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(<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 DISTINCT P.pname
FROM   Parts P, Catalog C
WHERE  P.pid = C.pid
```

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

3.
```
SELECT S.sname
FROM   Suppliers S
WHERE  NOT EXISTS (( SELECT P.pid
                     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    S.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(*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(<u>*eid:* `integer`</u>, *ename:* `string`, *age:* `integer`, *salary:* `real`)
Works(<u>*eid:* `integer`, *did:* `integer`</u>, *pct_time:* `integer`)
Dept(<u>*did:* `integer`</u>, *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 *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 The answers are given below:

1.
```
SELECT E.ename, E.age
FROM   Emp E, Works W1, Works W2, Dept D1, Dept D2
WHERE  E.eid = W1.eid AND W1.did = D1.did AND D1.dname = 'Hardware' AND
       E.eid = W2.eid AND W2.did = D2.did AND D2.dname = 'Software'
```

2.
```
SELECT   W.did, COUNT (W.eid)
FROM     Works W
GROUP BY W.did
HAVING   2000 < ( SELECT SUM (W1.pct_time)
                  FROM   Works W1
                  WHERE  W1.did = W.did )
```

3.
```
SELECT E.ename
FROM   Emp E
WHERE  E.salary > ALL (SELECT D.budget
                       FROM   Dept D, Works W
                       WHERE  E.eid = W.eid AND D.did = W.did)
```

4.
```
SELECT DISTINCT D.managerid
FROM   Dept D
WHERE  1000000 < ALL (SELECT D2.budget
                      FROM Dept D2
                      WHERE D2.managerid = D.managerid )
```

5.
```
SELECT E.ename
FROM   Emp E
WHERE  E.eid IN (SELECT D.managerid
                 FROM   Dept D
                 WHERE  D.budget = (SELECT MAX (D2.budget)
                                    FROM Dept D2))
```

6.
```
SELECT D.managerid
FROM   Dept D
WHERE  5000000 < (SELECT SUM (D2.budget)
                  FROM Dept D2
                  WHERE D2.managerid = D.managerid )
```

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

7.
```
SELECT DISTINCT tempD.managerid
FROM   (SELECT DISTINCT D.managerid, SUM (D.budget) AS tempBudget
        FROM   Dept D
        GROUP BY D.managerid ) AS tempD
WHERE  tempD.tempBudget = (SELECT MAX (tempD.tempBudget)
                           FROM   tempD)
```

8.
```
SELECT E.ename
FROM   Emp E, Dept D
WHERE  E.eid = D.managerid GROUP BY E.Eid, E.ename
HAVING EVERY (D.budget > 1000000) AND ANY (D.budget < 5000000)
```

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:

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

- 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:

(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