DBI202

SQL exercise 1

Question

Consider the Sailors-Boats-Reserves DB described in the text.
s (sid, sname, rating, age)
b (bid, bname, color)
r (sid, bid, date)
Write each of the following queries in SQL.

1/ Find the colors of boats reserved by Albert.
2/ Find all sailor id's of sailors who have a rating of at least 8 or reserved boat 103.
3/ Find the names of sailors who have not reserved a boat whose name contains the string "storm". Order the names in ascending order.
4/ Find the sailor id's of sailors with age over 20 who have not reserved a boat whose name includes the string "thunder".
5/ Find the names of sailors who have reserved at least two boats.
6/ Find the names of sailors who have reserved all boats.
7/ Find the names of sailors who have reserved all boats whose name starts with "typhoon".
8/ Find the sailor id's of sailors whose rating is better than some sailor called Bob.
9/ Find the sailor id's of sailors whose rating is better than every sailor called Bob.
10/ Find the sailor id's of sailors with the highest rating.
11/ Find the name and age of the oldest sailor.
12/ Find the names of sailors who have reserved every boat reserved by those with a lower rating.
13/ For each rating, find the average age of sailors at that level of rating.
14/ For each boat which was reserved by at least 5 distinct sailors, find the boat id and the average age of sailors who reserved it.
15/ For each boat which was reserved by at least 5 sailors with age >= 40, find the boat id and the average age of such sailors.
16/ For each boat which was reserved by at least 5 sailors with age >= 40, find the boat id and the average age of all sailors who reserved the boat.

SOLUTION EXERCISE 1

1. Find the colors of boats reserved by Albert.

```
SELECT color
FROM    s, b, r
WHERE r.sid=s.sid AND r.bid=b.bid AND
```

sname=`Albert’

2. Find all sailor id’s of sailors who have a rating of at least 8 or reserved boat 103.

```
(SELECT sid
FROM     s
WHERE  rating>=8)
UNION
(SELECT sid
FROM     r
WHERE  bid=103)
```

3. Find the names of sailors who have not reserved a boat whose name contains the string “storm”. Order the names in ascending order.

```
SELECT sname
FROM     s s1
WHERE  sid NOT IN
   (SELECT sid
    FROM      r, s
    WHERE r.sid=s.sid AND sname LIKE `%storm%’)
ORDER BY s1.sname
```

4. Find the sailor id’s of sailors with age over 20 who have not reserved a boat whose name includes the string “thunder”.

```
SELECT  sid
FROM     s
WHERE  age>20 AND sid NOT IN
   (SELECT sid
    FROM r, b
    WHERE r.bid=b.bid AND bname LIKE `%thunder%')
```

5. Find the names of sailors who have reserved at least two boats.

```
SELECT sname
FROM    s, r r1, r r2
WHERE s.sid=r1.sid AND s.sid=r2.sid AND
            r1.bid<>r2.bid
```

Note: If we want to eliminate duplicates, we SELECT DISTINCT sname. Alternatively, we could simply change the condition in the WHERE clause from r1.bid<>r2.bid to, say r1.bid<r2.bid. (Why would this have the same effect?)

6. Find the names of sailors who have reserved all boats.

```
SELECT sname
FROM    s
WHERE NOT EXISTS (
   SELECT *
   FROM b
  WHERE NOT EXISTS (
     SELECT *
     FROM     r
    WHERE r.sid=s.sid AND r.bid=b.bid))
```

Note: A general remark that applies to most queries involving EXISTS or NOT EXISTS is that you can often "not care" about which attributes are selected in the SELECT statement. The reason is using the above constructs you merely tests the non-emptiness or emptiness of a certain collection of tuples. In other words, we can just do a SELECT *. On the other hand, if you are taking the difference between two collections (e.g., using EXCEPT) then you **must** care about the attributes being selected, since difference is only defined between two union-compatible relations.

7. Find the names of sailors who have reserved all boats whose name starts with “typhoon”.

```
SELECT sname
FROM    s
WHERE NOT EXISTS (
   SELECT *
   FROM b
  WHERE bname LIKE `typhoon%’ AND NOT EXISTS (
     SELECT *
     FROM     r
    WHERE r.sid=s.sid AND r.bid=b.bid))
```

8. Find the sailor id’s of sailors whose rating is better than some sailor called Bob.

```
SELECT  s1.sid
FROM     s s1, s s2
WHERE s1.rating>s2.rating AND s2.sname=`Bob’
```

Alternatively:

```
SELECT sid
FROM    s
WHERE rating > any (
  SELECT rating
  FROM    s s2
  WHERE s2.sname=`Bob’))
```

9. Find the sailor id’s of sailors whose rating is better than every sailor called Bob.

```
SELECT sid
FROM    s
WHERE rating > all (
  SELECT rating
  FROM    s s2
  WHERE s2.sname=`Bob’)
```

10. Find the sailor id’s of sailors with the highest rating.

```
SELECT sid
FROM    s s1
WHERE s1.rating >= all (
  SELECT rating
  FROM    s)
```

11. Find the name and age of the oldest sailor.

```
SELECT s1.sname, s1.age
FROM    s s1
WHERE NOT EXISTS (
  SELECT *
  FROM     s s2
  WHERE s2.age>s1.age)
```

Alternatively:

```
SELECT s1.sname, s1.age
FROM    s s1
WHERE s1.age >= all (
  SELECT age
  FROM    s)
```

12. Find the names of sailors who have reserved every boat reserved by those with a lower rating.

```
SELECT s1.sname
FROM    s s1
WHERE NOT EXISTS (
  SELECT *
  FROM     b, r, s s2
  WHERE  s2.sid=r.sid AND s2.bid=b.bid AND s2.rating<s1.rating AND
              NOT EXISTS (
    SELECT *
    FROM    r r2
    WHERE s1.sid=r2.sid AND r2.bid=b.bid))
```

Alternatively:

```
SELECT s1.sname
FROM    s s1
WHERE NOT EXISTS (
  (SELECT b1.bid
   FROM     b b1, r, s s2
  WHERE  s2.rating<s1.rating AND s2.sid=r.sid)
    EXCEPT
  (SELECT b2.bid
   FROM     b b2, r r2
   WHERE r2.bid=b.bid AND r2.sid=s1.sid))
```

Note: Recall my note above regarding the use of EXISTS/NOT EXISTS in conjunction with EXCEPT. Make sure you understand why in the previous queries we could just SELECT * but in this query, we pay attention to the attributes being selected.

13. For each rating, find the average age of sailors at that level of rating.

```
SELECT  rating, AVG(age)
FROM    s
GROUP BY rating
```

14. For each boat which was reserved by at least 5 distinct sailors, find the boat id and the average age of sailors who reserved it.

```
SELECT  b.bid, AVG(age)
FROM    b, r, s
WHERE  b.bid=r.bid AND r.sid=s.sid
GROUP BY bid
HAVING 5<=COUNT(DISTINCT r.sid)
```

This is equivalent to the following more complex query, which many optimizers may not optimize well and hence the DBMS may take longer to evaluate the following query.

```
SELECT b1.bid, AVG(s.age)
FROM    r, s (SELECT bid
              FROM    b, r r2
              WHERE b.bid=r2.bid
              GROUP BY bid
              HAVING 5<=COUNT(DISTINCT r2.sid)) b1
WHERE b1.bid=r.bid AND r.sid=s.sid
GROUP BY b1.bid
```

15. For each boat which was reserved by at least 5 sailors with age >= 40, find the boat id and the average age of such sailors.

```
SELECT bid, AVG(age)
FROM    b, r, s
WHERE s.age>=40 AND b.bid=r.bid AND s.sid=r.sid
GROUP BY bid
HAVING 5<=COUNT(DISTINCT s.sid)
```

16. For each boat which was reserved by at least 5 sailors with age >= 40, find the boat id and the average age of all sailors who reserved the boat.

```
SELECT b1.bid, AVG(s.age)
FROM    r, s (SELECT bid
                FROM    b, r r2, s s2
                WHERE b.bid=r2.bid AND s2.sid=r.sid AND s.age>=40
                GROUP BY bid
                HAVING 5<=COUNT(DISTINCT r2.sid)) b1
WHERE b1.bid=r.bid AND r.sid=s.sid
GROUP BY b1.bid
```

Unlike the previous query, this query cannot be "flattened", i.e., cannot be expressed without a nested subquery. It is of course possible to express this query with the WITH construct for defining a temporary table. Similarly, it can also be done by defining a view. Practice writing those queries on your own.

SQL exercise 2

Social-Network Query Exercise

Highschooler (ID, name, grade) English: There is a high school student with unique ID and a given firstname in a certain grade.

Friend (ID1, ID2) English: The student with ID1 is friends with the student with ID2. Friendship is mutual, so if (123, 456) is in the Friend table, so is (456, 123).

Likes (ID1, ID2) English: The student with ID1 likes the student with ID2. Liking someone is not necessarilymutual, so if (123, 456) is in the Likes table, there is no guarantee that (456, 123) is also present.

Queries:

1/ Find the names of all students who are friends with someone named Gabriel.

2/ For every student who likes someone 2 or more grades younger than themselves, return that student's nameand grade, and the name and grade of the student they like.

3/ For every pair of students who both like each other, return the name and grade of both students. Include each pair only once, with the two names in alphabetical order.

4/ Find all students who do not appear in the Likes table (as a student who likes or is liked) and return theirnames and grades. Sort by grade, then by name within each grade.

5/ For every situation where student A likes student B, but we have no information about whom B likes (that is,B does not appear as an ID1 in the Likes table), return A and B's names and grades.

6/ Find names and grades of students who only have friends in the same grade. Return the result sorted bygrade, then by name within each grade.

7/ For each student A who likes a student B where the two are not friends, find if they have a friend C incommon (who can introduce them!). For all such trios, return the name and grade of A, B, and C.

8/ Find the difference between the number of students in the school and the number of different first names.

9/ Find the name and grade of all students who are liked by more than one other student.

10/ For every situation where student A likes student B, but student B likes a different student C, return thenames and grades of A, B, and C.

11/ Find those students for whom all of their friends are in different grades from themselves. Return the students' names and grades.

12/ What is the average number of friends per student? (Your result should be just one number.)

13/ Find the number of students who are either friends with Cassandra or are friends of friends of Cassandra. Donot count Cassandra, even though technically she is a friend of a friend.

14/ Find the name and grade of the student(s) with the greatest number of friends.

15/ It's time for the seniors to graduate. Remove all 12th graders from Highschooler.

16/ If two students A and B are friends, and A likes B but not vice-versa, remove the Likes tuple.

17/ For all cases where A is friends with B, and B is friends with C, add a new friendship for the pair A and C. Do not add duplicate friendships, friendships that already exist, or friendships with oneself.

SOLUTION EXERCISE 2

Find the names of all students who are friends with someone named Gabriel.

```
select name
from Highschooler
where ID in (select ID1 from Friend where ID2 in
      (select ID from Highschooler where name = "Gabriel"));
```

For every student who likes someone 2 or more grades younger than themselves, return that student's nameand grade, and the name and grade of the student they like.

```
select h1.name, h1.grade, h2.name,
h2.grade from Highschooler h1,
Highschooler h2, Likes l where h1.ID =
l.ID1
```

For every pair of students who both like each other, return the name and grade of both

```
select h1.name, h1.grade, h2.name, h2.grade
from Highschooler h1, Highschooler h2, Likes l1, Likes l2
where h1.ID = l1.ID1
and h2.ID = l1.ID2
and l1.ID1 = l2.ID2
and l1.ID2 = l2.ID1
and h1.name < h2.name;
```

students. Includeeach pair only once, with the two names in alphabetical order.

Find all students who do not appear in the Likes table (as a student who likes or is liked) and return theirnames and grades. Sort by grade, then by name within each grade.

```
select name, grade
from Highschooler
where ID not in (select distinct ID1 from Likes)
and ID not in (select distinct ID2 from Likes)
order by grade, name;
```

For every situation where student A likes student B, but we have no information about

```
select distinct h1.name, h1.grade, h2.name, h2.grade
from Highschooler h1, Highschooler h2, Likes l1, Likes l2
where h1.ID = l1.ID1
and h2.ID = l1.ID2
and h2.ID not in (select distinct ID1 from Likes);
```

whom B likes (that is,B does not appear as an ID1 in the Likes table), return A and B's names and grades.

Find names and grades of students who only have friends in the same grade. Return the

result sorted bygrade, then by name within each grade.

```
select h1.name, h1.grade
from Highschooler h1, Highschooler h2, Friend f
where h1.ID = f.ID1 and h2.ID = f.ID2
group by h1.name, h1.grade
having count(distinct h2.grade) = 1
order by h1.grade, h1.name;
```

For each student A who likes a student B where the two are not friends, find if they

```
select distinct h1.name, h1.grade, h2.name, h2.grade, h3.name,
h3.grade from Highschooler h1, Highschooler h2, Highschooler
h3, Friend f, Likes lwhere h1.ID = 1.ID1 and h2.ID = 1.ID2
and h2.ID not in (select distinct ID2 from Friend where ID1 = h1.ID)
and h3.ID in (select distinct ID2 from Friend where ID1 = h1.ID
          INTERSECT
              select distinct ID2 from Friend where ID1 = h2.ID);
```

have a friend C incommon (who can introduce them!). For all such trios, return the name and grade of A, B, and C.

Find the difference between the number of students in the school and the number of different first names.

```
select (select count(distinct ID) from Highschooler)
- (select count(distinct name) from Highschooler);
```

Find the name and grade of all students who are liked by more than one other student.

```
select h1.name, h1.grade
from Highschooler h1
where h1.ID in (select distinct ID2 from Likes group by ID2 having count(ID1) > 1);
```

For every situation where student A likes student B, but student B likes a different student C, return the names and grades of A, B, and C.

```
select distinct h1.name, h1.grade, h2.name, h2.grade, h3.name, h3.grade
from Highschooler h1, Highschooler h2, Highschooler h3, Likes l
where h1.ID = l.ID1
and h2.ID = l.ID2
and h1.ID not in (select ID2 from Likes where ID1 = h2.ID)
and h3.ID in (select ID2 from Likes where ID1 = h2.ID);
```

Find those students for whom all of their friends are in different grades from themselves.

```
select h1.name, h1.grade
from Highschooler h1
where h1.grade not in (select h2.grade from Highschooler h2,
Friend f where f.ID1 = h1.ID and f.ID2 = h2.ID);
```

Return the students' names and grades.

What is the average number of friends per student? (Your result should be just one number.)

```
select avg(fr) from
(select count(distinct ID2) as fr from Highschooler, Friend where ID = ID1 group by ID);
```

Find the number of students who are either friends with Cassandra or are friends of friends of Cassandra. Donot count Cassandra, even though technically she is a friend of a friend.

```
select count(*) -1 from (
select distinct f2.ID2 from Friend f2
where f2.ID1 in
(select distinct f1.ID2 as s1 from Friend f1
where f1.ID1 = (select ID from Highschooler where name = "Cassandra"))
union
select distinct f1.ID2 as s1 from Friend f1
where f1.ID1 = (select ID from Highschooler where name = "Cassandra") );
```

Find the name and grade of the student(s) with the greatest number of friends.

```
select h.name, h.grade
from Highschooler h
where ID in
(select ID from
(select ID, count(distinct ID2) as fr from Highschooler, Friend
where ID = ID1 group by ID)
where fr >= (select max(fr)
from
(select ID, count(distinct ID2) as fr from Highschooler, Friend
where ID = ID1 group by ID)));
```

It's time for the seniors to graduate. Remove all 12th graders from Highschooler.

```
delete from Highschooler
where grade = 12;
```

If two students A and B are friends, and A likes B but not vice-versa, remove the Likes tuple.

```
delete from Likes
where ID1 in (select f.ID1 from Friend f where f.ID2 = Likes.ID2)
and ID1 not in (select l.ID2 from Likes l where l.ID1 = Likes.ID2);
```

For all cases where A is friends with B, and B is friends with C, add a new friendship for the pair A and C.Do not add duplicate friendships, friendships that already exist, or friendships with oneself.

```
insert into Friend
select f1.ID1, f2.ID1 from Friend f1, Friend f2
where f1.ID2 = f2.ID2 and f1.ID1 <> f2.ID1
except select * from Friend;
```