PostgreSQL notes

Note:

- These commands are based off the lectures from the Udemy Course: **The Complete Web Developer Zero To Mastery**

Commands

- **Starter Commands**
    - Start postgreSQL using **brew**
        - `brew services start postgresql`
        - `brew services stop postgresql`
        - `brew services restart postgresql`
- Create a db
    - `createdb db_name`
- Connect to db createdb
    - `psql 'db_name'`
    - How to *describe the database*
        - `\d`
        - `\d table_name`
- Exit and go back to terminal
    - `\q`

*** SQL COMMANDS**

- **Create Table**
- `CREATE TABLE table_name(column_1 datatype, column_2 datatype)`
    - *ex:*
        - `CREATE TABLE users(name text, age smallint, year, date);
- **INSERT INTO && SELECT**
- `INSERT INTO table_name (column_1, column_2, column_3 ) VALUES (value_1, value_2, value_3);`
    - *ex:*
    - `INSERT INTO users (name, age, birthday)`
    - `VALUES ('james', 31, 'YYYY-MM-DD')`
- Note: use single quotes or else errors
- Also do no t ave to keep re-writing the **(column-1, columN-2, column_3)**

- **SELECT**
    - `SELECT * from <table>`
    - `SELECT name, age, birthday FROM users;`
    -
- **ALTER TABLE && UPDATE**
- **ALTER TABLE**
    - We can add columns to a table already createdb
        - `ALTER TABLE users ADD score smallint;`
            - This creates that new column in the *table users*
- **UPDATE**
    - `UPDATE table_name`
    - `UPDATE users SET some_column = some_value;`
    - `UPDATE users SET score=50 WHERE name='James';`
    - `UPDATE users SET score=0 WHERE score IS NULL;`
    - `UPDATE users set score=100 WHERE score IS NOT NULL;`
- **Conditional Statements**
- > What if li wanted to grab all users starting with name 'a'
    - `SELECT * from users WHERE name LIKE 'a%'`
        - grab users where name starts with a `%` is a wildcard regex search
    - `SELECT * from users wherer NAME LIKE '%y'
        - Name ends wiith y1
- `SELECT * FROM users ORDER BY score DESC;`
- `SELECT * FROM users ORDER by score ASC;`
- **SQL FUNCTIONS**
    - `SELECT * FROM users;`
    - > What if we wanted to get the average scores of the users?
        - We can use `AVG` FUNCTIONS**
            - `SELECT AVG(score) FROM users;`
    - > What if we wanted the sum of ages
    - `SELECT SUM(age) FROM users;
    - `SELECT COUNT(name) FROM users;
    - `SELECT COUNT(*) FROM users;`
- **JOINING TABLES**

    - > Primary keys and foreign keys link tables between one another > How dow e connect tables?

- `CREAtE TABLE login (`
- `ID serial NOT NULL PRIMARY KEY,`
- `secret VARCHAR(100) NoT NULL,`
- `name text UNIQUE NOT NULL`
- `);`
- > Datatype `serial` is an autoincrementing value: a unique value suitable to be the primary key
- Assume two tables exist **users** and **login**
- Where **login** holds the credentials of the users and has a foreign key where users.name === login.name *for some value*
- Join Tables through this **relationship**
- `SELECT * FROM users JOIN login ON users.name =login.name;`
- DELETE FROM + DROP TABLE*
- > What if we wanted too delete something?
- > Let's delete SALLY
- `DELETE FROM users WHERE name = 'Sally';`
- `DROP TABLE login;`
- `DROP TABLE users;`

Author(s)

- James Chhun WingChhun