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.

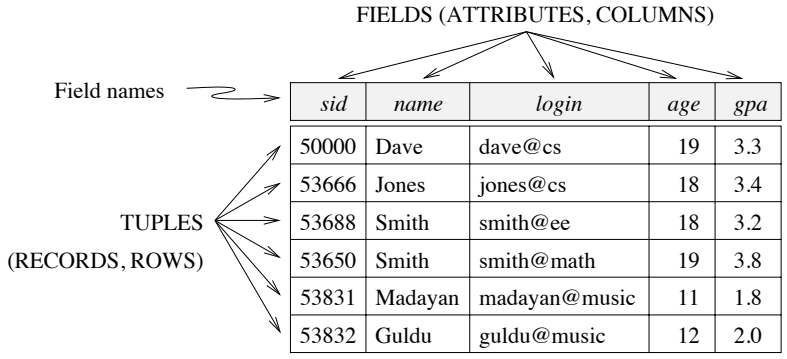
A *relation instance* is a set of tuples (also known as *rows* or *records*) that each conform to the schema of the relation.

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

Answer 3.4 The *primary key* is the key selected by the DBA from among the group of *candidate keys*, all of which uniquely identify a tuple. A *superkey* is a set of attributes that contains a key.



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 A *foreign key* constraint is a statement of the form that one or more fields of a relation, say R, together *refer* to a second relation, say S. That is, the values in these fields of a tuple in R are either *null*, or uniquely identify some tuple in S. Thus, these fields of R should be a (candidate or primary) key. For example, a student, uniquely identified by an *sid*, enrolled in a class must also be registered in the school's student database (say, in a relation called Students). Therefore, the *sid* of a legal entry in the Class_Enrollment relation must match an existing *sid* in the Students relation.

Foreign key constraints are important because they provide safeguards for insuring the integrity of data. Users are alerted/thwarted when they try to do something that does not make sense. This can help minimize errors in application programs or in data-entry.

Referential integrity means all foreign key constraints are enforced.

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(*eid:* `integer`, *ename:* `string`, *age:* `integer`, *salary:* `real`)
Works(*eid:* `integer`, *did:* `integer`, *pcttime:* `integer`)
Dept(*did:* `integer`, *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 The answers are given below:

1. Consider the following example. It is natural to require that the *did* field of Works should be a foreign key, and refer to Dept.

```
CREATE TABLE Works ( eid     INTEGER NOT NULL ,
                     did     INTEGER NOT NULL ,
```

```
                                pcttime INTEGER,
                                PRIMARY KEY (eid, did),
                                UNIQUE (eid),
                                FOREIGN KEY (did) REFERENCES Dept )
```

When a user attempts to delete a Dept tuple, There are four options:

- Also delete all Works tuples that refer to it.
- Disallow the deletion of the Dept tuple if some Works tuple refers to it.
- For every Works tuple that refers to it, set the *did* field to the *did* of some (existing) 'default' department.
- For every Works tuple that refers to it, set the *did* field to *null*.

2.
```
CREATE TABLE Emp (   eid          INTEGER,
                     ename        CHAR(10),
                     age          INTEGER,
                     salary       REAL,
                     PRIMARY KEY (eid) )
CREATE TABLE Works ( eid          INTEGER,
                     did          INTEGER,
                     pcttime      INTEGER,
                     PRIMARY KEY (eid, did),
                     FOREIGN KEY (did) REFERENCES Dept,
                     FOREIGN KEY (eid) REFERENCES Emp,
                     ON DELETE CASCADE)
CREATE TABLE Dept (  did          INTEGER,
                     budget       REAL,
                     managerid    INTEGER ,
                     PRIMARY KEY (did),
                     FOREIGN KEY (managerid) REFERENCES Emp,
                     ON DELETE SET NULL)
```

3.
```
CREATE TABLE Dept ( did        INTEGER,
                    budget     REAL,
                    managerid INTEGER  NOT NULL ,
                    PRIMARY KEY (did),
                    FOREIGN KEY (managerid) REFERENCES Emp)
```

4.
```
INSERT
INTO   Emp     (eid, ename, age, salary)
VALUES         (101, 'John Doe', 32, 15000)
```

5.
```
UPDATE Emp E
```

```
        SET     E.salary = E.salary * 1.10
```

6.
```
        DELETE
        FROM    Dept D
                WHERE D.dname = 'Toy'
```

The did field in the Works relation is a foreign key and references the Dept relation. This is the referential integrity constraint chosen. By adding the action `ON DELETE CASCADE` to this, when a department record is deleted, the Works record associated with that Dept is also deleted.

The query works as follows: The Dept relation is searched for a record with name = 'Toy' and that record is deleted. The did field of that record is then used to look in the Works relation for records with a matching did value. All such records are then deleted from the Works relation.

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