1 (~Exercise 3.11)

a. Find the names of all students who have taken at least one Comp. Sci. course; make sure there are no duplicate names in the result.

```
SELECT DISTINCT NAME
FROM STUDENT S
RIGHT JOIN TAKES T ON S.ID = T.ID
WHERE COURSE_ID LIKE 'CS-%'
```

b. Find the IDs and names of all students who have not taken any course offering before Spring 2009.

```
SELECT DISTINCT S.NAME,S.ID
FROM STUDENT S JOIN TAKES T
ON S.ID = T.ID
WHERE T.YEAR > '2009'
```

c. For each department, find the maximum salary of instructors in that department. You may assume that every department has at least one instructor.

```
SELECT DISTINCT DEPT_NAME,NAME, SALARY
FROM INSTRUCTOR
WHERE SALARY in
        (SELECT MAX(SALARY)
         FROM INSTRUCTOR
         GROUP BY DEPT_NAME)
```

d. Find the lowest, across all departments, of the per-department maximum salary computed by the preceding query.

```
SELECT DISTINCT DEPT_NAME,NAME, MIN(SALARY)
FROM INSTRUCTOR
WHERE SALARY in
        (SELECT MAX(SALARY)
         FROM INSTRUCTOR
        GROUP BY DEPT_NAME)
```

2 (~Exercise 3.12)

a. Create a new course “CS-001”, titled “Weekly Seminar”, with 0 credits.

```
INSERT INTO COURSE VALUES ("CS-001","Weekly Seminar","Comp. Sci.","0")
```

b. Create a section of this course in Autumn 2009, with section id of 1.

```
INSERT INTO SECTION VALUES ("CS-001","1","Fall","2009","Taylor","3128","F")
```

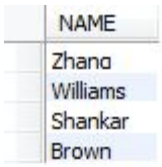
c. Enroll every student in the Comp. Sci. department in the above section.

```
INSERT INTO TAKES VALUES ("00128","CS-001","1","Fall","2009","");
INSERT INTO TAKES VALUES ("12345","CS-001","1","Fall","2009","");
INSERT INTO TAKES VALUES ("54321","CS-001","1","Fall","2009","");
INSERT INTO TAKES VALUES ("76543","CS-001","1","Fall","2009","");
```

d. Delete enrollments in the above section where the student’s name is Chavez.

By this query:

```
SELECT NAME
FROM STUDENT S
RIGHT JOIN TAKES T ON S.ID = T.ID
WHERE COURSE_ID LIKE 'CS-001'
```



NAME
Zhang
Williams
Shankar
Brown

(output)

We can see that there is no student named Chavez.

e. Delete the course CS-001. What will happen if you run this delete statement without first deleting offerings (sections) of this course?

ANSWER:
Normally this would cause an error because of the foreign key constraint, but “on delete cascade” is added, which means that when a record in a parent table is deleted, then the corresponding records in the child table will be deleted. So this should delete all the records which were added in exercise A,B and C.

f. Delete all takes tuples corresponding to any section of any course with the word "database" as a part of the title; ignore case when matching the word with the title.

```
DELETE FROM TAKES
WHERE COURSE_ID in
       (SELECT COURSE_ID
        FROM SECTION
     WHERE COURSE_ID IN
               (SELECT COURSE_ID
               FROM COURSE
               WHERE UPPER(TITLE) LIKE "DATABASE%"))
```

3 (~Exercise 4.12)

For the database of Figure 4.11, write a query to find those employees with no manager. Note that an employee may simply have no manager listed or may have a null manager.

a) Write your query using an outer join and then

```
SELECT EMPLOYEE_NAME
FROM EMPLOYEE
FULL JOIN MANAGES ON EMPLOYEE.EMPLOYEE_NAME = MANAGES.EMPLOYEE_NAME
WHERE MANAGER_NAME <> ''
       AND MANAGER_NAME IS NOT NULL
```

b) Write it again using no outer join at all.

```
SELECT EMPLOYEE_NAME
FROM EMPLOYEE
WHERE EMPLOYEE_NAME in
       (SELECT EMPLOYEE_NAME
        FROM MANAGES
        WHERE MANAGER_NAME <> ''
              AND MANAGER_NAME IS NOT NULL)
```

employee (employee name, street, city)
works (employee name, company name, salary)
company (company name, city)
manages (employee name, manager name)

Figure 4.11 Employee database

4 (~Exercise 4.14)

Show how to define a view tot_credits (year, num credits), giving the total number of credits taken by students in each year.

```
VIEW `university`.`tot_credits` AS
    SELECT
        `university`.`student`.`tot_cred` AS `TOT_CRED`,
        `university`.`takes`.`year` AS `YEAR`
    FROM
        (`university`.`student`
        JOIN `university`.`takes` ON ((`university`.`takes`.`ID` =
`university`.`student`.`ID`)))
```

JDBC

5 (~Exercise 5.12)

Consider the following relations for a company database:
• emp (ename, dname, salary)
• mgr (ename, mname)

and the Java code in Figure 5.26, which uses the JDBC API. Assume that the userid, password, machine name, etc. are all okay. Describe in concise English what the Java program does. (That is, produce an English sentence like “It finds the manager of the toy department,” not a line-by-line description of what each Java statement does.)

```
import java.sql.*;
public class Mystery {
public static void main(String[] args) {
try {
        Connection con=null;
        Class.forName("oracle.jdbc.driver.OracleDriver");
con=DriverManager.getConnection(
"jdbc:oracle:thin:star/X@//edgar.cse.lehigh.edu:1521/XE");
Statement s=con.createStatement();
String q;
String empName = "dog";
boolean more;
ResultSet result;
do {
q = "select mname from mgr where ename = '" + empName + "'";
result = s.executeQuery(q);
more = result.next();
if (more) {
empName = result.getString("mname");
System.out.println (empName);
}
} while (more);
s.close();
con.close();
} catch(Exception e){e.printStackTrace();} }}
```

ANSWER:
It selects the managers from the mgr table, who manages the employee with the name "dog".
It then prints all results found to the console.

PL/SQL and triggers
6.
Write a PL/SQL Procedure getInstructorInfo(id) that fetches all information about an instructor (in the University DB)

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `getInstructorInfo`(IN id varchar(5))
BEGIN
SELECT
    I.ID,
    I.NAME,
    I.DEPT_NAME,
    I.SALARY,
    A.S_ID AS ADVISING,
    C.TITLE AS TEACHES,
    T.COURSE_ID,
    T.SEC_ID,
    T.SEMESTER,
    T.YEAR
FROM
       INSTRUCTOR I
              LEFT OUTER JOIN ADVISOR A
                     ON A.I_ID=I.ID
              RIGHT OUTER JOIN TEACHES T
                     ON T.ID=I.ID
              JOIN COURSE C
                     ON C.COURSE_ID=T.COURSE_ID
WHERE
       I.ID=ID
           ;
END
```

7
Write a Trigger limitSalary that handles situations when instructors are added or updated. It should check that the salary for the affected instructor(s) is/are below an upper limit of 100000.
If this is not the case it should adjust the salary for these instructors so that it equals the highest paid instructor.

I am getting the **error**:
“Selected name conflicts with existing table “instructor””

I have tried both:

```
CREATE TRIGGER INSTRUCTOR_SALARY_INSERT
BEFORE INSERT ON `instructor`
FOR EACH ROW
BEGIN
        IF NEW.SALARY > "100000" THEN
    SET NEW.SALARY = (SELECT MAX(SALARY) FROM INSTRUCTOR);
    END IF;
END;
```

And

```
CREATE DEFINER = CURRENT_USER TRIGGER `university`.`instructor_BEFORE_INSERT`
BEFORE INSERT ON `instructor`
FOR EACH ROW
BEGIN
        IF NEW.SALARY > "100000" THEN
    SET NEW.SALARY = (SELECT MAX(SALARY) FROM INSTRUCTOR);
    END IF;
END
```