Database Guide

Table of Contents

General Information

Setting up Local Requirements

I used the following video to guide myself and other people for setting up the requirements for a local database.

https://codingpub.dev/ubuntu-install-postgresql-and-pgadmin/

Only the first 4 minutes are relevant. Once they start talking about multiple users then that's the end of what's needed.

It's highly recommended to make the username for your local database postgres, the password root, and the database name soccer for the sake of consistency.

Connecting to the Database

Log into the database through the terminal.

- Local:
  - psql -U postgres -h localhost
- AWS:
  - psql -U postgres -h database-1.cbumbixir8o8.us-east-1.rds.amazonaws.com

Some important commands:

- The list of databases can be viewed through the "\l" command.
- A database can be created through "CREATE DATABASE [db name];".
- A database can be deleted through "DROP DATABASE [db name];".
- Connect the database through the "\c [db name]" command.

Creating a Database File

- Local:
    - pg_dump soccer -U postgres -h localhost > db.pgsql
- AWS:
    - pg_dump soccer -U postgres -h database-1.cbumbixir8o8.us-east-1.rds.amazonaws.com > db.pgsql

This will create a file called db.pgsql which is effectively a copy of the db it was downloaded from. This file can then be used to set up another database or be used as a backup of the database at a certain moment.

Using the Database File

- Local:
    - psql -h localhost -d soccer -U postgres -f [INPUT FILE PATH HERE]
- AWS:
    - psql -h database-1.cbumbixir8o8.us-east-1.rds.amazonaws.com -d soccer -U postgres -f [INPUT FILE PATH HERE]

This command will take the database file (indicated after the -f) and attempt to upload it to the database (indicated after -d which in this case is soccer) found at the host (indicated after -h).

Creating a Local Database

If you wanted to download the file for the aws database and create a local database you would do the following:

1. Create the database file from the AWS database.
2. Log into psql.
3. Delete the relevant database if it already exists (named soccer).
4. Create the database again (or for the first time) and name it soccer.
5. Now in a separate terminal from the one currently logged into psql run the command under the “using the database file section” to set up the soccer db with the file.

Switching to a Different Database

To switch to a using a different database the connection variables in each of the four api files will need to be switched. These variables are at the top of each of these python files right below the imports. There will be four variables named database, user, host and password.

In order to use the local database the variables should be changed to whatever you set up your local database with. However, if you followed the guide above the variables will be as follows: (For a local db host would always be localhost)

```
database = 'soccer'
user = 'postgres'
host = 'localhost'
password = 'root'
```

To use the online database the variables should be changed to the following:

```
database = 'soccer'
user = 'postgres'
host = 'database-1.cbumbixir8o8.us-east-1.rds.amazonaws.com'
password = 'rootroot'
```

Database Structure

There is already some relevant documentation on this subject in the senior design document. However, some of that text is outdated due to some changes so the purpose of this.

Relevant Tables

- Game
    - This table is intended to have one entry corresponding to one game that has been uploaded to the database.
    - The key part of this table is the game_id column. The game_id is the key value that connects this entry to entries in the following tables:
        - Roster
        - Detections
        - Homography
        - Locations
    - video_original and video_processed are intended to be strings that contain links to locations of the original and processed videos stored in S3. However, the functionality of that was never implemented.
    - process_state is basically the state of the game within the application. The following states are a general layout for how the workflow through the application would look.
        - 0: PAGE - Initial Review (Game freshly uploaded)
        - 1: PROCESS - Converted to 15 fps
        - 2: PAGE - Video Editor (Being edited)
        - 3: PROCESS - Video being recompiled with removed frames (Then usually goes back to stage 2 unless no more editing needs to be done)
        - 4: PROCESS - Detection algorithm being run
        - 5: PAGE - Dashboard (Track matching + Manual annotation + Add track)
        - 6: PROCESS - Homography being run
        - 7: Finished Processing
    - The process state will be further explained in another section.
- Player
    - This table is basically a storage location for all player's basic information.
    - Players are searched from this table on the upload page to be inserted in the roster table.
        - Additional info like team_id are added in the transfer.

    - The team_id column in this table is no longer relevant. Players in this table should have no relation to any team.
- Roster
    - This table is in many ways an extension of the player table. A player should not exist in this table without first existing in the player table.
        - This is so each new player created and inserted in the player table gets a unique player_id.
    - The game_id in this table relates to the game_id in the game table. All entries in this table that have a certain game_id belong to that game's roster of players.
        - The same player can appear multiple times in the roster table with different information. However, multiple entries of the same player in this table must have different game_ids.
- Detections
    - This table is meant to house the camera detections and track information from the main detection algorithm. It is then visualized on the dashboard video player.
    - Game_id is once again used to relate the detections to which game they belong to.
    - The player_id column of this table is for which player the track is assigned to. A -1 means that the track is not currently assigned to a player.
- Homography
    - This table is meant to store the relevant homography mapping file for each frame of a game.
- Locations
    - The location table stores the actual x,y coordinate of the player on the field after being run through the homography algorithm.

Upload Page Relevancy

The upload page is intended to be used as a way to generate the first set of information a game will have when it's uploaded. This is mainly the following:

- An entry in the game table.
- A set of entries in the roster table.

It can also be used to fill in missing information in the following tables:

- Player table (because some players may need to be created for the roster)
- Team table (because some teams may be absent from the team table)

Process State Explanation

The main function that the process state serves is to track the current state of processing that the game is currently in. The state of a game can primarily take two forms:

- Page: The game is currently available to be worked on in a page.
- Process: An algorithm is currently being run on the game.

The current design of the list of states is as follows:

- 0: PAGE - Initial Review (Game freshly uploaded)
- 1: PROCESS - Being converted to 15 fps
- 2: PAGE - Video Editor (Being edited)
- 3: PROCESS - Video being recompiled with removed frames (Then usually goes back to stage 2 unless no more editing needs to be done)
- 4: PROCESS - Detection algorithm being run
- 5: PAGE - Dashboard (Track matching + Manual annotation + Add track)
- 6: PROCESS - Homography being run
- 7: Finished Processing

States are intended to be changed at the start and end of a process. For example, when the homography algorithm is run the state will be changed from 5 to 6 at the start of the algorithm and then from 6 to 7 at the end.

The process state is used on the home page to dynamically generate a link to the current page that the game is being worked on in. If the game is currently in a process then no link would be generated and instead the user would be informed that an algorithm is being run.