

**Lab 2 : Implementing "This database class @#\$&#\$\$@" (and other useless rants)**  
**CMSC 362**  
**Marmorstein**  
**Spring 2009**  
**Due Monday (2/2/2009) by 5:00pm**

The purpose of this lab is to reinforce the commands we learned in Lab 1 for building tables in SQL and to practice using SQL to build tables. We will do this by implementing a simple "rant" database which could serve as the backend to a blog or other web site.

**Step 1. Setting Up**

We are going to again use the postgresql database server on highlander. You should use the same account and login information. Remember, to connect to the server you can type:

```
psql -h highlander -U lastName -d lastName
```

To execute a database script on the server, you can use:

```
psql -h highlander -U lastName -d lastName -f scriptName.sql
```

*Create a folder named Lab2. In that folder create a file named "lab2.sql". All of your sql code will go in this file. At the top of your file type two dashes followed by a space and then your name, the text "CMSC 362" and the words "Lab 2".*

**Step 2. Some New Types**

Last time we looked at the integer, character, varchar, and some other types. In this lab, I would like to introduce you to some new types:

<b>NUMERIC</b>	<b>: stores fractional (floating point) values without rounding.</b>
<b>BIGINT</b>	<b>: stores a 64-bit signed integer.</b>
<b>SERIAL</b>	<b>: stores an integer value which has a default value that automatically increments with each new record.</b>
<b>TEXT</b>	<b>: stores a block of text of arbitrary length.</b>
<b>BYTEA</b>	<b>: stores binary data such as a picture.</b>
<b>BLOB</b>	<b>: similar to BYTEA, but part of the SQL Standard (Blob stands for "Binary Large Object").</b>
<b>BOOLEAN</b>	<b>: stores a single true or false value</b>
<b>POINT</b>	<b>: stores an (x,y) pair</b>
<b>BOX</b>	<b>: stores a pair of points (representing corners of a box)</b>

*In your lab2.sql file, add SQL code to perform the following operations:*

*Create a table named "rants". Your table should have the fields "username", "rant", "post\_id" (which should be a SERIAL) and "time\_posted". Use appropriate types selected from those in the list above and/or Lab1.*

*Create another table named "accounts". It should have two fields "username" and "password".*

*Create a third table named "comments" which has the fields "username", "post\_id", "comment\_id", and "comment\_text".*

### Step 3. Dropping Tables

Last time we deleted tables using the DROP TABLE command. This command generates an error if the table does not exist. A cleaner way to drop the table is to use the DROP TABLE IF EXISTS comand:

```
DROP TABLE IF EXISTS <tableName>;
```

*At the top of your lab2.sql file add SQL code to drop all three of your new tables if they already exist. Save your work and test, by running the .sql file with psql, that all three tables can be dropped and recreated.*

### Step 4. Calling SQL functions

Like most modern programming languages, SQL allows us to create and use functions. We will worry about creating our own functions some other day. For today, we are just going to use some built-in functions that SQL already provides for us.

In particular, we are going to use the "crypt" function. This function takes a string of text and uses one of several cryptographic hash functions to create a digest. This is useful, because we don't really want to store password in plain text. Even in the database, they would be vulnerable if our system is compromised.

To create an account in our rant database, we will use the insert command we learned in Lab 1. To insert the username "frank" with encrypted password "dumbPassword", you would type

```
INSERT INTO accounts (username, password) VALUES ('frank',  
crypt('dumbPassword', gen_salt('md5')) );
```

*Add this command to the bottom of your lab2.sql file now. Then add accounts for "tom", "harry", and "celeste" with encrypted passwords "fish1", "fish2", and "fish3" respectively.*

### Step 5. Constraints

Users are notoriously good at doing stupid things that violate the security of their accounts. In particular, they often forget to set a password or set a blank password. Let's add a constraint to the database that prevents the password field from ever being set to NULL.

The easy way to do this is to change the CREATE TABLE command for the account table.

Change the line

```
password    VARCHAR(8),
```

to read:

```
password    VARCHAR(8)    NOT NULL,
```

(Of course, if you implemented the password field differently, that's fine. Just add NOT NULL to the end before the comma).

To test, use the interactive mode of psql to try to insert a tuple with only a username.

## **Step 6. Glitter**

Populate your rant database with interesting and useful content using "INSERT INTO" statements at the bottom of your file. The more comments you have and the more interesting they are, the more points you will earn for glitter. Don't forget that in addition to the rants themselves, users can add comments to a rant.

### **Submitting:**

As usual, save your work and upload your lab2.psql file to  
<http://narnia.homeunix.com/~robert/submit> by 5:00pm on Monday.