CHAPTER 2

Introduction to the Relational Model

This chapter presents the relational model and a brief introduction to the relational-algebra query language. The short introduction to relational algebra is sufficient for courses that focus on application development, without going into database internals. In particular, the chapters on SQL do not require any further knowledge of relational algebra. However, courses that cover internals, in particular query processing, require a more detailed coverage of relational algebra, which is provided in Chapter 6.

Exercises

2.9 Consider the bank database of Figure 2.15.

a. What are the appropriate primary keys?

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.

b. Given your choice of primary keys, identify appropriate foreign keys.

Answer:

a. The primary keys of the various schema are underlined. Although in a real bank the customer name is unlikely to be a primary key, since two customers could have the same name, we use a simplified schema where we assume that names are unique. We allow customers to have more than one account, and more than one loan.

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*)

b. The foreign keys are as follows
   i. For *loan*: *branch_name* referencing *branch*.
   ii. For *borrower*: Attribute *customer_name* referencing *customer* and *loan_number* referencing *loan*
   iii. For *account*: *branch_name* referencing *branch*.
   iv. For *depositor*: Attribute *customer_name* referencing *customer* and *account_number* referencing *account*

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?
Answer: No, *s_id* would not be a primary key, since there may be two (or more) tuples for a single student, corresponding to two (or more) advisors. The primary key should then be *s_id, i_id*.

2.11 Describe the differences in meaning between the terms *relation* and *relation schema*.
Answer: A relation schema is a type definition, and a relation is an instance of that schema. For example, *student* (*ss#*, *name*) is a relation schema and

123-456-222	John
234-567-999	Mary

is a relation based on that 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:

a. Find the names of all employees who work for "First Bank Corporation".

b. Find the names and cities of residence of all employees who work for "First Bank Corporation".

c. Find the names, street address, and cities of residence of all employees who work for "First Bank Corporation" and earn more than $10,000.

Answer:

a. $\Pi_{person\_name}(\sigma_{company\_name = \text{"First Bank Corporation"}}(works))$

b. $\Pi_{person\_name, city}(employee \bowtie (\sigma_{company\_name = \text{"First Bank Corporation"}}(works)))$

c. $\Pi_{person\_name, street, city}(\sigma_{(company\_name = \text{"First Bank Corporation"} \wedge salary > 10000)}(works \bowtie employee))$

2.13 Consider the bank database of Figure 2.15. Give an expression in the relational algebra for each of the following queries:

a. Find all loan numbers with a loan value greater than $10,000.

b. Find the names of all depositors who have an account with a value greater than $6,000.

c. Find the names of all depositors who have an account with a value greater than $6,000 at the "Uptown" branch.

Answer:

a. $\Pi_{loan\_number}(\sigma_{amount > 10000}(loan))$

b. $\Pi_{customer\_name}(\sigma_{balance > 6000}(depositor \bowtie account))$

c. $\Pi_{customer\_name}(\sigma_{balance > 6000 \wedge branch\_name = \text{"Uptown"}}(depositor \bowtie account))$

2.14 List two reasons why null values might be introduced into the database.

Answer: Nulls may be introduced into the database because the actual value is either unknown or does not exist. For example, an employee whose address has changed and whose new address is not yet known should be retained with a null address. If employee tuples have a composite attribute *dependents*, and a particular employee has no dependents, then that tuple's *dependents* attribute should be given a null value.

2.15 Discuss the relative merits of procedural and nonprocedural languages.

Answer: Nonprocedural languages greatly simplify the specification of queries (at least, the types of queries they are designed to handle). The free the user from having to worry about how the query is to be evaluated; not only does this reduce programming effort, but in fact in most situations the query optimizer can do a much better task of choosing the best way to evaluate a query than a programmer working by trial and error.

On the other hand, procedural languages are far more powerful in terms of what computations they can perform. Some tasks can either not be

done using nonprocedural languages, or are very hard to express using nonprocedural languages, or execute very inefficiently if specified in a nonprocedural manner.