which is described in the next section. At this point, we will state the formal description of the product operation.

Given a relation R with attributes $<A_1, A_2, ..., A_n>$, where A_1 has domain $D_{A1} = <d_{a11}, d_{a12}, ..., d_{a1k}>$, A_2 has domain $D_{A2} = <d_{a21}, d_{a22}, ..., d_{a2m}>$, ..., and A_n has domain $D_{An} = <d_{an1}, D_{an2}, ... d_{anp}>$, and a relation S with attributes $<B_1, B_2, ..., B_m>$, where B_1 has domain $D_{B1} = <d_{b11}, d_{b12}, ..., d_{b1k}>$, B_2 has domain $D_{B2} = <d_{b21}, d_{b22}, ..., d_{b2m}>$, ..., and B_m has domain $D_{Bm} = <d_{bm1}, D_{bm2}, ... d_{bmp}>$, the product of R and S produces a relation T with attributes $<A_1, A_2, ..., A_n, B_1, B_2, ..., B_m>$ containing tuples such that if $<a_{1i}, a_{2j}, ..., a_{nk}>$ occurs in a tuple in R and if $<b_{1u}, b_{2v}, ..., b_{mw}>$ occurs in a tuple in S then the tuple $<a_{1i}, a_{2j}, ..., a_{nk}, b_{1u}, b_{2v}, ..., b_{mw}>$ occurs in T.

Natural Join

As we noted above, the product provides one means to combine data in two relations. However, the product is not a very useful operation because the result contains a mixture of tuples including some that have no logical definition. For example, what meaning can be attached to the second tuple in the result in Figure 6-13b?

Suppose that we wished to know the names of the employees working on each project. Obtaining this result requires that we somehow combine the data in EMPLOYEE and PROJECT. You have learned that the intersection relationship embodied in the ASSIGNMENTS relation provides the solution, since it "connects" to both EMPLOYEE and PROJECT. If we take the result of the product operation from above and keep only tuples where the value for the project identification attribute in ASSIGNMENTS is the same as the project identification attribute in PROJECT, we have the result of a **natural join** operation on ASSIGNMENTS and PROJECT. This operation is typically called simply a join operation, as the other types of join operations (described below) are rarely used. The result has combined the employee number for each employee with all occurrences of a project to which the employee is assigned. The P-ID attribute appears only once in the result.

To complete our task of obtaining the names of the employees on each project, we would join this result with the EMPLOYEE relation. Our result, shown in Figure 6-14b, retains only the attributes where the value

for the employee identification attribute in EMPLOYEE (E-ID) is the same as the value for employee identification attribute in the result from the first join operation. Once again, the result is another relation.

Formally, we can state the natural join operation as follows:

Given a relation R with attributes $<A_1, A_2, ..., A_i, ..., A_n>$, where A_1 has domain $D_{A1} = <d_{a11}, d_{a12}, ...>$, A_2 has domain $D_{A2} = <d_{a21}, d_{a22}, ...>$, ..., A_i has domain $D_{Ai} = <d_{ai1}, d_{ai2}, ...>$, ..., and A_n has domain $D_{An} = <d_{an1}, D_{an2}, ...>$, and a relation S with attributes $<A_i, B_1, B_2, ..., B_m>$, where A_i is defined as in R, B_1 has domain $D_{B1} = <d_{b11}, d_{b12}, ...>$, B_2 has domain $D_{B2} = <d_{b21}, d_{b22}, ...>$, ..., and B_m has domain $D_{Bm} = <d_{bm1}, D_{bm2}, ...>$, the natural join of R and S produces a relation T with attributes $<A_1, A_2, ..., A_i, ..., A_n, B_1, B_2, ..., B_m>$ containing tuples such that if $<a_{1i}, a_{2j}, ...>$ occurs in a tuple in R and if $<b_{1u}, b_{2v}, ...>$ occurs in a tuple in S then the tuple $<a_{1i}, a_{2j}, ..., a_{nk}, b_{1u}, b_{2v}, ..., b_{mw}>$ occurs in T if the value of A_i in R is the same as the value of A_i in S.

Theta-Join

The most common type of join is over a comparison of equality of two attributes, but any logical comparison can be made. The comparison operation is generically called "theta" and represented by the symbol Θ. In a natural join, "=" is substituted for Θ. When the comparison is equality the theta-join is called an **equi-join.**

The formal description of theta-join assumes a valid relationship operation Θ defined on a common attribute and differs from natural join only in the last few phrases:

Given a relation R with attributes $<A_1, A_2, ..., A_i, ..., A_n>$, where A_1 has domain $D_{A1} = <d_{a11}, d_{a12}, ...>$, A_2 has domain $D_{A2} = <d_{a21}, d_{a22}, ...>$, ..., A_i has domain $D_{Ai} = <d_{ai1}, d_{ai2}, ...>$, ..., and A_n has domain $D_{An} = <d_{an1}, D_{an2}, ...>$, and a relation S with attributes $<A_i, B_1, B_2, ..., B_m>$, where A_i is defined as in R, B_1 has domain $D_{B1} = <d_{b11}, d_{b12}, ...>$, B_2 has domain $D_{B2} = <d_{b21}, d_{b22}, ...>$, ..., and B_m has domain $D_{Bm} = <d_{bm1}, D_{bm2}, ...>$, the natural join of R and S produces a relation T with attributes $<A_1, A_2, ..., A_i, ..., A_n, B_1, B_2, ..., B_m>$ containing tuples such that if $<a_{1i}, a_{2j}, ...>$ occurs in a tuple in R and if $<b_{1u}, b_{2v}, ...>$ occurs in a tuple in S then the tuple $<a_{1i}, a_{2j}, ..., a_{nk}, b_{1u}, b_{2v}, ..., b_{mw}>$ occurs in T if the relationship Θ holds between the value of A_i in R and the value of A_i in S.

Maybe-Join

In all of the discussions thus far, the existence of a null value would not impact the data manipulation operations described. However, when a null value occurs for a foreign key, the definition of the join operation must be carefully considered. For an example, we will use the data in Figure 6-15a.

In this modified vacation data, we have a null value for one of the authorizations. Suppose we want to resolve a query which asks for the names of the managers that authorized each vacation. We will join VACATION TAKEN with EMPLOYEE to obtain the names of the managers that have approved these vacations. But, how should the null value be handled?

Codd (1979) illustrates how this condition is handled by introducing a three-valued logic as defined in the following tables:

AND	F	-0-	T
F	F	F	F
-0-	F	-0-	-0-

OR	F	-0-	T
F	F	-0-	T
-0-	-0-	-0-	T

T | F -0- T T | T T T

NOT(F) = T NOT(-0-) = -0- NOT(T) = F

Now, two join operations are defined. The true-join is the join with which you are familiar. The true-join produces the result which indicates that the operation is true. Figure 6-15b shows the result of the true-join. A second join, called the maybe-join, is used with the revised truth tables to determine the rest of the query (see Figure 6-15c). This join produces the result where the null values are involved. Note that the maybe-join does not include the results of the true-join.

RELATIONAL CALCULUS*

Relational calculus is simply an alternative expression of the operations defined in relational algebra. Whereas relational algebra is defined in terms of a set of operators for constructing new relations from existing relations, relational calculus simply specifies the desired result. In many cases, relational calculus can specify in one statement a result that requires several relational algebra statements.

Students are often intimidated by the term "calculus." One of our goals in writing this section is to remove this intimidation by demonstrating that relational calculus is really quite simple in its operation. On a practical note, data manipulation in a relational database system is occasionally described in terms of relational calculus instead of relational algebra. Rarely is data manipulation actually implemented in terms of relational calculus, although it certainly could be. Codd (1971) has developed such an implementation, and Date (1986) contains a discussion of implementations. Consequently, we believe an understanding of this notation to be to the student's advantage.

We present a limited subset of relational calculus in the following discussion. We illustrate only a few of the many available operations, and the examples are not complex. As you work through the following sections, think in terms of the desired result. This is the fundamental difference between relational algebra and relational calculus. Earlier you were required to think in terms of how a relation was constructed. Now you must think in terms of what the resulting relation contains.

Projection with Relational Calculus

When a projection is performed, the desired result is a relation containing (typically a subset of) attributes arranged in a specified order. Hence a projection is specified by describing the attributes to be retained in the result, along with their order. If R is a relation containing attributes A, B, and C, and r is a tuple in R, then the projection of R over A and C is described in relational calculus as the set of all the r's such that r's values for A and C appear in R. Formally, this is written

In case you are rusty when it comes to reading this type of set notation, the colon is translated "such that" and the ε is translated "is an element of." The /\ symbol denotes logical conjunction and is read "and." Thus, the statement may be read: "The projection of the relation R over the attributes A and C is equal to the set of all the tuples (denoted by r) such that the tuple comes from (in other words, is an element of) the relation R and the tuple containing the attributes A and C.

The example of relational algebra illustrated in Figure 6-8 projects EMPLOYEE over E-ID and Name. This would be specified by substituting EMPLOYEE for R, E-ID for B, and Name for C in the statement above. The r has no direct translation, since it represents tuples in general that produce the desired result.

Selection with Relational Calculus

Selection is used to isolate tuples that satisfy a certain constraint or set of constraints. In selection using relational calculus, then, specification of constraints is necessary. Let R be a relation containing attributes A, B, and C, and r a tuple in R. The constraint that the selected tuples must satisfy is U. A constant value v is also needed.

In relational calculus two cases are distinguished. The first is a selection in which the value of an attribute is compared to a specific value--this is called **restriction.** The term **selection** is reserved for the second case, in which the value of an attribute is compared to the value of another attribute. The example in Figure 6-9 is a restriction; it is defined by selection from R of all the r tuples where A satisfies a constraint specified by Θ on a value v. Formally, this is written

Our example requires substitution of EMPLOYEE for R, Type for A, "E" for v, and "equals" for Θ.

Selection is formally defined as follows:

The only difference between this and restriction occurs in the last term, where the A attribute is compared to the B attribute instead of to a constant value. This selection can be illustrated with the PROJECT relation in Figure 6-2. If selection of all tuples for projects with actual hours greater than or equal to expected hours is needed, this can be specified by substituting PROJECT for R, Actual Hours for A, Expected Hours for B, and "greater than or equal to" for Θ. The result contains the tuples for projects 2686 and 2671.

Join with Relational Calculus

The join operation can be defined with the definitions used above: let R be a relation containing attributes A, B, and C, r a tuple in R, and Θ a constraint that the selected tuples must satisfy; in addition, let S be a relation containing tuples represented by s defined on the same domains as R and let ➢◅ define the concatenation of two tuples, so that r ➢◅ s constructs a tuple consisting of all the attributes of r followed by all the attributes of s. The join operation is defined in relational calculus by

In other words, the join of R and S over the attributes A and B is defined by the concatenated tuples r and s such that r is in R, s is in S, and the values of A in r and B in s satisfy a constraint specified by Θ.

The join illustrated in Figure 6-14 can be obtained by substituting ASSIGNMENTS for R, PROJECT for S, ASSIGNMENTS P-ID for A, PROJECT P-ID for B, and "equals" for Θ. As above, Θ could be any other conditional operator specifying a constraint.

Case Example: Relational Models of the Omega Company Database

This section is devoted to more examples of operations in relational algebra. The two relations in Figure 6-16 are used as the basis for these examples. One, ENVIRONMENT, contains typical data from the economic environment of a business organization. The other, ACCOUNTING-DATA, contains some accounting information for several quarters for two firms. Note that these relations are not union compatible; some relational algebra operations are not applicable. Examples of selection, projection, and join, however, can still be given.

These examples illustrate how a user may "browse" through a database, searching for answers to specific questions. The information retrieved may lead to new questions and additional queries. On-line browsing is extremely valuable in searching for information in a database.

Formulating a Forecasting Model

Suppose that you are an executive in one of the illustrated firms and you are interested in formulating a sales forecasting model. Desiring to look at some historical data, but not too much, you select from ENVIRONMENT tuples in which Year is greater than 81, producing the relation RECENT-DATA (Figure 6-17).

After looking at this relation, you decide to examine the Economic Index for the first quarter of every year for which you have data, because in the first quarter of last year sales for your firm, the Omega Company, were a little lower than expected. You select from ENVIRONMENT tuples in which Quarter is equal to 1 and project that result over Year and Economic Index to produce ECON-INDEX.

This data (Figure 6-18) does not seem to indicate any reason for sales to have been slow. You decide, therefore, to compare Sales with Economic Index on a quarterly basis. This could be a first step in deciding if a more sophisticated statistical technique should be considered. This manipulation requires a join of ENVIRONMENT with ACCOUNTING-DATA where ENVIRONMENT Quarter is equal to ACCOUNTING-DATA Quarter and ENVIRONMENT Year is equal to ACCOUNTING-DATA Year, with the result projected over Quarter, Year, Firm, Economic Index, and Sales. COMPARISON (Figure 6-19) is the result.

After examining the resulting relation, you recall that the stock market was unusually depressed in that quarter--or was it the year before? What are the values for the Stock Market Index for each quarter in the database? To find out, you project ENVIRONMENT over Year, Quarter, and Stock Market Index to obtain SMI-VALUE (Figure 6-20). There does not seem to be anything significant in the stock market indices. But now you remember that quarter was the second quarter of the big advertising campaign mounted by your competitor, Alpha Incorporated. What accounting data is available for Alpha Incorporated? You can select tuples from ACCOUNTING-DATA where FIRM = "ALPHA," resulting in ALPHA-DATA (Figure 6-21).

While you were busy working on this mystery of Omega's sluggish performance, you did not notice that Jose Garza, one of your best employees in the finance department, had come in and was watching over your shoulder. Jose thinks that there may have been a problem in securing a short-term loan for the company during the time you are analyzing and suggests that you look for some relatively extreme short-term bill rates. You decide to retrieve the Year, Quarter, and Short Term Bill Rate for quarters in which the bill rate exceeded 15 percent. This query is obtained by first projecting ENVIRONMENT over Year, Quarter, and Short Term Bill Rate and then selecting tuples in which the Short Term Bill Rate is greater than 15 from the intermediate result. The final result, STBR-OVER-15 (Figure 6-22), shows something worth noticing. The short-term bill rates were a little higher than 15 percent in the first quarter of 1983. You make a note of this and decide to pursue the matter in more detail later.

Formulating a Complex Query

Because Jose is such a conscientious worker, you know he has come into your office for a reason. Jose has been working on a model that he believes will tie the company's sales to the current stock market activity. He is having some trouble getting the data needed to test his model, however. He knows he needs the data for Quarter, Year, Firm, and Sales for all quarters in which the Stock Market Index was less than 97. Extracting this information requires a selection, a projection, a join, and another projection. You decide to break this query into steps (Figure 6-23a-d). You first select tuples from ENVIRONMENT where Stock Market Index is less than 97 and store the result in STEP-1. Next you project STEP-1 over Quarter and Year, producing STEP-2. You then join ACCOUNTING-DATA with STEP-2 where ACCOUNTING-DATA Quarter is equal to STEP-2 Quarter and ACCOUNTING-DATA Year is equal to STEP-2 Year, thus generating STEP-3. Finally you project STEP-3 Quarter, Year, Firm, and Sales, producing END. This gives Jose the information he needs. He is confident his model will have a positive impact on the operation of the company.

This short scenario demonstrates how a manager can interact with a database in an exploratory fashion, seeking data that might provide information. This exploratory aspect of data retrieval is one of the greatest advantages of the database environment.

Managers are able to get data from the database as fast as the proper query can be formulated. No programs must be written, run, debugged, rerun, debugged again, and so on. Also, a manager need not know anything about the physical nature of the data. These characteristics of database management systems free managers to do what they have been hired to do: manage.

DISTINGUISHING RELATIONAL FROM RELATIONAL-LIKE SYSTEMS

We conclude this chapter with a brief discussion of methods for distinguishing relational from relational-like database systems. As you are probably aware, many systems are advertised as relational database systems. Following Codd (1982), we can define four categories. A **tabular** system supports only tables as data structures and does not support any of the operations defined in relational algebra. A **minimally relational** system is tabular and supports only the selection, projection, and join operations of relational algebra. A system that is tabular and supports all the operations of relational algebra is **relationally complete.** Finally, a system that is relationally complete and also supports entity and referential integrity is **fully relational.**

Codd (1985a, 1985b) outlines twelve rules that define a fully relational database management system. These rules emphasize and extend the basic premise that a relational database management system must be able to manage databases entirely through its relational capabilities. At the time he wrote these articles, Codd was unable to find a single system claiming to be relational that satisfied all twelve rules. These twelve rules should be considered guidelines for making a rough distinction in relational database systems. Codd (1990) has recently published a set of 333 "features" of the relational model (version two).

All of the rules below are based on what Codd calls Rule Zero:

For any system that is advertised as, or claims to be, a relational database management system, that system must be able to manage databases entirely through its relational capabilities.

This rule must hold regardless of whatever non-relational features the database management system might have. If this rule is not met, Codd believes there is no justification in calling the system a relational system. We will now repeat the twelve rules and give a brief explanation of each.

Rule 1, the information rule. All information in a relational database is represented explicitly at the logical level and in exactly one way -- by values in tables.

This rule requires that system related information, such as table names, column names, and domain information, must be maintained as data in relations. There are no special structures for maintaining any of the data in a relational database. Therefore, the database administrator's job is executed by using the same relational data manipulation techniques that any other database user uses. Codd argues that this approach should lead to increased productivity for database administrators.

Rule 2, the guaranteed access rule. Each and every (atomic) value in a relational database is guaranteed to be logically accessible by resorting to a combination of table name, primary key value, and column name.

This rule guarantees that all data in the database can be accessed in at least one way. This rule exists

because the relational model does not incorporate any of the traditional methods of accessing data, such as looping through successive memory locations. Therefore, a rule that guarantees that every data value is accessible is important. We mentioned this concept earlier in our discussion of primary keys.

Rule 3, the systematic treatment of null values rule. Null values (distinct from the empty character string, a string of blanks, a zero, or any other number) are supported in fully relational database management systems for representing missing information and inapplicable information in a systematic way, independent of data type.

Database integrity is closely related to this rule. The database administrator should be able to specify that null values are not allowed for primary keys and for certain foreign keys (as determined by the organization's specific database processing requirements). Also, the systematic treatment of nulls leads to increased productivity. The implication here is that a common null value is used regardless of the domain. This approach implies database users need to learn only one null value representation, which is more efficient than learning, for instance, one null value for numeric data, another null value for character data, and so forth. Additionally, nulls should be treated in a consistent fashion when functions (e.g., SUM and AVERAGE) are applied to data.

Rule 4, the dynamic on-line catalog based on the relational model rule. The database description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data.

This rule implies database users need learn only one data model. Also, users can extend the on-line catalog into a data dictionary (see Chapter 11) if the vendor of a system does not offer one. Rules one and four allow database administrators to access the database easily to determine what information is stored in the database.

Rule 5, the comprehensive data sublanguage rule. There must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and that is comprehensive in supporting all of the following items:

- data definition
- view definition
- data manipulation (interactive and by the program)
- integrity constraints
- authorization
- transaction boundaries

The intent of this rule is to provide for a single database language that addresses all of the actions required against a database, without requiring the database to be taken off-line for any reason. Other database appraoches provide these functions in separate languages. Codd states that the ANSI/SPARC/X3 model discussed in Chapter 5, with over 40 separate interfaces defined, could have over 40 different languages associated with it. SQL, discussed in Chapter 7, is a comprehensive data manipulation language.

Rule 6, the view updating rule. All views that are theoretically updatable are also updatable in the system.

A view is "theoretically updatable" if there exists a time-independent algorithm for making changes in the base relations so that a desired effect occurs in a user's view. This rule states that changes in a view are reflected in the base relations used to create that view. Logical data independence, discussed in Chapter 5, is closely related to this rule.

Rule 7, the high-level insert, update, and delete rule. The capability of handling a base relation or a

derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update, and deletion of data.

This is another user productivity rule. This rule provides a means for users to manipulate data at the table level, rather than requiring separate subqueries to handle data in each row of a table. Also, in distributed database environments (see Chapter 12), the data for a single table may be partitioned and reside at different locations. This rule implies a user need not be concerned with the location of the rows in a table.

Rule 8, the physical data independence rule. Application programs and terminal activity remain logically unimpaired whenever any changes are made in either storage representations or access methods.

We discussed this concept in Chapter 5. Codd states it here to emphasize that applications in a relational environment deal strictly with the logical representation of data, not the physical implementation.

Rule 9, the logical data independence rule. Application programs and terminal activities remain logically unimpaired when information-perserving changes of any kind that theoretically permit unimpairment are made to the base tables.

This rule implies that changes in base tables are allowed as long as the original information is preserved. Key preserving decompositions and non-loss joins, both illustrated in Chapter 5, are two ways the base tables might be altered. Consequently, the database design may be changed dynamically without incurring major reorganization costs.

Rule 10, the integrity independence rule. Integrity constraints specific to a particular database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

A need exists to record all integrity constraints related to a database in one place, outside of application programs. The constraints mentioned in this rule include any business policy, governmental regulation, or other constraint including the entity integrity and referential integrity rules discussed earlier in this chapter. This approach allows changes in these constraints to be made without interupting applications.

Rule 11, the distribution independence rule. A relational database management system has distribution independence.

This rule ensures that applications are unaffected by a decision to distribute data (if it is originally centralized) or redistribute data (if it is already distributed). Note that an existing relational database management system that does not operate in a distributed environment may still be considered relational. This rule does not impact non-distributed systems. Distributed database management systems are covered in Chapter 12. Rules eight through eleven, all independence rules, protect against the problems discussed in Chapters 1 and 5.

Rule 12, the nonsubversion rule. If a relational system has a low-level (i.e., single-record-at-a-time) language, that low level cannot be used to subvert or by-pass the integrity rules and constraints expressed in the higher level (multiple-records-at-a-time) relational language.

The intent of this rule is to preserve the integrity aspects of the relational model in a manner that is independent of logical data structures. According to Codd, many systems that have been "converted" from a different underlying database approach to the relational approach have difficulty satisfying this rule because the prior approach was tightly coupled to low level implementation details. If low level langauges

are allowed to subvert the independence aspects of the relational model, administrators will soon lose control of the database.

As we stated at the beginning of this section, these twelve rules are meant as rough guidelines regarding how well a system meets the criteria for relational database management systems. Codd's recent work describes over 300 criteria in 18 categories to define the relational model in detail. No other approach to database menagement is so thoroughly defined.

THE USER'S VIEW

The relational database model has been presented here in much more depth than many users encounter. Most database management systems claiming to be relational do not adhere to even the most basic relational terminology. Relations are frequently referred to as "tables," tuples as "rows," and attributes as "columns." These terms are less threatening to many users, especially those without technical background.

Similarly, the operations of relational algebra are often defined in less formal terms than those given here. Projection, selection, and join operations are typically variations of one basic command, as can be seen in Chapter 7 in the SQL discussion. Furthermore, the operations are often simply explained in terms like "retrieving a column" (for projection) or "retrieving a row" (for selection). The fact that data in the relational database model can be pictured as two-dimensional tables facilitates these types of explanations.

Users may be tempted to think that the way they picture data in the database is the way it is actually stored. It is not. Complicated list structures may be required to manage additions and deletions from the database. Logical pointer fields may be implemented to reduce redundancy. Whatever internal data structures are used, however, the result must conform to the characteristics of relations specified earlier.

A great deal has been written regarding the "performance myth," according to which relational database systems are inherently inefficient. Many experienced database systems users remember early implementations of the relational model that may have been inefficient. However, optimization techniques exist that improve the efficiency of relational systems. As more vendors develop relational systems, the efficiency of relational systems is expected to increase. As with any software package, issues such as efficiency and response time should be examined using benchmark tests representative of the normal operating conditions under which the database system is expected to operate.

Metropolitan National Bank

The next step in your project is designing a relational implementation of the integrated MetNat database. The Entity-Relationship Model and the conceptual designs developed in earlier chapters provide starting points for your implementation. However, you should be very careful about translating either directly into a relational design. Be sure that your implementation meets the criteria for constructing valid relations. Use the sample reports provided in Appendix C for data to illustrate your implementation.
Be sure you consider the goals you outlined in the master plan that you developed earlier in the project. If response time should be fast, for example, what is implied about the number of operations that should be performed (projects, selections, and joins) in accessing a particular data item? Include a brief discussion for each relation that describes how your implementation supports aspects of the master plan and/or the typical operating procedures described in earlier chapters.

You should prepare a report on the relational approach to MetNat's database system which includes a complete schema and view descriptions. Specify all tables and attributes that appear in the conceptual schema. Specify all tables and attributes that exist in individual user views. Identify the attributes that

compose the candidate, primary, and foreign keys in your relations. Also identify derived tables and attributes. Finally, distinguish between the relatively stable tables that represent account and employee information and the relatively volatile tables that represent transactions on accounts and credit cards.

CHAPTER SUMMARY

The relational database model presents data in a familiar form, that of two-dimensional tables. This model has its foundation in set theory.

There are two ways of building a relationship in a relational database environment. One type of relationship is constructed by the existence of data within a single relation. In this case, an entity has a specific property because a value for an attribute corresponding to that property exists in the relation. The other type of relationship is constructed by replicating primary key values from one relation in another relation.

The relational database model can be used to represent all the relationships defined in Chapter 3--tree, simple network, and complex network. These are all cases of relationships between entities.

Three major aspects of the relational model are data structure, rules for integrity, and methods for data manipulation. Relations are the only data structures provided by the relational model, the relational model requires entity and referential integrity, and data may be manipulated using relational algebra.

There are at least four ways of manipulating data in a relational database management system: relational algebra, relational calculus, transform languages, and graphic languages. The basic operations available in relational algebra are projection, selection, union, intersection, difference, division, and join. These operations are much like those for manipulating sets. Relational calculus differs from relational algebra in that the desired result is specified instead of constructed.

Database systems that claim to be relational systems may be categorized as tabular, minimally relational, relationally complete, or fully relational. These categories require increasingly closer adherence to the relational model. Tabular systems need only represent data in a tabular format, but fully relational systems follow all of the specifications of the relational model. As we stated at the beginning of this chapter, it is the theoretical foundation of the relational model that allows one to make distinctions between truly relational database systems and database systems that are "relational-like." No other database system model has this characteristic.

QUESTIONS AND EXERCISES

1. Define the following terms: relation, tuple, cardinality, attribute, domain, degree.

2. What is the difference between a candidate key and a primary key?

3. Does a relation always have a primary key? Why or why not?

4. Should a primary key in a database ever be allowed to have a null value?

5. Why might it be a bad idea to have an employee's last name as a primary key in a relation?

6. Consider a relation called Sales with attributes Date, Amount, and Salesperson. Assume that a tuple exists in the relation for Boisvert. How does one determine what amount Boisvert sold on a particular date?

7. How are relationships between relations derived?

8. Name three types of relationships that can exist between relations. Define each and give examples.

9. Consider the following relations: EMPLOYEE has attributes E-Number, Name, and Spouse's-name; SPOUSE has attributes Name and Child's-name.

SPOUSE
E-
Number
Name
Spouse's-name

Name
Child's-name

1010
Fraser
Barbara

Barbara
Robert
1003
Clarke
Lisa

Barbara
Kathy
1012
Mason
Robert

Lisa
John

There are at least two major problems with this database design. List all of the problems you can find.

10. What is a data manipulation language? Name four types of data manipulation language that have been developed for use with the relational database model.

11. What is closure?

12. What mathematical operations does relational algebra closely resemble? Why?

13. When would you want to perform a union operation on two relations?

14. What must be true of two relations in order to perform a union operation on them?

15. Compare the difference operation and the intersection operation of relational algebra. How are they similar? How do they differ?

16. Is the difference of two relations the same regardless of the order in which the difference is taken? Give an example to support your answer.

17. What is the difference between the selection operation and the projection operation?

18. Describe the join operation in relational algebra. When would one want to perform a join operation?

19. What is the difference between an equi-join and a natural join?

20. Can one perform a join operation over two attributes when the comparison between the two attributes is something other than equality? If the answer is yes, construct two relations for which such a join makes sense. If the answer is no, explain why.

21. Consider these two relations: SALE has attributes Invoice-number, Salesperson, and Amount (of sale); QUANTITY has attributes Invoice-number, Item-number, and Amount (ordered). Assume that Invoice-number is a unique number assigned to each invoice processed, and that many items can be ordered on a single invoice. Create sample data for these relations.

22. For the relations created in Question 21, show the result of a selection on the SALE relation for Amount greater than some value in your data.

23. For the relations created in Question 21, show the result of a join operation over Invoice-number. What does this represent?

24. What type of relationship is modeled in Question 21?

25. Show relational calculus constructions for the queries formulated in the case example titled "Relational Models of the Omega Company Database."

FOR FURTHER READING

The relational model was first presented in:
Codd, E. F. "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM, Vol. 13, No. 6, June 1970.
Dr. Codd (and others) have refined, extended, and in some cases clarified the relational approach in numerous articles since the original publication. See, for example,
Codd, E.F. "Extending the Database Relational Model to Capture More Meaning." ACM Transactions on Database Systems, Vol. 4, No. 4, December 1979, pp. 397-434.
More recently, Dr. Codd authored two somewhat controversial articles in Computerworld:
Codd, E. F. "Is Your DBMS Really Relational?" Computerworld, October 14, 1985, Vol. 19, Issue 41, and
Codd, E.F. "Does Your DBMS Run By The Rules?" Computerworld, October 21, 1985, Vol. 19, Issue 42.
Version 2 of the relational model is described in:
Codd, E.F. The Relational Model for Database Management - Version 2, Reading: Addison-Wesley Publishing Company, Inc, 1990.
As mentioned in the text, there are few detailed descriptions of a data manipulation language for the relational model which is based on relational calculus. However, one of the best treatments is given in an early paper by Dr. Codd:

Codd, E. F. "A Data Base Sublanguage Founded on the Relational Calculus," Proceedings of the ACM SIGFIDET Workshop on Data Description, Access and Control," 1971, pp.35-68.
For a very theoretical treatment of the relational model, the reader is referred to:
Yang, Chao-Chih. Relational Databases. Prentice Hall, Englewood Cliffs, New Jersey, 1986.

*Note to instructors: This section may be omitted without loss of continuity in the chapter content.

FIGURE 6-1
E-R Model for Employees and Projects in a Hypothetical Database

FIGURE 6-2
Sample Organizational Data in a Relational Format.

FIGURE 6-3
An example of a view (or derived relation). This view was derived from the VACATION TAKEN and SICK LEAVE TAKEN relations.

FIGURE 6-4
A relation with a null value in a foreign key.

FIGURE 6-5
An example of a foreign key existing in the same relation as the primary key that it references.

FIGURE 6-6
An example of a questionable foreign key value. The last tuple contains a value that does not correspond to a current manager's identification number.

FIGURE 6-7
Explicit representation of relationships using links.

FIGURE 6-8
(a) Original EMPLOYEE relation; (b) EMPLOYEE projected over E-ID and Name.

FIGURE 6-9
(a) Original PROJECT relation; (b) selection from PROJECT where Type = "E"; (c) Selection from EMPLOYEE where Title = "Programmer", projected over Name.

FIGURE 6-10
(a) Original relations; (b) union of VACATION TAKEN and SICK LEAVE TAKEN.

FIGURE 6-11
(a) Data to illustrate an intersection. Note the identical last tuple in each relation; (b) Intersection of VACATION TAKEN and SICK LEAVE TAKEN.

FIGURE 6-12
(a) Original EMPLOYEE relation; (b) Selection from EMPLOYEE where Title = "Manager"; (c) EMPLOYEE - MANAGER determines employees that are not managers.

FIGURE 6-13
(a) Relations to illustrate the PRODUCT operation; (b) The PRODUCT of PROJECT and ASSIGNMENTS.

FIGURE 6-14
(a) the JOIN of PROJECT and ASSIGNMENTS (b) join of result in (a) and EMPLOYEE.

FIGURE 6-15
(a) A relation with a null value in a foreign key; (b) The true-join of VACATION TAKEN (modified) and EMPLOYEE (some attributes from EMPLOYEE are omitted).

FIGURE 6-16
Sample Relations from the Omega Company Database

FIGURE 6-17
Selection from ENVIRONMENT Where Year >
Selection from ENVIRONMENT Where Year > 81

FIGURE 6-18
Selection from ENVIRONMENT Where Quarter = 1, Projected Over Year and Economic Index

FIGURE 6-19
Join of ENVIRONMENT and ACCOUNTING-DATA Over Quarter and Year, Projected Over Quarter, Year, Firm, Economic Index, and Sales

FIGURE 6-20
Projection of ENVIRONMENT Over Year, Quarter, and Stock Market Index

FIGURE 6-21
Selection from ACCOUNTING-DATA Where Firm = "ALPHA"

FIGURE 6-22
Projection of ENVIRONMENT Over Year, Quarter, and Short Term Bill Rate, Followed by Selection with Short Term Bill Rate >
Projection of ENVIRONMENT Over Year, Quarter, and Short Term Bill Rate, Followed by Selection with Short Term Bill Rate > 15

FIGURE 6-23
A Complex Query. (a) selection from ENVIRONMENT where Stock Market Index < 97; (b) projection of STEP-1 over Quarter and Year; (c) join of STEP-2 and ACCOUNTING-DATA over Quarter and Year; (d) projection of STEP-3 over Quarter, Year, Firm, and Sales.

EMPLOYEE

E-ID	Name	Address	City	Zip	Phone	Title	Salary	Vacation Available	Sick Leave Available
134572698	Jones	12 Maple St.	Atlanta	30311	758-1267	Programmer	2200	3	8
983749493	Smith	28 Alma St.	Smyrna	30364	748-4847	Programmer	2200	5	8
847478599	Clarke	373 Ave. A	Smyrna	30364	837-9383	Consultant	2500	5	7
252930423	Dexter	3888 6th St.	Smyrna	30364	837-2552	Consultant	2500	4	5
383783993	Hayet	149 Oakside Dr.	Atlanta	30311	758-7399	Programmer	2300	3	7
252928462	Gofman	2333 Briarcrest Dr.	Atlanta	30312	766-6752	Programmer	2350	7	8
252022010	Wray	2911 Windgate St.	Smyrna	30364	837-7288	Manager	3500	7	7
252887324	Bennett	131 Rhodes Dr.	Smyrna	30367	977-0979	Manager	4200	3	8

PROJECT

P-ID	Name	Type	Actual Hours	Expected Hours	Project Leader
6301	Marketing report	E	12	120	252928462
2376	New billing system	E	80	160	252928462
2876	Common stock report	C	12	8	383783993
2671	Bad debt report	D	40	20	252930423
2651	Manpower report bug	C	8	16	983749493

Project types: E = Enhancement, C = Correction, D = Development

VACATION TAKEN

E-ID	Begin Date	End Date	Authorization
134572698	7/1	7/5	252022010
252930423	8/10	8/10	252022010
383783993	9/4	9/7	252022010
983749493	9/4	9/4	252887324
383783993	11/20	11/21	252887324

SICK LEAVE TAKEN

E-ID	Begin Date	End Date	Authorization
134572698	3/2	3/2	252022010
383783993	5/10	5/14	252022010

Figure 6-2

Sample organizational data in a relational format.

TIMESHEETS

E-ID	P-ID	Date	Hours
134572698	6301	10/2	3.5
134572698	2376	10/2	5.5
252930423	2671	10/2	8.0
983749493	2376	10/2	8.0
847478599	2876	10/2	5.0
847478599	2671	10/2	3.0
252928462	2376	10/2	3.5
252928462	6301	10/2	4.5
134572698	6301	10/3	7.5
134572698	2376	10/3	0.5
252930423	2671	10/3	8.0
983749493	2376	10/3	8.0
847478599	2876	10/3	4.0
847478599	2671	10/3	4.0

ASSIGNMENTS

E-ID	P-ID	Total Hours	Evaluation
134572698	6301	11.0	On schedule
134572698	2376	6.0	On schedule
252930423	2671	16.0	Slipping - needs analysis
983749493	2376	16.0	Late start - OK now
847478599	2876	9.0	Analysis problems
847478599	2671	7.0	Appears on schedule - Need formal review
252928462	2376	3.5	OK
252928462	6301	4.5	-0-
383783993	2651	-0-	-0-

Figure 6-2 (continued)

Sample organizational data in a relational format.

Paid Leave Taken

E-ID	Begin Date	End Date	Authorization
134572698	7/1	7/5	252022010
252930423	8/10	8/10	252022010
383783993	9/4	9/7	252022010
983749493	9/4	9/4	252887324
383783993	11/20	11/21	252887324
134572698	3/2	3/2	252022010
383783993	5/10	5/14	252022010

Figure 6-3

An example of a view (or derived relation). This view was derived from the Vacation Taken and Sick Leave Taken relations.

Employees (modified)

E-ID	Name	Address	City	Zip	Phone	Title		Manager ID
134572698	Jones	12 Maple St.	Atlanta	30311	758-1267	Programmer	...	252022010
983749493	Smith	28 Alma St.	Smyrna	30364	748-4847	Programmer	...	252022010
847478599	Clarke	373 Ave. A	Smyrna	30364	837-9383	Consultant	...	252887324
252930423	Dexter	3888 6th St.	Smyrna	30364	837-2552	Consultant	...	252887324
383783993	Hayet	149 Oakside Dr.	Atlanta	30311	758-7399	Programmer	...	252022010
252928462	Gofman	2333 Briarcrest Dr.	Atlanta	30312	766-6752	Programmer	...	252022010
252022010	Wray	2911 Windgate St.	Smyrna	30364	837-7288	Manager	...	349942839
252887324	Bennett	131 Rhodes Dr.	Smyrna	30367	977-0979	Manager	...	349942839

Figure 6-5

An example of a foreign key existing in the same relation as the primary key that it references.

Vacation Taken

E-ID	Begin Date	End Date	Authorization
134572698	7/1	7/5	252022010
252930423	8/10	8/10	252022010
383783993	9/4	9/7	252022010
983749493	9/4	9/4	252887324
383783993	9/10	9/14	837954503

Figure 6-6

An example of a questionable foreign key value. The last tuple contains a value that does not correspond to a current manager's identification value.

VACATION TAKEN

E-ID	Begin Date	End Date	Authorization
134572698	7/1	7/5	252022010
252930423	8/10	8/10	252022010
383783993	9/4	9/7	252022010
983749493	9/4	9/4	252887324
383783993	11/20	11/21	252887324

SICK LEAVE TAKEN

E-ID	Begin Date	End Date	Authorization
134572698	3/2	3/2	252022010
383783993	5/10	5/14	252022010
383783993	11/20	11/21	252887324

Figure 6-11a

Data to illustrate an intersection. Note the identical last tuple in each relation.

DOUBLE COUNTED ABSENCES

E-ID	Begin Date	End Date	Authorization
383783993	11/20	11/21	252887324

Figure 6-11b

Intersection of VACATION TAKEN and SICK LEAVE TAKEN.

VACATION TAKEN (modified)

E-ID	Begin Date	End Date	Authorization
134572698	7/1	7/5	252022010
252930423	8/10	8/10	252022010
383783993	9/4	9/7	252022010
983749493	9/4	9/4	252887324
383783993	11/20	11/21	-0-

Figure 6-4

A relation with a null value in a foreign key.

VACATION TAKEN (modified)

E-ID	Begin Date	End Date	Authorization
134572698	7/1	7/5	252022010
252930423	8/10	8/10	252022010
383783993	9/4	9/7	252022010
983749493	9/4	9/4	252887324
383783993	11/20	11/21	-0-

Figure 6-15a

A relation with a null value in a foreign key.

VACATION APPROVALS

E-ID	Begin Date	End Date	Authorization		
134572698	7/1	7/5	252022010	Wray	...
252930423	8/10	8/10	252022010	Wray	...
383783993	9/4	9/7	252022010	Wray	...
983749493	9/4	9/4	252887324	Bennett	...

Figure 6-15b

The true-join of VACATION TAKEN (modified) and EMPLOYEE (some attributes from EMPLOYEE are omitted).

VACATION APPROVALS

E-ID	Begin Date	End Date	Authorization		
383783993	11/20	11/21	-0-	Wray	...
383783993	11/20	11/21	-0-	Bennett	...

Figure 6-15c

The maybe-join of VACATION TAKEN (modified) and EMPLOYEE (some attributes from EMPLOYEE are omitted).

EMPLOYEE

E-ID	Name	Address	City	Zip	Phone	Title	Salary	Vacation Available	Sick Leave Available
134572698	Jones	12 Maple St.	Atlanta	30311	758-1267	Programmer	2200	3	8
983749493	Smith	28 Alma St.	Smyrna	30364	748-4847	Programmer	2200	5	8
847478599	Clarke	373 Ave. A	Smyrna	30364	837-9383	Consultant	2500	5	7
252930423	Dexter	3888 6th St.	Smyrna	30364	837-2552	Consultant	2500	4	5
383783993	Hayet	149 Oakside Dr.	Atlanta	30311	758-7399	Programmer	2300	3	7
252928462	Gofman	2333 Briarcrest Dr.	Atlanta	30312	766-6752	Programmer	2350	7	8
252022010	Wray	2911 Windgate St.	Smyrna	30364	837-7288	Manager	3500	7	7
252887324	Bennett	131 Rhodes Dr.	Smyrna	30367	977-0979	Manager	4200	3	8

VACATION TAKEN

Begin Date	End Date	Authorization
7/1	7/5	252022010
8/10	8/10	252022010
9/4	9/7	252022010
9/4	9/4	252887324
11/20	11/21	252887324

Figure 6-7

Explicit representation of relationships using links.

EMPLOYEE

E-ID	Name	Address	City	Zip	Phone	Title	Salary	Vacation Available	Sick Leave Available
134572698	Jones	12 Maple St.	Atlanta	30311	758-1267	Programmer	2200	3	8
983749493	Smith	28 Alma St.	Smyrna	30364	748-4847	Programmer	2200	5	8
847478599	Clarke	373 Ave. A	Smyrna	30364	837-9383	Consultant	2500	5	7
252930423	Dexter	3888 6th St.	Smyrna	30364	837-2552	Consultant	2500	4	5
383783993	Hayet	149 Oakside Dr.	Atlanta	30311	758-7399	Programmer	2300	3	7
252928462	Gofman	2333 Briarcrest Dr.	Atlanta	30312	766-6752	Programmer	2350	7	8
252022010	Wray	2911 Windgate St.	Smyrna	30364	837-7288	Manager	3500	7	7
252887324	Bennett	131 Rhodes Dr.	Smyrna	30367	977-0979	Manager	4200	3	8

Figure 6-8a

Original employee data.

EMPLOYEE-NUMBER-NAME

E-ID	Name
134572698	Jones
983749493	Smith
847478599	Clarke
252930423	Dexter
383783993	Hayet
252928462	Gofman
252022010	Wray
252887324	Bennett

Figure 6-8b

EMPLOYEE projected over E-ID and Name.

PROJECT

P-ID	Name	Type	Actual Hours	Expected Hours	Project Leader
6301	Marketing report	E	12	120	252928462
2376	New billing system	E	80	160	252928462
2876	Common stock report	C	12	8	383783993
2671	Bad debt report	D	40	20	252930423
2651	Manpower report bug	C	8	16	983749493

Project types: E = Enhancement, C = Correction, D = Development

Figure 6-9a

Original project data.

ENHANCEMENTS

P-ID	Name	Type	Actual Hours	Expected Hours	Project Leader
6301	Marketing report	E	12	120	252928462
2376	New billing system	E	80	160	252928462

Project types: E = Enhancement, C = Correction, D = Development

Figure 6-9b

Selection from PROJECT where Type = 'E'

PROGRAMMERS

Name
Jones
Smith
Hayet
Gofman

Figure 6-9c

Selection from EMPLOYEE where Title = 'Programmer', projected over Name.

VACATION TAKEN

E-ID	Begin Date	End Date	Authorization
134572698	7/1	7/5	252022010
252930423	8/10	8/10	252022010
383783993	9/4	9/7	252022010
983749493	9/4	9/4	252887324
383783993	11/20	11/21	252887324

SICK LEAVE TAKEN

E-ID	Begin Date	End Date	Authorization
134572698	3/2	3/2	252022010
383783993	5/10	5/14	252022010

Figure 6-10a

Original vacation and sick leave data.

PAID LEAVE TAKEN

E-ID	Begin Date	End Date	Authorization
134572698	7/1	7/5	252022010
252930423	8/10	8/10	252022010
383783993	9/4	9/7	252022010
983749493	9/4	9/4	252887324
383783993	11/20	11/21	252887324
134572698	3/2	3/2	252022010
383783993	5/10	5/14	252022010

Figure 6-10b

Union of VACATION TAKEN and SICK LEAVE TAKEN relations.

EMPLOYEE

E-ID	Name	Address	City	Zip	Phone	Title	Salary	Vacation Available	Sick Leave Available
134572698	Jones	12 Maple St.	Atlanta	30311	758-1267	Programmer	2200	3	8
983749493	Smith	28 Alma St.	Smyrna	30364	748-4847	Programmer	2200	5	8
847478599	Clarke	373 Ave. A	Smyrna	30364	837-9383	Consultant	2500	5	7
252930423	Dexter	3888 6th St.	Smyrna	30364	837-2552	Consultant	2500	4	5
383783993	Hayet	149 Oakside Dr.	Atlanta	30311	758-7399	Programmer	2300	3	7
252928462	Gofman	2333 Briarcrest Dr.	Atlanta	30312	766-6752	Programmer	2350	7	8
252022010	Wray	2911 Windgate St.	Smyrna	30364	837-7288	Manager	3500	7	7
252887324	Bennett	131 Rhodes Dr.	Smyrna	30367	977-0979	Manager	4200	3	8

Figure 6-12a

Original employee data.

MANAGERS

E-ID	Name	Address	City	Zip	Phone	Title	Salary	Vacation Available	Sick Leave Available
252022010	Wray	2911 Windgate St.	Smyrna	30364	837-7288	Manager	3500	7	7
252887324	Bennett	131 Rhodes Dr.	Smyrna	30367	977-0979	Manager	4200	3	8

Figure 6-12b

E-ID	Name	Address	City	Zip	Phone	Title	Salary	Vacation Available	Sick Leave Available
134572698	Jones	12 Maple St.	Atlanta	30311	758-1267	Programmer	2200	3	8
983749493	Smith	28 Alma St.	Smyrna	30364	748-4847	Programmer	2200	5	8
847478599	Clarke	373 Ave. A	Smyrna	30364	837-9383	Consultant	2500	5	7
252930423	Dexter	3888 6th St.	Smyrna	30364	837-2552	Consultant	2500	4	5
383783993	Hayet	149 Oakside Dr.	Atlanta	30311	758-7399	Programmer	2300	3	7
252928462	Gofman	2333 Briarcrest Dr.	Atlanta	30312	766-6752	Programmer	2350	7	8

Figure 6-12c

The Difference of EMPLOYEE and MANAGER.

ASSIGNMENTS (modified)

E-ID	P-ID
134572698	6301
134572698	2376
252930423	2671
983749493	2376

PROJECT

P-ID	Name	Type	Actual Hours	Expected Hours	Project Leader
6301	Marketing report	E	12	120	252928462
2376	New billing system	E	80	160	252928462
2876	Common stock report	C	12	8	383783993
2671	Bad debt report	D	40	20	252930423
2651	Manpower report bug	C	8	16	983749493

Project types: E = Enhancement, C = Correction, D = Development

Figure 6-13a

Relations to illustrate the PRODUCT operation.

ASSIGNMENTS-AND-PROJECTS (product operation)

		PROJECT.		Actual	Expected	Project	
E-ID	**P-ID**	**P-ID**	Name	Type	Hours	Hours	Leader
134572698	6301	6301	Marketing report	E	12	120	252928462
134572698	6301	2376	New billing system	E	80	160	252928462
134572698	6301	2876	Common stock report	C	12	8	383783993
134572698	6301	2671	Bad debt report	D	40	20	252930423
134572698	6301	2651	Manpower report bug	C	8	16	983749493
134572698	2376	6301	Marketing report	E	12	120	252928462
134572698	2376	2376	New billing system	E	80	160	252928462
134572698	2376	2876	Common stock report	C	12	8	383783993
134572698	2376	2671	Bad debt report	D	40	20	252930423
134572698	2376	2651	Manpower report bug	C	8	16	983749493
252930423	2671	6301	Marketing report	E	12	120	252928462
252930423	2671	2376	New billing system	E	80	160	252928462
252930423	2671	2876	Common stock report	C	12	8	383783993
252930423	2671	2671	Bad debt report	D	40	20	252930423
252930423	2671	2651	Manpower report bug	C	8	16	983749493
983749493	2376	6301	Marketing report	E	12	120	252928462
983749493	2376	2376	New billing system	E	80	160	252928462
983749493	2376	2876	Common stock report	C	12	8	383783993
983749493	2376	2671	Bad debt report	D	40	20	252930423
983749493	2376	2651	Manpower report bug	C	8	16	983749493

Figure 6-13b

The PRODUCT of PROJECT and ASSIGNMENTS.

ASSIGNMENTS-AND-PROJECTS (join operation)

E-ID	P-ID	Name	Type	Actual Hours	Expected Hours	Project Leader
134572698	6301	Marketing report	E	12	120	252928462
134572698	2376	New billing system	E	80	160	252928462
252930423	2671	Bad debt report	D	40	20	252930423
983749493	2376	New billing system	E	80	160	252928462

Figure 6-14a

The JOIN of PROJECT and ASSIGNMENTS.

(Final join operation)

Figure 6-14b

The JOIN of EMPLOYEE and ASSIGNMENTS-AND-PROJECTS.