DATABASE MANAGEMENT SYSTEMS SOLUTIONS MANUAL

Raghu Ramakrishnan et al.

University of Wisconsin

Madison, WI, USA

2
THE ENTITY-RELATIONSHIP MODEL

Exercise 2.1 Explain the following terms briefly: *attribute, domain, entity, relationship, entity set, relationship set, one-to-many relationship, many-to-many relationship, participation constraint, overlap constraint, covering constraint, weak entity set, aggregation*, and *role indicator*.

Answer 2.1 No answer provided yet.

Exercise 2.2 A university database contains information about professors (identified by social security number, or SSN) and courses (identified by courseid). Professors teach courses; each of the following situations concerns the Teaches relationship set. For each situation, draw an ER diagram that describes it (assuming that no further constraints hold).

1. Professors can teach the same course in several semesters, and each offering must be recorded.
2. Professors can teach the same course in several semesters, and only the most recent such offering needs to be recorded. (Assume this condition applies in all subsequent questions.)
3. Every professor must teach some course.
4. Every professor teaches exactly one course (no more, no less).
5. Every professor teaches exactly one course (no more, no less), and every course must be taught by some professor.
6. Now suppose that certain courses can be taught by a team of professors jointly, but it is possible that no one professor in a team can teach the course. Model this situation, introducing additional entity sets and relationship sets if necessary.

Answer 2.2 Answer omitted.

Exercise 2.3 Consider the following information about a university database:

- Professors have an SSN, a name, an age, a rank, and a research specialty.
- Projects have a project number, a sponsor name (e.g., NSF), a starting date, an ending date, and a budget.
- Graduate students have an SSN, a name, an age, and a degree program (e.g., M.S. or Ph.D.).
- Each project is managed by one professor (known as the project's principal investigator).
- Each project is worked on by one or more professors (known as the project's co-investigators).
- Professors can manage and/or work on multiple projects.
- Each project is worked on by one or more graduate students (known as the project's research assistants).
- When graduate students work on a project, a professor must supervise their work on the project. Graduate students can work on multiple projects, in which case they will have a (potentially different) supervisor for each one.
- Departments have a department number, a department name, and a main office.
- Departments have a professor (known as the chairman) who runs the department.
- Professors work in one or more departments, and for each department that they work in, a time percentage is associated with their job.
- Graduate students have one major department in which they are working on their degree.
- Each graduate student has another, more senior graduate student (known as a student advisor) who advises him or her on what courses to take.

Design and draw an ER diagram that captures the information about the university. Use only the basic ER model here, that is, entities, relationships, and attributes. Be sure to indicate any key and participation constraints.

Answer 2.3 The ER diagram is shown in Figure 2.1.

Exercise 2.4 A company database needs to store information about employees (identified by *ssn*, with *salary* and *phone* as attributes); departments (identified by *dno*, with *dname* and *budget* as attributes); and children of employees (with *name* and *age* as attributes). Employees *work* in departments; each department is *managed by* an

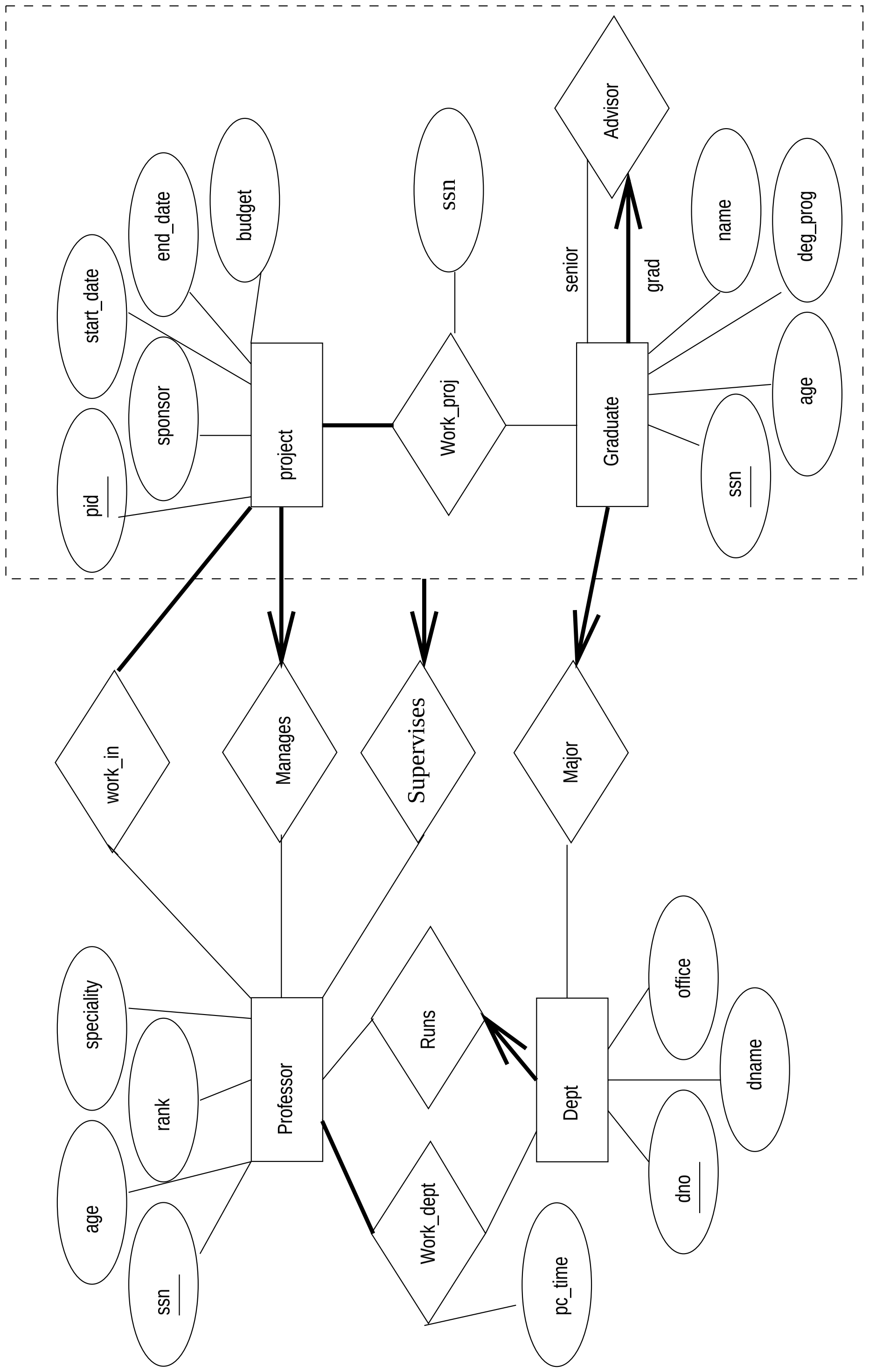


Figure 2.1 ER Diagram for Exercise 2.3

employee; a child must be identified uniquely by *name* when the parent (who is an employee; assume that only one parent works for the company) is known. We are not interested in information about a child once the parent leaves the company.

Draw an ER diagram that captures this information.

Answer 2.4 Answer omitted.

Exercise 2.5 Notown Records has decided to store information about musicians who perform on its albums (as well as other company data) in a database. The company has wisely chosen to hire you as a database designer (at your usual consulting fee of $2,500/day).

- Each musician that records at Notown has an SSN, a name, an address, and a phone number. Poorly paid musicians often share the same address, and no address has more than one phone.
- Each instrument that is used in songs recorded at Notown has a name (e.g., guitar, synthesizer, flute) and a musical key (e.g., C, B-flat, E-flat).
- Each album that is recorded on the Notown label has a title, a copyright date, a format (e.g., CD or MC), and an album identifier.
- Each song recorded at Notown has a title and an author.
- Each musician may play several instruments, and a given instrument may be played by several musicians.
- Each album has a number of songs on it, but no song may appear on more than one album.
- Each song is performed by one or more musicians, and a musician may perform a number of songs.
- Each album has exactly one musician who acts as its producer. A musician may produce several albums, of course.

Design a conceptual schema for Notown and draw an ER diagram for your schema. The following information describes the situation that the Notown database must model. Be sure to indicate all key and cardinality constraints and any assumptions that you make. Identify any constraints that you are unable to capture in the ER diagram and briefly explain why you could not express them.

Answer 2.5 The ER diagram is shown in Figure 2.2.

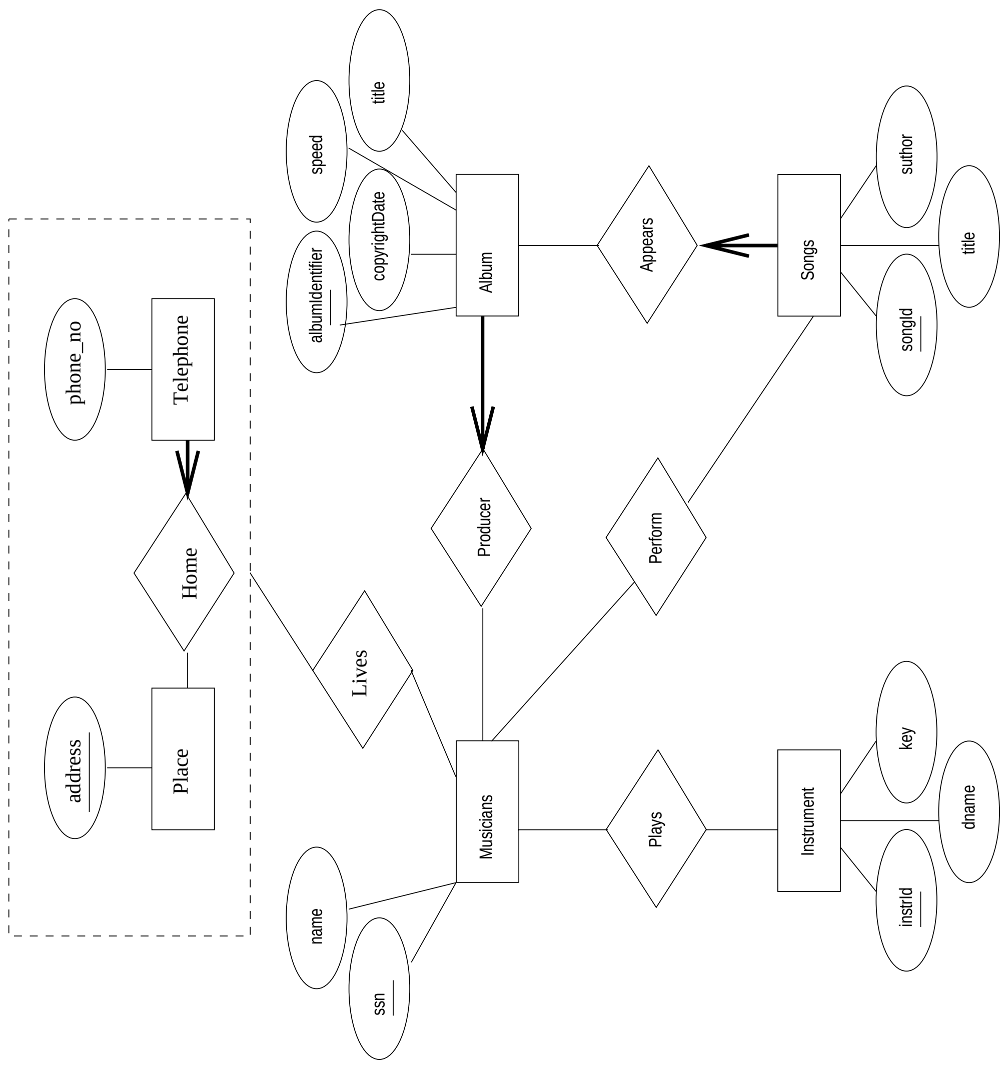


Figure 2.2 ER Diagram for Exercise 2.5

Exercise 2.6 Computer Sciences Department frequent fliers have been complaining to Dane County Airport officials about the poor organization at the airport. As a result, the officials have decided that all information related to the airport should be organized using a DBMS, and you've been hired to design the database. Your first task is to organize the information about all the airplanes that are stationed and maintained at the airport. The relevant information is as follows:

- Every airplane has a registration number, and each airplane is of a specific model.
- The airport accommodates a number of airplane models, and each model is identified by a model number (e.g., DC-10) and has a capacity and a weight.
- A number of technicians work at the airport. You need to store the name, SSN, address, phone number, and salary of each technician.
- Each technician is an expert on one or more plane model(s), and his or her expertise may overlap with that of other technicians. This information about technicians must also be recorded.
- Traffic controllers must have an annual medical examination. For each traffic controller, you must store the date of the most recent exam.
- All airport employees (including technicians) belong to a union. You must store the union membership number of each employee. You can assume that each employee is uniquely identified by the social security number.
- The airport has a number of tests that are used periodically to ensure that airplanes are still airworthy. Each test has a Federal Aviation Administration (FAA) test number, a name, and a maximum possible score.
- The FAA requires the airport to keep track of each time that a given airplane is tested by a given technician using a given test. For each testing event, the information needed is the date, the number of hours the technician spent doing the test, and the score that the airplane received on the test.

1. Draw an ER diagram for the airport database. Be sure to indicate the various attributes of each entity and relationship set; also specify the key and participation constraints for each relationship set. Specify any necessary overlap and covering constraints as well (in English).
2. The FAA passes a regulation that tests on a plane must be conducted by a technician who is an expert on that model. How would you express this constraint in the ER diagram? If you cannot express it, explain briefly.

Answer 2.6 Answer omitted.

Exercise 2.7 The Prescriptions-R-X chain of pharmacies has offered to give you a free lifetime supply of medicines if you design its database. Given the rising cost of health care, you agree. Here's the information that you gather:

- Patients are identified by an SSN, and their names, addresses, and ages must be recorded.
- Doctors are identified by an SSN. For each doctor, the name, specialty, and years of experience must be recorded.
- Each pharmaceutical company is identified by name and has a phone number.
- For each drug, the trade name and formula must be recorded. Each drug is sold by a given pharmaceutical company, and the trade name identifies a drug uniquely from among the products of that company. If a pharmaceutical company is deleted, you need not keep track of its products any longer.
- Each pharmacy has a name, address, and phone number.
- Every patient has a primary physician. Every doctor has at least one patient.
- Each pharmacy sells several drugs and has a price for each. A drug could be sold at several pharmacies, and the price could vary from one pharmacy to another.
- Doctors prescribe drugs for patients. A doctor could prescribe one or more drugs for several patients, and a patient could obtain prescriptions from several doctors. Each prescription has a date and a quantity associated with it. You can assume that if a doctor prescribes the same drug for the same patient more than once, only the last such prescription needs to be stored.
- Pharmaceutical companies have long-term contracts with pharmacies. A pharmaceutical company can contract with several pharmacies, and a pharmacy can contract with several pharmaceutical companies. For each contract, you have to store a start date, an end date, and the text of the contract.
- Pharmacies appoint a supervisor for each contract. There must always be a supervisor for each contract, but the contract supervisor can change over the lifetime of the contract.

1. Draw an ER diagram that captures the above information. Identify any constraints that are not captured by the ER diagram.
2. How would your design change if each drug must be sold at a fixed price by all pharmacies?
3. How would your design change if the design requirements change as follows: If a doctor prescribes the same drug for the same patient more than once, several such prescriptions may have to be stored.

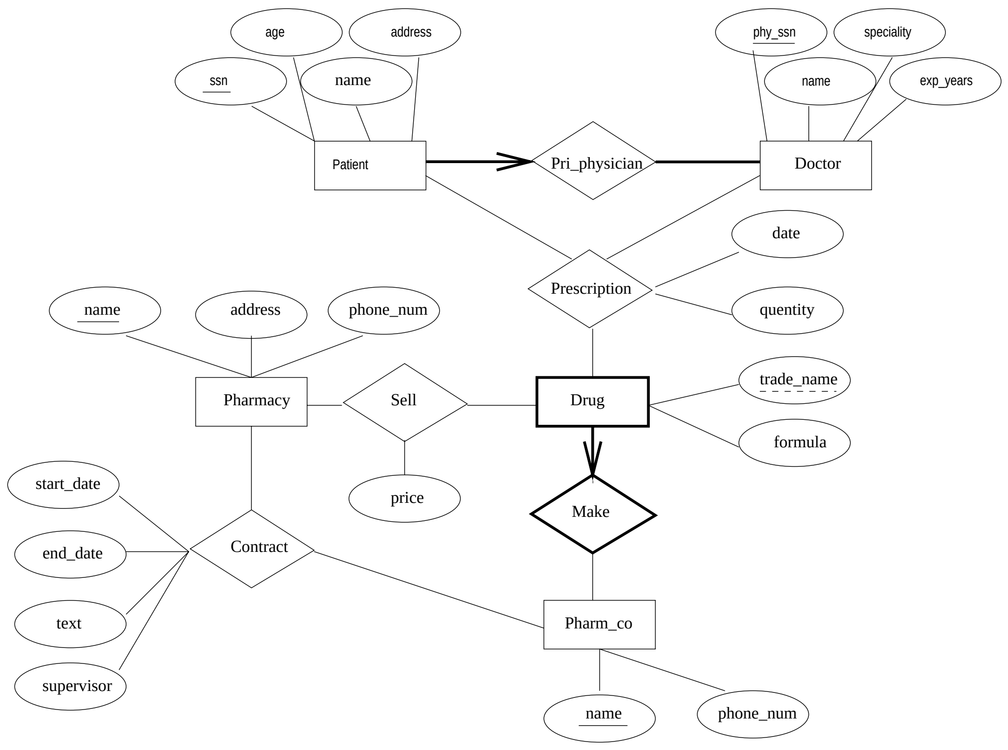


Figure 2.3 ER Diagram for Exercise 2.8

Answer 2.7 1. The ER diagram is shown in Figure 2.3.

2. If the drug is to be sold at a fixed price we can add the price attribute to the Drug entity set and eliminate the Sell relationship set.

3. The date information can no longer be modeled as an attribute of Prescription. We have to create a new entity set called Prescription_date and make Prescription a 4-way relationship set that involves this additional entity set.

Exercise 2.8 Although you always wanted to be an artist, you ended up being an expert on databases because you love to cook data and you somehow confused 'data base' with 'data baste.' Your old love is still there, however, so you set up a database company, ArtBase, that builds a product for art galleries. The core of this product is a database with a schema that captures all the information that galleries need to maintain. Galleries keep information about artists, their names (which are unique), birthplaces, age, and style of art. For each piece of artwork, the artist, the year it was made, its unique title, its type of art (e.g., painting, lithograph, sculpture, photograph), and its price must be stored. Pieces of artwork are also classified into groups of various kinds, for example, portraits, still lifes, works by Picasso, or works of the 19th century; a given piece may belong to more than one group. Each group is identified by a name (like those above) that describes the group. Finally, galleries keep information about customers. For each customer, galleries keep their unique name, address, total amount of dollars they have spent in the gallery (very important!), and the artists and groups of art that each customer tends to like.

Draw the ER diagram for the database.

Answer 2.8 Answer omitted.

3
THE RELATIONAL MODEL

Exercise 3.1 Define the following terms: *relation schema, relational database schema, 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?

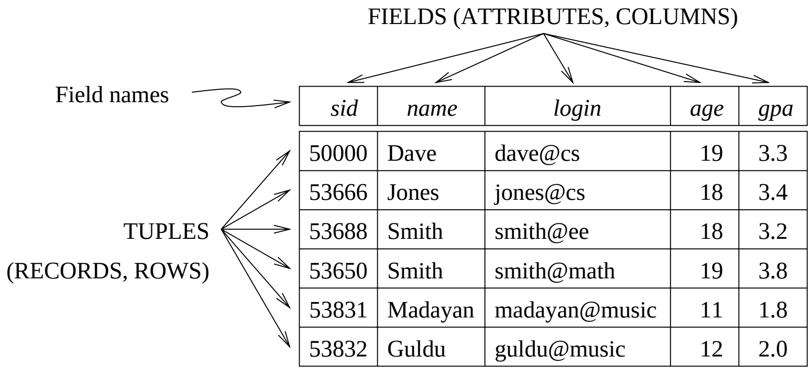
Answer 3.2 Answer omitted.

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 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 that were 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`, *pct_time:* `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 above 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% 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 $S1$

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

2. The answer tuple for Madayan is omitted.

Exercise 3.10 Explain why the addition of `NOT NULL` constraints to the SQL definition of the Manages relation (in Section 3.5.3) would 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 *a*1 and *a*2, with *a*1 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 statement creates Table A:

```
CREATE TABLE A (  a1     CHAR(10),
                  a2     CHAR(10),
                  PRIMARY KEY (a1) )
```

Tables B and C are similar to A.

```
CREATE TABLE R (  a1     CHAR(10),
                  b1     CHAR(10) NOT NULL ,
                  c1     CHAR(10) ,
                  PRIMARY KEY (a1),
                  UNIQUE (b1)
                  FOREIGN KEY (a1) REFERENCES A,
                  FOREIGN KEY (b1) REFERENCES B )
                  FOREIGN KEY (c1) REFERENCES C )
```

We cannot capture the total participation constraint of A in R. This is because we cannot ensure that every key a1 appears in R without the use of checks.

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 that 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 Answer omitted.

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 2.4. 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 (
```
   ssn             CHAR(10) NOT NULL,
   albumIdentifier INTEGER,
   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       INTGER,
                           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.

4 RELATIONAL ALGEBRA AND CALCULUS

Exercise 4.1 Explain the statement that relational algebra operators can be *composed*. Why is the ability to compose operators important?

Answer 4.1 Every operator in relational algebra accepts one or more relation instances as arguments and the result is always an relation instance. So the argument of one operator could be the result of another operator. This is important because, this makes it easy to write complex queries by simply composing the relational algebra operators.

Exercise 4.2 Given two relations $R1$ and $R2$, where $R1$ contains N1 tuples, $R2$ contains N2 tuples, and N2 > N1 > 0, give the minimum and maximum possible sizes (in tuples) for the result relation produced by each of the following relational algebra expressions. In each case, state any assumptions about the schemas for $R1$ and $R2$ that are needed to make the expression meaningful:

> (1) $R1 \cup R2$, (2) $R1 \cap R2$, (3) $R1 - R2$, (4) $R1 \times R2$, (5) $\sigma_{a=5}(R1)$, (6) $\pi_a(R1)$, and (7) $R1/R2$

Answer 4.2 Answer omitted.

Exercise 4.3 Consider the following schema:

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

The key fields are underlined, and the domain of each field is listed after the field name. Thus *sid* is the key for Suppliers, *pid* is the key for Parts, and *sid* and *pid* together form the key for Catalog. The Catalog relation lists the prices charged for parts by Suppliers. Write the following queries in relational algebra, tuple relational calculus, and domain relational calculus:

1. Find the *names* of suppliers who supply some red part.
2. Find the *sids* of suppliers who supply some red or green part.
3. Find the *sids* of suppliers who supply some red part or are at 221 Packer Ave.
4. Find the *sids* of suppliers who supply some red part and some green part.
5. Find the *sids* of suppliers who supply every part.
6. Find the *sids* of suppliers who supply every red part.
7. Find the *sids* of suppliers who supply every red or green part.
8. Find the *sids* of suppliers who supply every red part or supply every green part.
9. Find pairs of *sids* such that the supplier with the first *sid* charges more for some part than the supplier with the second *sid*.
10. Find the *pids* of parts that are supplied by at least two different suppliers.
11. Find the *pids* of the most expensive parts supplied by suppliers named Yosemite Sham.
12. Find the *pids* of parts supplied by every supplier at less than $200. (If any supplier either does not supply the part or charges more than $200 for it, the part is not selected.)

Answer 4.3 In the answers below RA refers to Relational Algebra, TRC refers to Tuple Relational Calculus and DRC refers to Domain Relational Calculus.

1. ■ RA

$$\pi_{sname}(\pi_{sid}((\pi_{pid}\sigma_{color='red'}Parts) \bowtie Catalog) \bowtie Suppliers)$$

■ TRC

$$\{T \mid \exists T1 \in Suppliers(\exists X \in Parts(X.color =' red' \wedge \exists Y \in Catalog$$
$$(Y.pid = X.pid \wedge Y.sid = T1.sid)) \wedge T.sname = T1.sname)\}$$

■ DRC

$$\{\langle Y\rangle \mid \langle X, Y, Z\rangle \in Suppliers \wedge \exists P, Q, R(\langle P, Q, R\rangle \in Parts$$
$$\wedge R =' red' \wedge \exists I, J, K(\langle I, J, K\rangle \in Catalog \wedge J = P \wedge I = X))\}$$

■ SQL

```
SELECT S.sname
FROM   Suppliers S, Parts P, Catalog C
WHERE  P.color='red' AND C.pid=P.pid AND C.sid=S.sid
```

2. ■ RA

$$\pi_{sid}(\pi_{pid}(\sigma_{color='red' \vee color='green'} Parts) \bowtie catalog)$$

■ TRC

$$\{T \mid \exists T1 \in Catalog(\exists X \in Parts((X.color = `red' \vee X.color = `green') \\ \wedge X.pid = T1.pid) \wedge T.sid = T1.sid)\}$$

■ DRC

$$\{\langle X \rangle \mid \langle X, Y, Z \rangle \in Catalog \wedge \exists A, B, C(\langle A, B, C \rangle \in Parts \\ \wedge (C =' red' \vee C =' green') \wedge A = Y)\}$$

■ SQL

```
SELECT C.sid
FROM   Catalog C, Parts P
WHERE  (P.color = 'red' OR P.color = 'green')
       AND P.pid = C.pid
```

3. ■ RA

$$\rho(R1, \pi_{sid}((\pi_{pid}\sigma_{color='red'} Parts) \bowtie Catalog))$$
$$\rho(R2, \pi_{sid}\sigma_{address='221PackerStreet'} Suppliers)$$
$$R1 \cup R2$$

■ TRC

$$\{T \mid \exists T1 \in Catalog(\exists X \in Parts(X.color = `red' \wedge X.pid = T1.pid) \\ \wedge T.sid = T1.sid) \\ \vee \exists T2 \in Suppliers(T2.address =' 221PackerStreet' \wedge T.sid = T2.sid)\}$$

■ DRC

$$\{\langle X \rangle \mid \langle X, Y, Z \rangle \in Catalog \wedge \exists A, B, C(\langle A, B, C \rangle \in Parts \\ \wedge C =' red' \wedge A = Y) \\ \vee \exists P, Q(\langle X, P, Q \rangle \in Suppliers \wedge Q =' 221PackerStreet')\}$$

■ SQL

```
SELECT S.sid
FROM   Suppliers S
WHERE  S.address = ‘221 Packer street’
       OR S.sid IN ( SELECT C.sid
                     FROM   Parts P, Catalog C
                     WHERE  P.color=’red’ AND P.pid = C.pid )
```

4. ■ RA

$$\rho(R1, \pi_{sid}((\pi_{pid}\sigma_{color='red'}Parts) \bowtie Catalog))$$
$$\rho(R2, \pi_{sid}((\pi_{pid}\sigma_{color='green'}Parts) \bowtie Catalog))$$
$$R1 \cap R2$$

■ TRC

$$\{T \mid \exists T1 \in Catalog(\exists X \in Parts(X.color = `red' \wedge X.pid = T1.pid)$$
$$\wedge \exists T2 \in Catalog(\exists Y \in Parts(Y.color =' green' \wedge Y.pid = T2.pid)$$
$$\wedge T2.sid = T1.sid) \wedge T.sid = T1.sid)\}$$

■ DRC

$$\{\langle X\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge \exists A, B, C(\langle A, B, C\rangle \in Parts$$
$$\wedge C =' red' \wedge A = Y)$$
$$\wedge \exists P, Q, R(\langle P, Q, R\rangle \in Catalog \wedge \exists E, F, G(\langle E, F, G\rangle \in Parts$$
$$\wedge G =' green' \wedge E = Q) \wedge P = X)\}$$

■ SQL

```
SELECT C.sid
FROM   Parts P, Catalog C
WHERE  P.color = ‘red’ AND P.pid = C.pid
       AND EXISTS ( SELECT P2.pid
                    FROM   Parts P2, Catalog C2
                    WHERE  P2.color = ‘green’ AND C2.sid = C.sid
                           AND P2.pid = C2.pid )
```

5. ■ RA

$$(\pi_{sid,pid}Catalog)/(\pi_{pid}Parts)$$

■ TRC

$$\{T \mid \exists T1 \in Catalog(\forall X \in Parts(\exists T2 \in Catalog$$
$$(T2.pid = X.pid \wedge T2.sid = T1.sid)) \wedge T.sid = T1.sid)\}$$

- DRC

$$\{\langle X\rangle \mid \langle X,Y,Z\rangle \in Catalog \land \forall\langle A,B,C\rangle \in Parts$$
$$(\exists\langle P,Q,R\rangle \in Catalog(Q = A \land P = X))\}$$

- SQL

```
SELECT C.sid
FROM   Catalog C
WHERE  NOT EXISTS (SELECT P.pid
                   FROM   Parts P
                   WHERE  NOT EXISTS (SELECT C1.sid
                                      FROM   Catalog C1
                                      WHERE  C1.sid = C.sid
                                             AND C1.pid = P.pid))
```

6. - RA

$$(\pi_{sid,pid}Catalog)/(\pi_{pid}\sigma_{color='red'}Parts)$$

- TRC

$$\{T \mid \exists T1 \in Catalog(\forall X \in Parts(X.color \neq \text{'}red\text{'}$$
$$\lor \exists T2 \in Catalog(T2.pid = X.pid \land T2.sid = T1.sid))$$
$$\land T.sid = T1.sid)\}$$

- DRC

$$\{\langle X\rangle \mid \langle X,Y,Z\rangle \in Catalog \land \forall\langle A,B,C\rangle \in Parts$$
$$(C \neq \text{'}red\text{'} \lor \exists\langle P,Q,R\rangle \in Catalog(Q = A \land P = X))\}$$

- SQL

```
SELECT C.sid
FROM   Catalog C
WHERE  NOT EXISTS (SELECT P.pid
                   FROM   Parts P
                   WHERE  P.color = 'red'
                   AND (NOT EXISTS (SELECT C1.sid
                                    FROM   Catalog C1
                                    WHERE  C1.sid = C.sid AND
                                           C1.pid = P.pid)))
```

7. - RA

$$(\pi_{sid,pid}Catalog)/(\pi_{pid}\sigma_{color='red' \lor color='green'}Parts)$$

- TRC

$$\{T \mid \exists T1 \in Catalog(\forall X \in Parts((X.color \neq \text{‘}red' \\ \wedge X.color \neq \text{‘}green') \vee \exists T2 \in Catalog \\ (T2.pid = X.pid \wedge T2.sid = T1.sid)) \wedge T.sid = T1.sid)\}$$

- DRC

$$\{\langle X\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge \forall \langle A, B, C\rangle \in Parts \\ ((C \neq \text{‘}red' \wedge C \neq \text{‘}green') \vee \exists \langle P, Q, R\rangle \in Catalog \\ (Q = A \wedge P = X))\}$$

- SQL

```
SELECT C.sid
FROM   Catalog C
WHERE  NOT EXISTS (SELECT P.pid
                   FROM   Parts P
                   WHERE  (P.color = ‘red’ OR P.color = ‘green’)
                   AND (NOT EXISTS (SELECT C1.sid
                                    FROM   Catalog C1
                                    WHERE  C1.sid = C.sid AND
                                           C1.pid = P.pid)))
```

8. - RA

$$\rho(R1, ((\pi_{sid,pid}Catalog)/(\pi_{pid}\sigma_{color='red'}Parts))) \\ \rho(R2, ((\pi_{sid,pid}Catalog)/(\pi_{pid}\sigma_{color='green'}Parts))) \\ R1 \cup R2$$

- TRC

$$\{T \mid \exists T1 \in Catalog((\forall X \in Parts \\ (X.color \neq \text{‘}red' \vee \exists Y \in Catalog(Y.pid = X.pid \wedge Y.sid = T1.sid)) \\ \vee \forall Z \in Parts(Z.color \neq \text{‘}green' \vee \exists P \in Catalog \\ (P.pid = Z.pid \wedge P.sid = T1.sid))) \wedge T.sid = T1.sid)\}$$

- DRC

$$\{\langle X\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge (\forall \langle A, B, C\rangle \in Parts \\ (C \neq \text{‘}red' \vee \exists \langle P, Q, R\rangle \in Catalog(Q = A \wedge P = X)) \\ \vee \forall \langle U, V, W\rangle \in Parts(W \neq \text{‘}green' \vee \langle M, N, L\rangle \in Catalog \\ (N = U \wedge M = X)))\}$$

- SQL

```
SELECT C.sid
FROM   Catalog C
WHERE  (NOT EXISTS (SELECT P.pid
                    FROM   Parts P
                    WHERE  P.color = 'red' AND
                    (NOT EXISTS (SELECT C1.sid
                                 FROM   Catalog C1
                                 WHERE  C1.sid = C.sid AND
                                        C1.pid = P.pid))))
       OR ( NOT EXISTS (SELECT P1.pid
                        FROM   Parts P1
                        WHERE  P1.color = 'green' AND
                        (NOT EXISTS (SELECT C2.sid
                                     FROM   Catalog C2
                                     WHERE  C2.sid = C.sid AND
                                            C2.pid = P1.pid))))
```

9. - RA

$$\rho(R1, Catalog)$$
$$\rho(R2, Catalog)$$
$$\pi_{R1.sid,R2.sid}(\sigma_{R1.pid=R2.pid \wedge R1.sid \neq R2.sid \wedge R1.cost > R2.cost}(R1 \times R2))$$

- TRC

$$\begin{aligned}\{T \quad | \quad & \exists T1 \in Catalog(\exists T2 \in Catalog \\ & (T2.pid = T1.pid \wedge T2.sid \neq T1.sid \\ & \wedge T2.cost < T1.cost \wedge T.sid2 = T2.sid) \\ & \wedge T.sid1 = T1.sid)\}\end{aligned}$$

- DRC

$$\begin{aligned}&\{\langle X, P\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge \exists P, Q, R \\ &(\langle P, Q, R\rangle \in Catalog \wedge Q = Y \wedge P \neq X \wedge R < Z)\}\end{aligned}$$

- SQL

```
SELECT C1.sid, C2.sid
FROM   Catalog C1, Catalog C2
WHERE  C1.pid = C2.pid AND C1.sid ≠ C2.sid
       AND C1.cost > C2.cost
```

10. ■ RA

$$\rho(R1, Catalog)$$
$$\rho(R2, Catalog)$$
$$\pi_{R1.pid}\sigma_{R1.pid=R2.pid \wedge R1.sid \neq R2.sid}(R1 \times R2)$$

■ TRC

$$\{T \mid \exists T1 \in Catalog(\exists T2 \in Catalog$$
$$(T2.pid = T1.pid \wedge T2.sid \neq T1.sid)$$
$$\wedge T.pid = T1.pid)\}$$

■ DRC

$$\{\langle X \rangle \mid \langle X, Y, Z \rangle \in Catalog \wedge \exists A, B, C$$
$$(\langle A, B, C \rangle \in Catalog \wedge B = Y \wedge A \neq X)\}$$

■ SQL

```
SELECT C.sid
FROM   Catalog C
WHERE  EXISTS (SELECT C1.sid
               FROM   Catalog C1
               WHERE  C1.pid = C.pid AND C1.sid ≠ C.sid )
```

11. ■ RA

$$\rho(R1, \pi_{sid}\sigma_{sname='YosemiteSham'}Suppliers)$$
$$\rho(R2, R1 \bowtie Catalog)$$
$$\rho(R3, R2)$$
$$\rho(R4(1 \rightarrow sid, 2 \rightarrow pid, 3 \rightarrow cost), \sigma_{R3.cost<R2.cost}(R3 \times R2))$$
$$\pi_{pid}(R2 - \pi_{sid,pid,cost}R4)$$

■ TRC

$$\{T \mid \exists T1 \in Catalog(\exists X \in Suppliers$$
$$(X.sname =' YosemiteSham' \wedge X.sid = T1.sid) \wedge \neg(\exists S \in Suppliers$$
$$(S.sname =' YosemiteSham' \wedge \exists Z \in Catalog$$
$$(Z.sid = S.sid \wedge Z.cost > T1.cost))) \wedge T.pid = T1.pid)$$

■ DRC

$$\{\langle Y \rangle \mid \langle X, Y, Z \rangle \in Catalog \wedge \exists A, B, C$$
$$(\langle A, B, C \rangle \in Suppliers \wedge C =' YosemiteSham' \wedge A = X)$$
$$\wedge \neg(\exists P, Q, R(\langle P, Q, R \rangle \in Suppliers \wedge R =' YosemiteSham'$$
$$\wedge \exists I, J, K(\langle I, J, K \rangle \in Catalog(I = P \wedge K > Z))))\}$$

- SQL

```
SELECT C.pid
FROM   Catalog C, Suppliers S
WHERE  S.sname = 'Yosemite Sham' AND C.sid = S.sid
       AND C.cost ≥ ALL (Select C2.cost
                         FROM  Catalog C2, Suppliers S2
                         WHERE S2.sname = 'Yosemite Sham'
                               AND C2.sid = S2.sid)
```

Exercise 4.4 Consider the Supplier-Parts-Catalog schema from the previous question. State what the following queries compute:

1. $\pi_{sname}(\pi_{sid}(\sigma_{color='red'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers)$
2. $\pi_{sname}(\pi_{sid}((\sigma_{color='red'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers))$
3. $(\pi_{sname}((\sigma_{color='red'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers)) \cap$

   $(\pi_{sname}((\sigma_{color='green'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers))$
4. $(\pi_{sid}((\sigma_{color='red'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers)) \cap$

   $(\pi_{sid}((\sigma_{color='green'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers))$
5. $\pi_{sname}((\pi_{sid,sname}((\sigma_{color='red'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers)) \cap$

   $(\pi_{sid,sname}((\sigma_{color='green'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers)))$

Answer 4.4 Answer not available yet.

Exercise 4.5 Consider the following relations containing airline flight information:

Flights(<u>*flno:* `integer`</u>, *from:* `string`, *to:* `string`,
distance: `integer`, *departs:* `time`, *arrives:* `time`)
Aircraft(<u>*aid:* `integer`</u>, *aname:* `string`, *cruisingrange:* `integer`)
Certified(<u>*eid:* `integer`, *aid:* `integer`</u>)
Employees(<u>*eid:* `integer`</u>, *ename:* `string`, *salary:* `integer`)

Note that the Employees relation describes pilots and other kinds of employees as well; every pilot is certified for some aircraft (otherwise, he or she would not qualify as a pilot), and only pilots are certified to fly.

Write the following queries in relational algebra, tuple relational calculus, and domain relational calculus. Note that some of these queries may not be expressible in relational algebra (and, therefore, also not expressible in tuple and domain relational calculus)! For such queries, informally explain why they cannot be expressed. (See the exercises at the end of Chapter 5 for additional queries over the airline schema.)

1. Find the *eid*s of pilots certified for some Boeing aircraft.
2. Find the *name*s of pilots certified for some Boeing aircraft.
3. Find the *aid*s of all aircraft that can be used on non-stop flights from Bonn to Madras.
4. Identify the flights that can be piloted by every pilot whose salary is more than $100,000.
5. Find the names of pilots who can operate planes with a range greater than 3,000 miles but are not certified on any Boeing aircraft.
6. Find the *eid*s of employees who make the highest salary.
7. Find the *eid*s of employees who make the second highest salary.
8. Find the *eid*s of employees who are certified for the largest number of aircraft.
9. Find the *eid*s of employees who are certified for exactly three aircraft.
10. Find the total amount paid to employees as salaries.
11. Is there a sequence of flights from Madison to Timbuktu? Each flight in the sequence is required to depart from the city that is the destination of the previous flight; the first flight must leave Madison, the last flight must reach Timbuktu, and there is no restriction on the number of intermediate flights. Your query must determine whether a sequence of flights from Madison to Timbuktu exists for *any* input Flights relation instance.

Answer 4.5 In the answers below RA refers to Relational Algebra, TRC refers to Tuple Relational Calculus and DRC refers to Domain Relational Calculus.

1. ■ RA

$$\pi_{eid}(\sigma_{aname=\text{`}Boeing'}(Aircraft \bowtie Certified))$$

■ TRC

$$\begin{aligned}\{C.eid \quad | \quad & C \in Certified \wedge \\ & \exists A \in Aircraft(A.aid = C.aid \wedge A.aname = \text{`}Boeing')\}\end{aligned}$$

■ DRC

$$\begin{aligned}\{\langle Ceid\rangle \quad | \quad & \langle Ceid, Caid\rangle \in Certified \wedge \\ & \exists Aid, AN, AR(\langle Aid, AN, AR\rangle \in Aircraft \\ & \wedge Aid = Caid \wedge AN = \text{`}Boeing')\}\end{aligned}$$

- SQL

```
SELECT C.eid
FROM   Aircraft A, Certified C
WHERE  A.aid = C.aid AND A.aname = 'Boeing'
```

2. - RA

$$\pi_{ename}(\sigma_{aname='Boeing'}(Aircraft \bowtie Certified \bowtie Employees))$$

- TRC

$\{E.ename \mid E \in Employees \wedge \exists C \in Certified$
$(\exists A \in Aircraft(A.aid = C.aid \wedge A.aname = \text{`}Boeing' \wedge E.eid = C.eid))\}$

- DRC

$\{\langle EN\rangle \mid \langle Eid, EN, ES\rangle \in Employess\wedge$
$\exists Ceid, Caid(\langle Ceid, Caid\rangle \in Certified\wedge$
$\exists Aid, AN, AR(\langle Aid, AN, AR\rangle \in Aircraft\wedge$
$Aid = Caid \wedge AN = \text{`}Boeing' \wedge Eid = Ceid)\}$

- SQL

```
SELECT E.ename
FROM   Aircraft A, Certified C, Employees E
WHERE  A.aid = C.aid AND A.aname = 'Boeing' AND E.eid = C.eid
```

3. - RA

$\rho(LAtoNY, \sigma_{from=\text{`}L.A.'\wedge to=\text{`}N.Y.'}(Flights))$
$\pi_{aid}(\sigma_{cruisingrange>distance}(Aircraft \times LAtoNY))$

- TRC

$\{A.aid \mid A \in Aircraft \wedge \exists F \in Flights$
$(F.from = \text{`}L.A.' \wedge F.to = \text{`}N.Y.' \wedge A.cruisingrange > F.distance)\}$

- DRC

$\{Aid \mid \langle Aid, AN, AR\rangle \in Aircraft\wedge$
$(\exists FN, FF, FT, FDi, FDe, FA(\langle FN, FF, FT, FDi, FDe, FA\rangle \in Flights\wedge$
$FF = \text{`}L.A.' \wedge FT = \text{`}N.Y.' \wedge FDi < AR))\}$

- SQL

```
SELECT A.aid
FROM   Aircraft A, Flights F
WHERE  F.from = ‘L.A.’ AND F.to = ‘N.Y.’ AND
       A.cruisingrange > F.distance
```

4. ■ RA

   $\pi_{flno}(\sigma_{distance<cruisingrange \wedge salary>100{,}000}(Flights \bowtie Aircraft \bowtie Certified \bowtie Employees)))$

   ■ TRC $\{F.flno \mid F \in Flights \wedge \exists A \in Aircraft \exists C \in Certified$
   $\exists E \in Employees(A.cruisingrange > F.distance \wedge E.salary > 100,000 \wedge$
   $A.aid = C.aid \wedge E.eid = C.eid)\}$

   ■ DRC

   $\{FN \mid \langle FN, FF, FT, FDi, FDe, FA \rangle \in Flights \wedge$
   $\exists Ceid, Caid(\langle Ceid, Caid \rangle \in Certified \wedge$
   $\exists Aid, AN, AR(\langle Aid, AN, AR \rangle \in Aircraft \wedge$
   $\exists Eid, EN, ES(\langle Eid, EN, ES \rangle \in Employees$
   $(AR > FDi \wedge ES > 100,000 \wedge Aid = Caid \wedge Eid = Ceid)\}$

   ■ SQL

```
SELECT E.ename
FROM   Aircraft A, Certified C, Employees E, Flights F
WHERE  A.aid = C.aid AND E.eid = C.eid AND
       distance < cruisingrange AND salary > 100,000
```

5. ■ RA $\rho(R1, \pi_{eid}(\sigma_{cruisingrange>3000}(Aircraft \bowtie Certified)))$
   $\pi_{ename}(Employees \bowtie (R1 - \pi_{eid}(\sigma_{aname=\text{‘}Boeing'}(Aircraft \bowtie Certified))))$

   ■ TRC

   $\{E.ename \mid E \in Employees \wedge \exists C \in Certified(\exists A \in Aircraft$
   $(A.aid = C.aid \wedge E.eid = C.eid \wedge A.cruisingrange > 3000)) \wedge$
   $\neg(\exists C2 \in Certified(\exists A2 \in Aircraft(A2.aname = \text{‘}Boeing' \wedge C2.aid =$
   $A2.aid \wedge C2.eid = E.eid)))\}$

   ■ DRC

   $\{\langle EN \rangle \mid \langle Eid, EN, ES \rangle \in Employess \wedge$
   $\exists Ceid, Caid(\langle Ceid, Caid \rangle \in Certified \wedge$
   $\exists Aid, AN, AR(\langle Aid, AN, AR \rangle \in Aircraft \wedge$
   $Aid = Caid \wedge Eid = Ceid \wedge AR > 3000)) \wedge$

$\neg(\exists Aid2, AN2, AR2(\langle Aid2, AN2, AR2\rangle \in Aircraft \wedge$
$\exists Ceid2, Caid2(\langle Ceid2, Caid2\rangle \in Certified$
$\wedge Aid2 = Caid2 \wedge Eid = Ceid2 \wedge AN2 = \text{`}Boeing'))))\}$

- SQL

```
SELECT E.ename
FROM   Certified C, Employees E, Aircraft A
WHERE  A.aid = C.aid AND E.eid = C.eid AND A.cruisingrange > 3000
AND E.eid NOT IN ( SELECT C2.eid
FROM Certified C2, Aircraft A2
WHERE C2.aid = A2.aid AND A2.aname = 'Boeing' )
```

6. - RA

   The approach to take is first find all the employees who do not have the highest salary. Subtract these from the original list of employees and what is left is the highest paid employees.

   $\rho(E1, Employees)$
   $\rho(E2, Employees)$
   $\rho(E3, \pi_{E2.eid}(E1 \bowtie_{E1.salary>E2.salary} E2)$
   $(\pi_{eid}E1) - E3$

   - TRC

   $\{E1.eid \mid E1 \in Employees \wedge \neg(\exists E2 \in Employees(E2.salary > E1.salary))\}$

   - DRC

   $\{\langle Eid1\rangle \mid \langle Eid1, EN1, ES1\rangle \in Employess \wedge$
   $\neg(\exists Eid2, EN2, ES2(\langle Eid2, EN2, ES2\rangle \in Employess \wedge ES2 > ES1))\}$

   - SQL

```
SELECT E.eid
FROM   Employees E
WHERE  E.salary = ( Select MAX (E2.salary)
                    FROM Employees E2 )
```

7. - RA

   The approach taken is similar to the solution for the previous exercise. First find all the employees who do not have the highest salary. Remove these from the original list of employees and what is left is the highest paid employees. Remove the highest paid employees from the original list. What is left is the

second highest paid employees together with the rest of the employees. Then find the highest paid employees of this new list. This is the list of the second highest paid employees. $\rho(E1, Employees)$
$\rho(E2, Employees)$
$\rho(E3, \pi_{E2.eid}(E1 \bowtie_{E1.salary>E2.salary} E2)$
$\rho(E4, E2 \bowtie E3)$
$\rho(E5, E2 \bowtie E3)$
$\rho(E6, \pi_{E5.eid}(E4 \bowtie_{E1.salary>E5.salary} E5)$
$(\pi_{eid}E3) - E6$

- TRC
  $\{E1.eid \mid E1 \in Employees \wedge \exists E2 \in Employees(E2.salary > E1.salary$
  $\wedge\neg(\exists E3 \in Employees(E3.salary > E2.salary)))\}$

- DRC
  $\{\langle Eid1\rangle \mid \langle Eid1, EN1, ES1\rangle \in Employess\wedge$
  $\exists Eid2, EN2, ES2(\langle Eid2, EN2, ES2\rangle \in Employess(ES2 > ES1)$
  $\wedge\neg(\exists Eid3, EN3, ES3(\langle Eid3, EN3, ES3\rangle \in Employess(ES3 > ES2))))\}$

- SQL

```
SELECT E.eid
FROM   Employees E
WHERE  E.salary = (SELECT MAX (E2.salary)
                   FROM   Employees E2
                   WHERE  E2.salary ≠ (SELECT MAX (E3.salary)
                                       FROM   Employees E3 ))
```

8. This cannot be expressed in relational algebra (or calculus) because there is no operator to count, and this query requires the ability to count upto a number that depends on the data. The query can however be expressed in SQL as follows:

```
SELECT Temp.eid
FROM   ( SELECT    C.eid AS eid, COUNT (C.aid) AS cnt,
         FROM      Certified C
         GROUP BY  C.eid) AS Temp
WHERE  Temp.cnt = ( SELECT   MAX (Temp.cnt)
                    FROM     Temp)
```

9. - RA
     The approach behind this query is to first find the employees who are certified for at least three aircraft (they appear at least three times in the Certified

relation). Then find the employees who are certified for at least four aircraft. Subtract the second from the first and what is left is the employees who are certified for exactly three aircraft.

$\rho(R1, Certified)$
$\rho(R2, Certified)$
$\rho(R3, Certified)$
$\rho(R4, Certified)$
$\rho(R5, \pi_{eid}(\sigma_{(R1.eid=R2.eid=R3.eid) \wedge (R1.aid \neq R2.aid \neq R3.aid)}(R1 \times R2 \times R3)))$
$\rho(R6, \pi_{eid}(\sigma_{(R1.eid=R2.eid=R3.eid=R4.eid) \wedge (R1.aid \neq R2.aid \neq R3.aid \neq R4.aid)}$
$(R1 \times R2 \times R3 \times R4)))$
$R5 - R6$

- TRC

  $\{C1.eid \mid C1 \in Certified \wedge \exists C2 \in Certified(\exists C3 \in Certified$
  $(C1.eid = C2.eid \wedge C2.eid = C3.eid \wedge$
  $C1.aid \neq C2.aid \wedge C2.aid \neq C3.aid \wedge C3.aid \neq C1.aid \wedge$
  $\neg(\exists C4 \in Certified$
  $(C3.eid = C4.eid \wedge C1.aid \neq C4.aid \wedge$
  $C2.aid \neq C4.aid \wedge C3.aid \neq C4.aid))))\}$

- DRC

  $\{\langle CE1 \rangle \mid \langle CE1, CA1 \rangle \in Certified \wedge$
  $\exists CE2, CA2(\langle CE2, CA2 \rangle \in Certified \wedge$
  $\exists CE3, CA3(\langle CE3, CA3 \rangle \in Certified \wedge$
  $(CE1 = CE2 \wedge CE2 = CE3 \wedge$
  $CA1 \neq CA2 \wedge CA2 \neq CA3 \wedge CA3 \neq CA1 \wedge$
  $\neg(\exists CE4, CA4(\langle CE4, CA4 \rangle \in Certified \wedge$
  $(CE3 = CE4 \wedge CA1 \neq CA4 \wedge$
  $CA2 \neq CA4 \wedge CA3 \neq CA4))))\}$

- SQL

```
SELECT C1.eid
FROM   Certified C1, Certified C2, Certified C3
WHERE  (C1.eid = C2.eid AND C2.eid = C3.eid AND
        C1.aid ≠ C2.aid AND C2.aid ≠ C3.aid AND C3.aid ≠ C1.aid)
EXCEPT
SELECT C4.eid
FROM   Certified C4, Certified C5, Certified C6, Certified C7,
WHERE  (C4.eid = C5.eid AND C5.eid = C6.eid AND C6.eid = C7.eid AND
        C4.aid ≠ C5.aid AND C4.aid ≠ C6.aid AND C4.aid ≠ C7.aid AND
        C5.aid ≠ C6.aid AND C5.aid ≠ C7.aid AND C6.aid ≠ C7.aid )
```

This could also be done in SQL using COUNT.

10. This cannot be expressed in relational algebra (or calculus) because there is no operator to sum values. The query can however be expressed in SQL as follows:

    ```
    SELECT SUM (E.salaries)
    FROM   Employees E
    ```

11. This cannot be expressed in relational algebra or relational calculus or SQL. The problem is that there is no restriction on the number of intermediate flights. All of the query methods could find if there was a flight directly from Madison to Timbuktu and if there was a sequence of two flights that started in Madison and ended in Timbuktu. They could even find a sequence of n flights that started in Madison and ended in Timbuktu as long as there is a static (i.e., data-independent) upper bound on the number of intermediate flights. (For large n, this would of course be long and impractical, but at least possible.) In this query, however, the upper bound is not static but dynamic (based upon the set of tuples in the Flights relation).

    In summary, if we had a static upper bound (say k), we could write an algebra or SQL query that repeatedly computes (upto k) joins on the Flights relation. If the upper bound is dynamic, then we cannot write such a query because k is not known when writing the query.

12. This cannot be expressed in relational algebra (or calculus). If we had the constraint that every employee has a unique salary, then the query could be expressed with much difficulty in relational algebra and calculus. To do this, the 20 highest paid employees would removed from the list one by one and saved as in questions 6 and 7. Then the saved tuples could be added together with union. In practice, this is tedious. In SQL however, ORDER BY could be used to list the employees in sorted order by salary, and the user could see the top 20 rows. (Of course, this could be done in SQL even without the constraint mentioned above.)

Exercise 4.6 What is *relational completeness*? If a query language is relationally complete, can you write any desired query in that language?

Answer 4.6 Answer omitted.

Exercise 4.7 What is an *unsafe* query? Give an example and explain why it is important to disallow such queries.

Answer 4.7 An *unsafe* query is a query in relational calculus that has an infinite number of results. An example of such a query is:

$$\{S \mid \neg(S \in Sailors)\}$$

The query is for all things that are not sailors which of course is everything else. Clearly there is an infinite number of answers, and this query is *unsafe*. It is important to disallow *unsafe* queries because we want to be able to get back to users with a list of all the answers to a query after a finite amount of time.

5

SQL: QUERIES, PROGRAMMING, TRIGGERS

Exercise 5.1 Consider the following relations:

> Student(*snum:* `integer`, *sname:* `string`, *major:* `string`, *level:* `string`, *age:* `integer`)
> Class(*name:* `string`, *meets_at:* `time`, *room:* `string`, *fid:* `integer`)
> Enrolled(*snum:* `integer`, *cname:* `string`)
> Faculty(*fid*: `integer`, *fname:* `string`, *deptid:* `integer`)

The meaning of these relations is straightforward; for example, Enrolled has one record per student-class pair such that the student is enrolled in the class.

Write the following queries in SQL. No duplicates should be printed in any of the answers.

1. Find the names of all Juniors (Level = JR) who are enrolled in a class taught by I. Teach.
2. Find the age of the oldest student who is either a History major or is enrolled in a course taught by I. Teach.
3. Find the names of all classes that either meet in room R128 or have five or more students enrolled.
4. Find the names of all students who are enrolled in two classes that meet at the same time.
5. Find the names of faculty members who teach in every room in which some class is taught.
6. Find the names of faculty members for whom the combined enrollment of the courses that they teach is less than five.
7. Print the Level and the average age of students for that Level, for each Level.

8. Print the Level and the average age of students for that Level, for all Levels except JR.

9. Find the names of students who are enrolled in the maximum number of classes.

10. Find the names of students who are not enrolled in any class.

11. For each age value that appears in Students, find the level value that appears most often. For example, if there are more FR level students aged 18 than SR, JR, or SO students aged 18, you should print the pair (18, FR).

Answer 5.1 The answers are given below:

1.
```
SELECT DISTINCT S.Sname
FROM   Student S, Class C, Enrolled E, Faculty F
WHERE  S.snum = E.snum AND E.cname = C.name AND C.fid = F.fid AND
       F.fname = 'I.Teach' AND S.level = 'JR'
```

2.
```
SELECT MAX(S.age)
FROM   Student S
WHERE  (S.major = 'History')
       OR S.num IN (SELECT E.snum
                    FROM   Class C, Enrolled E, Faculty F
                    WHERE  E.cname = C.name AND C.fid = F.fid
                           AND F.fname = 'I.Teach' )
```

3.
```
SELECT   C.name
FROM     Class C
WHERE    C.room = 'R128'
         OR C.name IN (SELECT   E.cname
                       FROM     Enrolled E
                       GROUP BY E.cname
                       HAVING   COUNT (*) >= 5)
```

4.
```
SELECT DISTINCT S.sname
FROM   Student S
WHERE  S.snum IN (SELECT E1.snum
                  FROM   Enrolled E1, Enrolled E2, Class C1, Class C2
                  WHERE  E1.snum = E2.snum AND E1.cname <> E2.cname
                  AND E1.cname = C1.name
                  AND E2.cname = C2.name AND C1.time = C2.time)
```

5.
```
SELECT DISTINCT F.fname
FROM   Faculty F
WHERE  NOT EXISTS (( SELECT *
                    FROM    Class C )
                    EXCEPT
                    (SELECT C1.room
                    FROM    Class C1
                    WHERE   C1.fid = F.fid ))
```

6.
```
SELECT   DISTINCT F.fname
FROM     Faculty F
WHERE    5 > (SELECT E.snum
              FROM   Class C, Enrolled E
              WHERE  C.name = E.cname
              AND    C.fid = F.fid)
```

7.
```
SELECT   S.level, AVG(S.age)
FROM     Student S
GROUP BY S.level
```

8.
```
SELECT   S.level, AVG(S.age)
FROM     Student S
WHERE    S.level <> 'JR'
GROUP BY S.level
```

9.
```
SELECT   DISTINCT S.sname
FROM     Student S
WHERE    S.snum IN (SELECT   E.snum
                    FROM     Enrolled E
                    GROUP BY E.snum
                    HAVING   COUNT (*) >= ALL (SELECT   COUNT (*)
                                               FROM     Enrolled E2
                                               GROUP BY E2.snum ))
```

10.
```
SELECT DISTINCT S.sname
FROM   Student S
WHERE  S.snum NOT IN (SELECT E.snum
                      FROM   Enrolled E )
```

Exercise 5.2 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. Write the following queries in SQL:

1. Find the *pname*s of parts for which there is some supplier.
2. Find the *sname*s of suppliers who supply every part.
3. Find the *sname*s of suppliers who supply every red part.
4. Find the *pname*s of parts supplied by Acme Widget Suppliers and by no one else.
5. Find the *sid*s of suppliers who charge more for some part than the average cost of that part (averaged over all the suppliers who supply that part).
6. For each part, find the *sname* of the supplier who charges the most for that part.
7. Find the *sid*s of suppliers who supply only red parts.
8. Find the *sid*s of suppliers who supply a red part and a green part.
9. Find the *sid*s of suppliers who supply a red part or a green part.

Answer 5.2 Answer omitted.

Exercise 5.3 The following relations keep track of airline flight information:

Flights(<u>*flno:* `integer`</u>, *from:* `string`, *to:* `string`, *distance:* `integer`,
departs: `time`, *arrives:* `time`, *price:* `integer`)
Aircraft(<u>*aid:* `integer`</u>, *aname:* `string`, *cruisingrange:* `integer`)
Certified(<u>*eid:* `integer`, *aid:* `integer`</u>)
Employees(<u>*eid:* `integer`</u>, *ename:* `string`, *salary:* `integer`)

Note that the Employees relation describes pilots and other kinds of employees as well; every pilot is certified for some aircraft, and only pilots are certified to fly. Write each of the following queries in SQL. (*Additional queries using the same schema are listed in the exercises for Chapter 4.*)

1. Find the names of aircraft such that all pilots certified to operate them earn more than 80,000.
2. For each pilot who is certified for more than three aircraft, find the *eid* and the maximum *cruisingrange* of the aircraft that he (or she) is certified for.
3. Find the names of pilots whose *salary* is less than the price of the cheapest route from Los Angeles to Honolulu.
4. For all aircraft with *cruisingrange* over 1,000 miles, find the name of the aircraft and the average salary of all pilots certified for this aircraft.
5. Find the names of pilots certified for some Boeing aircraft.

6. Find the *aid*s of all aircraft that can be used on routes from Los Angeles to Chicago.
7. Identify the routes that can be piloted by every pilot who makes more than \$100,000.
8. Print the *ename*s of pilots who can operate planes with *cruisingrange* greater than 3,000 miles, but are not certified on any Boeing aircraft.
9. A customer wants to travel from Madison to New York with no more than two changes of flight. List the choice of departure times from Madison if the customer wants to arrive in New York by 6 p.m.
10. Compute the difference between the average salary of a pilot and the average salary of all employees (including pilots).
11. Print the name and salary of every nonpilot whose salary is more than the average salary for pilots.

Answer 5.3 The answers are given below:

1.
```
SELECT DISTINCT A.aname
FROM   Aircraft A
WHERE  A.Aid IN (SELECT C.aid
                 FROM   Certified C, Employees E
                 WHERE  C.eid = E.eid AND
                 NOT EXISTS ( SELECT *
                              FROM   Employees E1
                              WHERE  E1.eid = E.eid AND E1.salary < 80000 ))
```

2.
```
SELECT   C.eid, MAX (A.cruisingrange)
FROM     Certified C, Aircraft A
WHERE    C.aid = A.aid
GROUP BY C.eid
HAVING   COUNT (*) > 3
```

3.
```
SELECT DISTINCT E.aname
FROM   Employee E
WHERE  E.salary < ( SELECT MIN (F.price)
FROM   Flights F
WHERE  F.from = ‘LA’ AND F.to = ‘Honolulu’ )
```

4. Observe that *aid* is the key for Aircraft, but the question asks for aircraft names; we deal with this complication by using an intermediate relation Temp:

```
SELECT Temp.name, Temp.AvgSalary
FROM   ( SELECT   A.aid, A.aname AS name,
                  AVG (E.salary) AS AvgSalary
         FROM     Aircraft A, Certified C, Employes E
         WHERE    A.aid = C.aid AND
                  C.eid = E.eid AND A.cruisingrange > 1000
         GROUP BY A.aid, A.aname ) AS Temp
```

5.
```
SELECT DISTINCT E.ename
FROM   Employees E, Certified C, Aircraft A
WHERE  E.eid = C.eid AND
       C.aid = A.aid AND
       A.aname = ‘Boeing’
```

6.
```
SELECT A.aid
FROM   Aircraft A
WHERE  A.cruisingrange > ( SELECT MIN (F.distance)
                           FROM   Flights F
                           WHERE  F.from = ‘L.A.’ AND F.to = ‘Chicago’ )
```

7.
```
SELECT DISTINCT F.from, F.to
FROM   Flights F
WHERE  NOT EXISTS ( SELECT *
                    FROM   Employees E
                    WHERE  E.salary > 100000
                    AND
                    NOT EXISTS (SELECT *
                                FROM   Aircraft A, Certified C
                                WHERE  A.cruisingrange > F.distance
                                AND E.eid = C.eid
                                AND A.eid = C.aid) )
```

8.
```
SELECT DISTINCT E.ename
FROM   Employees E, Certified C, Aircraft A
WHERE  C.eid = E.eid
AND    C.aid = A.aid
AND    A.cruisingrange > 3000
AND    E.eid NOT IN ( SELECT C1.eid
FROM Certified C1, Aircraft A1
WHERE C1.aid = A1.aid
AND A1.aname = ‘Boeing’ )
```

9.
```
SELECT F.departs
FROM   Flights F
WHERE  F.flno IN ( ( SELECT F0.flno
                     FROM   Flights F0
                     WHERE  F0.from = ‘Madison’ AND F0.to = ‘NY’ AND
                            AND F0.arrives < 1800 )
                   UNION
                   ( SELECT F0.flno
                     FROM   Flights F0, Flights F1
                     WHERE  F0.from = ‘Madison’ AND F0.to <> ‘NY’ AND
                     AND    F0.to = F1.from AND F1.to = ‘NY’
                            F1.departs > F0.arrives AND
                            F1.arrives < 1800 )
                     UNION
                     (      SELECT F0.flno
                            FROM Flights F0, Flights F1, Flights F2
                            WHERE F0.from = ‘Madison’
                            WHERE F0.to = F1.from
                            AND F1.to = F2.from
                            AND F2.to = ‘NY’
                            AND F0.to <> ‘NY’
                            AND F1.to <> ‘NY’
                            AND F1.departs > F0.arrives
                            AND F2.departs > F1.arrives
                            AND F2.arrives < 1800 ))
```

10.
```
SELECT Temp1.avg - Temp2.avg
FROM   ( SELECT AVG (E.salary) AS avg
         FROM   Employees E
         WHERE  E.eid IN (SELECT DISTINCT C.eid
                FROM Certified C )) AS Temp1,
         ( SELECTAVG (E1.salary) AS avg
         FROM   Employees E1 ) AS Temp2
```

11.
```
SELECT E.ename, E.salary
FROM   Employees E
WHERE  E.eid NOT IN ( SELECT DISTINCT C.eid
                      FROM   Certified C )
AND E.salary > ( SELECT AVG (E1.salary)
                 FROM   Employees E1
                 WHERE  E1.eid IN
                        ( SELECT DISTINCT C1.eid
                          FROM   Certified C1 ) )
```

Exercise 5.4 Consider the following relational schema. An employee can work in more than one department; the *pct_time* field of the Works relation shows the percentage of time that a given employee works in a given department.

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

Write the following queries in SQL:

1. Print the names and ages of each employee who works in both the Hardware department and the Software department.
2. For each department with more than 20 full-time-equivalent employees (i.e., where the part-time and full-time employees add up to at least that many full-time employees), print the *did* together with the number of employees that work in that department.
3. Print the name of each employee whose salary exceeds the budget of all of the departments that he or she works in.
4. Find the *managerid*s of managers who manage only departments with budgets greater than $1,000,000.
5. Find the *ename*s of managers who manage the departments with the largest budget.
6. If a manager manages more than one department, he or she *controls* the sum of all the budgets for those departments. Find the *managerid*s of managers who control more than $5,000,000.
7. Find the *managerid*s of managers who control the largest amount.

Answer 5.4 Answer omitted.

Exercise 5.5 Consider the instance of the Sailors relation shown in Figure 5.1.

1. Write SQL queries to compute the average rating, using `AVG`; the sum of the ratings, using `SUM`; and the number of ratings, using `COUNT`.
2. If you divide the sum computed above by the count, would the result be the same as the average? How would your answer change if the above steps were carried out with respect to the *age* field instead of *rating*?

sid	*sname*	*rating*	*age*
18	jones	3	30.0
41	jonah	6	56.0
22	ahab	7	44.0
63	moby	*null*	15.0

Figure 5.1 An Instance of Sailors

3. Consider the following query: *Find the names of sailors with a higher rating than all sailors with age* < 21. The following two SQL queries attempt to obtain the answer to this question. Do they both compute the result? If not, explain why. Under what conditions would they compute the same result?

```
SELECT S.sname
FROM   Sailors S
WHERE  NOT EXISTS ( SELECT *
                    FROM   Sailors S2
                    WHERE  S2.age < 21
                           AND S.rating <= S2.rating )

SELECT *
FROM   Sailors S
WHERE  S.rating > ANY ( SELECT S2.rating
                        FROM   Sailors S2
                        WHERE  S2.age < 21 )
```

4. Consider the instance of Sailors shown in Figure 5.1. Let us define instance S1 of Sailors to consist of the first two tuples, instance S2 to be the last two tuples, and S to be the given instance.

   (a) Show the left outer join of S with itself, with the join condition being *sid=sid*.

   (b) Show the right outer join of S with itself, with the join condition being *sid=sid*.

   (c) Show the full outer join of S with itself, with the join condition being *sid=sid*.

   (d) Show the left outer join of S1 with S2, with the join condition being *sid=sid*.

   (e) Show the right outer join of S1 with S2, with the join condition being *sid=sid*.

   (f) Show the full outer join of S1 with S2, with the join condition being *sid=sid*.

Answer 5.5 The answers are shown below:

4. (a)

sid	*sname*	*rating*	*age*	*sid*	*sname*	*rating*	*age*
18	jones	3	30.0	18	jones	3	30.0
41	jonah	6	56.0	41	jonah	6	56.0
22	ahab	7	44.0	22	ahab	7	44.0
63	moby	*null*	15.0	63	moby	*null*	15.0

1.
```
SELECT AVG (S.rating) AS AVERAGE
FROM   Sailors S

SELECT SUM (S.rating)
FROM   Sailors S

SELECT COUNT (S.rating)
FROM   Sailors S
```

2. The result using SUM and COUNT would be smaller than the result using AVERAGE if there are tuples with rating = NULL. This is because all the aggregate operators, except for COUNT, ignore NULL values. So the first approach would compute the average over all tuples while the second approach would compute the average over all tuples with non-NULL rating values. However, if the aggregation is done on the age field, the answers using both approaches would be the same since the age field does not take NULL values.
3. Only the first query is correct. The second query returns the names of sailors with a higher rating than *at least one* sailor with age < 21. Note that the answer to the second query does not necessarily contain the answer to the first query. In particular, if all the sailors are at least 21 years old, the second query will return an empty set while the first query will return all the sailors. This is because the NOT EXISTS predicate in the first query will evaluate to *true* if its subquery evaluates to an empty set, while the ANY predicate in the second query will evaluate to *false* if its subquery evaluates to an empty set. The two queries give the same results if and only if one of the following two conditions hold:
   - The *Sailors* relation is empty, or
   - There is at least one sailor with age > 21 in the *Sailors* relation, and for every sailor s, either s has a higher rating than all sailors under 21 or s has a rating no higher than all sailors under 21.

Exercise 5.6 Answer the following questions.

1. Explain the term *impedance mismatch* in the context of embedding SQL commands in a host language such as C.

(b)

sid	*sname*	*rating*	*age*	*sid*	*sname*	*rating*	*age*
18	jones	3	30.0	18	jones	3	30.0
41	jonah	6	56.0	41	jonah	6	56.0
22	ahab	7	44.0	22	ahab	7	44.0
63	moby	*null*	15.0	63	moby	*null*	15.0

(c)

sid	*sname*	*rating*	*age*	*sid*	*sname*	*rating*	*age*
18	jones	3	30.0	18	jones	3	30.0
41	jonah	6	56.0	41	jonah	6	56.0
22	ahab	7	44.0	22	ahab	7	44.0
63	moby	*null*	15.0	63	moby	*null*	15.0

(d)

sid	*sname*	*rating*	*age*	*sid*	*sname*	*rating*	*age*
18	jones	3	30.0	*null*	*null*	*null*	*null*
41	jonah	6	56.0	*null*	*null*	*null*	*null*

(e)

sid	*sname*	*rating*	*age*	*sid*	*sname*	*rating*	*age*
null	*null*	*null*	*null*	22	ahab	7	44.0
null	*null*	*null*	*null*	63	moby	*null*	15.0

(f)

sid	*sname*	*rating*	*age*	*sid*	*sname*	*rating*	*age*
18	jones	3	30.0	*null*	*null*	*null*	*null*
41	jonah	6	56.0	*null*	*null*	*null*	*null*
null	*null*	*null*	*null*	22	ahab	7	44.0
null	*null*	*null*	*null*	63	moby	*null*	15.0

2. How can the value of a host language variable be passed to an embedded SQL command?
3. Explain the `WHENEVER` command's use in error and exception handling.
4. Explain the need for cursors.
5. Give an example of a situation that calls for the use of embedded SQL, that is, interactive use of SQL commands is not enough, and some host language capabilities are needed.
6. Write a C program with embedded SQL commands to address your example in the previous answer.
7. Write a C program with embedded SQL commands to find the standard deviation of sailors' ages.
8. Extend the previous program to find all sailors whose age is within one standard deviation of the average age of all sailors.
9. Explain how you would write a C program to compute the transitive closure of a graph, represented as an SQL relation Edges(*from, to*), using embedded SQL commands. (You don't have to write the program; just explain the main points to be dealt with.)
10. Explain the following terms with respect to cursors: *updatability, sensitivity*, and *scrollability*.
11. Define a cursor on the Sailors relation that is updatable, scrollable, and returns answers sorted by *age*. Which fields of Sailors can such a cursor *not* update? Why?
12. Give an example of a situation that calls for dynamic SQL, that is, even embedded SQL is not sufficient.

Answer 5.6 Answer omitted.

Exercise 5.7 Consider the following relational schema and briefly answer the questions that follow:

> 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. Define a table constraint on Emp that will ensure that every employee makes at least $10,000.

2. Define a table constraint on Dept that will ensure that all managers have *age* > 30.
3. Define an assertion on Dept that will ensure that all managers have *age* > 30. Compare this assertion with the equivalent table constraint. Explain which is better.
4. Write SQL statements to delete all information about employees whose salaries exceed that of the manager of one or more departments that they work in. Be sure to ensure that all the relevant integrity constraints are satisfied after your updates.

Answer 5.7 The answers are given below:

1. Define a table constraint on Emp that will ensure that every employee makes at least 10,000

```
CREATE TABLE Emp ( eid      INTEGER,
                   ename    CHAR(10),
                   age      INTEGER ,
                   salary   REAL,
                   PRIMARY KEY (eid),
                   CHECK ( salary >= 10000 ))
```

2. Define a table constraint on Dept that will ensure that all managers have age > 30

```
CREATE TABLE Dept ( did         INTEGER,
                    buget       REAL,
                    managerid   INTEGER ,
                    PRIMARY KEY (did),
                    FOREIGN KEY (managerid) REFERENCES Emp,
                    CHECK (     ( SELECT E.age FROM Emp E, Dept D)
                                WHERE E.eid = D.managerid ) > 30 )
```

3. Define an assertion on Dept that will ensure that all managers have age > 30

```
CREATE TABLE Dept ( did         INTEGER,
                    budget      REAL,
                    managerid   INTEGER ,
                    PRIMARY KEY (did) )


CREATE ASSERTION managerAge
CHECK ((SELECT E.age
        FROM   Emp E, Dept D
        WHERE  E.eid = D.managerid ) > 30 )
```

Since the constraint involves two relations, it is better to define it as an assertion, independent of any one relation, rather than as a check condition on the Dept relation. The limitation of the latter approach is that the condition is checked only when the Dept relation is being updated. However, since age is an attribute of the Emp relation, it is possible to update the age of a manager which violates the constraint. So the former approach is better since it checks for potential violation of the assertion whenever one of the relations is updated.

4. To write such statements, it is necessary to consider the constraints defined over the tables. We will assume the following:

```
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,
                       buget      REAL,
                       managerid  INTEGER ,
                       PRIMARY KEY (did),
                       FOREIGN KEY (managerid) REFERENCES Emp,
                       ON DELETE SET NULL)
```

Now, we can define statements to delete employees who make more than one of their managers:

```
DELETE
FROM   Emp E
       WHERE E.eid IN ( SELECT W.eid
       FROM   Work W, Emp E2, Dept D
       WHERE  W.did = D.did
       AND    D.managerid = E2.eid
       AND    E.salary > E2.salary )
```

Exercise 5.8 Consider the following relations:

Student(<u>*snum:* `integer`</u>, *sname:* `string`, *major:* `string`,

level: `string`, *age:* `integer`)
Class(*name:* `string`, *meets_at:* `time`, *room:* `string`, *fid:* `integer`)
Enrolled(*snum:* `integer`, *cname:* `string`)
Faculty(*fid*: `integer`, *fname:* `string`, *deptid:* `integer`)

The meaning of these relations is straightforward; for example, Enrolled has one record per student-class pair such that the student is enrolled in the class.

1. Write the SQL statements required to create the above relations, including appropriate versions of all primary and foreign key integrity constraints.
2. Express each of the following integrity constraints in SQL unless it is implied by the primary and foreign key constraint; if so, explain how it is implied. If the constraint cannot be expressed in SQL, say so. For each constraint, state what operations (inserts, deletes, and updates on specific relations) must be monitored to enforce the constraint.

   (a) Every class has a minimum enrollment of 5 students and a maximum enrollment of 30 students.

   (b) At least one class meets in each room.

   (c) Every faculty member must teach at least two courses.

   (d) Only faculty in the department with *deptid=33* teach more than three courses.

   (e) Every student must be enrolled in the course called Math101.

   (f) The room in which the earliest scheduled class (i.e., the class with the smallest *meets_at* value) meets should not be the same as the room in which the latest scheduled class meets.

   (g) Two classes cannot meet in the same room at the same time.

   (h) The department with the most faculty members must have fewer than twice the number of faculty members in the department with the fewest faculty members.

   (i) No department can have more than 10 faculty members.

   (j) A student cannot add more than two courses at a time (i.e., in a single update).

   (k) The number of CS majors must be more than the number of Math majors.

   (l) The number of distinct courses in which CS majors are enrolled is greater than the number of distinct courses in which Math majors are enrolled.

   (m) The total enrollment in courses taught by faculty in the department with *deptid=33* is greater than the number of Math majors.

   (n) There must be at least one CS major if there are any students whatsoever.

   (o) Faculty members from different departments cannot teach in the same room.

Answer 5.8 Answer omitted.

Exercise 5.9 Discuss the strengths and weaknesses of the trigger mechanism. Contrast triggers with other integrity constraints supported by SQL.

Answer 5.9 Answer not available yet.

Exercise 5.10 Consider the following relational schema. An employee can work in more than one department; the *pct_time* field of the Works relation shows the percentage of time that a given employee works in a given department.

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

Write SQL-92 integrity constraints (domain, key, foreign key, or `CHECK` constraints; or assertions) or SQL:1999 triggers to ensure each of the following requirements, considered independently.

1. Employees must make a minimum salary of $1,000.
2. Every manager must be also be an employee.
3. The total percentage of all appointments for an employee must be under 100%.
4. A manager must always have a higher salary than any employee that he or she manages.
5. Whenever an employee is given a raise, the manager's salary must be increased to be at least as much.
6. Whenever an employee is given a raise, the manager's salary must be increased to be at least as much. Further, whenever an employee is given a raise, the department's budget must be increased to be greater than the sum of salaries of all employees in the department.

Answer 5.10 Answer omitted.

7 STORING DATA: DISKS AND FILES

Exercise 7.1 What is the most important difference between a disk and a tape?

Answer 7.1 *Tapes* are sequential devices that do not support direct access to a desired page. We must essentially step through all pages in order. *Disks* support direct access to a desired page.

Exercise 7.2 Explain the terms *seek time, rotational delay*, and *transfer time.*

Answer 7.2 Answer omitted.

Exercise 7.3 Both disks and main memory support direct access to any desired location (page). On average, main memory accesses are faster, of course. What is the other important difference (from the perspective of the time required to access a desired page)?

Answer 7.3 The time to access a disk page is not constant. It depends on the location of the data. Accessing to some data might be much faster than to others. It is different for memory. Access to memory is uniform for most computer systems.

Exercise 7.4 If you have a large file that is frequently scanned sequentially, explain how you would store the pages in the file on a disk.

Answer 7.4 Answer omitted.

Exercise 7.5 Consider a disk with a sector size of 512 bytes, 2,000 tracks per surface, 50 sectors per track, 5 double-sided platters, average seek time of 10 msec.

1. What is the capacity of a track in bytes? What is the capacity of each surface? What is the capacity of the disk?

2. How many cylinders does the disk have?
3. Give examples of valid block sizes. Is 256 bytes a valid block size? 2,048? 51,200?
4. If the disk platters rotate at 5,400 rpm (revolutions per minute), what is the maximum rotational delay?
5. Assuming that one track of data can be transferred per revolution, what is the transfer rate?

Answer 7.5 1.

$$bytes/track = bytes/sector \times sectors/track = 512 \times 50 = 25K$$

$$bytes/surface = bytes/track \times tracks/surface = 25K \times 2000 = 50,000K$$

$$bytes/disk = bytes/surface \times surfaces/disk = 50,000K \times 10 = 500,000K$$

2. The number of cylinders is the same as the number of tracks on each platter, which is 2000.
3. The block size should be a multiple of the sector size. We can see that 256 is not a valid block size while 2048 and 51200 are.
4. If the disk platters rotate at 5400rpm, the time required for a rotation, which is the maximum rotational delay, is

$$\frac{1}{5400} \times 60 = 0.011 seconds$$

. The average rotational delay is half of the rotation time, 0.006 seconds.

5. The capacity of a track is 25K bytes. Since one track of data can be transferred per revolution, the data transfer rate is

$$\frac{25K}{0.011} = 2,250K bytes per sec$$

Exercise 7.6 Consider again the disk specifications from Exercise 7.5 and suppose that a block size of 1,024 bytes is chosen. Suppose that a file containing 100,000 records of 100 bytes each is to be stored on such a disk and that no record is allowed to span two blocks.

1. How many records fit onto a block?
2. How many blocks are required to store the entire file? If the file is arranged sequentially on disk, how many surfaces are needed?
3. How many records of 100 bytes each can be stored using this disk?

4. If pages are stored sequentially on disk, with page 1 on block 1 of track 1, what is the page stored on block 1 of track 1 on the next disk surface? How would your answer change if the disk were capable of reading/writing from all heads in parallel?

5. What is the time required to read a file containing 100,000 records of 100 bytes each sequentially? Again, how would your answer change if the disk were capable of reading/writing from all heads in parallel (and the data was arranged optimally)?

6. What is the time required to read a file containing 100,000 records of 100 bytes each in some random order? Note that in order to read a record, the block containing the record has to be fetched from disk. Assume that each block request incurs the average seek time and rotational delay.

Answer 7.6 Answer omitted.

Exercise 7.7 Explain what the buffer manager must do to process a read request for a page. What happens if the requested page is in the pool but not pinned?

Answer 7.7 When a page is requested the buffer manager does the following:

1. The buffer pool is checked to see if it contains the requested page. If the page is not in the pool, it is brought in as follows:
   (a) A frame is chosen for replacement, using the replacement policy.
   (b) If the frame chosen for replacement is dirty, it is *flushed* (the page it contains is written out to disk).
   (c) The requested page is read into the frame chosen for replacement.
2. The requested page is *pinned* (the *pin_count* of its frame is incremented) and its address is returned to the requestor.

Note that if the page is not pinned,it could be removed from buffer pool even if it is actually needed in main memory.

Exercise 7.8 When does a buffer manager write a page to disk?

Answer 7.8 Answer omitted.

Exercise 7.9 What does it mean to say that a page is *pinned* in the buffer pool? Who is responsible for pinning pages? Who is responsible for unpinning pages?

Answer 7.9 1. *Pinning* a page means the *pin_count* of its frame is incremented. Pinning a page guarantees higher-level DBMS software that the page will not be removed from the buffer pool by the buffer manager. That is, another file page will not be read into the frame containing this page until it is unpinned by this requestor.

2. It is the buffer manager's responsibility to pin a page.

3. It is the responsibility of the requestor of that page to tell the buffer manager to unpin a page.

Exercise 7.10 When a page in the buffer pool is modified, how does the DBMS ensure that this change is propagated to disk? (Explain the role of the buffer manager as well as the modifier of the page.)

Answer 7.10 Answer omitted.

Exercise 7.11 What happens if there is a page request when all pages in the buffer pool are dirty?

Answer 7.11 If there are some unpinned pages, the buffer manager chooses one by using a *replacement policy*, flushes this page, and then replaces it with the requested page.

If there are no unpinned pages, the buffer manager has to wait until an unpinned page is available (or signal an error condition to the page requestor).

Exercise 7.12 What is *sequential flooding* of the buffer pool?

Answer 7.12 Answer omitted.

Exercise 7.13 Name an important capability of a DBMS buffer manager that is not supported by a typical operating system's buffer manager.

Answer 7.13 1. Pinning a page to prevent it from being replaced.

2. Ability to explicitly force a single page to disk.

Exercise 7.14 Explain the term *prefetching.* Why is it important?

Answer 7.14 Answer omitted.

Exercise 7.15 Modern disks often have their own main memory caches, typically about one MB, and use this to do prefetching of pages. The rationale for this technique is the empirical observation that if a disk page is requested by some (not necessarily database!) application, 80 percent of the time the next page is requested as well. So the disk gambles by reading ahead.

1. Give a nontechnical reason that a DBMS may not want to rely on prefetching controlled by the disk.
2. Explain the impact on the disk's cache of several queries running concurrently, each scanning a different file.
3. Can the above problem be addressed by the DBMS buffer manager doing its own prefetching? Explain.
4. Modern disks support *segmented caches*, with about four to six segments, each of which is used to cache pages from a different file. Does this technique help, with respect to the above problem? Given this technique, does it matter whether the DBMS buffer manager also does prefetching?

Answer 7.15

1. The pre-fetching done at the disk level varies widely across different drives and manufacturers, and pre-fetching is sufficiently important to a DBMS that one would like to be independent of specific hardware support.
2. If there are many queries running concurrently, the request of a page from different queries can be interleaved. In the worst case, it cause the cache miss on every page request, even with disk pre-fetching.
3. If we have pre-fetching offered by DBMS buffer manager, the buffer manager can predict the reference pattern more accurately. In particular, a certain number of buffer frames can be allocated *per* active scan for pre-fetching purposes, and interleaved requests would not compete for the same frames.

Exercise 7.16 Describe two possible record formats. What are the trade-offs between them?

Answer 7.16 Answer omitted.

Exercise 7.17 Describe two possible page formats. What are the trade-offs between them?

Answer 7.17 Two possible page formats are: *consecutive slots* and *slot directory*

The consecutive slots organization is mostly used for fixed length record formats. It handles the deletion by using bitmaps or linked lists.

The slot directory organization maintains a directory of slots for each page, with a ¡*record offset*, *record length*¿ pair per slot.

The slot directory is an indirect way to get the offset of a entry. Because of this indirection, deletion is easy. It is accomplished by setting the length field to 0. And records can easily be moved around on the page without changing their external identifier.

Exercise 7.18 Consider the page format for variable-length records that uses a slot directory.

1. One approach to managing the slot directory is to use a maximum size (i.e., a maximum number of slots) and to allocate the directory array when the page is created. Discuss the pros and cons of this approach with respect to the approach discussed in the text.
2. Suggest a modification to this page format that would allow us to sort records (according to the value in some field) without moving records and without changing the record ids.

Answer 7.18 Answer omitted.

Exercise 7.19 Consider the two internal organizations for heap files (using lists of pages and a directory of pages) discussed in the text.

1. Describe them briefly and explain the trade-offs. Which organization would you choose if records are variable in length?
2. Can you suggest a single page format to implement both internal file organizations?

Answer 7.19

1. The list of pages in shown in Fig 3.7. The directory of pages is shown in Fig 3.8.
2. The linked-list approach is a little simpler, but finding a page with sufficient free space for a new record (especially with variable length records) is harder. We have to essentially scan the list of pages until we find one with enough space, whereas the directory organization allows us to find such a page by simply scanning the directory, which is much smaller than the entire file. The directory organization is therefore better, especially with variable length records.
3. A page format with *previous* and *next* page pointers would help in both cases. Obviously, such a page format allows us to build the linked list organization; it is also useful for implementing the directory in the directory organization.

Exercise 7.20 Consider a list-based organization of the pages in a heap file in which two lists are maintained: a list of *all* pages in the file and a list of all pages with free space. In contrast, the list-based organization discussed in the text maintains a list of full pages and a list of pages with free space.

1. What are the trade-offs, if any? Is one of them clearly superior?
2. For each of these organizations, describe a page format that can be used to implement it.

Answer 7.20 Answer omitted.

Exercise 7.21 Modern disk drives store more sectors on the outer tracks than the inner tracks. Since the rotation speed is constant, the sequential data transfer rate is also higher on the outer tracks. The seek time and rotational delay are unchanged. Considering this information, explain good strategies for placing files with the following kinds of access patterns:

1. Frequent, random accesses to a small file (e.g., catalog relations).
2. Sequential scans of a large file (e.g., selection from a relation with no index).
3. Random accesses to a large file via an index (e.g., selection from a relation via the index).
4. Sequential scans of a small file.

Answer 7.21

1. Place the file in the middle tracks. Sequential speed is not an issue due to the small size of the file, and the seek time is minimized by placing files in the center.
2. Place the file in the outer tracks. Sequential speed is most important and outer tracks maximize it.
3. Place the file and index on the inner tracks. The DBMS will alternately access pages of the index and of the file, and so the two should reside in close proximity to reduce seek times. By placing the file and the index on the inner tracks we also save valuable space on the faster (outer) tracks for other files that are accessed sequentially.
4. Place small files in the inner half of the disk. A scan of a small file is effectively random I/O because the cost is dominated by the cost of the initial seek to the beginning of the file.

8

FILE ORGANIZATIONS AND INDEXES

Exercise 8.1 What are the main conclusions that you can draw from the discussion of the three file organizations?

Answer 8.1 The main conclusion about the three file organizations is that all three file organizations have their own advantages and disadvantages. No one file organization is uniformly superior in all situations. The choice of appropriate structures for a given data set can have a significant impact upon performance. An unordered file is best if only full file scans are desired. A hashed file is best if the most common operation is an equality selection. A sorted file is best (of the three alternatiaves considered in this chapter) if range selections are desired.

Exercise 8.2 Consider a delete specified using an equality condition. What is the cost if no record qualifies? What is the cost if the condition is not on a key?

Answer 8.2 Answer omitted.

Exercise 8.3 Which of the three basic file organizations would you choose for a file where the most frequent operations are as follows?

1. Search for records based on a range of field values.
2. Perform inserts and scans where the order of records does not matter.
3. Search for a record based on a particular field value.

Answer 8.3

1. Using these fields as the search key, we would choose a sorted file organization.
2. Heap file would be the best fit in this situation.
3. Using this particular field as the searach key, choosing a hashed file would be the best.

Exercise 8.4 Explain the difference between each of the following:

1. Primary versus secondary indexes.
2. Dense versus sparse indexes.
3. Clustered versus unclustered indexes.

If you were about to create an index on a relation, what considerations would guide your choice with respect to each pair of properties listed above?

Answer 8.4 Answer omitted.

Exercise 8.5 Consider a relation stored as a randomly ordered file for which the only index is an unclustered index on a field called *sal*. If you want to retrieve all records with $sal > 20$, is using the index always the best alternative? Explain.

Answer 8.5 No. In this case, the index is unclustered, each qualifying data entries could contain an rid that points to a distinct data page, leading to as many data page I/Os as the number of data entries that match the range query. At this time,using index is worse than file scan.

Exercise 8.6 If an index contains data records as 'data entries', is it clustered or unclustered? Dense or sparse?

Answer 8.6 Answer omitted.

Exercise 8.7 Consider Alternatives (1), (2) and (3) for 'data entries' in an index, as discussed in Section 8.3.1. Are they all suitable for secondary indexes? Explain.

Answer 8.7 Yes.All the three alternatives allow duplicate data entries.

Exercise 8.8 Consider the instance of the Students relation shown in Figure 8.1, sorted by *age*: For the purposes of this question, assume that these tuples are stored in a sorted file in the order shown; the first tuple is in page 1, slot 1; the second tuple is in page 1, slot 2; and so on. Each page can store up to three data records. You can use $\langle$*page-id, slot*$\rangle$ to identify a tuple.

List the data entries in each of the following indexes. If the order of entries is significant, say so and explain why. If such an index cannot be constructed, say so and explain why.

1. A dense index on *age* using Alternative (1).

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

Figure 8.1 An Instance of the Students Relation, Sorted by *age*

2. A dense index on *age* using Alternative (2).
3. A dense index on *age* using Alternative (3).
4. A sparse index on *age* using Alternative (1).
5. A sparse index on *age* using Alternative (2).
6. A sparse index on *age* using Alternative (3).
7. A dense index on *gpa* using Alternative (1).
8. A dense index on *gpa* using Alternative (2).
9. A dense index on *gpa* using Alternative (3).
10. A sparse index on *gpa* using Alternative (1).
11. A sparse index on *gpa* using Alternative (2).
12. A sparse index on *gpa* using Alternative (3).

Answer 8.8 Answer omitted.