CHAPTER

2

Relational Data Model

Objectives

- To become familiar with relational data structure.
- To understand the characteristics of relations.
- To know the definition of primary keys, candidate keys, superkeys, and foreign keys.
- To understand the relation constraints (primary key constraints, foreign key constraints, and domain constraints).
- To know relational algebra and relational calculus.

2.1 Introduction

A ***data model*** is used to describe data in a database. A ***relational data model*** describes data in terms of relations. The relational model is built around a simple and natural structure. A relation is actually a table. Tables are easy to understand and easy to use. The relational model provides a simple yet powerful way to represent data. The relational data model is used in the relational database systems. A substantial theory has been established on relational data model. This book focuses on the theory that has direct impact on developing database applications.

A relational data model has three key components: structure, integrity, and languages. ***Structure*** defines the representation of the data. ***Integrity*** imposes constraints on the data. ***Language*** provides the means for accessing and manipulating data. This chapter introduces the structure, characteristics, integrity constraints and theoretical query languages of the relational data model. Theoretical query languages such as relational algebra and relational calculus are the basis for the commercial SQL languages. Understanding the theoretical query languages helps to learn SQL.

2.2 Relational Structures

A relational database consists of a set of relations. A relation has two things in one: a ***schema*** and an ***instance*** of the schema. The schema defines the relation and an instance is the content of the relation at a given time. An instance of a relation is nothing more than a table with rows and named columns. For convenience with no confusion, we refer instances of relations as just ***relations*** or ***tables***. Tables describe the relationship among data. Each row in the table represents a record of related data. For example, "11111," "CSCI," "1301," "Introduction to Java I," and "4" are related to form a record (the first row in Figure 1.4 on Page XX) in the Course table. Not only the data in the same row are related, but also data in different tables may be related through common attributes. Suppose the database has another table named Subject that describes the subjects. A subject is described by subject ID, subject name, the department that sponsors the subject, and the subject start time. Figure 2.1 gives a sample Subject table. The Course table and the Subject table are related through their common attribute subjectId.

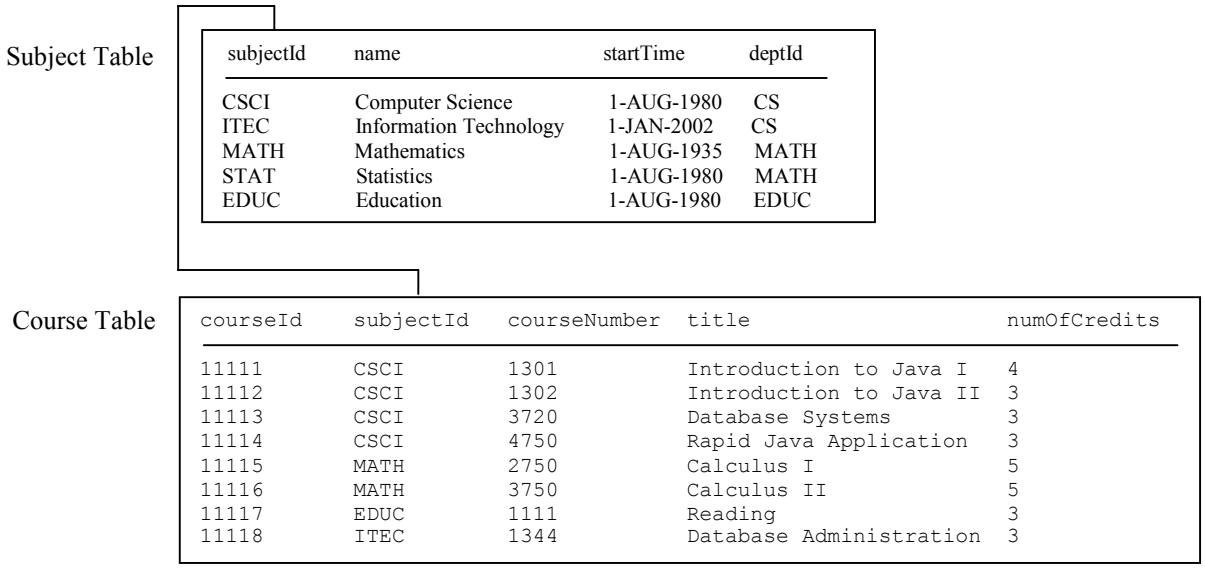


Figure 2.1

The Subject table and the Course table are related through the subjectId.

> NOTE: A table or a relation is not same as a file. Most of the relational database systems store multiple tables in a file.
>
> NOTE: All the tables used in the text are posted on the Companion Web site.

In the relational database theory, a row is called a ***tuple*** and a column is called an ***attribute***, as shown in Figure 2.2. The number of the columns is called the ***degree*** of the relation and the number of the rows is called the ***cardinality*** of the relation. The degree of the Subject table is 4 and the cardinality is 7, as shown in Figure 2.2.

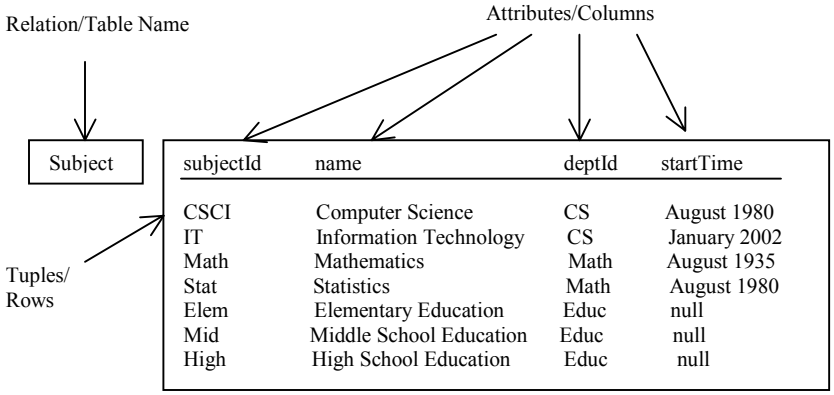


Figure 2.2

A table has a table name, column names, and rows.

Each attribute has a set of permissible values, called ***domain*** of that attribute. For example, the domain for the attribute subjectId in the Subject table is a set of subject IDs used in the university. Let D_1 denote the set of all subject IDs, D_2 the set of all subject names, D_3 the set of all department IDs, and D_4 the set of all start times. A row in the Subject table is a tuple (v_1, v_2, v_3, v_4), where $v_i \in D_i$ for (i = 1, 2, 3, 4). The Subject table is a subset of $D_1 \times D_2 \times D_3 \times D_4$. In general, a table of n attributes must be a subset of $D_1 \times D_2 \times \ldots \times D_n$.

2.2 Characteristics of Relations

A relation is a named table in a relational database. In the table, each column has a unique name and its domain is defined by an appropriate data type such as number, string, and date. Specifically, a relation has the following characteristics:

- No two records in the table are identical.
- The order of the records does not matter.
- The order of the columns does not matter.
- Each value in the column is ***atomic***, which means that it cannot be decomposed. If it is decomposed, its meaning is lost. For example, each row in the first table in Figure 2.3 describes a student and the courses took by the student. The values in the courseId column are not atomic. They can be decomposed into atomic values into the second table in Figure 2.3.

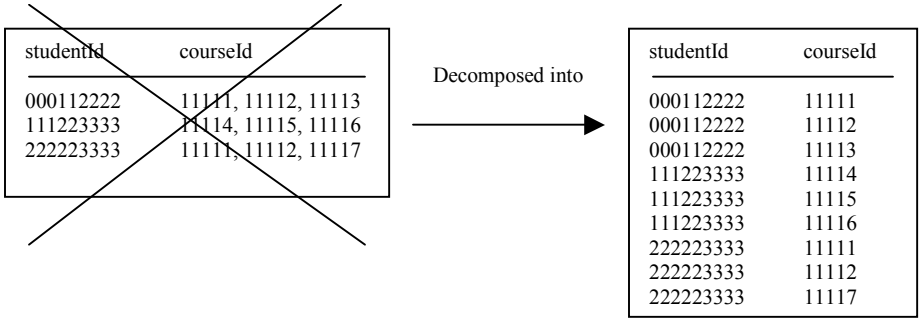


Figure 2.3

The first table contains non-atomic values in the courseId column. These values can be decomposed into single atomic values in the second table.

- A value may be ***null***, which means unknown or not applicable. For example, the start time for a subject may be null if it is not known and the aptNo value in the License table in Figure 2.4 may be null if the address does not have an apartment number.

License Table

ssn	licenseNo	Name	Street	aptNo	zipCode
111223333	133445555	John King	100 Main Stree	null	30211
211334444	233345535	George Yao	210 Mountain Rd	A	50211
321334445	313345545	Frank Khan	100 Washington St	null	30411
431334446	422345555	Jill Long	210 Franklin Rd	340	30551
531334447	522345595	Dick Frew	313 Deer Creek	341-c	60211
631334448	622345565	Sam Gross	210 Speed Way	340	70218

Figure 2.4

A license is described by ssn, licenseNo, Name, Street, aptNo, and zipCode.

2.3 Integrity Constraints

An integrity constraint imposes a condition that all legal instances of the relations must satisfy. Figure 2.5 shows an example of some integrity constraints:

Subject Table

subjecteId	name	startTime	deptId
CSCI	Computer Science	1-AUG-1980	CS
ITEC	Information Technology	1-JAN-2002	CS
MATH	Mathematics	1-AUG-1935	MATH
STAT	Statistics	1-AUG-1980	MATH
EDUC	Education	1-AUG-1980	EDUC

Each row must have a value for subjectId, and the value must be unique

Each value in subjectId must match a value in subjectId in the Subject table

Each value in the numOfCredits column must be greater than 0 and less than 5

Course Table

courseId	subjectId	number	title	numOfCredits
11111	CSCI	1301	Introduction to Java I	4
11112	CSCI	1302	Introduction to Java II	3
11113	CSCI	3720	Database Systems	3
11114	CSCI	4750	Rapid Java Application	3

Each row must have a value for couserId, and the value must be unique

Figure 2.5

The Subject table and the Course table have integrity constraints.

In general, there are three types of constraints: ***domain constraint, primary key constraint***, and ***foreign key constraint***. Domain constraints and primary key constraints are known as ***intra-relational constraints***, meaning that a constraint involves only one relation. The foreign key constraint is known as ***inter-relational***, meaning that a constraint involves more than one relation.

2.3.1 Domain Constraints

Domain constraints specify the permissible values for an attribute. Domains can be specified using standard data types such as integers, floating-point numbers, fixed-length strings, and variant-length strings. The standard data type specifies a broad range of values. Additional constraints can be specified to narrow the ranges. You can also specify whether an attribute can be null. For example, the following SQL statement specifies the data type for each attribute, specifies that the attribute cannot be null, and specifies numOfCredits must be greater than or equal to 1.

```
create table Course(
  courseId char(5),
  subjectId char(4) not null,
  number int not null,
  title varchar(50) not null,
  numOfCredits int not null
    constraint greaterThanOne check (numOfCredits >= 1));
```

2.3.2 Primary Key Constraints

To understand primary keys, it is helpful to know superkeys, keys, and candidate keys. A ***superkey*** is an attribute or a set of attributes that uniquely identify the relation. That is, no two tuples have the same values on the superkey. By definition, a relation consists of a set of distinct tuples. The set of all attributes in the relation forms a superkey.

A ***key*** K is a minimal superkey, meaning that any proper subset of K is not a superkey. It is possible that a relation has several keys. In this case, each of the keys is called a ***candidate key***. For example, in the License table in Figure 2.4, SSN and license number are the candidate keys. The ***primary*** key is one of the candidate keys designated by the database designer. The primary key is often used to identify tuples in a relation.

The primary key can be defined in the logical database

schema using the create table statement in SQL. For example, the following statement creates the Course table with subjectId and number together as the primary key.

```
create table Course(
  courseId char(5),
  subjectId char(4),
  number int,
  title varchar(50), numOfCredits int
    constraint greaterThanOne check (numOfCredits >= 1),
  primary key (subjectId, number));
```

The ***primary key constraint*** specifies that the primary key value of a tuple cannot be null and no two tuples in the relation can have the same value on the primary key.

2.3.3 Foreign Key Constraints

In a relational database, data are related. Tuples in a relation are related and tuples in different relations are related through their common attributes. Informally speaking, the common attributes are foreign keys. The ***foreign key constraints*** define the relationships among relations.

Formally, a set of attributes ***FK*** is a ***foreign key*** in a relation ***R*** that references relation ***T*** if it satisfies the following two rules:

- The attributes in ***FK*** have the same domain as the primary key in ***T***.
- A non-null value on ***FK*** in ***R*** must match a primary key value in ***T***.

As shown in Figure 2.6, collegeId is the foreign key in Department that references the primary key collegeId in College. Every collegeId value must match a collegeId value in College.

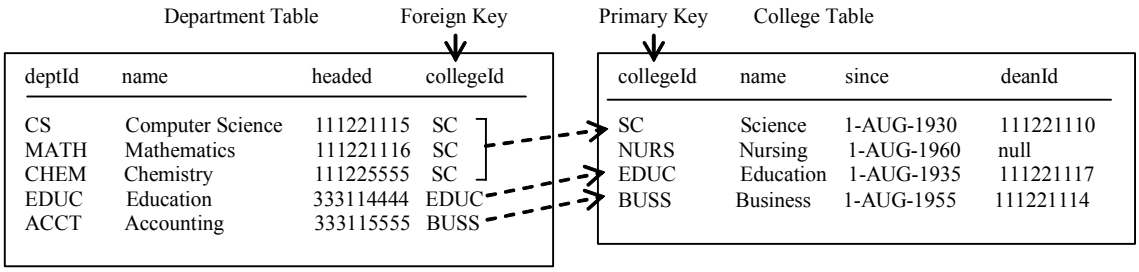


Figure 2.6

The non-null foreign key value must match its referenced primary key value.

R is called a ***referencing relation*** and *T* is called a ***referenced relation***. The graphical notation in Figure 2.7 is used to denote that R references T through foreign key FK.

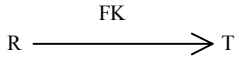
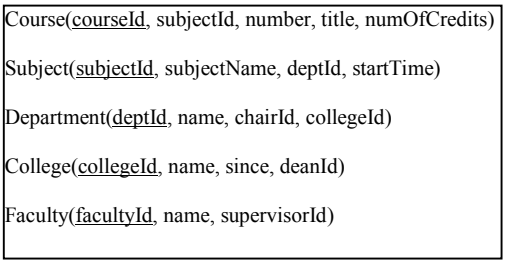


Figure 2.7

The foreign key relationship can be described graphically using an arrowed line.

Figure 2.8 shows a list of related tables and their foreign keys. The primary keys are underlined.



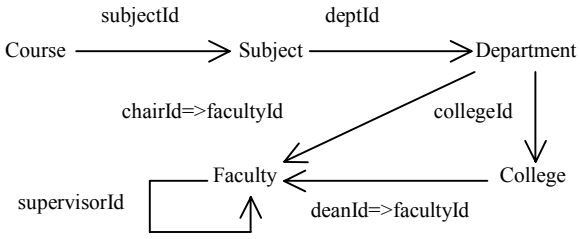


Figure 2.8

Tables in a relational database are related through foreign keys.

Several points arise with the foreign keys:

- A foreign key is not necessarily the primary key or part of the primary in the relation. For example, subjectId is a foreign key in the Course table that references the Subject table, but it is not the primary key in Course. departmentId is a foreign key in the Subject table that references Department, but it is not the primary key in Subject.
- The referencing relation and the referenced relation may be the same table. For example, supervisorId is a foreign key in Faculty that references facultyId in Faculty.
- The foreign key is not necessary to have the same name as its referenced primary key as long as they have the same domain. For example, headId is a foreign key in Department that references facultyId in Faculty.
- A relation may have more than one foreign key. For example, headId and collegeId are both foreign keys in Department.
- A foreign key value may be null. For example, if the

department head position is vacated for a department, the headID is null.

The foreign key can be defined in the logical database schema using the create table statement in SQL. For example, the following statement creates the Course table with subjectId as a foreign key that references the Subject table.

```
create table Course(
  courseId char(5),
  subjectId char(4),
  number int,
  title varchar(50),
  numOfCredits int
    constraint greaterThanOne check (numOfCredits >= 1),
  primary key (subjectId, number),
  foreign key (subjectId) references Subject(subjectId));
```

20.2.2.4 Enforcing Integrity Constraints

DBMS enforces the integrity constraints and rejects any operations that would violate the constraints. For example, if you attempt to insert a new record ('11113', '3272', 'Database Systems', 0) into the Course table, it would fail because the credit hours must be greater than or equal to 1; if you attempt to insert a record with the same primary key as an existing record in the table, the DBMS would report an error and reject the operation; if you attempt to delete a record in the Course table whose primary key value is referenced by the records in the Enrollment table, the DBMS would reject this operation.

> NOTE: All relational database systems support the primary key constraints and the foreign key constraints. Not all database systems support the domain constraints. For example, you cannot specify the constraint that numOfCredits is greater than or equal to 1 on Microsoft Access database.

2.4 Query Languages (Optional)

In the preceding sections, you learned to represent data and impose constraints on the data in the tables. This section turns attention to how to retrieve data from the tables. The commercial database systems use the SQL language to retrieve data. SQL is based on the theoretical database languages – relational algebra and relation calculus. Becoming familiar with these two languages will give you a better understanding on the database system and the SQL language.

2.4.1 Relational Algebra

The relational model enables you to define relations and specify constraints on the relations. It also supports the operations, known as the ***relational algebra***, to manipulate the relations. Relational algebra has seven essential operators: + (***union)***, ∩ (***intersect)***, - (difference), σ (selection), π (projection), × (product), and (join). These operators can be used to formulate basic queries.

> NOTE: The commercial counterpart of the language is SQL. Though relational algebra is not used by the user, knowing it helps understand relational models and learn SQL. The concept of relational algebra is used to implement and optimize SQL inside the DBMS.

2.4.1.1 Set Operators

Now you know a relation is a set of tuples and each tuple in a relation has the same number and types of attributes. The mathematical set operators (union, intersect, and difference) can be applied on the relations that have the same types of attributes.

The ***union*** of two relations yields a relation that combines all distinct tuples from both relations. The ***intersect*** of two relations is a set of tuples, each of which appears in both relations. The ***difference*** between two relations is a set of tuples that are in the first relation but not in the second. Figure 2.9 demonstrates these operators.

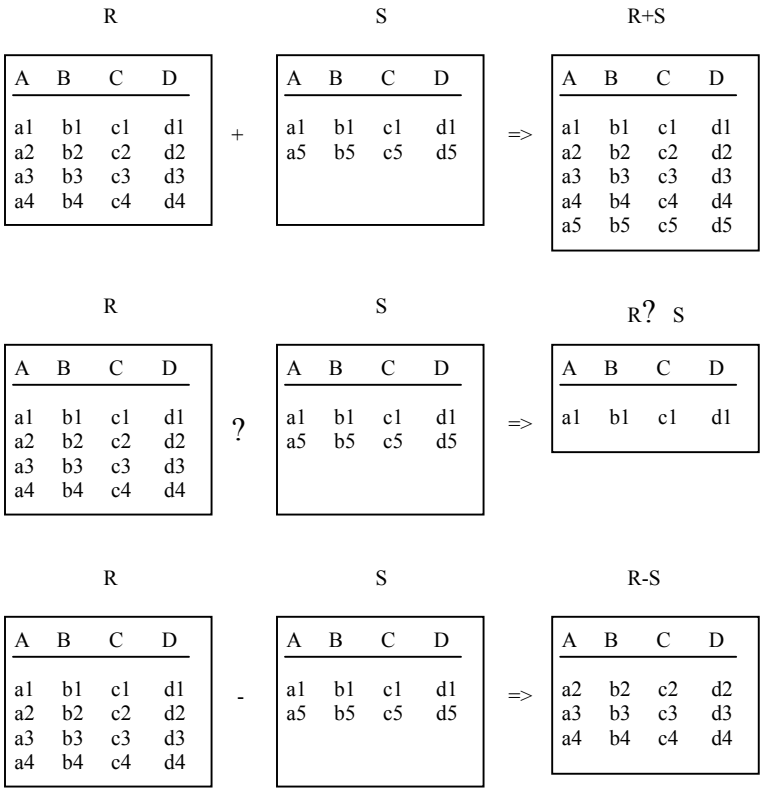


Figure 2.9

Relational algebra has three set operators.

2.4.1.2 Relational Operators

The selection (σ), project (π), product (×), and join (□) are known as ***relational operators***. The ***selection operator*** selects the tuples from a relation that satisfies a condition. For example, to select all three-credit hour courses, you can write an expression using the selection operator as follows:

$\sigma_{(numOfCredits = 3)}(Course)$

The equivalent SQL statement is

```
select *
from Course
where numOfCredits = 3;
```

Figure 2.10 shows the result relation of the expression for the Course table in Figure 1.4. The new relation has the same attributes as the input relation, but may have fewer rows.

The result of $\sigma_{(numOfCredits = 3)}$(Course)

courseId	subjectId	courseNumber	title	numOfCredits
11112	CSCI	1302	Introduction to Java II	3
11113	CSCI	3720	Database Systems	3
11114	CSCI	4750	Rapid Java Application	3
11117	EDUC	1111	Reading	3
11118	ITEC	1344	Database Administration	3

Figure 2.10

The selection operator selects the tuples from a relation with a condition.

> NOTE: You can use the comparison operators and Boolean operators to form the selection condition. The comparison operators are = (equal), < (less than), <= (less than or equal), > (greater than), >= (greater than or equal), and <> (not equal) and the Boolean operators are and, or, not.

The ***projection operator*** produces a new relation with the specified attributes from the input relation. For example, to obtain the course titles, you can write an expression using the projection operator as follows:

π_{title}(Course)

The equivalent SQL statement is

```
select title
from Course
```

Figure 2.11 shows the result relation of the expression.

The result of π_{title}(Course)

title
Introduction to Java I
Introduction to Java II
Database Systems
Rapid Java Application
Calculus I
Calculus II
Reading
Database Administration

Figure 2.11

The projection operator selects the attributes from a relation.

The ***product*** operator performs the Cartesian product on two relations. It produces a tuple of the new relation for each combination of tuples from both input relations. The product of Department and College can be written in relational algebra as follows:

```
Department × College
```

The equivalent SQL statement is

```
select Department.*, College.*
from Department, College
```

Figure 2.12 shows the result of the product of Department and College (Figure 2.6). All the attributes from the two relations are in the new relation.

The result of Department × College

deptId	name	headId	collegeId	College.collegeId	name	since	deanId
CS	Computer Science	111221115	SC	SC	Science	1-AUG-1930	999001111
CS	Computer Science	111221115	SC	NURS	Nursing	1-AUG-1960	777001111
CS	Computer Science	111221115	SC	EDUC	Education	1-AUG-1935	888001111
CS	Computer Science	111221115	SC	BUSS	Business	1-AUG-1955	666001111
MATH	Mathematics	111221116	SC	SC	Science	1-AUG-1930	999001111
MATH	Mathematics	111221116	SC	NURS	Nursing	1-AUG-1960	777001111
MATH	Mathematics	111221116	SC	EDUC	Education	1-AUG-1935	888001111
MATH	Mathematics	111221116	SC	BUSS	Business	1-AUG-1955	666001111
CHEM	Chemistry	111225555	SC	SC	Science	1-AUG-1930	999001111
CHEM	Chemistry	111225555	SC	NURS	Nursing	1-AUG-1960	777001111
CHEM	Chemistry	111225555	SC	EDUC	Education	1-AUG-1935	888001111
CHEM	Chemistry	111225555	SC	BUSS	Business	1-AUG-1955	666001111
EDUC	Education	333114444	EDUC	SC	Science	1-AUG-1930	999001111
EDUC	Education	333114444	EDUC	NURS	Nursing	1-AUG-1960	777001111
EDUC	Education	333114444	EDUC	EDUC	Education	1-AUG-1935	888001111
EDUC	Education	333114444	EDUC	BUSS	Business	1-AUG-1955	666001111
ACCT	Accounting	333115555	BUSS	SC	Science	1-AUG-1930	999001111
ACCT	Accounting	333115555	BUSS	NURS	Nursing	1-AUG-1960	777001111
ACCT	Accounting	333115555	BUSS	EDUC	Education	1-AUG-1935	888001111
ACCT	Accounting	333115555	BUSS	BUSS	Business	1-AUG-1955	666001111

Figure 2.12

The product of two relations produces a set of tuples that is the combination of tuples for the two relations.

The product in Figure 2.12 produces the complete combination of all tuples from the input relations. A more useful product is the one the produces the tuples with the same collegeId. This type of product is known as ***join***. collegeId is called the joined attribute. A special case of join is called ***natural join*** where the joined attribute appears only once in the result. Figure 2.13 shows the result of the

natural join of Department and College on collegeId.

The result of Department ? $_{collegeId}$ College

deptId	name	headed	collegeId	name	since	deanId
CS	Computer Science	111221115	SC	Science	1-AUG-1930	999001111
MATH	Mathematics	111221116	SC	Science	1-AUG-1930	999001111
CHEM	Chemistry	111225555	SC	Science	1-AUG-1930	999001111
EDUC	Education	333114444	EDUC	Education	1-AUG-1935	888001111
ACCT	Accounting	333115555	BUSS	Bussiness	1-AUG-1935	666001111

Figure 2.13

The product of two relations produces a set of tuples that is the combination of tuples for the two relations.

2.4.1.3 Combining Operators

The result of a relational algebra operation is a new relation, which can be further manipulated using the relational algebra. You can combine relational operators to produce query results. Here are three examples:

1. Display the course title with four credit hours.

$$\pi_{title}(\sigma_{numOfCredits = 4}(Course))$$

The equivalent SQL statement is

```
select title
from Course
where numOfCredits = 4;
```

2. Find all the departments in the Science college.

$\pi_{Department.name}$ ($\sigma_{College.name = 'Science'}$ (Department □ $_{collegeId}$ College)

The equivalent SQL statement is

```
select Department.name
from Department, College
where College.name = 'Science';
```

3. Display the course title with either three credit hours or four credit hours.

π_{title} ($\sigma_{numOfCredits = 3}$(Course)) union π_{title} ($\sigma_{numOfCredits = 4}$(Course))

The equivalent SQL statement is

```
select title from Course where numOfCredits = 3
union
select title from Course where numOfCredits = 4;
```

The relational algebra queries are not unique. This query can be rewritten as follows:

$\pi_{\text{title}}(\sigma_{\text{numOfCredits = 3 or numOfCredits = 4}}(\text{Course}))$

The equivalent SQL statement is

```
select title
from Course
where numOfCredits = 3 or numOfCredits = 4;
```

2.4.2 Relational Calculus

When you write the relational algebra queries, you use the set operators and relational operators to specify a sequence of actions that generates the result. Relational algebra is a procedural language in the sense that it specifies the procedure for obtaining the queries. Another relational language is called the ***relational calculus***. The relational calculus is based on the predicate calculus. It is a non-procedural language, in the sense that it specifies what the result is, but not how to obtain the result. There are several forms of relational calculus. This section introduces the ***tuple relational calculus***.

The tuple relational calculus uses tuple variables to references the tuples in the relations and uses the Boolean expressions to specify the conditions for the tuples. A query in the tuple relational calculus is expressed as

```
{result | boolean-condition}
```

where result is a set of tuples such as the Boolean-condition is true.

Here are three examples to demonstrate the relational calculus.

1. Display the course title with four credit hours.

```
{t.title | t ∈ Course and t.numOfCredits = 4}
```

Here t is a tuple variable in the Course table, as shown in Figure 2.14. The result of the query is t.title such that t ∈ Course and t.numOfCredits = 4 is true, i.e. for every tuple variable t in Course, if t.numOfCredits is 4, t.title is selected in the result.

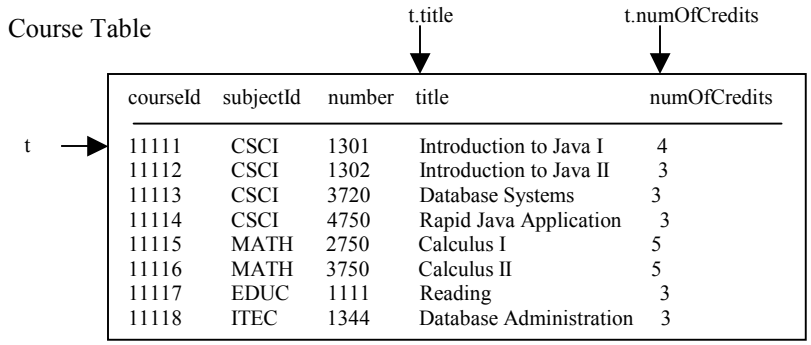


Figure 2.14

The tuple variable references to a tuple in the relation.

2. Display all the departments in the Science college.

```
{d.name | d ∈ Department and ∃c(c ∈ College
  and d.collegeId = c.collegeId and c.name = 'Science')}
```

Here d is a tuple variable in the Department table and c is a tuple in the College table, as shown in Figure 2.15. The result of the query is d.name such that for every tuple variable d in Department, ***there exists*** a tuple variable c in College such that d.collegeId = c.collegeId and c.name is 'Science'. The mathematical symbol ∃ expresses the meaning of existence of such a tuple in the College table.

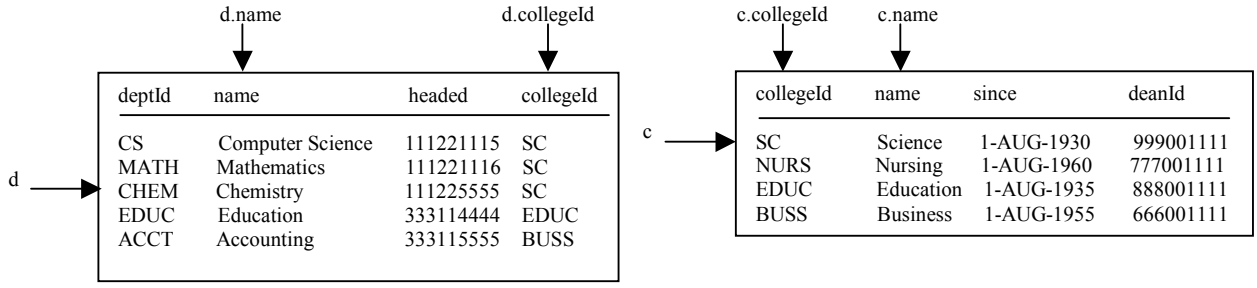


Figure 2.15

d.name is in the result if there exists c such that c.name is Science and c.collegeId is the same as d.collegeId.

3. Display the course title offered by the CS department.

```
{c.title | c ∈ Course and ∃s(s ∈ Subject
  and s.subjectId = c.subjectId and s.deptId = 'CS')}
```

Here c is a tuple variable in the Course table and s is a tuple in the Subject table, as shown in Figure 2.16. The

result of the query is c.title such that for every tuple variable c in Course, *there exists* a tuple variable s in Subject such that c.subjectId = s.subjectId and s.deptId is 'CS'.

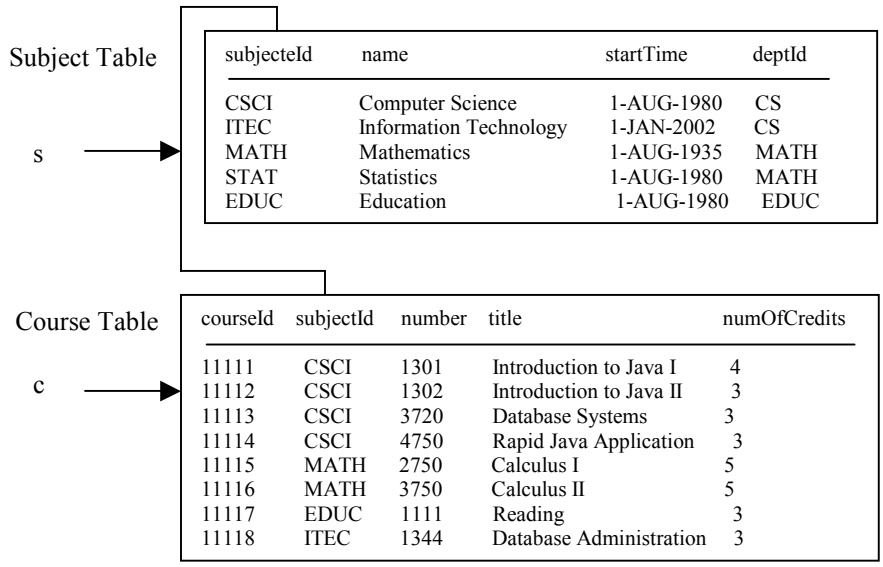


Figure 2.16

c.title is in the result if there exists s such that s.deptId is CS and c.subjectID is the same as s.subjectId.

Chapter Summary

This chapter introduced the relational data model – data structures, integrity, and query language. You learned the characteristics of relations, superkeys, keys, candidate keys, primary keys, and foreign keys. You learned the three types of integrity constraints – domain constraints, primary key constraints, and foreign key constraints. You also learned two theoretical query languages – relational algebra and relational calculus.

Review Questions

2.1 What are the characteristics of a relation?
2.2 What are superkeys, candidate keys, and primary keys? How do you create a table with a primary key?
2.3 What is a foreign key? How do you create a table with a foreign key?
2.4 Can a relation have more than one primary key or foreign key?
2.5 Does a foreign key need to be a primary key in the same relation?
2.6 Does a foreign key need to have the same name as its

referenced primary key?
2.7 Can a foreign key value be null?
2.8 What are the operators on relations?
2.9 Suppose relation R has *m* tuples and relation S has *n* tuples, answer the following questions:

a. How many tuples are in R × S?
b. How many columns are in R × S?
c. Suppose R and S have a common attribute, T is the natural join of R and S on the common attribute. What is the maximum number of tuples in T and what is the minimum number of tuples in T? How many columns are in T?
d. Suppose the common attribute is the primary key in R, what is the maximum number of tuples in T and what is the minimum number of tuples in T?
e. Suppose the common attribute in S is a foreign key that references the primary key in R. What is the number of tuples in T?

Exercises

2.1 Write the following queries using relational algebra and relational calculus, respectively. (The queries use the Subject and Course tables in Figure 2.1)

a. Find the subject start time for the subjectId CSCI.
b. Find the course titles in the CS department.
c. Find the course titles in the CS department excluding the course whose ID is 11111.
d. List the course titles in both MATH and CSCI subjects.