Lab 3 : Aggregate Queries
CMSC 362
Marmorstein
Spring 2014
Due Friday, 2/28/2014

So far, we have worked with simple SQL queries. In this lab, you will work with queries that use aggregate functions and sorting.

Step 1. Setting Up

We are going to use the rant database you created in Lab 2. *Make a copy of your Lab2 folder named Lab3. In that folder, rename lab2.sql to lab3.sql*. As before, you can use the command:

```
psql -h torvalds -U lastName -d lastName -f lab3.sql
```

to connect to the database and execute the script.

Edit lab3.sql and change the comment to reflect that this is Lab 3 instead of Lab 2.

Step 2. Ratings

It would be nice if, in addition to commenting on a post, users could give each rant a numerical score between 1 and 10, where 1 means "disagree completely" and 10 means "agree completely.

Modify lab3.sql so that it creates a new table called "ratings". The table should have the following attributes: (post_id, username, and rating).

The post_id identifies the post which is being rated. The username identifies the user who is changing the rating. The rating is a numerical value between 1 and 10.

You should choose appropriate types for these attributes (the post_id and username should be the same type as the corresponding field in the account and rant tables). The rating should be an integer with a default value of 5.

Step 3. More Constraints

In order to protect the database from data corruption, it is useful to add constraints to the tables. In particular, it is useful to add a PRIMARY KEY constraint. The primary key constraint ensures data in a column (or columns) is unique and non-null. To add a primary key constraint, simply place the words PRIMARY KEY after the attribute type, but before the comma in the create table definition. You can also use a combination of attributes as a primary key. To do this, you simply add the words PRIMARY KEY after both of them.

Make username a primary key of the accounts table. Make *post_id* a primary key of the rants table. Make the combination *post_id, comment_id* a primary key of the comments table. Finally, make the combination *post_id, username* a primary key of the ratings table you created in step 2.

We also want the rating to be between 1 and 10 (inclusive). We can do this with a CHECK constraint. To add a check constraint, *type "CONSTRAINT rating_range CHECK (rating >=1 AND rating <=10)" after the type, but before the comma, of the rating attribute.*

Step 4. Aggregate Queries

In class, we talked about the SUM, COUNT, MIN, and MAX functions. Use these functions to implement the following queries in lab3.sql:

A. A query which gives the total number of users in the accounts database.

B. A query which gives the average rating of each post. You will need to use the **GROUP BY** keyword to group the query results by post_id.

C. Another query which gives the average rating after dropping the highest and lowest values. Since the average function doesn't drop highest and lowest, you hae a couple choices for implementing this query. You can use the WHERE clause and a subselect or you can calculate this value by summing the ratings, subtracting the min and the max values and dividing the result by the count minus 2.

D. A query which displays the username, rant, and time_posted of each post and tells how many replies (comments) have been posted in response to that post. You will need to "join" the comments and rants tables to perform the query. You will also need to use the **GROUP BY** keyword to make sure that the count is correct.

E. A query which displays each rant sorted by the name of the user who posted it.

Step 5. Submitting

As usual, save your work and upload your lab3.sql file to
http://narnia.homeunix.com/~robert/submit