1

INTRODUCTION TO DATABASE SYSTEMS

Exercise 1.1 Why would you choose a database system instead of simply storing data in operating system files? When would it make sense *not* to use a database system?

Answer 1.1 A *database* is an integrated collection of data, usually so large that it has to be stored on secondary storage devices such as disks or tapes. This data can be maintained as a collection of operating system files, or stored in a *DBMS* (database management system). The advantages of using a DBMS are:

- *Data independence and efficient access.* Database application programs are independent of the details of data representation and storage. The conceptual and external schemas provide independence from physical storage decisions and logical design decisions respectively. In addition, a DBMS provides efficient storage and retrieval mechanisms, including support for very large files, index structures and query optimization.
- *Reduced application development time.* Since the DBMS provides several important functions required by applications, such as concurrency control and crash recovery, high level query facilities, etc., only application-specific code needs to be written. Even this is facilitated by suites of application development tools available from vendors for many database management systems.
- *Data integrity and security.* The view mechanism and the authorization facilities of a DBMS provide a powerful access control mechanism. Further, updates to the data that violate the semantics of the data can be detected and rejected by the DBMS if users specify the appropriate *integrity constraints.*
- *Data administration.* By providing a common umbrella for a large collection of data that is shared by several users, a DBMS facilitates maintenance and data administration tasks. A good DBA can effectively shield end-users from the chores of fine-tuning the data representation, periodic back-ups etc.

- *Concurrent access and crash recovery.* A DBMS supports the notion of a *transaction*, which is conceptually a single user's sequential program. Users can write transactions as if their programs were running in isolation against the database. The DBMS executes the actions of transactions in an interleaved fashion to obtain good performance, but schedules them in such a way as to ensure that conflicting operations are not permitted to proceed concurrently. Further, the DBMS maintains a continuous log of the changes to the data, and if there is a system crash, it can restore the database to a *transaction-consistent* state. That is, the actions of incomplete transactions are undone, so that the database state reflects only the actions of completed transactions. Thus, if each complete transaction, executing alone, maintains the consistency criteria, then the database state after recovery from a crash is consistent.

If these advantages are not important for the application at hand, using a collection of files may be a better solution because of the increased cost and overhead of purchasing and maintaining a DBMS.

Exercise 1.2 What is logical data independence and why is it important?

Answer 1.2 Answer omitted.

Exercise 1.3 Explain the difference between logical and physical data independence.

Answer 1.3 Logical data independence means that users are shielded from changes in the logical structure of the data, while physical data independence insulates users from changes in the physical storage of the data. We saw an example of logical data independence in the answer to Exercise 1.2. Consider the Students relation from that example (and now assume that it is not replaced by the two smaller relations). We could choose to store Students tuples in a heap file, with a clustered index on the sname field. Alternatively, we could choose to store it with an index on the gpa field, or to create indexes on both fields, or to store it as a file sorted by gpa. These storage alternatives are not visible to users, except in terms of improved performance, since they simply see a relation as a set of tuples. This is what is meant by physical data independence.

Exercise 1.4 Explain the difference between external, internal, and conceptual schemas. How are these different schema layers related to the concepts of logical and physical data independence?

Answer 1.4 Answer omitted.

Exercise 1.5 What are the responsibilities of a DBA? If we assume that the DBA is never interested in running his or her own queries, does the DBA still need to understand query optimization? Why?

Answer 1.5 The DBA is responsible for:

- *Designing the logical and physical schemas, as well as widely-used portions of the external schema.*
- *Security and authorization.*
- *Data availability and recovery from failures.*
- *Database tuning:* The DBA is responsible for evolving the database, in particular the conceptual and physical schemas, to ensure adequate performance as user requirements change.

A DBA needs to understand query optimization even if s/he is not interested in running his or her own queries because some of these responsibilities (database design and tuning) are related to query optimization. Unless the DBA understands the performance needs of widely used queries, and how the DBMS will optimize and execute these queries, good design and tuning decisions cannot be made.

Exercise 1.6 Scrooge McNugget wants to store information (names, addresses, descriptions of embarrassing moments, etc.) about the many ducks on his payroll. Not surprisingly, the volume of data compels him to buy a database system. To save money, he wants to buy one with the fewest possible features, and he plans to run it as a stand-alone application on his PC clone. Of course, Scrooge does not plan to share his list with anyone. Indicate which of the following DBMS features Scrooge should pay for; in each case, also indicate why Scrooge should (or should not) pay for that feature in the system he buys.

1. A security facility.
2. Concurrency control.
3. Crash recovery.
4. A view mechanism.
5. A query language.

Answer 1.6 Answer omitted.

Exercise 1.7 Which of the following plays an important role in *representing* information about the real world in a database? Explain briefly.

1. The data definition language.

2. The data manipulation language.
3. The buffer manager.
4. The data model.

Answer 1.7 Let us discuss the choices in turn.

- The data definition language is important in representing information because it is used to describe external and logical schemas.
- The data manipulation language is used to access and update data; it is not important for representing the data. (Of course, the data manipulation language must be aware of how data is represented, and reflects this in the constructs that it supports.)
- The buffer manager is not very important for representation because it brings arbitrary disk pages into main memory, independent of any data representation.
- The data model is fundamental to representing information. The data model determines what data representation mechanisms are supported by the DBMS. The data definition language is just the specific set of language constructs available to describe an actual application's data in terms of the *data model*.

Exercise 1.8 Describe the structure of a DBMS. If your operating system is upgraded to support some new functions on OS files (e.g., the ability to force some sequence of bytes to disk), which layer(s) of the DBMS would you have to rewrite to take advantage of these new functions?

Answer 1.8 Answer omitted.

Exercise 1.9 Answer the following questions:

1. What is a transaction?
2. Why does a DBMS interleave the actions of different transactions instead of executing transactions one after the other?
3. What must a user guarantee with respect to a transaction and database consistency? What should a DBMS guarantee with respect to concurrent execution of several transactions and database consistency?
4. Explain the strict two-phase locking protocol.
5. What is the WAL property, and why is it important?

Answer 1.9 Let us answer each question in turn:

1. A transaction is any one execution of a user program in a DBMS. This is the basic unit of change in a DBMS.

2. A DBMS is typically shared among many users. Transactions from these users can be interleaved to improve the execution time of users' queries. By interleaving queries, users do not have to wait for other user's transactions to complete fully before their own transaction begins. Without interleaving, if user A begins a transaction that will take 10 seconds to complete, and user B wants to begin a transaction, user B would have to wait an additional 10 seconds for user A's transaction to complete before the database would begin processing user B's request.

3. A user must guarantee that his or her transaction does not corrupt data or insert nonsense in the database. For example, in a banking database, a user must guarantee that a cash withdraw transaction accurately models the amount a person removes from his or her account. A database application would be worthless if a person removed 20 dollars from an ATM but the transaction set their balance to zero! A DBMS must guarantee that transactions are executed fully and independently of other transactions. An essential property of a DBMS is that a transaction should execute atomically, or as if it is the only transaction running. Also, transactions will either complete fully, or will be aborted and the database returned to it's initial state. This ensures that the database remains consistent.

4. Strict two-phase locking uses shared and exclusive locks to protect data. A transaction must hold all the required locks before executing, and does not release any lock until the transaction has completely finished.

5. The WAL property affects the logging strategy in a DBMS. The WAL, Write-Ahead Log, property states that each write action must be recorded in the log (on disk) before the corresponding change is reflected in the database itself. This protects the database from system crashes that happen during a transaction's execution. By recording the change in a log before the change is truly made, the database knows to undo the changes to recover from a system crash. Otherwise, if the system crashes just after making the change in the database but before the database logs the change, then the database would not be able to detect his change during crash recovery.

2

INTRODUCTION TO DATABASE DESIGN

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 Term explanations:

- *Attribute* - a property or description of an entity. A toy department employee entity could have attributes describing the employee's name, salary, and years of service.
- *Domain* - a set of possible values for an attribute.
- *Entity* - an object in the real world that is distinguishable from other objects such as the green dragon toy.
- *Relationship* - an association among two or more entities.
- *Entity set* - a collection of similar entities such as all of the toys in the toy department.
- *Relationship set* - a collection of similar relationships
- *One-to-many relationship* - a key constraint that indicates that one entity can be associated with many of another entity. An example of a one-to-many relationship is when an employee can work for only one department, and a department can have many employees.
- *Many-to-many relationship* - a key constraint that indicates that many of one entity can be associated with many of another entity. An example of a many-to-many relationship is employees and their hobbies: a person can have many different hobbies, and many people can have the same hobby.

- *Participation constraint* - a participation constraint determines whether relationships must involve certain entities. An example is if every department entity has a manager entity. Participation constraints can either be total or partial. A total participation constraint says that every department has a manager. A partial participation constraint says that every employee does not have to be a manager.
- *Overlap constraint* - within an ISA hierarchy, an overlap constraint determines whether or not two subclasses can contain the same entity.
- *Covering constraint* - within an ISA hierarchy, a covering constraint determines where the entities in the subclasses collectively include all entities in the superclass. For example, with an Employees entity set with subclasses HourlyEmployee and SalaryEmployee, does every Employee entity necessarily have to be within either HourlyEmployee or SalaryEmployee?
- *Weak entity set* - an entity that cannot be identified uniquely without considering some primary key attributes of another identifying owner entity. An example is including Dependent information for employees for insurance purposes.
- *Aggregation* - a feature of the entity relationship model that allows a relationship set to participate in another relationship set. This is indicated on an ER diagram by drawing a dashed box around the aggregation.
- *Role indicator* - If an entity set plays more than one role, role indicators describe the different purpose in the relationship. An example is a single Employee entity set with a relation Reports-To that relates supervisors and subordinates.

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 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.

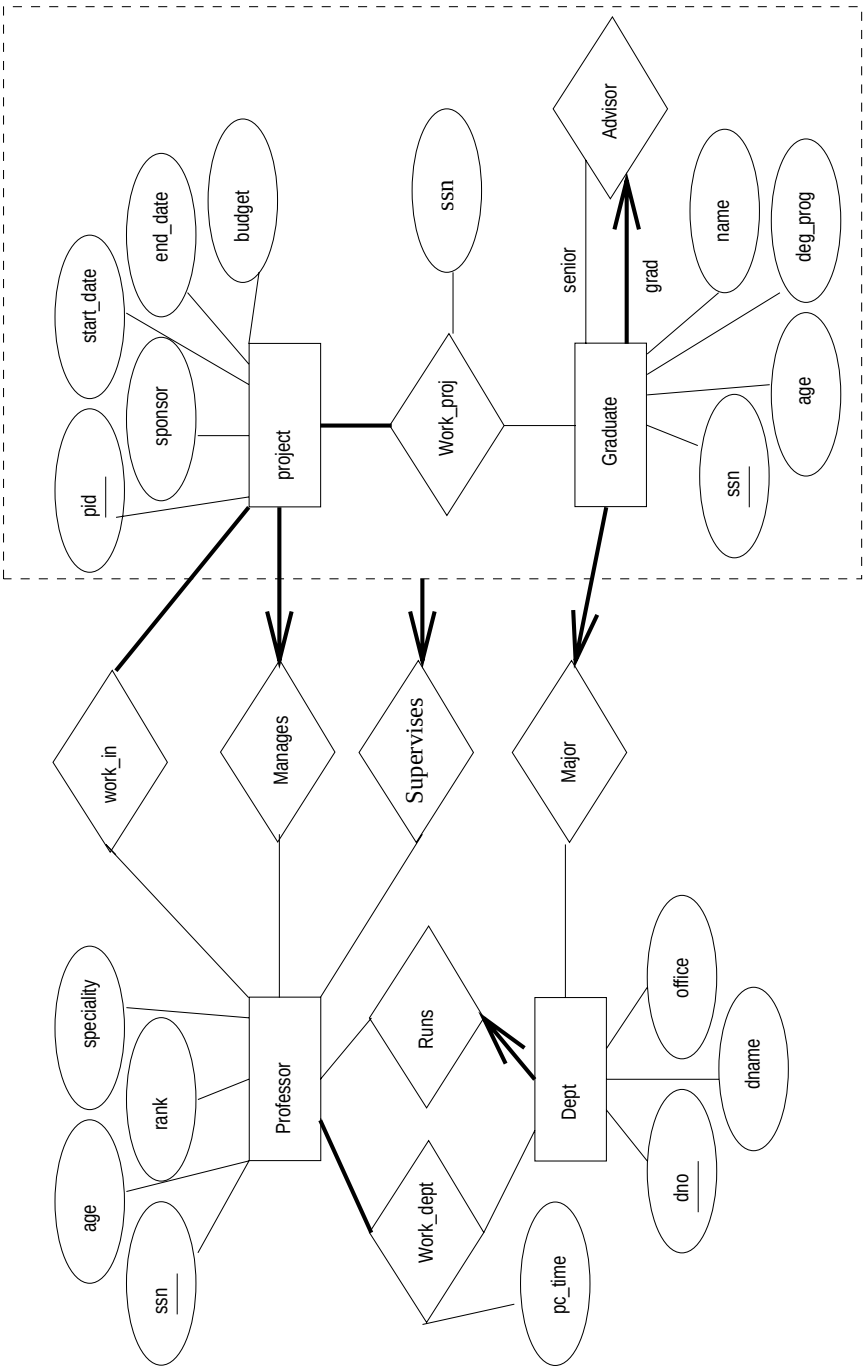


Figure 2.1 ER Diagram for Exercise 2.3

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 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 $2500/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 used in songs recorded at Notown has a unique identification number, a name (e.g., guitar, synthesizer, flute) and a musical key (e.g., C, B-flat, E-flat).
- Each album recorded on the Notown label has a unique identification number, 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.

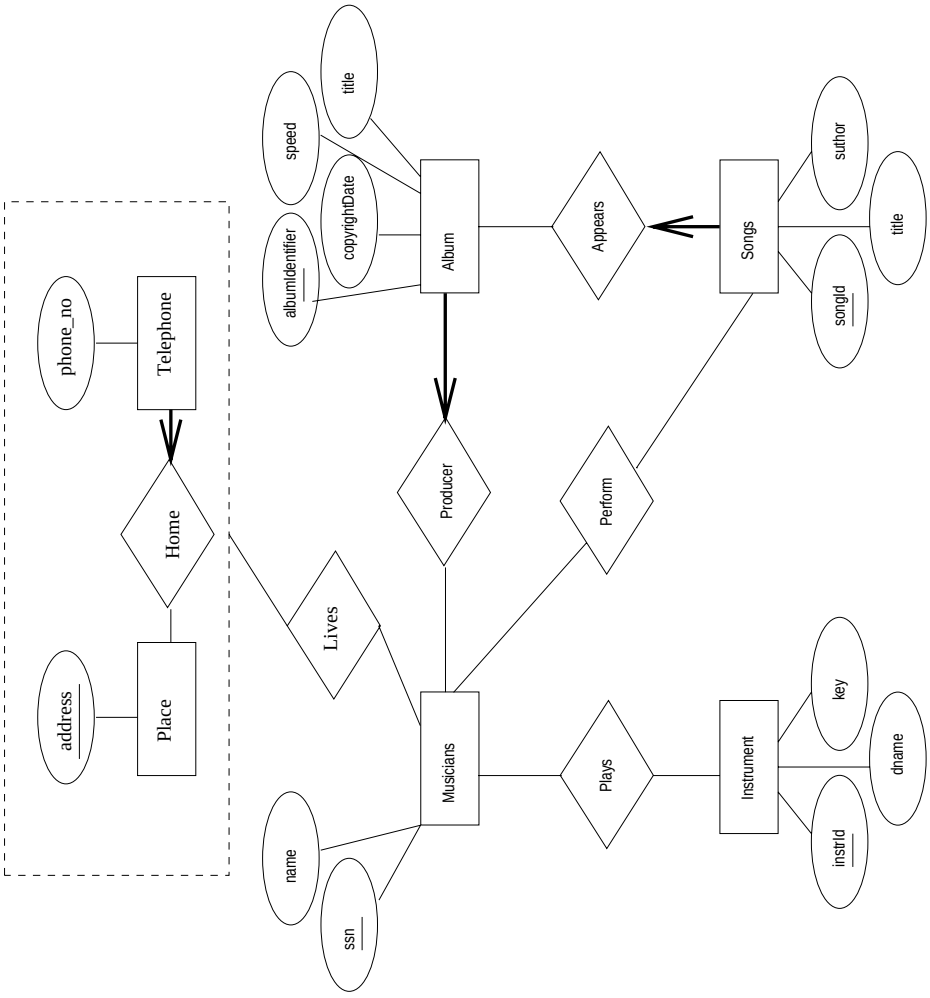


Figure 2.2 ER Diagram for Exercise 2.5

Design a conceptual schema for Notown and draw an ER diagram for your schema. The preceding information describes the situation that the Notown database must model. Be sure to indicate all key and cardinality constraints and any assumptions you make. Identify any constraints 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.

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 decided that all information related to the airport should be organized using a DBMS, and you have been hired to design the database. Your first task is to organize the information about all the airplanes 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 a 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 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 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 medicine 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 preceding information. Identify any constraints 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.

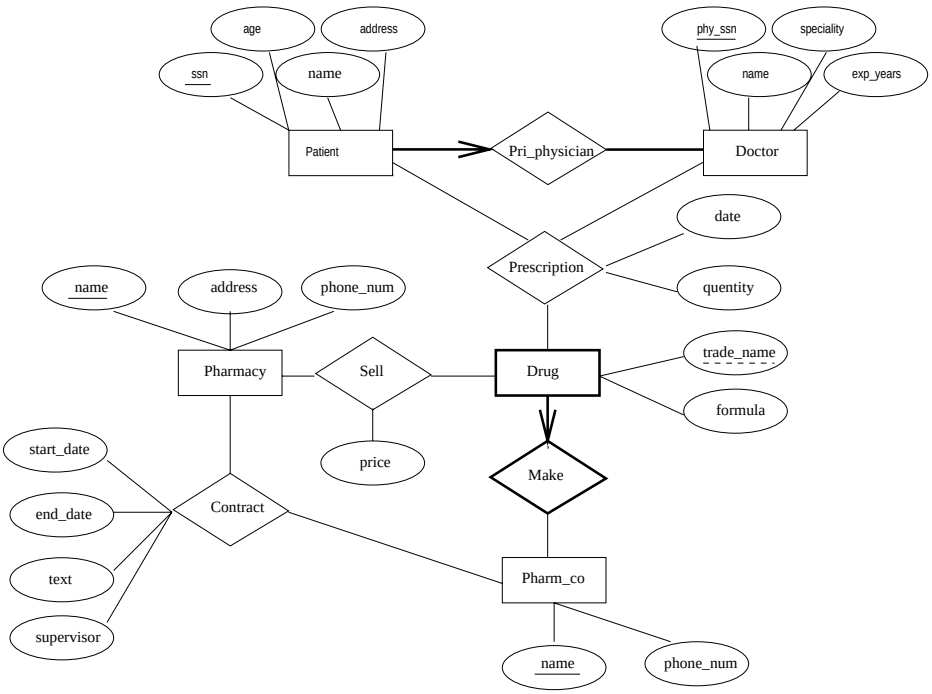


Figure 2.3 ER Diagram for Exercise 2.7

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 price from 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 *database* 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 just given) that describes the group. Finally, galleries keep information about customers. For each customer, galleries keep that person's unique name, address, total amount of dollars spent in the gallery (very important!), and the artists and groups of art that the customer tends to like.

Draw the ER diagram for the database.

Answer 2.8 Answer omitted.

Exercise 2.9 Answer the following questions.

- Explain the following terms briefly: *UML, use case diagrams, statechart diagrams, class diagrams, database diagrams, component diagrams*, and *deployment diagrams.*
- Explain the relationship between ER diagrams and UML.

Answer 2.9 Not yet done.

3

THE RELATIONAL MODEL

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

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

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

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

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

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

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

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

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

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

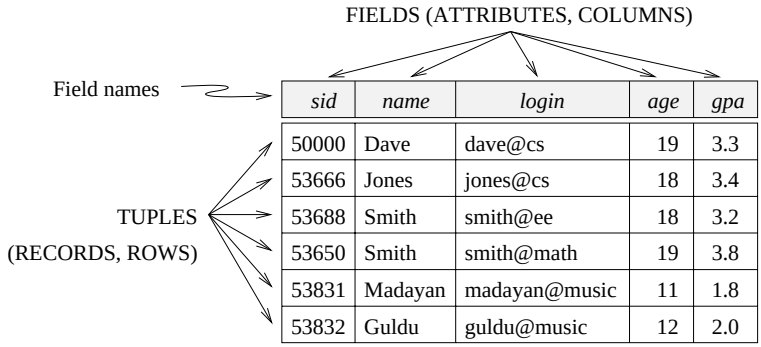
Answer 3.2 Answer omitted.

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

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

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

Answer 3.4 Answer omitted.



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

Figure 3.1 An Instance $S1$ of the Students Relation

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

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

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

You cannot determine a key of a relation given only one instance of the relation. The fact that the instance is "legal" is immaterial. A candidate key, as defined here, *is a*

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

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

Answer 3.6 Answer omitted.

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

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

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

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

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

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

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

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

2. Write the SQL statements required to create the preceding relations, including appropriate versions of all primary and foreign key integrity constraints.
3. Define the Dept relation in SQL so that every department is guaranteed to have a manager.
4. Write an SQL statement to add John Doe as an employee with $eid = 101$, $age = 32$ and $salary = 15,000$.
5. Write an SQL statement to give every employee a 10 percent raise.
6. Write an SQL statement to delete the Toy department. Given the referential integrity constraints you chose for this schema, explain what happens when this statement is executed.

Answer 3.8 Answer omitted.

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

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

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

1. Modify this query so that only the *login* column is included in the answer.
2. If the clause `WHERE` *S.gpa >= 2* is added to the original query, what is the set of tuples in the answer?

Answer 3.9 The answers are as follows:

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

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

2. The answer tuple for Madayan is omitted then.

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

Answer 3.10 Answer omitted.

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

Answer 3.11 The following SQL statements create the corresponding relations.

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

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


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

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

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

Answer 3.12 Answer omitted.

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

Answer 3.13 The following SQL statements create the corresponding relations.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Exercise 3.14 Consider the scenario from Exercise 2.4, where you designed an ER diagram for a company database. Write SQL statements to create the corresponding

relations and capture as many of the constraints as possible. If you cannot capture some constraints, explain why.

Answer 3.14 Answer omitted.

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

Answer 3.15 The following SQL statements create the corresponding relations.

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

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

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

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

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

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

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

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

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

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

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

Answer 3.16 Answer omitted.

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

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

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

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

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

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

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

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

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

Answer 3.18 Answer omitted.

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

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

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

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

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

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

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

Answer 3.19 The answer to each question is given below.

1. The system will do the following:

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

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

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

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

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

Exercise 3.20 Consider the following schema:

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

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

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

Answer 3.20 Answer omitted.

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 resulting relation produced by each of the following relational algebra expressions. In each case, state any assumptions about the schemas for $R1$ and $R2$ 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(*sid:* `integer`, *sname:* `string`, *address:* `string`)
Parts(*pid:* `integer`, *pname:* `string`, *color:* `string`)
Catalog(*sid:* `integer`, *pid:* `integer`, *cost:* `real`)

The key fields are underlined, and the domain of each field is listed after the field name. Therefore *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 *name*s of suppliers who supply some red part.
2. Find the *sid*s of suppliers who supply some red or green part.
3. Find the *sid*s of suppliers who supply some red part or are at 221 Packer Street.
4. Find the *sid*s of suppliers who supply some red part and some green part.
5. Find the *sid*s of suppliers who supply every part.
6. Find the *sid*s of suppliers who supply every red part.
7. Find the *sid*s of suppliers who supply every red or green part.
8. Find the *sid*s of suppliers who supply every red part or supply every green part.
9. Find pairs of *sid*s such that the supplier with the first *sid* charges more for some part than the supplier with the second *sid*.
10. Find the *pid*s of parts supplied by at least two different suppliers.
11. Find the *pid*s of the most expensive parts supplied by suppliers named Yosemite Sham.
12. Find the *pid*s 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

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

- SQL

```
SELECT C.sid
FROM   Catalog C
WHERE  NOT EXISTS (SELECT P.pid
                   FROM   Parts P
                   WHERE  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

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

- DRC

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

- SQL

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

7. - RA

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

- TRC

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

- DRC

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

- 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

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

- DRC

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

- 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

$$\{T \mid \exists T1 \in Catalog(\exists T2 \in Catalog (T2.pid = T1.pid \wedge T2.sid \neq T1.sid \wedge T2.cost < T1.cost \wedge T.sid2 = T2.sid) \wedge T.sid1 = T1.sid)\}$$

- DRC

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

- 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.pid
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 The statements can be interpreted as:

1. Find the Supplier names of the suppliers who supply a red part that costs less than 100 dollars.
2. This Relational Algebra statement does not return anything because of the sequence of projection operators. Once the sid is projected, it is the only field in the set. Therefore, projecting on sname will not return anything.
3. Find the Supplier names of the suppliers who supply a red part that costs less than 100 dollars and a green part that costs less than 100 dollars.
4. Find the Supplier ids of the suppliers who supply a red part that costs less than 100 dollars and a green part that costs less than 100 dollars.
5. Find the Supplier names of the suppliers who supply a red part that costs less than 100 dollars and a green part that costs less than 100 dollars.

Exercise 4.5 Consider the following relations containing airline flight information:

Flights(*flno:* `integer`, *from:* `string`, *to:* `string`,
distance: `integer`, *departs:* `time`, *arrives:* `time`)
Aircraft(*aid:* `integer`, *aname:* `string`, *cruisingrange:* `integer`)
Certified(*eid:* `integer`, *aid:* `integer`)
Employees(*eid:* `integer`, *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

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

■ TRC

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

■ DRC

$$\{\langle Ceid \rangle \mid \langle Ceid, Caid \rangle \in Certified \wedge \exists Aid, AN, AR(\langle Aid, AN, AR \rangle \in Aircraft \wedge Aid = Caid \wedge AN = 'Boeing')\}$$

■ 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

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

■ DRC

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

■ 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

$\rho(BonnToMadrid, \sigma_{from='Bonn' \wedge to='Madrid'}(Flights))$
$\pi_{aid}(\sigma_{cruisingrange>distance}(Aircraft \times BonnToMadrid))$

■ TRC

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

■ DRC

$\{Aid \mid \langle Aid, AN, AR\rangle \in Aircraft \wedge$
$(\exists FN, FF, FT, FDi, FDe, FA(\langle FN, FF, FT, FDi, FDe, FA\rangle \in Flights \wedge$
$FF = `Bonn' \wedge FT = `Madrid' \wedge FDi < AR))\}$

■ SQL

```
SELECT A.aid
FROM   Aircraft A, Flights F
WHERE  F.from = 'Bonn' AND F.to = 'Madrid' 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 $\rho(R1, \pi_{eid}(\sigma_{cruisingrange>3000}(Aircraft \bowtie Certified)))$
$\pi_{ename}(Employees \bowtie (R1 - \pi_{eid}(\sigma_{aname='Boeing'}(Aircraft \bowtie Certified))))$

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

   ■ DRC
   $\{\langle EN \rangle \mid \langle Eid, EN, ES \rangle \in Employees \wedge$
   $\exists Ceid, Caid(\langle Ceid, Caid \rangle \in Certified \wedge$
   $\exists Aid, AN, AR(\langle Aid, AN, AR \rangle \in Aircraft \wedge$
   $Aid = Caid \wedge Eid = Ceid \wedge AR > 3000)) \wedge$
   $\neg(\exists Aid2, AN2, AR2(\langle Aid2, AN2, AR2 \rangle \in Aircraft \wedge$
   $\exists Ceid2, Caid2(\langle Ceid2, Caid2 \rangle \in Certified$
   $\wedge Aid2 = Caid2 \wedge Eid = Ceid2 \wedge AN2 = `Boeing')))\}$

   ■ 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

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

- 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

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

- 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 up to 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.

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, CONSTRAINTS, TRIGGERS

Online material is available for all exercises in this chapter on the book's webpage at

`http://www.cs.wisc.edu/~dbbook`

This includes scripts to create tables for each exercise for use with Oracle, IBM DB2, Microsoft SQL Server, Microsoft Access and MySQL.

Exercise 5.1 Consider the following relations:

Student(*snum:* `integer`, *sname:* `string`, *major:* `string`, *level:* `string`, *age:* `integer`)
Class(*name:* `string`, *meets_at:* `string`, *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 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. For each level, print the level and the average age of students for that level.

8. For all levels except JR, print the level and the average age of students for that level.

9. For each faculty member that has taught classes only in room R128, print the faculty member's name and the total number of classes she or he has taught.

10. Find the names of students enrolled in the maximum number of classes.

11. Find the names of students not enrolled in any class.

12. 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.snum 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.meets_at = C2.meets_at)
```

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 COUNT (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   F.fname, COUNT(*) AS CourseCount
FROM     Faculty F, Class C
WHERE    F.fid = C.fid
GROUP BY F.fid, F.fname
HAVING   EVERY ( C.room = 'R128' )
```

10.
```
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 ))
```

11.
```
SELECT DISTINCT S.sname
FROM    Student S
WHERE   S.snum NOT IN (SELECT E.snum
                       FROM   Enrolled E )
```

12.
```
SELECT    S.age, S.level
FROM      Student S
GROUP BY S.age, S.level,
HAVING    S.level IN (SELECT   S1.level
                      FROM     Student S1
                      WHERE    S1.age = S.age
                      GROUP BY S1.level, S1.age
                      HAVING   COUNT (*) >= ALL (SELECT   COUNT (*)
                                                 FROM     Student S2
                                                 WHERE s1.age = S2.age
                                                 GROUP BY S2.level, S2.age))
```

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 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.
10. For every supplier that only supplies green parts, print the name of the supplier and the total number of parts that she supplies.
11. For every supplier that supplies a green part and a red part, print the name and price of the most expensive part that she supplies.

Answer 5.2 Answer omitted.

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

Flights(*flno:* `integer`, *from:* `string`, *to:* `string`, *distance:* `integer`, *departs:* `time`, *arrives:* `time`, *price:* `real`)
Aircraft(*aid:* `integer`, *aname:* `string`, *cruisingrange:* `integer`)
Certified(*eid:* `integer`, *aid:* `integer`)
Employees(*eid:* `integer`, *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 have salaries 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 for which she or he is certified.
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 1000 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 3000 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.

12. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles.

13. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles, but on at least two such aircrafts.

14. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles and who are certified on some Boeing aircraft.

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.ename
FROM   Employees E
WHERE  E.salary < ( SELECT MIN (F.price)
                    FROM   Flights F
                    WHERE  F.from = 'Los Angeles' 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, Employees 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 LIKE 'Boeing%'
```

6.

```
SELECT A.aid
FROM   Aircraft A
WHERE  A.cruisingrange > ( SELECT MIN (F.distance)
                           FROM   Flights F
                           WHERE  F.from = 'Los Angeles' 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.aid = C.aid) )
```

8.

```
SELECT DISTINCT E.ename
FROM   Employees E
WHERE  E.eid IN ( ( SELECT C.eid
                    FROM   Certified C
                    WHERE  EXISTS ( SELECT A.aid
                                    FROM   Aircraft A
                                    WHERE  A.aid = C.aid
                                    AND    A.cruisingrange > 3000 )
                    AND
                    NOT EXISTS ( SELECT A1.aid
```

```
                                    FROM    Aircraft A1
                                    WHERE   A1.aid = C.aid
                                    AND     A1.aname LIKE '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 = 'New York'
                            AND F0.arrives < '18:00' )
                   UNION
                   ( SELECT F0.flno
                     FROM   Flights F0, Flights F1
                     WHERE  F0.from = 'Madison' AND F0.to <> 'New York'
                            AND F0.to = F1.from AND F1.to = 'New York'
                            AND F1.departs > F0.arrives
                            AND F1.arrives < '18:00' )
                   UNION
                   ( SELECT F0.flno
                     FROM   Flights F0, Flights F1, Flights F2
                     WHERE  F0.from = 'Madison'
                            AND F0.to = F1.from
                            AND F1.to = F2.from
                            AND F2.to = 'New York'
                            AND F0.to <> 'New York'
                            AND F1.to <> 'New York'
                            AND F1.departs > F0.arrives
                            AND F2.departs > F1.arrives
                            AND F2.arrives < '18:00' ))
```

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,
       (SELECT AVG (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 ) )
```

12.
```
SELECT   E.ename
FROM     Employees E, Certified C, Aircraft A
WHERE    C.aid = A.aid AND E.eid = C.eid
GROUP BY E.eid, E.ename
HAVING   EVERY (A.cruisingrange > 1000)
```

13.
```
SELECT   E.ename
FROM     Employees E, Certified C, Aircraft A
WHERE    C.aid = A.aid AND E.eid = C.eid
GROUP BY E.eid, E.ename
HAVING   EVERY (A.cruisingrange > 1000) AND COUNT (*) > 1
```

14.
```
SELECT   E.ename
FROM     Employees E, Certified C, Aircraft A
WHERE    C.aid = A.aid AND E.eid = C.eid
GROUP BY E.eid, E.ename
HAVING   EVERY (A.cruisingrange > 1000) AND ANY (A.aname = 'Boeing')
```

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(*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`)

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.

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

4. Find the *managerid*s of managers who manage only departments with budgets greater than $1 million.
5. Find the *ename*s of managers who manage the departments with the largest budgets.
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 million.
7. Find the *managerid*s of managers who control the largest amounts.
8. Find the *ename*s of managers who manage only departments with budgets larger than $1 million, but at least one department with budget less than $5 million.

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 just computed by the count, would the result be the same as the average? How would your answer change if these steps were carried out with respect to the *age* field instead of *rating*?
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:

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

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

(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

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.

(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 need not 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: