# Lab 2: Interfacing Python to Postgres

## Instructions

You must write a single python program a2.py, which should support the following flags for each assignment part. **The output should be printed to the terminal unless otherwise specified.**

## Part 0 - Warmup with Postgres

### Connect to the postgres server using psql:

```
psql -U postgres -h localhost -p 5432 -W
```

You will be prompted to enter password. Enter **postgres**.

```
root:/home/labDirectory# psql -U postgres -h localhost -p 5432 -W
Password:
psql (14.10 (Ubuntu 14.10-0ubuntu0.22.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

postgres=#
```

### Basic psql commands

- Create a database lab2db:

```
CREATE DATABASE lab2db;
```

- View databases - Use \l to view the list of databases

```
postgres=# CREATE DATABASE lab2db;
CREATE DATABASE
postgres=# \l
                                  List of databases
   Name    |  Owner   | Encoding | Collate |  Ctype  |   Access privileges
-----------+----------+----------+---------+---------+-----------------------
 lab2db    | postgres | UTF8     | C.UTF-8 | C.UTF-8 |
 postgres  | postgres | UTF8     | C.UTF-8 | C.UTF-8 |
 template0 | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres          +
           |          |          |         |         | postgres=CTc/postgres
 template1 | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres          +
           |          |          |         |         | postgres=CTc/postgres
(4 rows)
```

- Connect to the database - Use \c to connect to a database

```
postgres=# \c lab2db
Password:
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
You are now connected to database "lab2db" as user "postgres".
lab2db=#
```

- Load the ddl file given to you in sample/ipl.ddl - `\i sample/ipl.ddl`

- View the list of tables in the database - \dt

```
lab2db=# \dt
            List of relations
 Schema |     Name     | Type  |  Owner
--------+--------------+-------+----------
 public | ball_by_ball | table | postgres
 public | match        | table | postgres
 public | owner        | table | postgres
 public | player       | table | postgres
 public | player_match | table | postgres
 public | team         | table | postgres
 public | umpire       | table | postgres
 public | umpire_match | table | postgres
 public | venue        | table | postgres
(9 rows)
```

You can use psql for running sql queries to your database. Link to tutorial: psql tutorial

## Using pgAdmin4

Go to the link: http://127.0.0.1:5050/pgadmin4

Login using credentials:

- Email: pgadmin4@dbis.com
- Password: pgadmin4

Note: Above PostgreSQL server is already added in pgAdmin4 for this user

pgAdmin4 tutorial

Psycopg helpful links:

- Psycopg docs
- Psycopg tutorial

You can take a look at the section for the **testing** flag in a2.py.

# Part 1 (40 marks)

In this part, you will be given a set of database tables in csv format. Your task is to output the ddl file that can be loaded into postgres for creating the tables whose schema will match the csv files you have been provided with.

**Guidelines:**

- There should be a table corresponding to each csv file. For eg, player.csv should have a table "player" in the database.
- The first row of each csv has the column names for that particular table.
- You only have to deal with 3 data types: integer(INT), text, and date. For the purposes of this assignment, no data in a column with TEXT type will be pure numbers. Dates will be mentioned in "YYYY-MM-DD" format. Only TEXT types will have empty values for this assignment.
- Your ddl file can begin with "DROP TABLE IF EXISTS **table_name** CASCADE" for each table in the database. This ensures that you can load the ddl file multiple times for testing.
- **PRIMARY_KEY:**
    - If a file a.csv contains a column **a_id**, then this will be the primary key for the table. (Example: player.csv)
    - Else, the primary key will include all the fields suffixed with "_id". (Example: ball_by_ball.csv)
- **FOREIGN KEY:**
    - All columns of the form **table _id** (where **table** is a table in the database) should have a foreign key relation to the corresponding table. For example, in table **ball_by_ball**, the column **match_id** is a foreign key that refers to table **match**. Note that not all columns of the form **A_id** will have **A** as some table in the database.
    - All columns suffixed with **_ _table_key** should have a foreign key relation to the corresponding **table**. For example, in **ball_by_ball**, the column **striker__player_key** is a foreign key that refers to the table **player.** Note the double underscore before the table name: table names may contain a single underscore, but never double underscores for this assignment.
    - You can set the on DELETE condition to be "ON DELETE set null"

**Flag:** "`--export-ddl --csv_dir <csv directory path> --output_path <filepath>`"

**Sample command:** "`python3 a2.py --name lab2db --user postgres --pswd postgres --host localhost --port 5432 --export-ddl --csv_dir ../csvs/ --output_path ./ipl_out.ddl`"

The program should take the file path to the directory that contains the CSV files and output the ddl to create all the tables of the database. **If no output_path is specified, the output should be printed to the terminal.**

Sample output is given in samples/outs/ipl_out.ddl

**IMPORTANT:** You must ensure that foreign key constraints are not violated. Thus, the order of CREATE TABLE statements is important. For example, because **ball_by_ball** refers to **match**, the table **match** has to be created before **ball_by_ball.**

# Part 2 (25 marks)

**Flag:** "`--import-table-data --table <table_name> --path <csv_path>`"

**Sample command:** "`python3 a2.py --name lab2db --user postgres --pswd postgres --host localhost --port 5432 --import-table-data --table player --path ../csvs/player.csv`"

In this part, your program will load the csv file at "<csv_path>" into the table "<table_name>". The first row in the csv files contains the header.

**Before attempting part 2, load the ddl file you generated in part 1 into the Postgres database with \i in psql, not as part of your python code. (A sample ipl.ddl is provided so that this part can be done independently.)**

**Guidelines:**

- You can assume the schema will match in the final auto-grader, and you do not have to check for schema differences.
- Use execute_values, which is faster than the execute API for multiple inserts.
- Empty strings, if any are in the input csv file, are to be stored as empty strings in the database.
- Make sure you perform a commit() on the connection after inserting all the rows to reflect all the updates in the database.

# Part 3 (35 marks)

**Flag:** "`--export-table-data --table <table_name> --format <format> --output_path <filepath>`"

**Sample command:** "`python3 a2.py --name lab2db --user postgres --pswd postgres --host localhost --port 5432 --export-table-data --table player --format sql --output_path ./player.sql`"

In this part, your program will export the table data in the format specified. **If no output_path is provided, the output should be printed to the terminal.**

**Guidelines:**

- If format = "csv", include the column names in the first row in the right order, comma-separated, followed by the data in the table in csv format. (**15 marks**)
- If format = "sql", then output the insert statements used to create the rows of the table. (**20 marks**)
- Sample csv and sql output files are given in the outs/ directory.
- **DO NOT** use the copy command for this part.

# General Guidelines and Submission Instructions

The program will be run as:

```
python3 a2.py --name <database_name> --user <username> --pswd <password>
--host <host_address> --port <port>
```

followed by one of the above set of flags. **Each part will be auto-graded independently of the other parts.**

For testing your solution, you can export a ddl file with part 1, then load some csvs into the db with part 2, and then check if export-tables works with part 3. **It is possible that the Vlab client-side evaluation may spuriously fail in some cases currently.**

Submit a single python file a2.py (sample submission file provided in the submission directory) via VLab.