

CHAPTER 2



Introduction to the Relational Model

The relational model is today the primary data model for commercial data-processing applications. It attained its primary position because of its simplicity, which eases the job of the programmer, compared to earlier data models such as the network model or the hierarchical model.

In this chapter, we first study the fundamentals of the relational model. A substantial theory exists for relational databases. We study the part of this theory dealing with queries in Chapter 6. In Chapters 7 through 8, we shall examine aspects of database theory that help in the design of relational database schemas, while in Chapters 12 and 13 we discuss aspects of the theory dealing with efficient processing of queries.

2.1 Structure of Relational Databases

A relational database consists of a collection of **tables**, each of which is assigned a unique name. For example, consider the *instructor* table of Figure 2.1, which stores information about instructors. The table has four column headers: *ID*, *name*, *dept_name*, and *salary*. Each row of this table records information about an instructor, consisting of the instructor's *ID*, *name*, *dept_name*, and *salary*. Similarly, the *course* table of Figure 2.2 stores information about courses, consisting of a *course_id*, *title*, *dept_name*, and *credits*, for each course. Note that each instructor is identified by the value of the column *ID*, while each course is identified by the value of the column *course_id*.

Figure 2.3 shows a third table, *prereq*, which stores the prerequisite courses for each course. The table has two columns, *course_id* and *prereq_id*. Each row consists of a pair of course identifiers such that the second course is a prerequisite for the first course.

Thus, a row in the *prereq* table indicates that two courses are *related* in the sense that one course is a prerequisite for the other. As another example, we consider the table *instructor*, a row in the table can be thought of as representing

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Figure 2.1 The *instructor* relation.

the relationship between a specified *ID* and the corresponding values for *name*, *dept_name*, and *salary* values.

In general, a row in a table represents a *relationship* among a set of values. Since a table is a collection of such relationships, there is a close correspondence between the concept of *table* and the mathematical concept of *relation*, from which the relational data model takes its name. In mathematical terminology, a *tuple* is simply a sequence (or list) of values. A relationship between n values is represented mathematically by an *n -tuple* of values, i.e., a tuple with n values, which corresponds to a row in a table.

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

Figure 2.2 The *course* relation.

<i>course_id</i>	<i>prereq_id</i>
BIO-301	BIO-101
BIO-399	BIO-101
CS-190	CS-101
CS-315	CS-101
CS-319	CS-101
CS-347	CS-101
EE-181	PHY-101

Figure 2.3 The *prereq* relation.

Thus, in the relational model the term **relation** is used to refer to a table, while the term **tuple** is used to refer to a row. Similarly, the term **attribute** refers to a column of a table.

Examining Figure 2.1, we can see that the relation *instructor* has four attributes: *ID*, *name*, *dept_name*, and *salary*.

We use the term **relation instance** to refer to a specific instance of a relation, i.e., containing a specific set of rows. The instance of *instructor* shown in Figure 2.1 has 12 tuples, corresponding to 12 instructors.

In this chapter, we shall be using a number of different relations to illustrate the various concepts underlying the relational data model. These relations represent part of a university. They do not include all the data an actual university database would contain, in order to simplify our presentation. We shall discuss criteria for the appropriateness of relational structures in great detail in Chapters 7 and 8.

The order in which tuples appear in a relation is irrelevant, since a relation is a *set* of tuples. Thus, whether the tuples of a relation are listed in sorted order, as in Figure 2.1, or are unsorted, as in Figure 2.4, does not matter; the relations in

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Figure 2.4 Unsorted display of the *instructor* relation.

the two figures are the same, since both contain the same set of tuples. For ease of exposition, we will mostly show the relations sorted by their first attribute.

For each attribute of a relation, there is a set of permitted values, called the **domain** of that attribute. Thus, the domain of the *salary* attribute of the *instructor* relation is the set of all possible salary values, while the domain of the *name* attribute is the set of all possible instructor names.

We require that, for all relations r , the domains of all attributes of r be atomic. A domain is **atomic** if elements of the domain are considered to be indivisible units. For example, suppose the table *instructor* had an attribute *phone_number*, which can store a set of phone numbers corresponding to the instructor. Then the domain of *phone_number* would not be atomic, since an element of the domain is a set of phone numbers, and it has subparts, namely the individual phone numbers in the set.

The important issue is not what the domain itself is, but rather how we use domain elements in our database. Suppose now that the *phone_number* attribute stores a single phone number. Even then, if we split the value from the phone number attribute into a country code, an area code and a local number, we would be treating it as a nonatomic value. If we treat each phone number as a single indivisible unit, then the attribute *phone_number* would have an atomic domain.

In this chapter, as well as in Chapters 3 through 6, we assume that all attributes have atomic domains. In Chapter 22, we shall discuss extensions to the relational data model to permit nonatomic domains.

The **null** value is a special value that signifies that the value is unknown or does not exist. For example, suppose as before that we include the attribute *phone_number* in the *instructor* relation. It may be that an instructor does not have a phone number at all, or that the telephone number is unlisted. We would then have to use the null value to signify that the value is unknown or does not exist. We shall see later that null values cause a number of difficulties when we access or update the database, and thus should be eliminated if at all possible. We shall assume null values are absent initially, and in Section 3.6 we describe the effect of nulls on different operations.

2.2 Database Schema

When we talk about a database, we must differentiate between the **database schema**, which is the logical design of the database, and the **database instance**, which is a snapshot of the data in the database at a given instant in time.

The concept of a relation corresponds to the programming-language notion of a variable, while the concept of a **relation schema** corresponds to the programming-language notion of type definition.

In general, a relation schema consists of a list of attributes and their corresponding domains. We shall not be concerned about the precise definition of the domain of each attribute until we discuss the SQL language in Chapter 3.

The concept of a relation instance corresponds to the programming-language notion of a value of a variable. The value of a given variable may change with time;

<i>dept_name</i>	<i>building</i>	<i>budget</i>
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

Figure 2.5 The *department* relation.

similarly the contents of a relation instance may change with time as the relation is updated. In contrast, the schema of a relation does not generally change.

Although it is important to know the difference between a relation schema and a relation instance, we often use the same name, such as *instructor*, to refer to both the schema and the instance. Where required, we explicitly refer to the schema or to the instance, for example “the *instructor* schema,” or “an instance of the *instructor* relation.” However, where it is clear whether we mean the schema or the instance, we simply use the relation name.

Consider the *department* relation of Figure 2.5. The schema for that relation is

department (*dept_name*, *building*, *budget*)

Note that the attribute *dept_name* appears in both the *instructor* schema and the *department* schema. This duplication is not a coincidence. Rather, using common attributes in relation schemas is one way of relating tuples of distinct relations. For example, suppose we wish to find the information about all the instructors who work in the Watson building. We look first at the *department* relation to find the *dept_name* of all the departments housed in Watson. Then, for each such department, we look in the *instructor* relation to find the information about the instructor associated with the corresponding *dept_name*.

Let us continue with our university database example.

Each course in a university may be offered multiple times, across different semesters, or even within a semester. We need a relation to describe each individual offering, or section, of the class. The schema is

section (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)

Figure 2.6 shows a sample instance of the *section* relation.

We need a relation to describe the association between instructors and the class sections that they teach. The relation schema to describe this association is

teaches (*ID*, *course_id*, *sec_id*, *semester*, *year*)

<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>	<i>building</i>	<i>room_number</i>	<i>time_slot_id</i>
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

Figure 2.6 The *section* relation.

Figure 2.7 shows a sample instance of the *teaches* relation.

As you can imagine, there are many more relations maintained in a real university database. In addition to those relations we have listed already, *instructor*, *department*, *course*, *section*, *prereq*, and *teaches*, we use the following relations in this text:

<i>ID</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

Figure 2.7 The *teaches* relation.

- *student* (*ID*, *name*, *dept_name*, *tot_cred*)
- *advisor* (*s_id*, *i_id*)
- *takes* (*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)
- *classroom* (*building*, *room_number*, *capacity*)
- *time_slot* (*time_slot_id*, *day*, *start_time*, *end_time*)

2.3 Keys

We must have a way to specify how tuples within a given relation are distinguished. This is expressed in terms of their attributes. That is, the values of the attribute values of a tuple must be such that they can *uniquely identify* the tuple. In other words, no two tuples in a relation are allowed to have exactly the same value for all attributes.

A **superkey** is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation. For example, the *ID* attribute of the relation *instructor* is sufficient to distinguish one instructor tuple from another. Thus, *ID* is a superkey. The *name* attribute of *instructor*, on the other hand, is not a superkey, because several instructors might have the same name.

Formally, let R denote the set of attributes in the schema of relation r . If we say that a subset K of R is a *superkey* for r , we are restricting consideration to instances of relations r in which no two distinct tuples have the same values on all attributes in K . That is, if t_1 and t_2 are in r and $t_1 \neq t_2$, then $t_1.K \neq t_2.K$.

A superkey may contain extraneous attributes. For example, the combination of *ID* and *name* is a superkey for the relation *instructor*. If K is a superkey, then so is any superset of K . We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called **candidate keys**.

It is possible that several distinct sets of attributes could serve as a candidate key. Suppose that a combination of *name* and *dept_name* is sufficient to distinguish among members of the *instructor* relation. Then, both $\{ID\}$ and $\{name, dept_name\}$ are candidate keys. Although the attributes *ID* and *name* together can distinguish *instructor* tuples, their combination, $\{ID, name\}$, does not form a candidate key, since the attribute *ID* alone is a candidate key.

We shall use the term **primary key** to denote a candidate key that is chosen by the database designer as the principal means of identifying tuples within a relation. A key (whether primary, candidate, or super) is a property of the entire relation, rather than of the individual tuples. Any two individual tuples in the relation are prohibited from having the same value on the key attributes at the same time. The designation of a key represents a constraint in the real-world enterprise being modeled.

Primary keys must be chosen with care. As we noted, the name of a person is obviously not sufficient, because there may be many people with the same name. In the United States, the social-security number attribute of a person would be a candidate key. Since non-U.S. residents usually do not have social-security

numbers, international enterprises must generate their own unique identifiers. An alternative is to use some unique combination of other attributes as a key.

The primary key should be chosen such that its attribute values are never, or very rarely, changed. For instance, the address field of a person should not be part of the primary key, since it is likely to change. Social-security numbers, on the other hand, are guaranteed never to change. Unique identifiers generated by enterprises generally do not change, except if two enterprises merge; in such a case the same identifier may have been issued by both enterprises, and a reallocation of identifiers may be required to make sure they are unique.

It is customary to list the primary key attributes of a relation schema before the other attributes; for example, the *dept_name* attribute of *department* is listed first, since it is the primary key. Primary key attributes are also underlined.

A relation, say r_1 , may include among its attributes the primary key of another relation, say r_2 . This attribute is called a **foreign key** from r_1 , referencing r_2 . The relation r_1 is also called the **referencing relation** of the foreign key dependency, and r_2 is called the **referenced relation** of the foreign key. For example, the attribute *dept_name* in *instructor* is a foreign key from *instructor*, referencing *department*, since *dept_name* is the primary key of *department*. In any database instance, given any tuple, say t_a , from the *instructor* relation, there must be some tuple, say t_b , in the *department* relation such that the value of the *dept_name* attribute of t_a is the same as the value of the primary key, *dept_name*, of t_b .

Now consider the *section* and *teaches* relations. It would be reasonable to require that if a section exists for a course, it must be taught by at least one instructor; however, it could possibly be taught by more than one instructor. To enforce this constraint, we would require that if a particular (*course_id*, *sec_id*, *semester*, *year*) combination appears in *section*, then the same combination must appear in *teaches*. However, this set of values does not form a primary key for *teaches*, since more than one instructor may teach one such section. As a result, we cannot declare a foreign key constraint from *section* to *teaches* (although we can define a foreign key constraint in the other direction, from *teaches* to *section*).

The constraint from *section* to *teaches* is an example of a **referential integrity constraint**; a referential integrity constraint requires that the values appearing in specified attributes of any tuple in the referencing relation also appear in specified attributes of at least one tuple in the referenced relation.

2.4 Schema Diagrams

A database schema, along with primary key and foreign key dependencies, can be depicted by **schema diagrams**. Figure 2.8 shows the schema diagram for our university organization. Each relation appears as a box, with the relation name at the top in blue, and the attributes listed inside the box. Primary key attributes are shown underlined. Foreign key dependencies appear as arrows from the foreign key attributes of the referencing relation to the primary key of the referenced relation.

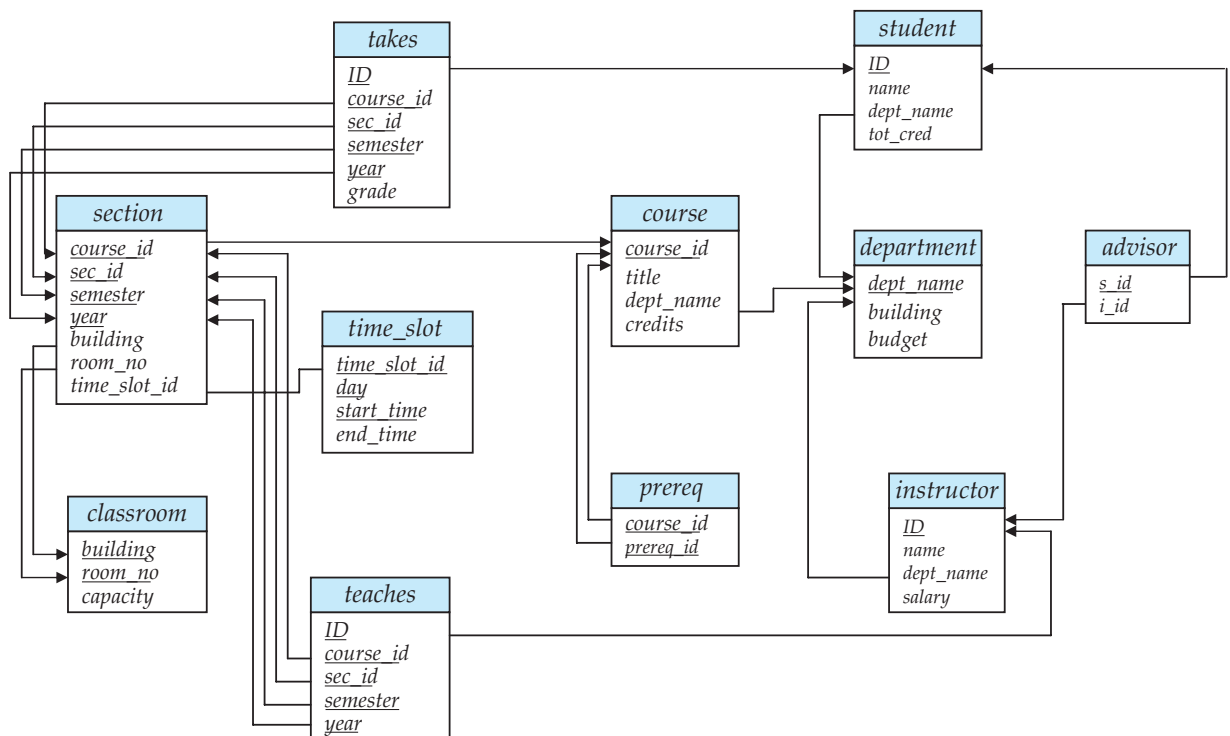


Figure 2.8 Schema diagram for the university database.

Referential integrity constraints other than foreign key constraints are not shown explicitly in schema diagrams. We will study a different diagrammatic representation called the entity-relationship diagram later, in Chapter 7. Entity-relationship diagrams let us represent several kinds of constraints, including general referential integrity constraints.

Many database systems provide design tools with a graphical user interface for creating schema diagrams. We shall discuss diagrammatic representation of schemas at length in Chapter 7.

The enterprise that we use in the examples in later chapters is a university. Figure 2.9 gives the relational schema that we use in our examples, with primary-key attributes underlined. As we shall see in Chapter 3, this corresponds to the approach to defining relations in the SQL data-definition language.

2.5 Relational Query Languages

A **query language** is a language in which a user requests information from the database. These languages are usually on a level higher than that of a standard programming language. Query languages can be categorized as either procedural or nonprocedural. In a **procedural language**, the user instructs the system to perform a sequence of operations on the database to compute the desired result. In a **nonprocedural language**, the user describes the desired information without giving a specific procedure for obtaining that information.

```

classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building, room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(ID, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)

```

Figure 2.9 Schema of the university database.

Query languages used in practice include elements of both the procedural and the nonprocedural approaches. We study the very widely used query language SQL in Chapters 3 through 5.

There are a number of “pure” query languages: The relational algebra is procedural, whereas the tuple relational calculus and domain relational calculus are nonprocedural. These query languages are terse and formal, lacking the “syntactic sugar” of commercial languages, but they illustrate the fundamental techniques for extracting data from the database. In Chapter 6, we examine in detail the relational algebra and the two versions of the relational calculus, the tuple relational calculus and domain relational calculus. The relational algebra consists of a set of operations that take one or two relations as input and produce a new relation as their result. The relational calculus uses predicate logic to define the result desired without giving any specific algebraic procedure for obtaining that result.

2.6 Relational Operations

All procedural relational query languages provide a set of operations that can be applied to either a single relation or a pair of relations. These operations have the nice and desired property that their result is always a single relation. This property allows one to combine several of these operations in a modular way. Specifically, since the result of a relational query is itself a relation, relational operations can be applied to the results of queries as well as to the given set of relations.

The specific relational operations are expressed differently depending on the language, but fit the general framework we describe in this section. In Chapter 3, we show the specific way the operations are expressed in SQL.

The most frequent operation is the selection of specific tuples from a single relation (say *instructor*) that satisfies some particular predicate (say *salary* > \$85,000). The result is a new relation that is a subset of the original relation (*in-*

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
12121	Wu	Finance	90000
22222	Einstein	Physics	95000
33456	Gold	Physics	87000
83821	Brandt	Comp. Sci.	92000

Figure 2.10 Result of query selecting *instructor* tuples with salary greater than \$85000.

structor). For example, if we select tuples from the *instructor* relation of Figure 2.1, satisfying the predicate “*salary* is greater than \$85000”, we get the result shown in Figure 2.10.

Another frequent operation is to select certain attributes (columns) from a relation. The result is a new relation having only those selected attributes. For example, suppose we want a list of instructor *IDs* and salaries without listing the *name* and *dept_name* values from the *instructor* relation of Figure 2.1, then the result, shown in Figure 2.11, has the two attributes *ID* and *salary*. Each tuple in the result is derived from a tuple of the *instructor* relation but with only selected attributes shown.

The *join* operation allows the combining of two relations by merging pairs of tuples, one from each relation, into a single tuple. There are a number of different ways to join relations (as we shall see in Chapter 3). Figure 2.12 shows an example of joining the tuples from the *instructor* and *department* tables with the new tuples showing the information about each instructor and the department in which she is working. This result was formed by combining each tuple in the *instructor* relation with the tuple in the *department* relation for the instructor’s department.

In the form of join shown in Figure 2.12, which is called a *natural join*, a tuple from the *instructor* relation matches a tuple in the *department* relation if the values

<i>ID</i>	<i>salary</i>
10101	65000
12121	90000
15151	40000
22222	95000
32343	60000
33456	87000
45565	75000
58583	62000
76543	80000
76766	72000
83821	92000
98345	80000

Figure 2.11 Result of query selecting attributes *ID* and *salary* from the *instructor* relation.

<i>ID</i>	<i>name</i>	<i>salary</i>	<i>dept_name</i>	<i>building</i>	<i>budget</i>
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
12121	Wu	90000	Finance	Painter	120000
15151	Mozart	40000	Music	Packard	80000
22222	Einstein	95000	Physics	Watson	70000
32343	El Said	60000	History	Painter	50000
33456	Gold	87000	Physics	Watson	70000
45565	Katz	75000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
76543	Singh	80000	Finance	Painter	120000
76766	Crick	72000	Biology	Watson	90000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000

Figure 2.12 Result of natural join of the *instructor* and *department* relations.

of their *dept_name* attributes are the same. All such matching pairs of tuples are present in the join result. In general, the natural join operation on two relations matches tuples whose values are the same on all attribute names that are common to both relations.

The *Cartesian product* operation combines tuples from two relations, but unlike the join operation, its result contains *all* pairs of tuples from the two relations, regardless of whether their attribute values match.

Because relations are sets, we can perform normal set operations on relations. The *union* operation performs a set union of two “similarly structured” tables (say a table of all graduate students and a table of all undergraduate students). For example, one can obtain the set of all students in a department. Other set operations, such as *intersection* and *set difference* can be performed as well.

As we noted earlier, we can perform operations on the results of queries. For example, if we want to find the *ID* and *salary* for those instructors who have salary greater than \$85,000, we would perform the first two operations in our example above. First we select those tuples from the *instructor* relation where the *salary* value is greater than \$85,000 and then, from that result, select the two attributes *ID* and *salary*, resulting in the relation shown in Figure 2.13 consisting of the *ID*

<i>ID</i>	<i>salary</i>
12121	90000
22222	95000
33456	87000
83821	92000

Figure 2.13 Result of selecting attributes *ID* and *salary* of instructors with salary greater than \$85,000.

RELATIONAL ALGEBRA

The relational algebra defines a set of operations on relations, paralleling the usual algebraic operations such as addition, subtraction or multiplication, which operate on numbers. Just as algebraic operations on numbers take one or more numbers as input and return a number as output, the relational algebra operations typically take one or two relations as input and return a relation as output.

Relational algebra is covered in detail in Chapter 6, but we outline a few of the operations below.

Symbol (Name)	Example of Use
σ (Selection)	$\sigma_{\text{salary} \geq 85000}(\text{instructor})$ Return rows of the input relation that satisfy the predicate.
Π (Projection)	$\Pi_{ID, salary}(\text{instructor})$ Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
\bowtie (Natural join)	$\text{instructor} \bowtie \text{department}$ Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
\times (Cartesian product)	$\text{instructor} \times \text{department}$ Output all pairs of rows from the two input relations (regardless of whether or not they have the same values on common attributes)
\cup (Union)	$\Pi_{name}(\text{instructor}) \cup \Pi_{name}(\text{student})$ Output the union of tuples from the two input relations.

and *salary*. In this example, we could have performed the operations in either order, but that is not the case for all situations, as we shall see.

Sometimes, the result of a query contains duplicate tuples. For example, if we select the *dept_name* attribute from the *instructor* relation, there are several cases of duplication, including “Comp. Sci.”, which shows up three times. Certain relational languages adhere strictly to the mathematical definition of a set and remove duplicates. Others, in consideration of the relatively large amount of processing required to remove duplicates from large result relations, retain duplicates. In these latter cases, the relations are not truly relations in the pure mathematical sense of the term.

Of course, data in a database must be changed over time. A relation can be updated by inserting new tuples, deleting existing tuples, or modifying tuples by

changing the values of certain attributes. Entire relations can be deleted and new ones created.

We shall discuss relational queries and updates using the SQL language in Chapters 3 through 5.

2.7 Summary

- The **relational data model** is based on a collection of tables. The user of the database system may query these tables, insert new tuples, delete tuples, and update (modify) tuples. There are several languages for expressing these operations.
- The **schema** of a relation refers to its logical design, while an **instance** of the relation refers to its contents at a point in time. The schema of a database and an instance of a database are similarly defined. The schema of a relation includes its attributes, and optionally the types of the attributes and constraints on the relation such as primary and foreign key constraints.
- A **superkey** of a relation is a set of one or more attributes whose values are guaranteed to identify tuples in the relation uniquely. A candidate key is a minimal superkey, that is, a set of attributes that forms a superkey, but none of whose subsets is a superkey. One of the candidate keys of a relation is chosen as its **primary key**.
- A **foreign key** is a set of attributes in a referencing relation, such that for each tuple in the referencing relation, the values of the foreign key attributes are guaranteed to occur as the primary key value of a tuple in the referenced relation.
- A **schema diagram** is a pictorial depiction of the schema of a database that shows the relations in the database, their attributes, and primary keys and foreign keys.
- The **relational query languages** define a set of operations that operate on tables, and output tables as their results. These operations can be combined to get expressions that express desired queries.
- The **relational algebra** provides a set of operations that take one or more relations as input and return a relation as an output. Practical query languages such as SQL are based on the relational algebra, but add a number of useful syntactic features.

Review Terms

- | | |
|------------|-----------------|
| • Table | • Attribute |
| • Relation | • Domain |
| • Tuple | • Atomic domain |

- Null value
- Database schema
- Database instance
- Relation schema
- Relation instance
- Keys
 - Superkey
 - Candidate key
 - Primary key
- Foreign key
 - Referencing relation
 - Referenced relation
- Referential integrity constraint
- Schema diagram
- Query language
 - Procedural language
 - Nonprocedural language
- Operations on relations
 - Selection of tuples
 - Selection of attributes
 - Natural join
 - Cartesian product
 - Set operations
- Relational algebra

Practice Exercises

- 2.1 Consider the relational database of Figure 2.14. What are the appropriate primary keys?
- 2.2 Consider the foreign key constraint from the *dept_name* attribute of *instructor* to the *department* relation. Give examples of inserts and deletes to these relations, which can cause a violation of the foreign key constraint.
- 2.3 Consider the *time_slot* relation. Given that a particular time slot can meet more than once in a week, explain why *day* and *start_time* are part of the primary key of this relation, while *end_time* is not.
- 2.4 In the instance of *instructor* shown in Figure 2.1, no two instructors have the same name. From this, can we conclude that *name* can be used as a superkey (or primary key) of *instructor*?
- 2.5 What is the result of first performing the cross product of *student* and *advisor*, and then performing a selection operation on the result with the predicate $s.id = ID$? (Using the symbolic notation of relational algebra, this query can be written as $\sigma_{s.id=ID}(student \times advisor)$.)

employee (*person_name*, *street*, *city*)
works (*person_name*, *company_name*, *salary*)
company (*company_name*, *city*)

Figure 2.14 Relational database for Exercises 2.1, 2.7, and 2.12.

```

branch(branch_name, branch_city, assets)
customer(customer_name, customer_street, customer_city)
loan(loan_number, branch_name, amount)
borrower(customer_name, loan_number)
account(account_number, branch_name, balance)
depositor(customer_name, account_number)

```

Figure 2.15 Banking database for Exercises 2.8, 2.9, and 2.13.

- 2.6** Consider the following expressions, which use the result of a relational algebra operation as the input to another operation. For each expression, explain in words what the expression does.
- $\sigma_{year \geq 2009}(takes) \bowtie student$
 - $\sigma_{year \geq 2009}(takes \bowtie student)$
 - $\Pi_{ID, name, course_id}(student \bowtie takes)$
- 2.7** Consider the relational database of Figure 2.14. Give an expression in the relational algebra to express each of the following queries:
- Find the names of all employees who live in city “Miami”.
 - Find the names of all employees whose salary is greater than \$100,000.
 - Find the names of all employees who live in “Miami” and whose salary is greater than \$100,000.
- 2.8** Consider the bank database of Figure 2.15. Give an expression in the relational algebra for each of the following queries.
- Find the names of all branches located in “Chicago”.
 - Find the names of all borrowers who have a loan in branch “Downtown”.

Exercises

- 2.9** Consider the bank database of Figure 2.15.
- What are the appropriate primary keys?
 - Given your choice of primary keys, identify appropriate foreign keys.
- 2.10** Consider the *advisor* relation shown in Figure 2.8, with *s_id* as the primary key of *advisor*. Suppose a student can have more than one advisor. Then, would *s_id* still be a primary key of the *advisor* relation? If not, what should the primary key of *advisor* be?
- 2.11** Describe the differences in meaning between the terms *relation* and *relation schema*.

- 2.12** Consider the relational database of Figure 2.14. Give an expression in the relational algebra to express each of the following queries:
- Find the names of all employees who work for “First Bank Corporation”.
 - Find the names and cities of residence of all employees who work for “First Bank Corporation”.
 - Find the names, street address, and cities of residence of all employees who work for “First Bank Corporation” and earn more than \$10,000.
- 2.13** Consider the bank database of Figure 2.15. Give an expression in the relational algebra for each of the following queries:
- Find all loan numbers with a loan value greater than \$10,000.
 - Find the names of all depositors who have an account with a value greater than \$6,000.
 - Find the names of all depositors who have an account with a value greater than \$6,000 at the “Uptown” branch.
- 2.14** List two reasons why null values might be introduced into the database.
- 2.15** Discuss the relative merits of procedural and nonprocedural languages.

Bibliographical Notes

E. F. Codd of the IBM San Jose Research Laboratory proposed the relational model in the late 1960s (Codd [1970]). This work led to the prestigious ACM Turing Award to Codd in 1981 (Codd [1982]).

After Codd published his original paper, several research projects were formed with the goal of constructing practical relational database systems, including System R at the IBM San Jose Research Laboratory, Ingres at the University of California at Berkeley, and Query-by-Example at the IBM T. J. Watson Research Center.

Many relational database products are now commercially available. These include IBM’s DB2 and Informix, Oracle, Sybase, and Microsoft SQL Server. Open source relational database systems include MySQL and PostgreSQL. Microsoft Access is a single-user database product that is part of the Microsoft Office suite.

Atzeni and Antonellis [1993], Maier [1983], and Abiteboul et al. [1995] are texts devoted exclusively to the theory of the relational data model.