CS 386/586 Week 1 Activities with Suggested Answers and Comments    January 9, 2013

Part 1:

I have predefined accounts for each student in 386/586 (with an associated database) on the postgreql installation at PSU MCECS. Your userid (and your database name) has the form w13db? where w13 stands for Winter 2013 and the ? is a number between 4 and 50.

We passed out the account information to all students who attended class on January 9, 2013.

If you do not yet have your account, please e-mail the instructor (lmd with the normal cs department e-mail address so that she can tell you your account name and password.)

Use your postgresql database:

Hint: it is easiest in postgresql if you use lowercase names for your tables and attributes.

This is because when you type an SQL statement into the SQL window, it translates your tablenames and attribute names to lowercase (automatically), so if you name your table something like Employee, then when you enter a query like SELECT * FROM Employee, postgresql will tell you table “employee” not found because there is no table named employee. If you do have capital letters in your table name, then you need to surround it by double quotes like this: SELECT * FROM “Employee” (or select * from “Employee”). Thus, it is simpler if you use lowercase names. The same is true of attributes.

1. Go to dbclass.cs.pdx.edu
   Click on “dbclass” on the left menu to be prompted to log in. Log in with your username and password.

2. Click on the Account tab and change your password. Note: as soon as you do, you will have to log in again the next time you want to do something on this site.

3. Expand the navigation bar on the left side; choose a database name, click on public in the main panel and you’ll see a list of tables that have been defined.

   For example, click on introdb_spy. You’ll see 13 tables. Try clicking on the table names – to see which attributes are defined for the tables. Check out the agent table and the skill table. Look at the mission table and consider the team_id attribute. Would you expect that to match an attribute in the team table? Which one? (Answer: it’s the team_id attribute in the team table.)

4. Within your db (say w13db21 – if that’s yours), create a new schema called company. Define a table called employee with a few attributes including:

emp_id – integer – where emp_id is the primary key for the table,
name – character of varying length with a maximum length of, say, 40
gender – character of length 1

Insert a few rows into the table using the forms available in phppgadmin.

5. Create another table in the company database called position with two attributes: id (the position id), and title (the position title) but do so using the SQL create table statement. Click on the SQL button in the upper right; type your SQL statement into the main window on the screen.

   Add a few rows to the position table but do so using an SQL insert statement. The insert statement should be typed into the SQL window – just like the create table statement.

   Try to violate the primary key by attempting to put in two positions with the same id.

6. Alter the employee table by adding an attribute called job (which should hold an id from the position table). Make sure that this job attribute has the same data type as the id attribute in the position table.

   You can alter a table by clicking "alter" on the phppgadmin interface. Or you can use an alter table SQL statement in the SQL window. You can look up the syntax for the ALTER TABLE statement online. Or, here's an example:

   alter table employee add job integer

   Once you add the attribute, you can edit the individual rows in your table to give them a value for the job attribute.

   Or, you can use the SQL update statement to do so. You can look up the syntax online. Here's a simple example:

   UPDATE employee SET job = 1 WHERE emp_id = 101

   This would work if you have an employee in your database whose emp_id is 101.

Part 2: queries

7. Write all of the following queries in relational algebra.
   a. List all rows from the employee table

      select *
      from employee

   b. List all rows from the employee table that have a job of 7 (or choose some other job value – one that you have in your position table and that DOES appear in your employee table).

      select *
      from employee
      where job = 7

   c. List all rows from the employee table that have a job of 99 (or chose some other job value – one that you DO NOT have in your position table but that DOES appear in your employee table).

      select *
      from employee
      where job = 99

      (I put parts b. and c. here because I want you to see that so far, based on the way the tables have been defined, there is no requirement and (thus no enforcement) of the idea that the value for the job attribute in the employee table MUST be in the position table. We probably want that constraint; we'll show later how to put the constraint in the schema so that it is enforced.

   d. List all rows from the employee table but do NOT include the emp_id attribute in your list.

      select name, gender, job
      from employee

   e. List all possible pairs of two employee ids.

      select e1.emp_id, e1.name, e1.gender, e1.job, e2.emp_id, e2.name, e2.gender, e2.job
      from employee e1, employee e2

f. List all possible pairs of two employee ids but make sure that the two employee ids are different. (This would list all possible two-person teams from your company.)

```
select e1.emp_id, e1.name, e1.gender, e1.job, e2.emp_id, e2.name,
e2.gender, e2.job
from employee e1, employee e2
where e1.emp_id != e2.emp_id
```

g. List all possible pairs of two employee ids but make sure that the two employee ids are different and make sure that the two employees have different genders.

```
select e1.emp_id, e1.name, e1.gender, e1.job, e2.emp_id, e2.name,
e2.gender, e2.job
from employee e1, employee e2
where e1.emp_id != e2.emp_id and e1.gender != e2.gender
```

h. Modify either of your two previous queries so that no pair of employees is listed twice.

```
select e1.emp_id, e1.name, e1.gender, e1.job, e2.emp_id, e2.name,
e2.gender, e2.job
from employee e1, employee e2
where e1.emp_id < e2.emp_id and e1.gender != e2.gender
```

or

```
select e1.emp_id, e1.name, e1.gender, e1.job, e2.emp_id, e2.name,
e2.gender, e2.job
from employee e1, employee e2
where e1.emp_id > e2.emp_id and e1.gender != e2.gender
```

The choice between these two depends on whether or not you want the lower or the higher emp_id listed first in the pair.

i. List all employee ids (with the names of the employees) along with their job and their job title. (Hint: use a cross product of employee and position followed by a select operator to find the matches.)

```
select emp_id, name, job, title
from employee, position
where job = id
```

8. Using the SQL window in phppgadmin, write the queries above SQL and run them in your company DB.

    a. List all rows from the employee table

       employee

       This – a relation name by itself – simply returns the content of the relation. So this is the simplest and the best way to write this query in relational algebra.

       You can also do it this way (although the operator shown is unnecessary and thus less efficient):

       $\sigma_{7=7}\ employee$

       or

       $\sigma_{true}\ employee$

       $\pi_{emp\_id,name,gender,job} employee$

    b. List all rows from the employee table that have a job of 7 (or choose some other job value – one that you have in your position table and that DOES appear in your employee table).

       $\sigma_{job=7}\ employee$

    c. List all rows from the employee table that have a job of 99 (or chose some other job value – one that you DO NOT have in your position table but that DOES appear in your employee table).

       $\sigma_{job=99}\ employee$

       (I put parts b. and c. here because I want you to see that so far, based on the way the tables have

    d. List all rows from the employee table but do NOT include the emp_id attribute in your list.

       $\pi_{name,gender,job} employee$

e. List all possible pairs of two employee ids.

We can use the rename operator, described in the book to provide a different name for a relation or for any of its attributes. Here, I rename the employee relation (and its attributes) twice. You can think of it as if you get a copy of the employee table in emp1 and a separate copy of the employee table in emp2.

$$\pi_{empid1,empid2}(\rho_{emp1(empid1,name1,gender1,job1}employee \\ \times \rho_{emp2(empid2,name2,gender2,job2}employee)$$

f. List all possible pairs of two employee ids but make sure that the two employee ids are different. (This would list all possible two-person teams from your company.)

$$\sigma_{empid1\ !=empid2}(\pi_{empid1,empid2}(\rho_{emp1(empid1,name1,gender1,job1}employee \\ \times \rho_{emp2(empid2,name2,gender2,job2}employee))$$

g. List all possible pairs of two employee ids but make sure that the two employee ids are different and make sure that the two employees have different genders.

$$\sigma_{empid1\ !=empid2\ AND\ gender1\ !=gender2} \\ (\pi_{empid1,empid2}(\rho_{emp1(empid1,name1,gender1,job1}employee \\ \times \rho_{emp2(empid2,name2,gender2,job2}employee))$$

h. Modify either of your two previous queries so that no pair of employees is listed twice.

$$\sigma_{empid1< empid2\ AND\ gender1\ !=gender2} \\ (\pi_{empid1,empid2}(\rho_{emp1(empid1,name1,gender1,job1}employee \\ \times \rho_{emp2(empid2,name2,gender2,job2}employee))$$

or

$$\sigma_{empid1 > empid2\ AND\ gender1\ !=gender2} \\ (\pi_{empid1,empid2}(\rho_{emp1(empid1,name1,gender1,job1}employee \\ \times \rho_{emp2(empid2,name2,gender2,job2}employee))$$

The choice between these two depends on whether or not you want the lower or the higher emp_id listed first in the pair.

i. List all employee ids (with the names of the employees) along with their job and their job title. (Hint: use a cross product of employee and position followed by a select operator to find the matches.)

$\sigma_{job=id}(employee \times position)$

9. Try out the union, intersect, and except operator in SQL by writing an SQL query that is equivalent to the following relational algebra queries:

   a. $(\sigma_{name='Susan'} employee) \cup (\sigma_{name='John'} employee)$

      select *
      from employee
      where name = 'Susan'
      union
      select *
      from employee
      where name = 'John'

      This could have been written more simply (without using the union operator) like this:

      select *
      from employee
      where name = 'Susan' or name = 'John'

   b. $(\sigma_{name='Susan'} employee) \cap (\sigma_{name='John'} employee)$

      select *
      from employee
      where name = 'Susan'
      intersect
      select *
      from employee
      where name = 'John'

      Will your answer always be empty – no matter how many rows are in the DB? Why or why not?

      This answer will necessarily, always, be empty because each employee has just one name. Each attribute value in a relation is one, simple, atomic value – for all attributes.

      Whenever a query is necessarily empty (regardless of what it is in the DB instance, we say that the query is unsatisfiable. You should never run

queries that are unsatisfiable; there is not need to run them; the answer is always empty.
Note: you could have written the query this way (and this one is also unsatisfiable for the same reason):

```
select *
from employee
where name = 'Susan' and name = 'John'
```

c. $(\sigma_{gender='m'} employee) - (\sigma_{name='John'} employee)$

```
select *
from employee
where gender = 'm'
except
select *
from employee
where name = 'John'
```

This could have been done more simply (but wouldn't have introduced you to the syntax for except) like this:

```
select *
from employee
where gender = 'm' and name != 'John'
```

10. For each of the three queries shown just above in item 9., explain what they do in English.
    a. This query finds all employees whose name is either Susan or John. The query lists all of the attributes (the emp_id, name, gender, and job) for such employees.

    b. This query finds all employees whose name is Susan and whose name is John. (This is impossible; the name can only have one value; that's why this query is unsatisfiable.) The query lists all of the attributes (the emp_id, name, gender, and job) for such employees.

    c. This query finds all employees that are male provided their name is not John. The query lists all of the attributes (the emp_id, name, gender, and job) for such employees.

11. Write a query in relational algebra and also in SQL against the Spy database that lists names that are used for missions or for teams.

In relational algebra:

$\pi_{name} team) \ \cup (\pi_{name} mission)$

In SQL:

```
select name
from team
union
select name
from mission
```

Notice that for this query (in relational algebra and in SQL), I must do the project first so that the two relations involved in the union are union-compatible. Union-compatible is when the two relations have the same number of attributes (one, in this case), corresponding attributes have the same name, and corresponding attributes are defined on the same domain (character varying, in this case).

Note: if the corresponding attributes in the two relations that you would like to take the union, intersection, or set difference of don't have the same name, use the rename operator to change the names so that they make the two relations union-compatible.

Note: in SQL, the two tables must have the same number of attributes and corresponding attributes must be defined on the same (or compatible) domains. But the corresponding attributes do NOT have to have the same name. SQL uses the names from the first table (from the answer to the first query) as the attribute names in the query answer.

Note: in English, this query lists all names that have been used for either a mission or a team.

Note: this query CANNOT be written without a union operator.

Part 3: Piazza: for questions, answers, announcements, and for turning in assignments.

1. Visit piazza.com and register for the CS386/586 class. Type your full name so that I can identify you in Piazza.

2. Post a question, edit an existing question, provide a student answer for a question, or edit a student answer.

   Note that we have predefined folders as follows:
   assign1 assign2 assign3 assign4 assign5 assign6
   turn-in-assign-here1 turn-in-assign-here2 turn-in-assign-here3
   turn-in-assign-here4 turn-in-assign-here5 turn-in-assign-here6
   test1 test2
   demo1 demo2 demo3 demo4 demo5 demo6
   activity1 activity2 activity3 activity4
   activity5 activity6 activity7 activity8
   sql
   rel-algebra
   plus there is one more called other and I created one for tonight called fake-question

   Note: posts will have "Instr" if it has been posted by me or one of the TAs.

   Note: the tag pin causes the question or note to stay at the top of the list – regardless of the date when it was posted.

3. Click on a folder name to see all of the questions in that folder.

4. Note: I will delete all of the informal, test messages that get posted during this exercise. Instructors can delete posts; students are not able to delete posts.

5. Try creating a follow-up comment on a question. (or a follow-up to a follow-up)

6. Post a message to the instructors; they are the only ones who can see your question. Attach a file to your post. You must post a message to the instructors, placed in the appropriate "turn-in-assignment" folder. Make SURE it's sent to "instructors"! Note: you can't delete questions or attachments. So, please submit it when you're ready. (Or you'll have to let us know that you need an earlier submission deleted – by hand, by an instructor.)

7. Try the preformatted tag when you enter a question.

8. Click in the upper right corner (to the right of your username) on the triangle symbol; choose Account Settings to set your e-mail preferences.