Stanford SQL exercise

Rui Wang

3/14/2017

Not the best code, but as a good reference for people are stuck with these problems. Thanks for Standford online course.

Movie-Rating Query Exercise

Movie (mID, title, year, director) English: There is a movie with ID number mID, a title, a release year, and a director.

Reviewer (rID, name) English: The reviewer with ID number rID has a certain name.

Rating (rID, mID, stars, ratingDate) English: The reviewer rID gave the movie mID a number of stars rating (1-5) on a certain ratingDate.

Find the titles of all movies directed by Steven Spielberg.

```sql
select title from Movie
where director = 'Steven Spielberg';
```

Find all years that have a movie that received a rating of 4 or 5, and sort them in increasing order.

```sql
select distinct year from Movie
join Rating on Movie.mID = Rating.mID
where stars = 4 or stars = 5
order by year;
```

Find the titles of all movies that have no ratings.

```sql
select title from Movie
left outer join Rating on Movie.mID = Rating.mID
where stars is null;
```

Some reviewers didn't provide a date with their rating. Find the names of all reviewers who have ratings with a NULL value for the date.

```sql
select name from Rating
left outer join Reviewer
on Reviewer.rID = Rating.rID
where ratingDate is null;
```

Write a query to return the ratings data in a more readable format: reviewer name, movie title, stars, and ratingDate. Also, sort the data, first by reviewer name, then by movie title, and lastly by number of stars.

```sql
select name, title, stars, ratingDate
from Rating join Movie using (mID)
join Reviewer using (rID)
order by name, title, stars;
```

For all cases where the same reviewer rated the same movie twice and gave it a higher rating the second time, return the reviewer's name and the title of the movie.

```sql
select name, title from Rating r1
join Rating r2 on(r1.rID=r2.rID and r1.mID=r2.mID)
join Movie on(Movie.mID = r1.mID)
join Reviewer on(Reviewer.rID = r1.rID)
where r1.stars < r2.stars
and r1.ratingDate < r2.ratingDate
and r1.rID in ( select rID from Rating
                group by rID, mID
                having count(*) = 2);
```

For each movie that has at least one rating, find the highest number of stars that movie received. Return the movie title and number of stars. Sort by movie title.

```
select title, max(stars)
from Rating
join Movie on(Rating.mID = Movie.mID)
join Reviewer on(Rating.rID = Reviewer.rID)
where Rating.mID in
(select mID from Rating group by mID
      having count(stars) >= 1)
group by title
order by title
```

For each movie, return the title and the 'rating spread', that is, the difference between highest and lowest ratings given to that movie. Sort by rating spread from highest to lowest, then by movie title.

```
select title, max(stars) - min(stars) as rating_spread
from Rating
join Movie using(mID)
join Reviewer using(rID)
group by title
order by rating_spread desc;
```

Find the difference between the average rating of movies released before 1980 and the average rating of movies released after 1980. (Make sure to calculate the average rating for each movie, then the average of those averages for movies before 1980 and movies after. Don't just calculate the overall average rating before and after 1980.)

```
select
    (select avg(avg_rating)
    from (select year, avg(stars) as avg_rating
          from Rating
          join Movie using(mID)
          join Reviewer using(rID)
          group by year)
    where year < 1980)
-
    (select avg(avg_rating)
    from (select year, avg(stars) as avg_rating
          from Rating
          join Movie using(mID)
          join Reviewer using(rID)
          group by year)
    where year > 1980);
```

Find the names of all reviewers who rated Gone with the Wind.

```sql
select distinct name
from Rating
join Movie using(mID)
join Reviewer using(rID)
where title = 'Gone with the Wind';
```

For any rating where the reviewer is the same as the director of the movie, return the reviewer name, movie title, and number of stars.

```sql
select distinct name, title, stars
from Rating
join Movie using(mID)
join Reviewer using(rID)
where name = director;
```

Return all reviewer names and movie names together in a single list, alphabetized. (Sorting by the first name of the reviewer and first word in the title is fine; no need for special processing on last names or removing "The".)

```sql
select name from
    (select distinct name
    from Reviewer
    union
    select distinct title
    from Movie)
order by name;
```

Find the titles of all movies not reviewed by Chris Jackson.

```sql
select title from Movie
where mID not in
(select mID from Rating where rID in
      (select rID from Reviewer
       where name = 'Chris Jackson'))
```

For all pairs of reviewers such that both reviewers gave a rating to the same movie, return the names of both reviewers. Eliminate duplicates, don't pair reviewers with themselves, and include each pair only once. For each pair, return the names in the pair in alphabetical order.

```sql
select distinct Re1.name, Re2.name
from Rating R1, Rating R2, Reviewer Re1, Reviewer Re2
where R1.mID = R2.mID
and R1.rID = Re1.rID
and R2.rID = Re2.rID
and Re1.name < Re2.name
order by Re1.name, Re2.name;
```

For each rating that is the lowest (fewest stars) currently in the database, return the reviewer name, movie title, and number of stars.

```sql
select name, title, stars
from Rating
join Movie using(mID)
join Reviewer using(rID)
where stars <= (select min(stars) from Rating);
```

List movie titles and average ratings, from highest-rated to lowest-rated. If two or more movies have the same average rating, list them in alphabetical order.

```sql
select title, avg(stars) as avg
from Rating
join Movie using(mID)
join Reviewer using(rID)
group by title
order by avg desc, title;
```

Find the names of all reviewers who have contributed three or more ratings. (As an extra challenge, try writing the query without HAVING or without COUNT.)

```sql
select name
from Rating
join Movie using(mID)
join Reviewer using(rID)
group by name
having count(mID) >= 3;
```

Some directors directed more than one movie. For all such directors, return the titles of all movies directed by them, along with the director name. Sort by director name, then movie title. (As an extra challenge, try writing the query both with and without COUNT.)

```sql
select title, director from Movie
where director in
  ( select director
    from movie
    group by director
    having count(*) > 1)
order by director, title;
```

Find the movie(s) with the highest average rating. Return the movie title(s) and average rating. (Hint: This query is more difficult to write in SQLite than other systems; you might think of it as finding the highest average rating and then choosing the movie(s) with that average rating.)

```sql
select title, avg from
      (select title,
      avg(stars) as avg
      from Rating
      join Movie using(mID)
      join Reviewer using(rID)
      group by title)
where avg >=
      (select max(avg) from
       (select title,
        avg(stars) as avg
        from Rating
        join Movie using(mID)
        join Reviewer using(rID)
        group by title));
```

Find the movie(s) with the lowest average rating. Return the movie title(s) and average rating. (Hint: This query may be more difficult to write in SQLite than other systems; you might think of it as finding the lowest average rating and then choosing the movie(s) with that average rating.)

```sql
select title, avg
      from
      (select title, avg(stars) as avg
      from Rating
      join Movie using(mID)
      join Reviewer using(rID)
      group by title)
where avg <=
      (select min(avg) from
      (select title,
      avg(stars) as avg
      from Rating
      join Movie using(mID)
      join Reviewer using(rID)
      group by title));
```

For each director, return the director's name together with the title(s) of the movie(s) they directed that received the highest rating among all of their movies, and the value of that rating. Ignore movies whose director is NULL.

```sql
select director, title, max(stars) from Rating
join Movie using(mID)
join Reviewer using(rID)
where director is not null
group by director;
```

Add the reviewer Roger Ebert to your database, with an rID of 209.

```sql
insert into Reviewer (rID, name)
values ('209', 'Roger Ebert');
```

Insert 5-star ratings by James Cameron for all movies in the database. Leave the review date as NULL.

```sql
insert into Rating (rID, mID, stars, ratingDate )
select Reviewer.rID , Movie.mID, 5, null from Movie
left outer join Reviewer
where Reviewer.name='James Cameron';
```

For all movies that have an average rating of 4 stars or higher, add 25 to the release year. (Update the existing tuples; don't insert new tuples.)

```sql
update Movie
set year = year + 25
where mID in
(select mID from
    (select mID, avg(stars) as avg
      from Movie join Rating using(mID)
      group by mID)
where avg >= 4);
```

Remove all ratings where the movie's year is before 1970 or after 2000, and the rating is fewer than 4 stars.

```sql
delete from rating
where mID in (select mID from movie where year < 1970 or year > 2000)
and stars < 4;
```

Social-Network Query Exercise

Highschooler (ID, name, grade) English: There is a high school student with unique ID and a given first name 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 necessarily mutual, so if (123, 456) is in the Likes table, there is no guarantee that (456, 123) is also present.

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

```sql
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 name and grade, and the name and grade of the student they like.

```sql
select h1.name, h1.grade, h2.name, h2.grade
from Highschooler h1, Highschooler h2, Likes l
where h1.ID = l.ID1
and h2.ID = l.ID2
and h1.grade >= h2.grade + 2
```

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.

```sql
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;
```

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

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

```sql
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);
```

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

```sql
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 have a friend C in common (who can introduce them!). For all such trios, return the name and grade 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, Friend f, Likes l
where h1.ID = l.ID1 and h2.ID = l.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);
```

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. Return the students' names and grades.

```
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);
```

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. Do not 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.

```sql
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.

```sql
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.

```sql
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. (This one is a bit challenging; congratulations if you get it right.)

```sql
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;
```