Lab 3: Potpourri, Part 2

Due date: Friday, April 28, 11:59pm

Lab Assignment

Assignment Preparation

This is an individual lab. Each student has to complete all work required in the lab, and submit all required materials **exactly as specified** in this assignment.

The lab continues the data manipulation tasks from Lab 3-1.

The tasks in this lab use five datasets, BAKERY, CSU, INN, KATZENJAMMER, and AIRLINES, which were not used in **Lab 3-1**.

Submission filenames. You will submit the following files for each dataset:

- `<DATABASE>-setup.sql`: your `CREATE TABLE` statements.
- `<DATABASE>-build-<file>.sql`: one script per table that inserts all tuples in the table. Each tuple must be inserted using a separate INSERT INTO statement.
- `<DATABASE>-insert.sql`: basically a script that, when run inserts ALL tuples into all tables of the database. (A script that consists of `source <DATABASE>-build-<file>.sql` commands usually works. So does a script that is the results of concatenating all `<DATABASE>-build-<file>.sql` files in the correct order).
- `<DATABASE>-cleanup.sql`: the `DROP TABLE` script.
- `<DATABASE>-test.sql`: the database test script.

In addition, for the datasets mentioned in this part of the assignment, you will submit a `<DATABASE>-modify.sql` script which performs all the required tasks.

Traceability

Your `<DATABASE-modify.sql>` scripts will consist of a number of SQL DDL and DML commands. Each script must have a top comment specifying your full name and cal poly email (login id).

Additionally, each SQL statement you place into the file must be prefaced with a short comment specifying its purpose. If you are skipping a command (e.g., because you were not able to make it work), place the comment specifying that you skipped a SQL statement into your script. For example, if you are asked to write a SQL command that adds a new attribute `foo` to table X, deletes a few records from this table, and then instantiates it `foo` to 10 for all remaining tuples, your script shall have comments similar to the ones shown below (assuming you skip the second command):

```
...

-- Add attribute foo to table X

ALTER TABLE ....
...
;

-- Delete tuples [Not Implemented]

-- Set value of foo to 10

UPDATE ....
...
;
```

Data Manipulation Tasks

The assignments in this part are specific to individual databases you created in **Lab 2**. Please execute them only on the specified datasets. The assignments may ask you to change both the schemas and the instances of the databases.

[AIRLINES dataset.] Create an SQL script `AIRLINES-modify.sql` which performs the actions described below.

You are modeling corporate takeover of a number of routes from one airport in the database.

1. Remove from the flights database all flights except for those to and from `AKI` (that's the airport code).
2. You will be modeling corporate takeover by `Continental` of all flights except those operated by `Virgin` and `AirTran` to and from `AKI`. For this problem **(and this problem ONLY)**[1], you will look up the

[1] As opposed to similar questions in future labs.

numeric IDs for each airline, and substitute them for airline names in the commands you need to develop.

3. For all flights NOT operated by `Continental`, `AirTran` or `Virgin`, increase the flight number by 2000 (this will ensure that after the corporate takeover, flight numbers are still unique).

4. All flights in the `Flights` table come in pairs. The first flight starts at Airport 1 and goes to Airport 2. The second flight goes from Airport 2 to Airport 1 (the return flight). The flight numbers of these two flights are different by 1 - one flight number is even, one is odd.

   For all pairs of flights to/from `AKI` NOT operated by `Continental`, `AirTran`, or `Virgin`, you need to *flip* the flight numbers. That is, if a flight from `AKI` to some other airport had an even flight number N, it needs to be replaced by $N + 1$, while, the flight number for a return flight will change from $N + 1$ to N. Conversely, if a flight from `AKI` has an odd flight number N, it needs to be replaced by $N - 1$, while the flight number of the return flight needs to change from $N - 1$ to N.

   (In other words, all even-numbered flights need to increase by 1, all odd-numbered flights need to decrease by 1.)

   Write a sequence of SQL commands that achieve this result.

   NOTE: You can use built-in MOD(N,D) function which returns the remainder of deleting N by D.

   Also, you can perform any *temporary* operations with the `Flights` table you want to achieve this task, as long as you *clean up* after yourself - i.e., as long as after all the commands are completed, the only change in the `Flights` table is the flipped flight numbers.

5. Complete the corporate takeover. Replace the airline for all flights to and from `AKI` except for `AirTran` and `Virgin` with `Continental`.

6. Output the contents of the `flights table` using the following SQL statement:

   ```
   SELECT *
   FROM <Flights-table>
   ORDER BY <Airline-column>, <flight-no-column>;
   ```

   replacing `<Flights-table>` with the name of the flights table in your database, and replacing `<Airline-column>` and `<flight-no-column>` with the name of the appropriate columns in that table.

[BAKERY dataset.] Create an SQL script `BAKERY-modify.sql` which performs the actions below.

1. Remove from the table containing the listing of the pastries, information about all pastries except for `Cakes`.

2. The bakery manager is looking to adjust the prices of the pastries to accommodate for seasonal changes in the prices of the ingredients.

3. Increase the prices of the `hocolate` and the `Opera Cake` by 20%.

4. Reduce the prices of `Lemon Cake` and `Napoleon Cake` by $2.

5. Reduce the price of all other cakes by 10%.

6. Show the contents of the table using the following SQL statement:

```
SELECT *
FROM <Pastry-table>
ORDER BY GID;
```

   replacing `<Pastry-table>` with the name of the table containing the list of pastries in your database.

[**CSU dataset.**] Create an SQL script `CSU-modify.sql` which performs the actions below.

For this assignment (and this assignment only), you need to look up the campus ID number for each campus name mentioned below, and the numeric discipline enrollment ID for each discipline mentioned below, and use these Id numbers in the commands where campus identity and/or discipline identity is used.

1. Keep in the table documenting *campus enrollments by discipline* only the information about the following enrollments:
   - Engineering majors from Cal Poly San Luis Obispo and Cal Poly Pomona.
   - Long Beach State enrollments for disciplines with more than 200 graduate students.
   - All enrollments in Computer and Information Sciences for schools with more than 500 undergraduate majors.

2. Output the remaining contents of the discipline enrollments table using the following SQL statement:

```
SELECT *
FROM <Discipline-Enrollments-Table>
ORDER BY <CampusID-column>, <DisciplineId-column>
```

   replacing `<Discipline-Enrollments-Table>` with the name of the discipline enrollments table in your database, and replacing `<Campus-ID-column>` and `<DisciplineId-column>` with the names of the appropriate attributes in that table.

3. Keep in the table documenting CSU fees only the information that matches ALL the conditions below:

- The fee is greater than \$2.500;
- The year is either 2002 or one of 2004—2006.
- The campus is either Cal Poly San Luis Obispo, Cal Poly Pomona or San Jose State.

4. Output the remaining contents of the fees table using the following SQL command:

```
SELECT *
FROM <Fees-table>
ORDER BY <CampusId-column>, <Year-column>
```

replacing `<Fees-table>` with the name of the CSU fees table in your database, and replacing `<CampusId-column>` and `<Year-column>` with the names of the appropriate columns in that table.

[INN dataset] Create an SQL script `INN-modify.sql` which performs the actions below.

- Add to the table storing information about the rooms an attribute to store the description of a room. The room description is a short text (about double the standard tweeter message length).
- Create a somewhat meaningful description for each room and place the description into the record for the respective room. The descriptions should be loosely based on the name of the room and other observable (from the table) room features. For example, for "Thrift and accolade" room, a description might be

  > *"Experience the luxury of our Bed & Breakfast for half the price! Cozy room overlooking picturesque garden."*

- Output the contents of the rooms table using the following SQL command:

```
SELECT *
FROM <rooms-table>
ORDER BY <Room-Id> \G
```

replacing `<rooms-table>` with the name of your rooms table and `<Room-id>` with the name of the attribute for the three-letter room code. Note the
`G` at the end of the query. This will force MySQL to output results in a row-by-row fashion, rather than in a table.

[KATZENJAMMER dataset.] Create a SQL script `KATZENJAMMER-modify.sql` which performs the following actions.

1. In the table specifying which instruments the band members play on each song (we refer to it as the "instruments table" below), replace all occurrences of `'bass balalaika'` with `'awesome bass balalaika'`, and all occurrences of `'guitar'` with `'acoustic guitar'`. (Please note that you may need to change the length of the instrument name field if it is not long enough in your table - do it using a table schema modification command, rather than in the CREATE TABLE statement).

2. Keep in the instruments table only the information about `'acoustic guitar'` players and the information about all instruments `Turid` (her band member id is 4 - you can use it directly) played on all songs.

3. Run the following SQL query:

   ```
   SELECT *
   FROM <instruments-table>
   ORDER BY <Song-id>, <Bandmate-Id>;
   ```

   where `<instruments-table>` is the name of your instruments table, and `<Song-id>` and `<Bandmate-Id>` are the columns in this table with the song and band member Ids respectively.

4. Add a new attribute to the table describing the albums released by Katzenjammer. The attribute should store the total number of songs on the album.

5. Based on information stored in the tracklists table (look up the CSV file if you have to), update each record in the albums table to show the correct number of tracks.

6. Run the following SQL query:

   ```
   SELECT *
   FROM <albums-table>
   ORDER BY <Year>;
   ```

   where `<albums-table>` is the name of your table of Katzenjammer albums, and `<Year>` is the name of the column in the albums table that stores the year of the album release.

Submission Instructions

Please, follow these instructions exactly. Up to 10% of the Lab 3-2 grade will be assigned for conformance to the assignment specifications, **including the subimssion instructions**.

Please, **name your files exactly as requested** (including capitalization), and submit all files **in a single archive**. Correct submission simplifies grading, and ensures its correctness.

Please include your name and Cal Poly email address in all files you are submitting. If you are submitting code/scripts, include, at the beginning of the file, a few comment lines with this information. Files that cannot be authentificated by observing their content will result in penalties assessed for your work.

Specific Instructions

You must submit all your files in a single archive. Accepted formats are `gzipped tar` (`.tar.gz`) or `zip` (`.zip`).

The file you are submitting must be named `lab3.zip` or `lab3.tar.gz`.

Inside it, the archive shall contain five dataset directories: `AIRLINES`, `CSU`, `BAKERY`, `KATZENJAMMER`, and `INN`. Each directory shall contain all Lab 2 scripts for the given dataset, as well as the appropriate `<DATABASE>-modify.sql` files. All directories must contain **the new versions** of the database creation, deletion and test files.

In addition, the root of the directory must contain a `README` file, which should, at a minimum, contain your name, Cal Poly email, and any specific comments concerning your submission.

Submit your archive using the following `handin` command:

```
handin dekhtyar lab03-2 <file>
```

Testing

The Lab 3-2 `try` script will be released to you early next week. Please make sure to use it prior to submission to test how it works. Instructions will be emailed to everyone once the `try` script is out.

Your submission will be tested by running all scripts you supply and checking the produced output for correctness. I may also use some extra scripts to verify the correctness of the databases you have constructed.

If you are aware of any bugs, or incorrect behavior of your SQL scripts, I strongly suggest that you mention it in the README file.