Lab 3: Potpourri

Due date: Tuesday, October 16, beginning of the lab

Lab Assignment

Please note, you will definetly get **Lab 4** assignment by labtime on October 16.

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 has three main assignments. First, you need to complete your database creation and population scripts and make certain that all errors found in their Lab 2 versions are fixed. You will be using these scripts for the rest of the quarter.

The second part of the assignment tests your ability to change the database schema and modify the contents of the tables.

The third part of the assignment establishes your mastery of the sql*plus environment, your overall knowledge of SQL DDL and DML and your ability to work effectively with DATE data in SQL.

Part 1: Finishing SQL scripts

Your task is to ensure that your scripts properly create all course datasets. If you were successful in doing so in Lab 2, then, you do not need to do anything else for this task.

If your Lab 2 submission yielded errors, you need to fix them, and ensure that your scripts create proper databases and insert correct data in them.

This part of the lab does not get graded, however, (a) you will need to complete this task in order to get the remainder of this lab to work and (b) you will be using the scripts you finalize in this lab in the labs that follow.

Note, that you will receive feedback on how your Lab 2 submissions performed on Tuesday, October 9.

Part 2: Working with databases: schema modification/database modification

The assignments in this part are specific to individual databases you create in Part 1 of the lab. Please execute them only on the specified datasets.

General note: all assignments for Part 2 and Part 3 shall be run independently of each other. To run each script you develop, create and populate the specific database using your scripts from Lab 2/Lab 3, Part 1. Then run the script you create. After that, delete the entire database. If there is another script relating to the same dataset, recreate the database to its original state, and run the other script.

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

Extend the database structure to include the information about the GPA for each student and the bus route each student takes to/from home.

For the GPA, make certain that only GPAs in the range between 0.0 and 5.0 are allowed. Update the database as follows:

- All kindergarden students are assigned GPA of 4.0.
- All students in classroom 112 are assigned GPA of 3.0.
- All 1st graders who are not in classroom 102 are assigned GPA of 2.8.
- All other students are assigned GPA of 3.2.

For the bus route, ensure that the only allowed values are either 0 (student does not take the bus), or number from 51 to 55. Assign bus numbers as follows.

- `ELTON FULVIO`, `ANIKA YUEN` and `JANEE DANESE` are the only students who take bus route 51.
- All second- and third- graders take bus route 52.
- All students in classroom 112 except `STORMY KRISTENSEN` take bus route 53.
- `STORMY KRISTENSEN` takes bus route 54.
- No student is taking bus route 55.

- All remaining students do not take bus to/from school.

You can use any way of setting the new values (and any number of commands) which results in correct assignments being made. Write an SQL script documenting the entire procedure, finish it with a `SELECT * FROM <StudentsTable>;` SQL statement.

[BAKERY dataset] .

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

Suppose the bakery is located in Guymon, OK in the five-sate area (OK-TX-NM-KS-CO). The customers come from locales in the 55-65 mile radius from these five states (study an on-line map of the area). Assign to each customer a location (town, state) of your choice. Ensure that at least one customer exists from each of the five states. Write an SQL script changing the database schema and updating the database appropriately.

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

Update the schema of the table containing the technical specs of the cars to include a new attribute representing the fuel efficiency of the vehicle in terms of liters on 100 km [1]. Populate the new attribute by converting the the MPG fuel efficiency.

Write SQL commands that clean the cars dataset, leaving in the `cars-data` tables only information about cars produced between 1977 and 1979 (inclusively). Prepare an SQL script with appropriate statements, end it with `SELECT * FROM <Cars-dataTable>;`.

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

Modify the table representing the list of campuses (i.e., the one originated from the `Campuses.csv`) to add to it information about the geographic region in which the campus is located and the university's athletic nickname.

Initialize the region information as follows. Split California into regions as follows. There are five regions: `'Southern California'`, `'Central Coast'`, `'Central Valley'`, `'Bay Area'` and `'Northern California'`.

`'Southern California'` contains all counties south of Santa Barbara, Kern and Inyo counties.

`'Central Coast'` is Santa Barbara, San Luis Obispo, Monterey, Santa Cruz and San Benito counties.

`'Bay Area'` are all counties that touch the Bay, except for Napa, Sonoma and Solano.

[1] Which is how fuel efficiency is measured in all countries using metric system.

`'Central Valley'` are all Central Valley, Sierra Foothills and Eastern Sierra counties south of Lake Tahoe. The northern edge of Central Valley is Solano, Sacramento, Amador and Alpine counties.

`'Northern California'` are all other counties.

Since county information is available for each campus, write SQL commands that set the region values for each campus. Use a map of Calfornia counties if you need to.

Write SQL commands that set the appropriate mascot for each university. Use the plural proper noun when appropriate (e.g., `'Mustangs'` for Cal Poly). The nickname information may be found on the CSU system's Wikipedia page (http://en.wikipedia.org/wiki/California_State_University).

Create an SQL script containing approrpiate statements, and end it with `SELECT * FROM <CampusesTable>;`.

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

Add a pair of wine drinking advice columns to the wine list table. In general, wine tasting professionals provide drinking advice in a form of "enjoy the wine from year X to year Y". The first column will be used to store the "year X" part of the advice and the second part — to store the "year Y" part of the advice.

Populate the columns as follows. For any white wines (see the list of grapes to figure out which wines are white — you may need to write one version of the apporipraite SQL statement for each white grape variety), the first column should be the the vintage year plus two and the second column should be the year that comes five years after the vintage year. For example, the drinking advice for the following wine:

```
8,'Sauvignon Blanc','Altamura','Napa Valley','California','Sauvignon Blanc',2007,48,92,500,
```

shall take values of 2009 and 2012 respectively.

For `Cabernet Sauvignion` wines, the drinking advice shall assign the first column the year that comes five years after the vintage year, and as second column, the year that comes fourteen years after the vintage year. For example, the drinking advice for the following wine:

```
49,'Cabernet Sauvingnon','Lewis','Napa Valley','California','Reserve',2007,130,95,1700,'now'
```

shall take values of 2012 and 2021 respectively.

For all other red wines, the drinking advice shall assign the first column the year that comes three years after the vintage year and assign the second column the year that comes ten years after the vintage year. For example, the drinking advice for the following wine:

```
156,'Syrah','Favia','Amador County','California','Quarzo',2008,65,95,154,'now'
```

shall take values of 2011 and 2018 respectively.

Finally, and this shall overwrite any other advice assignment, any wine rated 88 points or below shall have the drinking advice of the year after the vintage year for the first column, and the year two years after the vintage year for the second column, *for any grape variety.* For example, the drinking advice for the following wine:

```
208,'Chardonnay','CC:','California','California','Chardonnay',2009,15,86,5000,'now'
```

shall take values of 2010 and 2011.

Part 3: SQL*plus mastery

Each task in this part relates to a specific dataset, of this assignment, and should be performed only for that dataset.

This assignment is independent from the assignments in Part 2 of the lab. That is, I will be testing your scripts for Part 3 on the databases constructed by the scripts from Part 1.

[STUDENTS dataset]. Consider the following SQL query, which outputs the names of each student and his/her teacher:

```
SELECT S.<FirstName>, S.<LastName>, T.<FirstName>, T.<LastName>
FROM <listTable> S, <teachersTable> T
WHERE S.<Classroom> = T.<Classroom>
ORDER BY T.<LastName>;
```

Substitute appropriate column names for their placeholders in this query. Before proceeding with the tasks below, make sure this query runs and returns the correct result.

Create an SQL script, `STUDENTS-formatted.sql` containing SQL*plus commands which format the result of this query as follows:

- There are no pagebreaks in the provided output.
- The name of the teacher appears only once, and there is an empty line between the records for students with different teachers.
- Student name column headers are "Student: First name" and "Student: Last name". Teachers' first name column header is "Teacher", teachers' last name column header is empty.
- The columns are separated with a double vertical bar: (`'||'`).
- Each retrieved record fits in a single line.

[**AIRLINES dataset.**] Write an SQL script, `AIRLINES-change.sql` which performs the following actions.

1. Delete from the list of flights all flights except those flown by `Southwest Airlines` (`Southwest`) and `JetBlue Airways` (`JetBlue`).

   For this assignment (and this assignment ONLY: do not make this mistake in Lab 4 queries!!!) you may use the Id numbers for `Southwest` and `JetBlue` in your statements here and in all the statements you write below.

2. Output the list of flights in the database.

3. Prepare `JetBlue` for "corporate takeover." First, "flip" the flight numbers on `JetBlue` flights between each pair of airports. For example if the database contains flight `400` from `ANA` to `AEL` and flight `401` from `AEL` to `ANA`, after the issued commands, flight from `ANA` to `AEL` will have number `401`, and flight from `AEL` to `ANA` will have number `400`[2]

   Hint. You may have to perform this activity in more than one step. Note also, that in the current dataset, all flight numbers are smaller than 9999.

4. Output the list of flights in the database. (`SELECT * FROM <flights>;`).

5. Now, change the flight numbers on all `Southwest` flights to be larger by 2000. (this is done to guarantee no overlapping flight numbers with `JetBlue`).

6. Reassign all `JetBlue` flights to `Southwest`.

7. Finally, remove `JetBlue` from the list of airlines.

8. Output the list of flights in the database.

9. Output the list of airlines in the database.

[**CSU dataset.**] Write an SQL script, `CSU-change.sql` which does for following.

1. In the table of enrollements, keeps only the information about (a) Cal Poly enrollments in the years when it was over 15,000; (b) enrollments for any campus for any year between 1959 and 1961; (c) enrollments for any campus except for `San Diego State University` for years when the enrollment exceeded 29,000 and (d) enrollments for any campus except for `California State Univeristy - Channel Islands` for any year when the total enrollment was less then 400 students.

2. Outputs the contents of the enrollments table.

[2] Observe that all flights in this dataset come in pairs - an even-numbered flight from airport A to airport B and an odd-numbered flight from airport B to airport A. The flight numbers are always one apart, with the even number being the smaller of the two.

[**MARATHON dataset.**] Write an SQL script, `MARATHON-change.sql` which does the following. Note, the results of all actions are **cumulative**.

1. Keeps in the results table information only about runners from the state of Connecticut.
2. Outputs the contents of the results table formatted so that each record fits a single line, and the output fits one page. Time should be formatted as `[[ HH:MI --> SS]]` , pace should be formatted as `== MI,SS ==`.
3. Keeps in the results table information only about male runners in the `50-59` age group.
4. Outputs the contents of the results table formatted so that each record fits a single line, and the output fits one page. Time should be formatted as `HH:MI.SS` , pace should be formatted as `MI::SS`.
5. Keeps in the results table only the columns for the bib number, overall place of the runner his name and hometown.
6. Outputs the contents of the results table formatted so that each record fits a single line, and the output fits one page. Time should be formatted as `HH:MI:(SS)`.

Submission Instructions

Please, follow these instructions exactly. Up to 10% of the Lab 3 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 eight directories named `AIRLINES`, `CSU`, `CARS`, `BAKERY`, `MARATHON`, `STUDENTS`, `WINE` and `INN` after the dataset names. (Note, `INN` dataset is not used in any new assignments for this

lab, but please submit the corrected versions of your files for it too). 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.

Each directory shall contain all SQL scripts built by you for the specific dataset in response to all parts of the lab. The Lab 2 scripts/ Lab 3 part 1 scripts must be resubmitted resubmitted, with the same names. (these are the `<Dataset>-setup.sql, <Dataset>-build-<table>.sql` and `<Dataset>-cleanup.sql` files). SQL scripts for Part 2 and Part 3 shall be named as specified in the assignment.

Submit your archive using the following `handin` command:

```
handin dekhtyar lab03 <file>
```

Testing

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.