Lab 3: Potpourri

Due date: Thursday, April 21, midnight

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 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.

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. Make certain that only GPAs in the range between 0.0 and 5.0 are allowed. Update the database as follows:

- All 4th graders are assigned GPA of 2.5.
- All students in classroom 109 are assigned GPA of 3.5.
- All 5th graders who are not in classroom 109 are assigned GPA of 3.1.
- All other students are assigned GPA of 2.9.

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 Hayti, MO in the quad-sate area (MO-TN-KY-AR). The customers come from locales in the 55-65 mile radius from these four 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 four 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 in 1970. 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.

Univeristies comprising what is now known as the CSU system have been brought together in 1960. Since then, CSU system has been extended to include new campuses. The Campuses table contains information about the dates different universities were founded but not the dates when they joined the system.

Based on this data and information supplied below, you need to write SQL statements to (a) add a new column to the Campuses table for the year the campus joined the CSU system and (b) populate the new field.

For all campuses except for the Maritime Academy, if the campus was founded before 1960, it joined the CSU system in 1960. The Maritime Academy joined the CSU system in 1995. All campuses founded after 1960 joined the CSU system the year they were founded.

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

Note: in the SQL queries above (as well as below) a table name in angle brackets (e.g., `<Cars-dataTable>`) refers to the name you assign to the relational table storing the data of the specified `.csv` file (in our example - `cars-data.csv`). You will substute the actual table name for this placeholder. In the queries below, the attribute names in angle brackets are understood in a similar manner - they need to be replaced by the actual column names from your relational schema.

[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 appotypes SQL statement for each white grape variety), the first column should be the year immediately following the vintage year, and the second column should be the year that comes six years after the vintage year. For example, the drinking advice for the following wine:

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

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

shall take values of 2008 and 2013 respectively.

For `Cabernet Sauvignion` wines, the drinking advice shall assign the first column the year that comes four years after the vintage year, and as second column, the year that comes fifteen 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 2011 and 2022 respectively.

For all other red wines, the drinking advice shall assign the first column the year that comes two years after the vintage year and assign the second column the year that comes eight 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 2010 and 2016 respectively.

Finally, and this shall overwrite any other advice assignment, any wine rated 87 points or below shall have the drinking advice of the year after the vintage year for the first column, and the year three 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 2012.

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.

[BAKERY dataset]. Consider the following SQL statement:

```
SELECT R.<Date>, R.<RecieptNumber>, C.<FirstName>, C.<LastName>
FROM <RecieptsTable> R, <CustomersTable> C
WHERE R.<CustomerId> = C.<Id>
ORDER BY C.<LastName>, R.<Date>;
```

This statement outputs the list of reciepts (sales) with the names of the customers (rather than their Ids) sorted by customer's last name, and with a secondary sort by the purchase date.

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, `BAKERY-formatted.sql` containing SQL*plus commands which format the result of this query as follows.

- The result is broken into 4 pages, each page contains 50 records (there are 200 reciepts in the database). (make sure you set the appropriate parameters correctly).
- Each customer name appears exactly once (duplicate occurrences of the name are left blank).
- The column separator for the output is "(*)"; the column headers are separated from the data by a line of "-" characters.
- Each record in the output fits into a single line of the report;
- The column headers are as follows: "Sold On", "Reciept", "Bought by", "".

For each database create an SQL script (or scripts, where necessary) performing specified tasks.

AIRLINES database

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 `US Airways` (`US Air`) and `Frontier Airlines`.

   You may use the Id numbers for US Airways and Frontier in your statements here and in all the statements you write below.

2. Output the list of flights in the database.

3. Prepare `Frontier` for "corporate takeover." First, "flip" the flight numbers on `Frontier` 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`.

   Note, that 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.

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

6. Finally, reassign all `Frontier` flights to `US Airways`.

7. Remove `Frontier` from the list of airlines.

8. Output the list of flights in the database.

9. Output the list of airlines in the database.

BAKERY database

Do the following.

1. Re-write (if needed) your `CREATE TABLE` statements for the `BAKERY` dataset so that each constraint defined in the database is explicitly named.

2. Write an SQL script, `BAKERY-change.sql` that does the following (note: the actions have to be consecutive):

   (a) Keeps in the database information only about the pastry items (goods), whose price is greater than $8.50 or that have chocolate (`'Chocolate'`) flavor.

(b) Keeps in the database information only about the purchases (receipts) that occurred on October 12, 2007.

(c) Increases the prices of all pastries in the database by 9%.

(d) Outputs the list of pastries in the database.

(e) Outputs the list of receipts in the database.

Hint. There is a reason why each constraint in this database needs to have a name.

CSU database

Write an SQL script, `CSU-change.sql` which does for following.

1. In the tables of student fees, degrees and faculty information keeps only the information about Cal Poly (you may use Cal Poly's unique id - look it up.)
2. Outputs the contents of the student fees, degrees and faculty information tables.
3. In the table of enrollements, keeps only the information about (a) Cal Poly enrollments in the years when it was over 14,000; (b) enrollments for any campus for any year prior to 1935; (c) enrollments for any campus except for `San Diego State University` for years when the enrollment exceeded 30,000 and (d) enrollments for any campus except for `California State Univeristy - Channel Islands` for any year when the total enrollment was less then 300 students.
4. Outputs the contents of the enrollments table.

MARATHON database

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 Massachussets.
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 `40-49` 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 and SS` , pace should be formatted as `MI and SS`.

5. Keeps in the results table only the columns for the bib number, overall place of the runner, his name and time.

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)`.

STUDENTS database

Write an SQL script, `STUDENTS-change.sql` that does the following.

1. Replaces the names `AL`, `TOBIE`, `RODGER`, `RAY`, `SUMMER`, `KITTIE` with the names `ALEX`, `TOBIAS`, `ROGER`, `RAYMOND`, `AUTUMN` and `KATHERINE` respectively.

2. Outputs the list of students.

3. Converts all first and last names of students into lowercase, keeping the first letter capitalized.

4. Outputs the list of students.

5. Adds a new column to the table of students, representing the full name of the student. Sets the full name of the student to be his/her lastname, followed by the first name, separated by a comma. E.g., the full name of the student with first name `'Mel'` and last name `'Balboa'` is `'Balboa, Mel'`.

6. Removes first name and last name columns from the students table.

7. Outputs the list of students.

INN database

Write an SQL script, `INN-change.sql` that does the following.

1. Adds a new column to the table of reservations to store the total number of people staying in the room.

2. Populates the new column with the right information.

3. Adds a new column to the table of reservations to store the total number of nights spent by the guests.

4. Populates the new column with the right information (note: you can subtract `DATE` values from each other).

5. Keeps in the database only the reservations that are for seven nights or longer.

6. Keeps in the reservations table only the columns containing the room code, the last name of the person making the reservation, the starting date of the reservation, the number nights and the total number of people staying.
7. Outputs the contents of the reservations table.

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. 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-grader lab03-365 <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.