Exercise 5.2 Consider the following schema:

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

The 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 The answers are given below:

1.
```
SELECT P.pname
FROM   Parts P, Catalog C
WHERE  P.pid = C.pid
```

2.
```
SELECT S.sname
FROM   Suppiers S
WHERE  NOT EXISTS (( SELECT *
                     FROM   Parts P )
                    EXCEPT
                    ( SELECT C.pid
                      FROM   Catalog C
                      WHERE  C.sid = S.sid ))
```

3.
```
SELECT S.sname
FROM   Suppiers S
WHERE  NOT EXISTS (( SELECT *
                     FROM   Parts P
                     WHERE  P.color = 'red' )
                    EXCEPT
                    ( SELECT C.pid
                      FROM   Catalog C, Parts P
                      WHERE  C.sid = S.sid AND
                             C.pid = P.pid AND P.color = 'red' ))
```

4.
```
SELECT P.pname
FROM   Parts P, Catalog C, Suppliers S
WHERE  P.pid = C.pid AND C.sid = S.sid
AND    S1.sname = 'Acme Widget Suppliers'
AND    NOT EXISTS ( SELECT *
                    FROM   Catalog C1, Suppliers S1
                    WHERE  P.pid = C1.pid AND C1.sid = S1.sid AND
                           S1.sname <> 'Acme Widget Suppliers' )
```

5.
```
SELECT DISTINCT C.sid
FROM   Catalog C
```

```
WHERE  C.cost > ( SELECT AVG (C1.cost)
                  FROM   Catalog C1 )
                  WHERE  C1.pid = C.pid
```

6.

```
SELECT P.pid, S.sname
FROM   Parts P, Suppliers S, Catalog C
WHERE  C.pid = P.pid
AND    C.sid = S.sid
AND    C.cost = (SELECT MAX (C1.cost)
                 FROM   Catalog C1
                 WHERE  C1.pid = P.pid)
```

7.

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

8.

```
SELECT DISTINCT C.sid
FROM   Catalog C, Parts P
WHERE  C.pid = P.pid AND P.color = 'red'
INTERSECT
SELECT DISTINCT C1.sid
FROM   Catalog C1, Parts P1
WHERE  C1.pid = P1.pid AND P1.color = 'green'
```

9.

```
SELECT DISTINCT C.sid
FROM   Catalog C, Parts P
WHERE  C.pid = P.pid AND P.color = 'red'
UNION
SELECT DISTINCT C1.sid
FROM   Catalog C1, Parts P1
WHERE  C1.pid = P1.pid AND P1.color = 'green'
```

10.

```
SELECT S.sname, COUNT(*) as PartCount
FROM   Suppliers S, Parts P, Catalog C
WHERE  P.pid = C.pid AND C.sid = S.sid
GROUP BY S.sname, S.sid
HAVING EVERY (P.color='green')
```

11.

```
SELECT S.sname, MAX(C.cost) as MaxCost
FROM   Suppliers S, Parts P, Catalog C
WHERE  P.pid = C.pid AND C.sid = S.sid
```

```
GROUP BY S.sname, S.sid
HAVING ANY ( P.color='green' ) AND ANY ( P.color = 'red' )
```

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

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

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

1. Find the names of aircraft such that all pilots certified to operate them earn more than $80,000.
2. For each pilot who is certified for more than three aircraft, find the *eid* and the maximum *cruisingrange* of the aircraft 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.aname
FROM   Employee E
WHERE  E.salary < ( SELECT MIN (F.price)
                    FROM   Flights F
                    WHERE  F.from = 'LA' AND F.to = 'Honolulu' )
```

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

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

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

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

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

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

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

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

10.
```
SELECT Temp1.avg - Temp2.avg
FROM   (SELECT AVG (E.salary) AS avg
        FROM   Employees E
        WHERE  E.eid IN (SELECT DISTINCT C.eid
                         FROM Certified C )) AS Temp1,
       (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 5.8 Consider the following relations:

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

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

1. Write the SQL statements required to create these relations, including appropriate versions of all primary and foreign key integrity constraints.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Answer 5.8 Answers are given below.

1. The SQL statements needed to create the tables are given below:

```
CREATE TABLE Student ( snum    INTEGER,
                       sname   CHAR(20),
                       major   CHAR(20),
                       level   CHAR(20),
                       age     INTEGER,
                       PRIMARY KEY (snum))

CREATE TABLE Faculty ( fid     INTEGER,
                       fname   CHAR(20),
                       deptid  INTEGER,
                       PRIMARY KEY (fnum))

CREATE TABLE Class (   name    CHAR(20),
                       meets_atTIME,
                       room    CHAR(10),
```

```
                         fid      INTEGER,
                         PRIMARY KEY (name),
                         FOREIGN KEY (fid) REFERENCES Faculty)

CREATE TABLE Enrolled (snum    INTEGER,
                         cname   CHAR(20),
                         PRIMARY KEY (snum, cname),
                         FOREIGN KEY (snum) REFERENCES Student),
                         FOREIGN KEY (cname) REFERENCES Class)
```

2. The answer to each question is given below

   (a) The Enrolled table should be modified as follows:

```
CREATE TABLE Enrolled (snum      INTEGER,
                       cname     CHAR(20),
                       PRIMARY KEY (snum, cname),
                       FOREIGN KEY (snum) REFERENCES Student),
                       FOREIGN KEY (cname) REFERENCES Class,),
                       CHECK ((  SELECT    COUNT (E.snum)
                                 FROM      Enrolled E
                                 GROUP BY  E.cname) >= 5),
                       CHECK ((  SELECT    COUNT (E.snum)
                                 FROM      Enrolled E
                                 GROUP BY  E.cname) <= 30))
```

   (b) This constraint is already guaranteed because rooms are associated with classes, and thus a new room cannot be declared without an associated class in it.

   (c) Create an assertion as follows:

```
CREATE ASSERTION TeachTwo
CHECK ( ( SELECT    COUNT (*)
          FROM      Facult F, Class C
          WHERE     F.fid = C.fid
          GROUP BY  C.fid
          HAVING    COUNT (*) < 2) = 0)
```

   (d) Create an assertion as follows:

```
CREATE ASSERTION NoTeachThree
CHECK ( ( SELECT    COUNT (*)
          FROM      Facult F, Class C
          WHERE     F.fid = C.fid AND F.deptid ≠ 33
          GROUP BY  C.fid
```

```
HAVING   COUNT (*) > 3) = 0)
```

(e) Create an assertion as follows:

```
CREATE ASSERTION InMath101
CHECK ((  SELECT   COUNT (*)
          FROM     Student S
          WHERE    S.snum NOT IN ( SELECT   E.snum
                                   FROM     Enrolled E
                                   WHERE    E.cname = 'Math101')) = 0)
```

(f) The Class table should be modified as follows:

```
CREATE TABLE Class (  name      CHAR(20),
                      meets_at  TIME,
                      room      CHAR(10),
                      fid       INTEGER,
                      PRIMARY KEY (name),
                      FOREIGN KEY (fid) REFERENCES Faculty),
                      CHECK (   (SELECT   MIN (meets_at)
                                FROM      Class) <>
                                (SELECT   MAX (meets_at)
                                FROM      Class)))
```

(g) The Class table should be modified as follows:

```
CREATE TABLE Class (  name      CHAR(20),
                      meets_at  TIME,
                      room      CHAR(10),
                      fid       INTEGER,
                      PRIMARY KEY (name),
                      FOREIGN KEY (fid) REFERENCES Faculty),
                      CHECK ((SELECT COUNT (*)
                                FROM (    SELECT C.room, C.meets
                                          FROM Class C
                                          GROUP BY C.room, C.meets
                                          HAVING COUNT (*) > 1)) = 0))
```

(h) The Faculty table should be modified as follows:

```
CREATE TABLE Faculty ( fid       INTEGER,
                       fname     CHAR(20),
                       deptid    INTEGER,
                       PRIMARY KEY (fnum),
                       CHECK (   (SELECT MAX (*)
                                 FROM ( SELECT COUNT (*)
                                        FROM Faculty F
```

```
                                 GROUP BY F.deptid))
                           < 2 *
                           (SELECT MIN (*)
                           FROM ( SELECT COUNT (*)
                                  FROM Faculty F
                                  GROUP BY F.deptid))))
```

(i) The Faculty table should be modified as follows:

```
CREATE TABLE Faculty (fid       INTEGER,
                      fname     CHAR(20),
                      deptid    INTEGER,
                      PRIMARY KEY (fnum),
                      CHECK ( ( SELECT    COUNT (*)
                                FROM      Faculty F
                                GROUP BY  F.deptid
                                HAVING    COUNT (*) > 10) = 0))
```

(j) This constraint cannot be done because integratey constraints and assertions only affect the content of a table, not how that content is manipulated.

(k) The Student table should be modified as follows:

```
CREATE TABLE Student ( snum    INTEGER,
                       sname   CHAR(20),
                       major   CHAR(20),
                       level   CHAR(20),
                       age     INTEGER,
                       PRIMARY KEY (snum),
                       CHECK ((SELECT COUNT (*)
                               FROM Student S
                               WHERE S.major = 'CS') >
                              (SELECT COUNT (*)
                               FROM Student S
                               WHERE S.major = 'Math')))
```

(l) Create an assertion as follows:

```
CREATE ASSERTION MoreCSMajors
CHECK (   (SELECT   COUNT (E.cname)
           FROM     Enrolled E, Student S
           WHERE    S.snum = E.snum AND S.major = 'CS') >
          (SELECT   COUNT (E.cname)
           FROM     Enrolled E, Student S
           WHERE    S.snum = E.snum AND S.major = 'Math'))
```

(m) Create an assertion as follows:

```
CREATE ASSERTION MoreEnrolledThanMath
CHECK (  (SELECT  COUNT (E.snum)
         FROM     Enrolled E, Faculty F, Class C
         WHERE    E.cname = C.name
         AND C.fid = F.fid AND F.deptid = 33) >
         (SELECT  COUNT (E.snum)
         FROM     Student S
         WHERE    S.major = 'Math'))
```

(n) The Student table should be modified as follows:

```
CREATE TABLE Student ( snum   INTEGER,
                       sname  CHAR(20),
                       major  CHAR(20),
                       level  CHAR(20),
                       age    INTEGER,
                       PRIMARY KEY (snum),
                       CHECK ( (SELECT COUNT (S.snum)
                              FROM Student S
                              WHERE S.major = 'CS') > 0 ))
```

(o) Create an assertion as follows:

```
CREATE ASSERTION NotSameRoom
CHECK (  (SELECT  COUNT (*)
         FROM     Faculty F1, Faculty F2, Class C1, Class C2
         WHERE    F1.fid = C1.fid
         AND      F2.fid = C2.fid
         AND      C1.room = C2.room
         AND      F1.deptid ≠ F2.deptid) = 0)
```

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

Answer 5.9 A trigger is a procedure that is automatically invoked in response to a specified change to the database. The advantages of the trigger mechanism include the ability to perform an action based on the result of a query condition. The set of actions that can be taken is a superset of the actions that integrity constraints can take (i.e. report an error). Actions can include invoking new update, delete, or insert queries, perform data definition statements to create new tables or views, or alter security policies. Triggers can also be executed before or after a change is made to the database (that is, use old or new data).

There are also disadvantages to triggers. These include the added complexity when trying to match database modifications to trigger events. Also, integrity constraints are incorporated into database performance optimization; it is more difficult for a database to perform automatic optimization with triggers. If database consistency is the primary goal, then integrity constraints offer the same power as triggers. Integrity constraints are often easier to understand than triggers.

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

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

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

1. Employees must make a minimum salary of $1000.
2. Every manager must be also be an employee.
3. The total percentage of all appointments for an employee must be under 100%.
4. A manager must always have a higher salary than any employee that he or she manages.
5. Whenever an employee is given a raise, the manager's salary must be increased to be at least as much.
6. Whenever an employee is given a raise, the manager's salary must be increased to be at least as much. Further, whenever an employee is given a raise, the department's budget must be increased to be greater than the sum of salaries of all employees in the department.

Answer 5.10 The answer to each question is given below.

1. This constraint can be added by modifying the Emp table:

```
CREATE TABLE Emp ( eid        INTEGER,
                   ename      CHAR(20),
                   age        INTEGER,
                   salary     REAL,
                   PRIMARY KEY (eid),
                   CHECK      ( salary > 1000))
```

2. Create an assertion as follows:

```
CREATE ASSERTION ManagerIsEmployee
CHECK ( ( SELECT   COUNT (*)
          FROM     Dept D
          WHERE    D.managerid NOT IN
                   (SELECT * FROM Emp))
          = 0)
```

3. This constraint can be added by modifying the Works table:

```
CREATE TABLE Works ( eid        INTEGER,
                     did        INTEGER,
                     pct_time   INTEGER,
                     PRIMARY KEY (eid, did),
                     CHECK (    (SELECT    COUNT (W.eid)
                                FROM       Works W
                                GROUP BY   W.eid
                                HAVING     Sum(pct_time) > 100) = 0))
```

4. Create an assertion as follows:

```
CREATE ASSERTION ManagerHigherSalary
CHECK (   SELECT   E.eid
          FROM     Emp E, Emp M, Works W, Dept D
          WHERE    E.eid = W.eid
          AND      W.did = D.did
          AND      D.managerid = M.eid
          AND      E.salary > M.salary)
```

5. This constraint can be satisfied by creating a trigger that increases a manager's salary to be equal to the employee who received the raise, if the manager's salary is less than the employee's new salary.

```
CREATE TRIGGER GiveRaise AFTER UPDATE ON Emp
        WHEN     old.salary < new.salary
        FOR EACH ROW
        BEGIN
                 UPDATE   Emp M
                 SET      M.Salary = new.salary
                 WHERE    M.salary < new.salary
```

```
          AND     M.eid IN (SELECT  D.mangerid
                            FROM    Emp E, Works W, Dept D
                            WHERE   E.eid = new.eid
                            AND     E.eid = W.eid
                            AND     W.did = D.did);
      END
```

6. This constraint can be satisfied by extending the trigger in the previous question. We must add an UPDATE command to increase the budget by the amount of the raise if the budget is less than the sum of all employee salaries.

```
CREATE TRIGGER GiveRaise AFTER UPDATE ON Emp
        WHEN    old.salary < new.salary
        FOR EACH ROW
        DECLARE
                _raise REAL;
        BEGIN
                _raise := new.salary - old.salary;
                UPDATE  Emp M
                SET     M.Salary = new.salary
                WHERE   M.salary < new.salary
                AND     M.eid IN (SELECT  D.mangerid
                                  FROM    Emp E, Works W, Dept D
                                  WHERE   E.eid = new.eid
                                  AND     E.eid = W.eid
                                  AND     W.did = D.did);
                UPDATE  Dept D
                SET     D.budget = D.budget + _raise
                WHERE   D.did IN (SELECT  W.did
                                  FROM    Emp E, Works W, Dept D
                                  WHERE   E.eid = new.eid
                                  AND     E.eid = W.eid
                                  AND     D.did = W.did
                                  AND     D.budget <
                                  (SELECT  Sum(E2.salary)
                                  FROM    Emp E2, Works W2
                                  WHERE   E2.eid = W2.eid
                                  AND     W2.dept = D.did));
                END
```