Assignment #4: Database Programming

ECE 650 – Spring 2022

See SAKAI course site for due date

General Instructions

1. You will work individually on this assignment. You should not collaborate, nor receive/provide help, nor look at the code of any other current or past student. Failure to adhere to these rules will be considered a violation of the Duke code of conduct.
2. The code for this assignment should be developed and tested using a Linux Virtual machine such as the VM that has been made available to you for ECE551 or on a Windows machine, as you find more convenient. Testing will be conducted on the Linux VM: You will be responsible for solving all the portability issues that may result from differences between the development and testing environments.
3. You will need to create a PostgreSQL database (therefore you need to have an installed server running) and interact with it through a C++ API. The package that you will be using is `libpqxx`.
4. You must follow this assignment spec carefully, and turn in everything that is asked in the proper formats, as described.

Overview

In this assignment, you have to develop a program that interfaces with a PostgreSQL database. The relation schema of the database will be provided, along with the database values. You will implement the following:

1. A C++ program to read text files containing the entries (rows) for each table, and builds the database by creating each table and adding entries.
2. A C++ library (set of functions) which will provide an interface for a program to interact with the database (e.g. add rows to the tables and query for certain information).

The database is a very simple one that relates to Atlantic Coast Conference (ACC) basketball teams, and will allow queries to discover information about things like player statistics, team attributes, etc. The database has 4 tables, specified as follows (an underlined attribute indicates the primary key for the table):

Relation PLAYER

PLAYER_ID, TEAM_ID, UNIFORM_NUM, FIRST_NAME, LAST_NAME, MPG, PPG, RPG, APG, SPG, BPG

Relation TEAM

TEAM_ID, NAME, STATE_ID, COLOR_ID, WINS, LOSSES

Relation STATE

STATE_ID, NAME

Relation COLOR

COLOR_ID, NAME

Meaning of attributes with abbreviations:

- MPG = minutes per game
- PPG = points per game
- RPG = rebounds per game
- APG = assists per game
- SPG = steals per game
- BGP = blocks per game

What You Will Implement

Several skeleton files, and database table information files are provided to get you started (see the homework kit provided). The following describes each file:

- **Database source text files: player.txt, team.txt, state.txt, color.txt**. These files contain table-like information for each entry and each attribute that should be inserted into the respective database table.
- **main.cpp**: The main function. Here you should implement the code that will setup the database on each execution of the program. Specifically, it should drop (if needed) and add each table to the database (named ACC_BBALL), and then read information from the source text files and add rows to each table as appropriate.
- **query_funcs.h and query_funcs.cpp**: Here you will implement functions to interact with the database (add new entries and print the results of 5 different queries specified below). You may add new functions to these files if desired (you don't have to), but you should not change the definitions of the existing functions.

- **exerciser.h and exerciser.cpp**: Here you can add code in the exercise() function to test your query functions. The exercise() function is called from main() after the database is initialized.
- **Makefile**: To compile all source files into an executable program named “test”

Your task is to do the following:

- Create a PostgreSQL database named ACC_BBALL
- Create a user for the ACC_BBALL database named “postgres” with password “passw0rd”.
- Implement support in main.cpp to connect (as user ‘postgres') to the ACC_BBALL database and initialize its tables and data. Of course, usually a database will live persistently, but for the purposes of evaluating the submissions, the program should re-create the database every time. You should drop tables that may already exist (e.g. from prior runs of your test) before creating an initializing the tables in the program.
- Implement the following query functions:
    - **query1()**: show all attributes of each player with average statistics that fall between the min and max (inclusive) for each enabled statistic
    - **query2()**: show the name of each team with the indicated uniform color
    - **query3()**: show the first and last name of each player that plays for the indicated team, ordered from highest to lowest ppg (points per game)
    - **query4()**: show first name, last name, and jersey number of each player that plays in the indicated state and wears the indicated uniform color
    - **query5()**: show first name and last name of each player, and team name and number of wins for each team that has won more than the indicated number of games
    - **Important Note:** Each query function should print its output. The format of this output should be as follows. The first row of output should contain each field name (separated by a single space character. The next rows of output should contain the values returned from the query, each also separated by a single space. The field name and each row of output should appear one line after the next.
- You may test your database and query functions by adding calls to the query functions inside the exerciser() function. For the purposes of grading assignments, we will replace the exerciser.cpp file with a new one that will test a variety of query calls.

Example Output and Checking Results

The following text shows an example output. This output corresponds to **query1** which is invoked to show all players with a minimum minutes per game (MPG) of 35 and a maximum minutes per game (MPG) of 40.

```
PLAYER_ID TEAM_ID UNIFORM_NUM FIRST_NAME LAST_NAME MPG PPG RPG APG SPG BPG
153 12 3 Andrew WhiteIII 37 19 5 1 1.6 0.4
154 12 20 Tyler Lydon 36 13 9 2 1.0 1.4
30 3 5 Luke Kennard 36 20 5 3 0.8 0.4
59 5 44 Ben Lammers 35 14 9 2 1.2 3.4
87 7 5 Davon Reed 35 15 5 2 1.3 0.5
98 8 4 Dennis SmithJr. 35 18 5 6 1.9 0.4
130 10 32 Steve Vasturia 35 13 4 3 1.2 0.1
```

Use ‘diff’ to compare results.

```
$ diff -w output.txt validation.txt
```

Note that the -w indicates that 'diff' should not report differences in whitespace (spaces, tabs) in the output. If you want to compare two files character for character, the -w flag can be omitted.

Detailed Submission Instructions

Your submission will include the following files:

1. **Source Code Files:** main.cpp, query_funcs.h, query_funcs.cpp, exerciser.h, exerciser.cpp
2. **Makefile** – **Even if you have not made changes to the one provided**
3. **Database source text files**: player.txt, team.txt, state.txt, color.txt

Submit a zip file named "hw4_<netID>.zip" to Sakai.

Extra Credit (+50%)

By implementing SQL transactions in C++, you may get a deeper understanding of SQL code and consider it as a common way to use databases. However, a typical use of databases in modern software architecture is using a higher-level framework (e.g. an Object Relational Mapping, or ORM) which obviates the need to worry about transaction details. Specifically, Object-Relational Mapping (ORM) is a technique that allows you to write transactions using an object-oriented paradigm. Many libraries have implemented this technique. For example, some of you may have used Django to develop web applications. Django contains a default ORM layer to interact with data from relational database management systems like PostgreSQL. You may refer to *django.db.models* for details. And in addition to Django ORM, there are other ORM libraries, such as Hibernate, SQLAlchemy, Doctrine, etc. Choose one ORM library and implement the five queries shown in previous statement. Submit your code as a zip file named "hw4_extra.zip" to Sakai. We will schedule to let you give a demo to TAs for grading this extra-credit part.