3

THE RELATIONAL MODEL

Exercise 3.1 Define the following terms: *relation schema, relational database schema, domain, attribute, attribute domain, relation instance, relation cardinality*, and *relation degree*.

Answer 3.1 A *relation schema* can be thought of as the basic information describing a table or *relation*. This includes a set of column names, the data types associated with each column, and the name associated with the entire table. For example, a relation schema for the relation called Students could be expressed using the following representation:

Students(*sid:* `string`, *name:* `string`, *login:* `string`,
age: `integer`, *gpa:* `real`)

There are five fields or columns, with names and types as shown above.

A *relational database schema* is a collection of relation schemas, describing one or more relations.

Domain is synonymous with *data type*. *Attributes* can be thought of as columns in a table. Therefore, an *attribute domain* refers to the data type associated with a column.

A *relation instance* is a set of tuples (also known as *rows* or *records*) that each conform to the schema of the relation.

The *relation cardinality* is the number of tuples in the relation.

The *relation degree* is the number of fields (or columns) in the relation.

Exercise 3.2 How many distinct tuples are in a relation instance with cardinality 22?

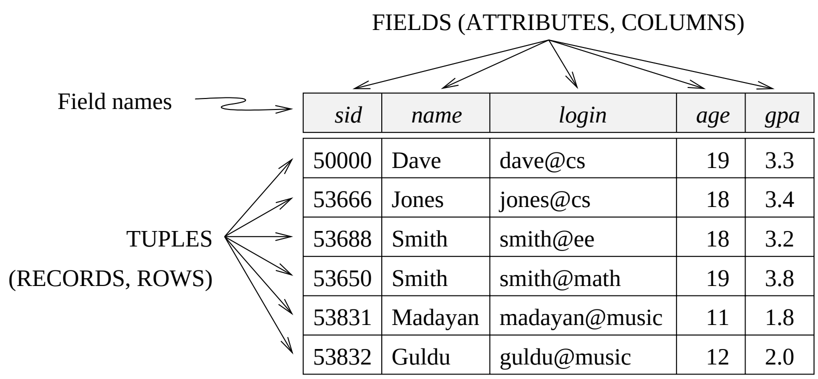
Answer 3.2 Answer omitted.

Exercise 3.3 Does the relational model, as seen by an SQL query writer, provide physical and logical data independence? Explain.

Answer 3.3 The user of SQL has no idea how the data is physically represented in the machine. He or she relies entirely on the relation abstraction for querying. Physical data independence is therefore assured. Since a user can define views, logical data independence can also be achieved by using view definitions to hide changes in the conceptual schema.

Exercise 3.4 What is the difference between a candidate key and the primary key for a given relation? What is a superkey?

Answer 3.4 Answer omitted.



sid	*name*	*login*	*age*	*gpa*
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Figure 3.1 An Instance $S1$ of the Students Relation

Exercise 3.5 Consider the instance of the Students relation shown in Figure 3.1.

1. Give an example of an attribute (or set of attributes) that you can deduce is *not* a candidate key, based on this instance being legal.
2. Is there any example of an attribute (or set of attributes) that you can deduce *is* a candidate key, based on this instance being legal?

Answer 3.5 Examples of non-candidate keys include the following: {name}, {age}. (Note that {gpa} can *not* be declared as a non-candidate key from this evidence alone even though common sense tells us that clearly more than one student could have the same grade point average.)

You cannot determine a key of a relation given only one instance of the relation. The fact that the instance is “legal” is immaterial. A candidate key, as defined here, *is a* *key*, not something that only *might* be a key. The instance shown is just one possible "snapshot" of the relation. At other times, the same relation may have an instance (or snapshot) that contains a totally different set of tuples, and we cannot make predictions about those instances based only upon the instance that we are given.

Exercise 3.6 What is a foreign key constraint? Why are such constraints important? What is referential integrity?

Answer 3.6 Answer omitted.

Exercise 3.7 Consider the relations Students, Faculty, Courses, Rooms, Enrolled, Teaches, and Meets_In defined in Section 1.5.2.

1. List all the foreign key constraints among these relations.
2. Give an example of a (plausible) constraint involving one or more of these relations that is not a primary key or foreign key constraint.

Answer 3.7 There is no reason for a foreign key constraint (FKC) on the Students, Faculty, Courses, or Rooms relations. These are the most basic relations and must be free-standing. Special care must be given to entering data into these base relations.

In the Enrolled relation, *sid* and *cid* should both have FKCs placed on them. (Real students must be enrolled in real courses.) Also, since real teachers must teach real courses, both the fid and the *cid* fields in the Teaches relation should have FKCs. Finally, Meets_In should place FKCs on both the *cid* and *rno* fields.

It would probably be wise to enforce a few other constraints on this DBMS: the length of *sid*, *cid*, and fid could be standardized; checksums could be added to these identification numbers; limits could be placed on the size of the numbers entered into the credits, capacity, and salary fields; an enumerated type should be assigned to the grade field (preventing a student from receiving a grade of G, among other things); etc.

Exercise 3.8 Answer each of the following questions briefly. The questions are based on the following relational schema:

> Emp(<u>*eid:* `integer`</u>, *ename:* `string`, *age:* `integer`, *salary:* `real`)
> Works(<u>*eid:* `integer`, *did:* `integer`</u>, *pcttime:* `integer`)
> Dept(<u>*did:* `integer`</u>, *dname:* `string`, *budget:* `real`, *managerid:* `integer`)

1. Give an example of a foreign key constraint that involves the Dept relation. What are the options for enforcing this constraint when a user attempts to delete a Dept tuple?

2. Write the SQL statements required to create the preceding relations, including appropriate versions of all primary and foreign key integrity constraints.

3. Define the Dept relation in SQL so that every department is guaranteed to have a manager.

4. Write an SQL statement to add John Doe as an employee with $eid = 101$, $age = 32$ and $salary = 15,000$.

5. Write an SQL statement to give every employee a 10 percent raise.

6. Write an SQL statement to delete the Toy department. Given the referential integrity constraints you chose for this schema, explain what happens when this statement is executed.

Answer 3.8 Answer omitted.

sid	*name*	*login*	*age*	*gpa*
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Figure 3.2 Students with $age < 18$ on Instance S

Exercise 3.9 Consider the SQL query whose answer is shown in Figure 3.2.

1. Modify this query so that only the *login* column is included in the answer.

2. If the clause `WHERE` *S.gpa >= 2* is added to the original query, what is the set of tuples in the answer?

Answer 3.9 The answers are as follows:

1. Only *login* is included in the answer:

```
SELECT S.login
FROM   Students S
WHERE  S.age < 18
```

2. The answer tuple for Madayan is omitted then.

Exercise 3.10 Explain why the addition of `NOT NULL` constraints to the SQL definition of the Manages relation (in Section 3.5.3) does not enforce the constraint that each department must have a manager. What, if anything, is achieved by requiring that the *ssn* field of Manages be non-*null*?

Answer 3.10 Answer omitted.

Exercise 3.11 Suppose that we have a ternary relationship R between entity sets A, B, and C such that A has a key constraint and total participation and B has a key constraint; these are the only constraints. A has attributes $a1$ and $a2$, with $a1$ being the key; B and C are similar. R has no descriptive attributes. Write SQL statements that create tables corresponding to this information so as to capture as many of the constraints as possible. If you cannot capture some constraint, explain why.

Answer 3.11 The following SQL statements create the corresponding relations.

```
CREATE TABLE A ( a1      CHAR(10),
                 a2      CHAR(10),
                 b1      CHAR(10),
                 c1      CHAR(10),
                 PRIMARY KEY (a1),
                 UNIQUE (b1),
                 FOREIGN KEY (b1) REFERENCES B,
                 FOREIGN KEY (c1) REFERENCES C )

CREATE TABLE B ( b1      CHAR(10),
                 b2      CHAR(10),
                 PRIMARY KEY (b1) )

CREATE TABLE C ( b1      CHAR(10),
                 c2      CHAR(10),
                 PRIMARY KEY (c1) )
```

The first SQL statement folds the relationship R into table A and thereby guarantees the participation constraint.

Exercise 3.12 Consider the scenario from Exercise 2.2, where you designed an ER diagram for a university database. Write SQL statements to create the corresponding relations and capture as many of the constraints as possible. If you cannot capture some constraints, explain why.

Answer 3.12 Answer omitted.

Exercise 3.13 Consider the university database from Exercise 2.3 and the ER diagram you designed. Write SQL statements to create the corresponding relations and capture as many of the constraints as possible. If you cannot capture some constraints, explain why.

Answer 3.13 The following SQL statements create the corresponding relations.

1. CREATE TABLE Professors (
```
CREATE TABLE Professors ( prof_ssn   CHAR(10),
                          name       CHAR(64),
                          age        INTEGER,
                          rank       INTEGER,
                          speciality CHAR(64),
                          PRIMARY KEY (prof_ssn) )
```

2.
```
CREATE TABLE Depts ( dno    INTEGER,
                     dname  CHAR(64),
                     office CHAR(10),
                     PRIMARY KEY (dno) )
```

3.
```
CREATE TABLE Runs ( dno      INTEGER,
                    prof_ssn CHAR(10),
                    PRIMARY KEY ( dno, prof_ssn),
                    FOREIGN KEY (prof_ssn) REFERENCES Professors,
                    FOREIGN KEY (dno) REFERENCES Depts )
```

4.
```
CREATE TABLE Work_Dept ( dno      INTEGER,
                         prof_ssn CHAR(10),
                         pc_time  INTEGER,
                         PRIMARY KEY (dno, prof_ssn),
                         FOREIGN KEY (prof_ssn) REFERENCES Professors,
                         FOREIGN KEY (dno) REFERENCES Depts )
```

   Observe that we would need check constraints or assertions in SQL to enforce the rule that Professors work in at least one department.

5.
```
CREATE TABLE Project ( pid        INTEGER,
                       sponsor    CHAR(32),
                       start_date DATE,
                       end_date   DATE,
                       budget     FLOAT,
                       PRIMARY KEY (pid) )
```

6.
```
CREATE TABLE Graduates ( grad_ssn CHAR(10),
                         age      INTEGER,
                         name     CHAR(64),
                         deg_prog CHAR(32),
```

```
major INTEGER,
PRIMARY KEY (grad_ssn),
FOREIGN KEY (major) REFERENCES Depts )
```

Note that the Major table is not necessary since each Graduate has only one major and so this can be an attribute in the Graduates table.

7. ```
   CREATE TABLE Advisor ( senior_ssn CHAR(10),
                          grad_ssn   CHAR(10),
                          PRIMARY KEY (senior_ssn, grad_ssn),
                          FOREIGN KEY (senior_ssn)
                                  REFERENCES Graduates (grad_ssn),
                          FOREIGN KEY (grad_ssn) REFERENCES Graduates )
   ```

8. ```
   CREATE TABLE Manages ( pid      INTEGER,
                          prof_ssn CHAR(10),
                          PRIMARY KEY (pid, prof_ssn),
                          FOREIGN KEY (prof_ssn) REFERENCES Professors,
                          FOREIGN KEY (pid) REFERENCES Projects )
   ```

9. ```
   CREATE TABLE Work_In ( pid      INTEGER,
                          prof_ssn CHAR(10),
                          PRIMARY KEY (pid, prof_ssn),
                          FOREIGN KEY (prof_ssn) REFERENCES Professors,
                          FOREIGN KEY (pid) REFERENCES Projects )
   ```

   Observe that we cannot enforce the participation constraint for Projects in the Work_In table without check constraints or assertions in SQL.

10. ```
    CREATE TABLE Supervises ( prof_ssn CHAR(10),
                              grad_ssn CHAR(10),
                              pid      INTEGER,
                              PRIMARY KEY (prof_ssn, grad_ssn, pid),
                              FOREIGN KEY (prof_ssn) REFERENCES Professors,
                              FOREIGN KEY (grad_ssn) REFERENCES Graduates,
                              FOREIGN KEY (pid) REFERENCES Projects )
    ```

    Note that we do not need an explicit table for the Work_Proj relation since every time a Graduate works on a Project, he or she must have a Supervisor.

Exercise 3.14 Consider the scenario from Exercise 2.4, where you designed an ER diagram for a company database. Write SQL statements to create the corresponding relations and capture as many of the constraints as possible. If you cannot capture some constraints, explain why.

Answer 3.14 Answer omitted.

Exercise 3.15 Consider the Notown database from Exercise 2.5. You have decided to recommend that Notown use a relational database system to store company data. Show the SQL statements for creating relations corresponding to the entity sets and relationship sets in your design. Identify any constraints in the ER diagram that you are unable to capture in the SQL statements and briefly explain why you could not express them.

Answer 3.15 The following SQL statements create the corresponding relations.

1. ```
   CREATE TABLE Musicians ( ssn     CHAR(10),
                            name    CHAR(30),
                            PRIMARY KEY (ssn))
   ```

2. ```
   CREATE TABLE Instruments ( instrId  CHAR(10),
                              dname    CHAR(30),
                              key      CHAR(5),
                              PRIMARY KEY (instrId))
   ```

3. ```
   CREATE TABLE Plays (     ssn      CHAR(10),
                            instrId  INTEGER,
                            PRIMARY KEY (ssn, instrId),
                            FOREIGN KEY (ssn) REFERENCES Musicians,
                            FOREIGN KEY (instrId) REFERENCES Instruments )
   ```

4. ```
   CREATE TABLE Songs_Appears ( songId          INTEGER,
                                author          CHAR(30),
                                title           CHAR(30),
                                albumIdentifier INTEGER NOT NULL,
                                PRIMARY KEY (songId),
                                FOREIGN KEY (albumIdentifier)
                                                References Album_Producer,
   ```

5. ```
   CREATE TABLE Telephone_Home ( phone           CHAR(11),
                                 address         CHAR(30),
                                 PRIMARY KEY (phone),
                                 FOREIGN KEY (address) REFERENCES Place,
   ```
   ```

6. CREATE TABLE Lives (
```
ssn      CHAR(10),
phone    CHAR(11),
address  CHAR(30),
PRIMARY KEY (ssn, address),
FOREIGN KEY (phone, address)
        References Telephone_Home,
        FOREIGN KEY (ssn) REFERENCES Musicians )
```

7. CREATE TABLE Place (
```
address  CHAR(30) )
```

8. CREATE TABLE Perform (
```
songId   INTEGER,
ssn      CHAR(10),
PRIMARY KEY (ssn, songId),
FOREIGN KEY (songId) REFERENCES Songs,
FOREIGN KEY (ssn) REFERENCES Musicians )
```

9. CREATE TABLE Album_Producer (
```
albumIdentifier INTEGER,
ssn             CHAR(10),
copyrightDate   DATE,
speed           INTEGER,
title           CHAR(30),
PRIMARY KEY (albumIdentifier),
FOREIGN KEY (ssn) REFERENCES Musicians )
```

Exercise 3.16 Translate your ER diagram from Exercise 2.6 into a relational schema, and show the SQL statements needed to create the relations, using only key and null constraints. If your translation cannot capture any constraints in the ER diagram, explain why.

In Exercise 2.6, you also modified the ER diagram to include the constraint that tests on a plane must be conducted by a technician who is an expert on that model. Can you modify the SQL statements defining the relations obtained by mapping the ER diagram to check this constraint?

Answer 3.16 Answer omitted.

Exercise 3.17 Consider the ER diagram that you designed for the Prescriptions-R-X chain of pharmacies in Exercise 2.7. Define relations corresponding to the entity sets and relationship sets in your design using SQL.

Answer 3.17 The statements to create tables corresponding to entity sets Doctor, Pharmacy, and Pharm_co are straightforward and omitted. The other required tables can be created as follows:

1. ```
   CREATE TABLE Pri_Phy_Patient ( ssn        CHAR(11),
                                  name       CHAR(20),
                                  age        INTEGER,
                                  address    CHAR(20),
                                  phy_ssn    CHAR(11),
                                  PRIMARY KEY (ssn),
                                  FOREIGN KEY (phy_ssn) REFERENCES Doctor )
   ```

2. ```
   CREATE TABLE Prescription ( ssn        CHAR(11),
                               phy_ssn    CHAR(11),
                               date       CHAR(11),
                               quantity   INTEGER,
                               trade_name CHAR(20),
                               pharm_id   CHAR(11),
                               PRIMARY KEY (ssn, phy_ssn),
                               FOREIGN KEY (ssn) REFERENCES Patient,
                               FOREIGN KEY (phy_ssn) REFERENCES Doctor,
                               FOREIGN KEY (trade_name, pharm_id)
                                          References Make_Drug)
   ```

3. ```
   CREATE TABLE Make_Drug (trade_name  CHAR(20),
                           pharm_id    CHAR(11),
                           PRIMARY KEY (trade_name, pharm_id),
                           FOREIGN KEY (trade_name) REFERENCES Drug,
                           FOREIGN KEY (pharm_id) REFERENCES Pharm_co)
   ```

4. ```
   CREATE TABLE Sell (     price       INTEGER,
                           name        CHAR(10),
                           trade_name  CHAR(10),
                           PRIMARY KEY (name, trade_name),
                           FOREIGN KEY (name) REFERENCES Pharmacy,
                           FOREIGN KEY (trade_name) REFERENCES Drug)
   ```

5. ```
   CREATE TABLE Contract ( name        CHAR(20),
                           pharm_id    CHAR(11),
                           start_date  CHAR(11),
                           end_date    CHAR(11),
   ```
   ```

```
text           CHAR(10000),
supervisor     CHAR(20),
PRIMARY KEY (name, pharm_id),
FOREIGN KEY (name) REFERENCES Pharmacy,
FOREIGN KEY (pharm_id) REFERENCES Pharm_co)
```

Exercise 3.18 Write SQL statements to create the corresponding relations to the ER diagram you designed for Exercise 2.8. If your translation cannot capture any constraints in the ER diagram, explain why.

Answer 3.18 Answer omitted.

Exercise 3.19 Briefly answer the following questions based on this schema:

Emp(*eid:* `integer`, *ename:* `string`, *age:* `integer`, *salary:* `real`)
Works(*eid:* `integer`, *did:* `integer`, *pct_time:* `integer`)
Dept(*did:* `integer`, *budget:* `real`, *managerid:* `integer`)

1. Suppose you have a view SeniorEmp defined as follows:

```
CREATE VIEW SeniorEmp (sname, sage, salary)
       AS SELECT E.ename, E.age, E.salary
          FROM   Emp E
          WHERE  E.age > 50
```

   Explain what the system will do to process the following query:

```
SELECT S.sname
FROM   SeniorEmp S
WHERE  S.salary > 100,000
```

2. Give an example of a view on Emp that could be automatically updated by updating Emp.

3. Give an example of a view on Emp that would be impossible to update (automatically) and explain why your example presents the update problem that it does.

Answer 3.19 The answer to each question is given below.

1. The system will do the following:

```
SELECT  S.name
FROM    ( SELECT E.ename AS name, E.age, E.salary
          FROM   Emp E
          WHERE  E.age > 50 ) AS S
WHERE   S.salary > 100000
```

2. The following view on Emp can be updated automatically by updating Emp:

```
CREATE VIEW SeniorEmp (eid, name, age, salary)
       AS SELECT E.eid, E.ename, E.age, E.salary
          FROM   Emp E
          WHERE  E.age > 50
```

3. The following view cannot be updated automatically because it is not clear which employee records will be affected by a given update:

```
CREATE VIEW AvgSalaryByAge (age, avgSalary)
       AS SELECT   E.eid, AVG (E.salary)
          FROM     Emp E
          GROUP BY E.age
```

Exercise 3.20 Consider the following schema:

Suppliers(*sid:* `integer`, *sname:* `string`, *address:* `string`)
Parts(*pid:* `integer`, *pname:* `string`, *color:* `string`)
Catalog(*sid:* `integer`, *pid:* `integer`, *cost:* `real`)

The Catalog relation lists the prices charged for parts by Suppliers. Answer the following questions:

- Give an example of an updatable view involving one relation.
- Give an example of an updatable view involving two relations.
- Give an example of an insertable-into view that is updatable.
- Give an example of an insertable-into view that is not updatable.

Answer 3.20 Answer omitted.