CS1555 Recitation 8a

Objective:

1. To practice Views
2. MT Q&A
3. Discuss some queries from HW4
4. Sample questions

PART 1:

Before we start:

- Download the SQL script studentdb.sql through an sFTP client (such as FileZilla) from the machine "class3.cs.pitt.edu" at the directory:
    - `/afs/pitt.edu/home/r/a/raa88/public/studentdb.sql`

1. Create a view called student_courses that lists the SIDs, student names, number of courses in the Course_taken table.

```
create or replace view student_courses as
select s.sid, s.name, count(course_no) as num_courses
from student s, course_taken ct
where s.sid = ct.sid
group by s.sid, s.name;
```

2. Create a materialized view called mv_student_courses that lists the SIDs, student names, number of courses in the Course_taken table.

```
drop materialized view if exists mv_student_courses;
create materialized view mv_student_courses as
select s.sid, s.name, count(course_no) as num_courses
from student s, course_taken ct
where s.sid = ct.sid
group by s.sid, s.name;
```

3. Execute the following commands. Compare the query results and time used of the two select statements.

```
insert into course_taken (course_no, sid, term, grade)
values ('CS1555', '129','Fall 19', null);

--REFRESH MATERIALIZED VIEW mv_student_courses;
select * from mv_student_courses;
select * from student_courses;

refresh materialized view mv_student_courses;

select * from mv_student_courses;
select * from student_courses;
```

- The result from the materialized view is incorrect because the materialized view was not refreshed after the insert statement.
- The result from the view is correct because what a normal view does is rewriting the query. It does not store a snapshot of the query result like the materialized view.
- The running time of the materialized view is shorter, because it does not need to rewrite the query and run the rewritten query on the original course_taken table.

PART 2: Discuss some queries from HW4

---

1. List the user id of users who have sent friend requests to other users, as 'sender_1', along with the user id who reciprocated the friend request as 'sender_2'. Note that a user who has not gotten a friend request back, should have NULL in the 'sender_2' attribute.

```
SELECT s.from_id AS sender_1, R.from_id AS sender_2
FROM PENDING_FRIEND S
LEFT JOIN PENDING_FRIEND R ON S.to_id = R.from_id
                                        AND S.from_id = R.to_id;
```

2. Display the average number of characters in messages sent by users as 'avg_characters', along with their user id and a category.
There are 3 categories:
i) Long Messages (the average is over 30 characters);
ii) Short Messages (the average is less than 30 characters);
and iii) Empty Messages (the average is NULL).
Hint: You can use the function LENGTH() that returns the number of characters of the input string.

```
SELECT from_id,
       AVG(LENGTH(message)) as avg_characters,
       CASE
           WHEN AVG(LENGTH(message)) >= 30 THEN 'Long Messages'
           WHEN AVG(LENGTH(message)) IS NULL THEN 'Empty Messages'
           ELSE 'Short Messages'
           END              as category
FROM MESSAGE_INFO
GROUP BY from_id
ORDER BY avg_characters;
```

3. Display the name and number of group memberships of users, who have the highest number of group memberships.

```
SELECT name, count(G.group_id)
FROM GROUP_MEMBER G
         NATURAL JOIN PROFILE P
GROUP BY name, user_id
having count(G.group_id) >=
       (SELECT max(MG.group_count)
        FROM (SELECT name, count(G.group_id) as group_count
              FROM GROUP_MEMBER G
                       NATURAL JOIN PROFILE P
              GROUP BY name, user_id) as MG);
```