**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;

1. 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;

1. 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);