CptS -451 Introduction to Database Systems
Spring 2022

Project Milestone-2 - Everett

Summary:

In this milestone you will:

- ✓ design the database schema for your application and provide the ER diagram for your database design,
- ✓ translate your entity relationship model into relations and produce DDL SQL statements for creating the corresponding tables in a relational DBMS,
- ✓ populate your database with the Yelp data and get to practice generating INSERT statements and running those to insert data into your DB,
- ✓ write triggers to enforce additional constraints,
- ✓ start developing your application UI. In Milestone3 you will develop the full final application with all required features.

Milestone Description:

You need to complete the following in milestone-2:

1) Revise the database schema that you created in milestone-1. Make sure that your database schema is complete (i.e., stores all data necessary for the application) and appropriate (i.e., all queries/data retrievals on/from the database can be run efficiently and effectively).

   Additional notes about the database schema:

   - ✓ Your business table should include the following attributes (in addition to the attributes in the business JSON objects):
     - "numCheckins" : the number of total check-ins to the business.
     - "numTips" : the number of tips provided for the business.
   - ✓ Your user table should include the following attributes (in addition to the attributes in the user JSON objects):
     - "totalLikes" : the total number of likes for the user's tips.
     - "tipCount" : the number of tips that user wrote for various businesses.
     - "lat"/ "long" : latitude/longitude coordinates of the user's location.

   The default value for numCheckins, numTips, totalLikes, and tipCount attributes should be 0.
   The above names are just suggestions.

2) (5%) Translate your revised ER model into relations and produce DDL SQL (CREATE TABLE) statements for creating the corresponding tables in a relational DBMS. Note the constraints, including primary key constraints, foreign key constraints, not NULL constraints, etc. needed for the relational schema to capture and enforce the semantics of your ER design. Write your CREATE TABLE statements to a file named "<*your-team-name*>_RELATIONS_v2.sql".

3) (42%) Populate your database with the Yelp data.

   a. Generate INSERT statements for your tables and run those to insert data onto your DB. You will use your JSON parsing code from Milestone-1 and use the data you extracted from JSON objects to generate the INSERT statements for your tables.

Note: Make sure to initialize "*numCheckins*", "*numTips*", "*totalLikes*", and "t*ipCount*" attributes to 0. In task-5, you will write UPDATE statements where you will calculate and update the values for these attributes.

b. You may populate your DB with data in 2 different ways:

   i. (*Recommended*) You may embed your INSERT statement inside your JSON parsing code and execute them one by one as you generate them. (Sample Python code for connecting to PostgreSQL database and executing SQL statements is available on Canvas).

   ii. Alternatively, you may write the INSERT statements to a SQL script file and then run this (large) script file. (You will find some information about how to generate and run SQL scripts in Appendix-A of this document).

Please note that due to foreign key constraints, the order matters. The INSERTs to referenced tables should be run before INSERTs to referencing tables.

Please do not create any additional INDEXES for your tables until you insert all the data. Indexes may slow down the data insertion.

4) (8%) Calculate and update the "*numCheckins*", "*numTips*", "*totalLikes*", and "t*ipCount*" attributes for each business.

a. "*numCheckins*" value for a business should be updated to the count of all check-in counts for that business. Similarly, "*numTips*" should be updated to the number of tips provided for that business. You should query the "Checkins" and "Tips" tables to calculate these values.

   "*totalLikes*" value for a user should be updated to the sum of all likes for the user's tips. And "t*ipCount*" should be updated to the number of tips that the user provided for various businesses. You should query the "Tips" table to calculate these values.

   In grading, points will be deducted if you don't update these values correctly.

b. Write your UPDATE statements to a file named *"<your-team-name>*_UPDATE.sql".

(**Note**: On some systems, the update statement may take a long time to complete. To speed up the process, you may calculate and store the calculated "*numCheckins*", "*numTips*", "*totalLikes*", and "t*ipCount*" values into temporary tables and then update the business/user tables using the values from these temporary tables.)

5) (20%) Create triggers to enforce the following constraints in your database:

   a. Whenever a user provides a tip for a business, the "*numTips*" value for that business and the "*tipCount*" value for the user should be updated.

   b. Similarly, when a customer checks-in a business, the "*numCheckins*" attribute value for that business should be updated.

   c. When a user likes a tip, the "*totalLikes*" attribute value for the user who wrote that tip should be updated.

   Test your triggers with INSERT/UPDATE statements, i.e.,
   - Add a tip for a business (insert to tips table) and make sure that the tip, business, and user tables are updated correctly.
   - Check-in to a business (insert to checkins table) and make sure that the checkin and business tables are updated correctly.
   - Like a tip (update the tips table) and make sure that the tips and user tables are updated correctly.

   Write your TRIGGER statements and test statements (INSERT, UPDATE statements) to a file named ""*<your-team-name>*_TRIGGER.sql".

6) (25%) Start implementing your user interface. You should complete the following features in milestone2. If you would like to replicate your DB on multiple machines, you can create a backup of your DB using pg_dump and restore it . See Appendix-B for more information.
   a. Retrieve all distinct states that appear in the Yelp business data and list them. User should be able to select a state from this list.
   b. When user selects a state, retrieve and display the cities that appear in the Yelp business data in the selected state (e.g. in a list a listbox.)
   c. When a city is selected, retrieve the zipcodes that appear in the Yelp business data in the selected city (e.g. in a list a listbox.)
   d. When a zipcode is selected, retrieve the **business categories for the businesses that appear in that zipcode**. (e.g. in a list a listbox.)
   e. When the user searches for businesses, all the businesses in the selected zipcode will be displayed (e.g. in a datagrid).
   f. When user selects one or more categories, the search results will be filtered based on the selected business categories. If more than one category is selected, the search should return the businesses that are associated with all the selected categories (Note that choosing more categories should narrow down the results.)
   g. User selects a business and displays the tips of the selected business.
   h. User adds a new tip for a selected business (i.e., user enters a tip text and the tip is inserted into the tips table.) The trigger you implemented in task 5(a) should update the "*numTips*" value for the business and "tipCount" value for the user after the new tip is added.)

   (Note: The mentioned interface components are only suggested ways to display the data. As long as it is functional and easy to use, any design is acceptable.)

Milestone-2 Deliverables - Checklist:

1. The revised E-R diagram for your database design. **Should be submitted in .pdf format.**
   - Name this file "*<your-team-name>*_ER_v2.*pdf*"
2. SQL script file containing all CREATE TABLE statements.
   - Name this file "*<your-team-name>*_RELATIONS_v2.sql"
3. SQL script file containing all UPDATE TABLE statements.
   - Name this file "*<your-team-name>*_UPDATE.sql"
4. SQL script file containing all TRIGGER statements and the test statements.
   - Name this file "*<your-team-name>*_TRIGGER.sql"
5. Your code that generates (or executes) 'INSERT' statements. (i.e., Task3 code) .
   - Name this file "*<your-team-name>*_parseandinsert.py"
6. The source code of your application. (Please include all source files including project or solution files.)

All milestone-2 deliverables should be committed to the "milestone2" branch in your team's GitHub (or GitLab) project repo. **The milestone2 branch in your GitHub repo should include the following folders:**

- **DBSchema**
  - The revised E-R diagram for your database design: "*<your-team-name>*_ER_v2.*pdf*"
  - SQL script file containing all CREATE TABLE statements: "<your-team-name>_RELATIONS_v2.sql"
- **DBPopulate**:
  - Your code that generates (or executes) 'INSERT' statements. (i.e., Task3 code) : "*<your-team-name>*_parseandinsert.sql"
- **YelpApp_v1**: this folder will include the prototype of your application that you will submit in milestone2
- **SQLScripts**: this folder will include your SQL script files for milestone2

- *"<your-team-name>*_UPDATE.sql"
- *"<your-team-name>*_TRIGGER.sql"

You will demonstrate your Milestone-2 to the instructor and the TA in early April.

Appendix A – How to create and run a SQL script file in PostgreSQL

Simply open a text editor and write all of your queries, separating them with empty lines. Make sure that each query is terminated by a ';'.
As an example, suppose that you have the following two queries, and your database name is yelpDB:
Q1:

```
select * from  reviewTable;
```

Q2:

```
select name from businessTable
where  state>'AZ';
```

Your script file should then look like as follows:

```
select * from reviewTable;
select name from businessTable
where  star>3;
```

You should save this file with a ".sql" extension.

Running The Script

In the command line, run the following :

```
psql -d yelpdb -U postgres
```

(on Windows: run 'cmd' to open command line window)
(if `psql` is not recognized, you need to add the PostgreSQL installation path to the PATH environment variable. Alternatively, you may browse to the installation directory of PostgreSQL and then run the above command).

- You have to supply a database name to connect to. The above statement assumes your database name is "yelpdb".
- If you would be running postgreSQL with another username (other than `postgres`), replace `postgres` with that username. You will be asked to enter your password for the username you specify.

Assuming that you have saved the script file in the folder c:\myfolder, run the following in command line:

```
yelpdb=#> \i ./myscript.sql
```

(update the path of the file if your script file is not in the current directory.)
(The "yelpdb=#" here is the command prompt. Yours will look different depending on your database name.)

The above command will execute all the queries in the `myscript.sql` file.

Check http://www.postgresqlforbeginners.com/2010/11/interacting-with-postgresql-psql.html for a brief tutorial about interacting with PostgreSQL in the command line.

Appendix-B Extract a PostgreSQL database into a script file

Assume the name of your database is “yelp”.

To dump the yelp database into a SQL-script file, run the following on the terminal (or Windows command line):

```
pg_dump -U postgres yelp > yelpdbbackup.sql
```

Copy the `yelpdbbackup.sqlfile` to the other machine, create the database “yelp”.
To reload such a script into the (freshly created) database named “yelp” :

```
psql -U postgres -d yelp < yelpdbbackup.sql
```

OR

```
psql -U postgres -d yelp -f yelpdbbackup.sql
```

If you want to create a “tar” or directory backup, check the pg_dump documentation at :
https://www.postgresql.org/docs/13/app-pgdump.html
https://www.postgresqltutorial.com/postgresql-backup-database/