Database and SQL Review

Web Resources

- SQL basics: https://www.w3schools.com/sql/
- Creating databases: https://www.postgresql.org/docs/9.3/sql-createdatabase.html
- Dropping databases: https://www.postgresql.org/docs/9.3/sql-dropdatabase.html
- Creating Tables: https://www.postgresql.org/docs/9.3/sql-createtable.html
- Dropping Tables: https://www.postgresql.org/docs/9.3/sql-droptable.html
- Granting permissions: https://www.postgresql.org/docs/9.3/sql-grant.html
- Meta-commands: https://www.postgresql.org/docs/9.3/app-psql.html

SQL (Structured Query Language)

- Allows you to create and delete tables
- Four basic things you can do to an existing table CRUD
  - Create: `INSERT` statement
  - Retrieve: `SELECT` statement
  - Update: `UPDATE` statement
  - Destroy: `DELETE` statement
- You can use clauses to narrow/format your result sets or the records to retrieve/modify

Creating Tables in a Database

- The command is as follows:

```
CREATE TABLE movies(
    movie_num INTEGER,
    title CHAR(35) NOT NULL,
    actor INTEGER,
    year INTEGER
    );
```

- This will create a table named movies with four columns, one of which will not support empty values (title)
- To delete this table the syntax is: DROP TABLE movies;

SQL Data Types

- Numeric
  - Integers of various ranges: INTEGER (or INT), SMALLINT
  - Real numbers of various precision: FLOAT, REAL, DOUBLE PRECISION,
  - or the preferred: NUMERIC(p, s) where p = precision and s = scale (the number before and after the decimal)
- Character Strings
  - Fixed length n: CHAR(n) or CHARACTER(n)
  - Variable length of maximum n: VARCHAR(n) or CHAR VARYING(n) (default n =1)
- Date/Time
  - Date: contains only date info
    - N.B. PostGreSQL on the opentech server takes the Date as a string with the format of 'YYYY-MM-DD' (e.g. '1989-10-23')
  - Time: contains only time info
  - Timestamp: contains both date and time information

For detailed list see:
http://www.postgresql.org/docs/7.3/interactive/datatype.html#DATATYPE-FLOAT

SQL Data Qualifiers

- When you set up (i.e. CREATE) your tables you can set conditions for the fields in your records:
  - PRIMARY KEY
    - Makes the field in the record required and has the extra condition that the field must be unique in the table (only one record can have a certain value for the field)
    - A DB table can only have one PRIMARY KEY
  - NOT NULL
    - Makes the field in the record mandatory (i.e. you cannot create records that do not have the field present)
    - Different from PRIMARY KEY as multiple records can have the same value

SELECT SQL Statements

- SELECT statements work on existing records in a database
- Example:

```
SELECT * FROM movies;
```

- For readability, should be more specific

```
SELECT movie_num, title, actor, year FROM movies;
```

SELECT Statement Qualifiers

- You can narrow down your results by using various qualifiers
  - `WHERE` *`column_name`* `LOGIC_OPERATOR` *`value`*
- The logic operators are the same as programming:
  - `<>, >=, =, <=, <, >`
    - N.B. single equal sign is the logic comparator
    - And <> to check inequaliy (not the !=)

SELECT Statement Sorting

- You can sort your select result set with the “ORDER BY” clause
  - ORDER BY *column_name directional_qualifier*
- The directional qualifier can be:
  - ASC for ascending (default)
  - DESC for descending

SELECT Statement Aliases

- For joining multiple tables in a single query it is sometimes easier (clearer) to give table names aliases:
- E.g.:

```
SELECT movies.title, movies.year, actors.name
FROM movies, actors
WHERE movies.actor = actors.id
ORDER BY movies.year ASC
```

- Could be:

```
SELECT m.title, m.year, a.name
FROM movies m, actors a
WHERE m.actor = a.id
ORDER BY m.year ASC
```

- In this case, `a` and `m` become aliases for the tables actors and movies respectively

Insert SQL Statements

- INSERT statements create records in a database
- Example:

```
INSERT INTO movies VALUES(21, 'Casino
  Royale', 'Daniel Craig', 2006);
```

- Should be more specific

```
INSERT INTO movies (movie_num, title, actor,
year) VALUES(21, 'Casino Royale', 'Daniel
Craig', 2006);
```

- N.B. strings are delimited by single quotes ('), not double quotes

Update SQL Statements

- UPDATE statements modify existing records in a database, and uses same clauses as SELECT statements
- Example:

```
UPDATE movies SET year = 2003
  WHERE title = 'Die Another Day';
```

- Be aware you can update multiple records with one UPDATE command (if not careful)

DELETE SQL Statements

- DELETE statements remove existing records in a database, and uses same clauses as SELECT statements
- Example:

```
DELETE FROM movies WHERE title = 'Die Another Day'
```

- Be aware you can DELETE multiple records with one DELETE command (if not careful)
- e.g.

DELETE FROM movies;

  - Removes all records from the movies table (but does not remove the table, you must use a DROP statement to delete)

SQL Comments

- SQL scripts support comments in two formats:
  - Single line comments start with -- (two hyphens)

```
-- this is a single line SQL comment
```

  - Multi-line comments are the same as c-style:

```
/*
     This is a  multi-line
     SQL comment
*/
```

Creating a New PostGreSQL DB

- For this course (WEBD2201) you will not need to create your own database (it is created for you)
- From the command line the command following command was executed:

```
createdb userid_db
```

  - Where userid is your user id (i.e. pufferd)
  - **When the database was successfully created the system displays a "`CREATE DATABASE`" message**
- Alternately, you can run:

```
CREATE DATABASE userid_db
```

from inside a database

Creating a New PostGreSQL DB

- Now that a database exists for you, and you have been given ownership, when you log onto the server with PuTTy, if you type: `psql userid_db`
  - You will be prompted for your password
  - After entering it, the system will take you into your database, where you can perform SQL commands (prompt will be => instead of the # or $ OS prompt)
- To change your password at the sql prompt => type:

```
ALTER USER your_user_id WITH ENCRYPTED PASSWORD 'your_new_password';
```

Removing a Database

- If you create a database (misnamed or unwanted), to remove it type:

  `dropdb nogood_db`

  where `nogood_db` is the unused/unwanted database

- **NOTE: do NOT execute this command on your `lastnamefirstinitial_db` database, you are able to remove your db, but your user does not have permission to create a new one**

Running SQL Script from the command line

- For large scale applications, it is advantageous to set up “build” scripts that drop/create databases as required
- Easier to implement changes than doing everything manually
- At the command prompt type:

```
psql –d userid_db –f script_file.sql
```

  - Where `userid_db` is the database to be modified and `script_file.sql` is the SQL file with the commands to be executed (must be co-located in the current directory, or else fully qualify the address)

Allowing Data Access to Users

- When a *.sql script is run by a PostgreSQL user, the user owns the table and the data in the table
- If a different user needs to access the data they must be given permission explicitly
  - E.g. in this class your instructor needs to see your tables/data
- The command to given certain access levels on your table to anpther user is: `GRANT`

GRANTing Privileges

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
         | ALL TABLES IN SCHEMA schema_name [, ...] }
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

- For this course add the following line in your SQL build script

```
GRANT ALL ON table_name TO faculty;
```

- Example:

```
DROP TABLE actors;
CREATE TABLE actors(
     id INTEGER,
     name CHAR(20)
);
GRANT ALL ON actors TO faculty;
INSERT INTO actors(id, name) VALUES…
```

PostGreSQL Meta-commands

- There are several commands that are defined for PostGreSQL that allow users some short-cuts to administer databases or runs scripts
- Some common and useful ones that can be executed, at the PostGreSQL => prompt type:
  - \q will quit or exit the database, takes you back to the OS
  - \d will “dump” (quickly preview) the database’s content
  - \d *table_name* will dump a specific table’s info
  - \i *db_script.sql* allows you to run a script from outside the db prompt

    NOTE: this will default go to the directory the user was in when they connected to the database, to use a file from a different directory the file path must be fully qualified:

    \i */var/www/users/webd2201/user_id/sql/db_script.sql*