

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.


```

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 *pnames* of parts for which there is some supplier.
2. Find the *snames* of suppliers who supply every part.
3. Find the *snames* of suppliers who supply every red part.
4. Find the *pnames* of parts supplied by Acme Widget Suppliers and no one else.
5. Find the *sids* 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 *sids* of suppliers who supply only red parts.
8. Find the *sids* of suppliers who supply a red part and a green part.

9. Find the *sids* 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(*fno*: 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 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 *aids* 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 *enames* 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, 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 = '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')

```





- ```

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.
4. Find the *managerids* of managers who manage only departments with budgets greater than \$1 million.
5. Find the *enames* of managers who manage the departments with the largest budgets.

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

Figure 5.1 An Instance of Sailors

6. If a manager manages more than one department, he or she *controls* the sum of all the budgets for those departments. Find the *managerids* of managers who control more than \$5 million.
7. Find the *managerids* of managers who control the largest amounts.
8. Find the *enames* 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 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

4. (a)

<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>	<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>
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	<i>null</i>	15.0	63	moby	<i>null</i>	15.0

(b)

<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>	<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>
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	<i>null</i>	15.0	63	moby	<i>null</i>	15.0

(c)

<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>	<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>
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	<i>null</i>	15.0	63	moby	<i>null</i>	15.0

(d)

<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>	<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>
18	jones	3	30.0	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>
41	jonah	6	56.0	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>

(e)

<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>	<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>
<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	22	ahab	7	44.0
<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	63	moby	<i>null</i>	15.0

	<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>	<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>
	18	jones	3	30.0	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>
(f)	41	jonah	6	56.0	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>
	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	22	ahab	7	44.0
	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	63	moby	<i>null</i>	15.0

- 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.
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?

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

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

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

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

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

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

```
CREATE TABLE Works ( eid      INTEGER,
                     did      INTEGER,
                     pcttime  INTEGER,
                     PRIMARY KEY (eid, did),
                     FOREIGN KEY (did) REFERENCES Dept,
                     FOREIGN KEY (eid) REFERENCES Emp,
                     ON DELETE CASCADE)
```

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

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

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

Exercise 5.8 Consider the following relations:

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

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

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 Answer omitted.

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

Answer 5.9 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 Answer omitted.