Weekend: SQL Basics; SQL ALchemy

SQL basics

Let us learn how to interact with our database.

We can access our database using the following command.

```
$ psql
james=#
```

1) Creating a Database

Let us create a new database

```
james=#  CREATE DATABASE school;
```

SQL commands are usually case insensitive but by convention we capitalize them. Every SQL command ends with a semicolon `;`.

```
james=# \c school
```

We use the `\c` psql command to connect to the `school` database. We can view all the databases we have using the `\l` command.

2) Creating Tables

Let us create our course tables

```
school=# CREATE TABLE courses (name varchar,full_time boolean);
```

We create a course table with two columns for name and whether it is a full time or not. After we create the column name we then have to define the datatype of the column `varchar` defines that all data stored in the column is will be a String. `Boolean` data type states all the data stored will be a boolean either `True` or `False`.

We can change the scheme of the table and add Items to it.

3) Altering Tables

```
school=# ALTER TABLE courses ADD student_number integer;
```

We add an Integer column to the table where we can store the number of students taking the course.

We can also drop columns

```
school=# ALTER TABLE courses DROP student_number;
```

4) Inserting into Tables

We can insert data into our table.

```
school=# INSERT INTO courses (name, full_time) VALUES ('Advanced Math',True);
school=# INSERT INTO courses (name, full_time) VALUES ('Literature',True);
school=# INSERT INTO courses (name, full_time) VALUES ('Computer science',True);
```

5) Retrieving data

We can select all the data from a particular column

```
school=# SELECT name FROM courses ;
       name
------------------
 Advanced Math
 Literature
 Computer science
(3 rows)
```

We can also select all entries from the table

```
school=# SELECT * FROM courses ;
full_time  |       name
-----------+------------------
 t         | Advanced Math
 t         | Literature
 t         | Computer science
(3 rows)
```

The wildcard `*` tells psql to pick all the tables.

6) Deleting data

We can delete values we set in our table

```
school=#  DELETE FROM courses * ;
```

This deletes all items from our table.

7) Adding a Primary Key

A **primary key** is a unique identifier inside a table. It is assigned to a new row and is not repeated anywhere else in the table.

```
school=# ALTER TABLE course ADD id serial PRIMARY KEY;
```

We use the `serial` data type to define an auto incremented integer.

Creating a Password

To connect to our psql server we need to create a password for the User Role we created when setting up:

```
school=# ALTER USER james password 'New Pasword'
```

Replace `james` with your username and `'New Pasword'` with your desired password

Common SQL commands

```
CREATE DATABASE database_name; # From the $USER database.

CREATE TABLE table_name (id serial PRIMARY KEY, some_column varchar, another_column int);
ALTER TABLE table_name ADD column_name boolean;

ALTER TABLE table_name DROP column_name;


SELECT * FROM table_name WHERE id >= 3;
SELECT * FROM contacts WHERE NOT id >= 5;

UPDATE courses SET name = 'Poetry' WHERE id = 2;
DROP TABLE table_name; # From the database that holds the table.
DROP DATABASE test_database; # From the $USER database.
```

Postgresql commands

- List all databases: \l.
- Exit out of help menu: \q
- Connect to database: \c database_name
- List tables in current database: \dt
- List columns in a table: \d table_name
- See a list of all psql commands: \? (Press the down arrow to scroll through, or q to exit list.)

SQLAlchemy

SQLAlchemy is an Object relational mapper that allows us to create a mapping between the database and the Python Objects in our application. It allows us to interact with the database using Python objects that will be translated and mapped into SQL statements.

Going back to our watchlist app, we will use the `Flask-SQLAlchemy` extension that will allow us to connect SQLAlchemy to our application

```
(virtual)$ pip install flask-SQLAlchemy
```

SQLAlchemy supports many kinds of databases. For our Postgres database, we will need a driver that will be helpful in connecting our application to our database

```
(virtual)$ pip install psycopg2
```

We install `psycopg2` which is our Postgres driver.

Setting up SQLAlchemy

First, we need to create a new database for our application.

```
$ psql
psql (9.5.8)
Type "help" for help.

james=#
```

We type in `psql` to connect to the postgres server.

```
james=#  CREATE DATABASE watchlist;
```

We then use an SQL command to create a database `watchlist`.Note that all SQL commands end with a semicolon.

```
james=# \c watchlist
You are now connected to database "watchlist" as user "james".
watchlist=#
```

We then connect to the database using the Postgres `\c` command followed by the database name.

```
watchlist=# \dt
No relations found.
```

We use the `\dt` command to check what tables are in our database. Since we have no tables created we get a `No relations found.` response.

You can get all PostgreSQL commands by typing `help` while in the Postgres server.

Next SQLAlchemy needs one important configuration. `SQLALCHEMY_DATABASE_URI` - this is the location of the database with authentication.

The format for the URI is `db+driver://username:password@host/database`. We can define this URI inside our configuration file.

config.py

```
# ...
class Config:

    MOVIE_API_BASE_URL ='https://api.themoviedb.org/3/movie/{}?api_key={}'
    MOVIE_API_KEY = os.environ.get('MOVIE_API_KEY')
    SECRET_KEY = os.environ.get('SECRET_KEY')
    SQLALCHEMY_DATABASE_URI = 'postgresql+psycopg2://username:password@localhost/watchlist'

# ...
```

We create this configuration inside our *config.py* file.Replace the `username` with user we created when configuring PostgreSQL and `password` with the user's password.

We can now initialize our extension.

app/__init__.py

```
from flask_sqlalchemy import SQLAlchemy

bootstrap = Bootstrap()
db = SQLAlchemy()
```

We first import the `SQLAlchemy` and then create a `db` instance.

```
# ...
def create_app(config_name):
```

```
    app = Flask(__name__)

    # ....

    # Initializing flask extensions
    bootstrap.init_app(app)
    db.init_app(app)

    # ...
```

While in the `create_app` application factory function we then call the `init_app`method and pass in the `app` instance.

We will next consider how we can create models for our database. And then we can see how we can authenticate users.

You can find the application at this point here :

https://github.com/mbuthiya/watchlist/tree/17-Configuring-database (https://github.com/mbuthiya/watchlist/tree/17-Configuring-database)