

Lab 3: Potpourri Part 1

Due date: Thursday, May 4, 7:00pm

Note: It is likely that Lab 4 will be assigned on Thursday, May 4, in class.

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.

Note on data. This lab will require you to use the files you prepared in **Lab 2**. You will complete three assignments, using three of the nine course databases.

You can use both command-line `mysql` client, and/or `LabThreeSixFive.com` portal (sandbox mode) to work on this assignment.

Dataset Submission.

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.py` Python script which performs all the required tasks.

Tasks

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

Task 1: Fix Your Tables

On Thursday, April 27, you will receive feedback on your `CREATE TABLE` statements from me. The feedback will indicate where any errors were made. By the Lab 3 submission deadline you need to fix your `CREATE TABLE` errors and have new versions of `<DATABASE>-setup.sql`.

You need to do the following:

1. fix all errors, and finalize the content of `<DATABASE>-setup.sql` files;

2. ensure that all insertions proceed correctly with the new `setup` scripts, fix any issues;
3. submit fixed script collections (`<DATABASE>-setup.sql`, `<DATABASE>-build-<table>.sql`, `<DATABASE>-cleanup.sql`, `<DATABASE>-test.sql`) for ALL nine datasets.

Task 2: Data Manipulation.

The assignments in this part are specific to individual databases you created in **Lab 2**. You will be writing a one Python script per dataset. That script shall contain code (both Python, and SQL) to perform all tasks.

Note: For this assignment, you are welcome to debug and test your Python code using Jupyter labs. However, the **the final version** of your submission shall be a Python (.py) script, **not** a notebook.

Each Python script you submit shall be named `<DATASET>-modify.py` and shall be submitted as part of your `<DATASET>` directory.

To ensure that your script properly authenticates **both for you and for the instructor/grader**, we provide the following instructions below.

- When started, each Python script shall read the contents of the file named `../account.info` (i.e., the `account.info` file located on directory up in the directory structure. This file therefore shall reside in the root of your Lab 3 directory.)
- The `accounts.info` file shall consist of two lines. The first line shall contain the name of the user and the second line shall contain the MySQL account password for the user. For example, for user `alex` with password `abc123321bca`, the `account.info` file will have the following content:

```
alex
abc123321bca
```

Note: DO NOT submit YOUR account.info file. We will use our own versions to test your code.

- Your Python script then shall connect to the Lab365 Python server, using the `loginId` and the password read from the `account.info`. It shall set the working database to the `loginId` read from the `account.info` as well.

Following the confirmation of successful connection, your script shall complete all required tasks: both by passing the DML/DDI statements to the DBMS, and by sending SQL `SELECT` statements specified in the task, and processing/displaying the output.

Each Python script shall assume that the database it is connecting to already has all tables for the specific dataset created and populated. Effectively, we will be testing your program by first creating all the tables for the dataset using your `setup` script, then populating them using your `build-itablei` scripts, then running your Python script for the dataset, and finally, cleaning up your work by running your `clean` script.

[**STUDENTS dataset.**] Create an Python script `STUDENTS-modify.py` which performs the actions below. First, extend the database structure to include the information about the GPA for each student. Then, update the database as follows:

- Keep in the database only the students from grades 2,3 and 4.
- Add a new classroom to the database. The classroom number is 114, and the teacher in that classroom is ROBERTA YANG.
- Move MARCS RUNKLE, SHANAE BIBB and MEL BALBOA to classroom 114.

- Set the GPA of second graders to 3.25.
- Set the GPA of first-grade students from room 102 to 3.1.
- Set the GPA of first-grade students from room 114 to 3.5.
- Set the GPA of all other first-grade students to 2.75.
- Set the GPA of third-grade students except for `CHRISSY NABOZNY` to 3.2.
- Set the GPA for `CHRISSY NABOZNY` to 3.3
- Add new first grade students to classroom 114: `YORAM LEVY`, `JUSTIN WASHINGTON`, `ASTRID NG` and set their GPA to 3.2.
- The following instructions apply to individual students and override all prior GPA assignments.
 - Set the GPAs of `EVELINA BROMLEY` and `MAUDE CAR` to be 15% higher than their current GPAs.
 - Set the GPAs of `RODGER RANSLKER` and `ROBBY PINNELL` to be 0.25 lower than their current GPAs.

Include all necessary SQL commands to achieve this result into the `STUDENTS-modify.py` script. Complete the script by running two queries:

```
SELECT * FROM <students-table>
ORDER BY <GPA-column>, <grade-column>, <student-lastname-column>;
```

and

```
SELECT *
FROM <teachers-table>;
```

replacing `<students-table>` with the name of your table containing the list of students and `<GPA-column>`, `<grade-column>` and `<student-lastname-column>` with the names of the columns storing the GPA, the grade level of each student and their last names respectively, and replacing `<teachers-table>` with the name of your table containing the list of teachers. Print the outputs of each of these queries.

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

1. Remove the columns storing the appellation name and the name of the wine from the table storing the list of wines (we refer to this table as "the wine table")¹.
2. Keep in the wine table only the Zinfandels with a score of 92 or higher.
3. Add a new column to the table called **Revenue**. It should have the same type as your price column.
4. A case is 12 bottles of wine. Using the information available to you, set the revenue for each wine left in the table to be equal to the total amount of money that can be made by selling all the cases of the wine.
5. Output the list of wines using the following SQL query:

```
SELECT * FROM <wine-table>
ORDER BY Revenue;
```

(replace `<wine-table>` with the appropriate table name).

¹This is largely for the eventual final result/output to be compact.

[**CARS dataset.**] Create a SQL script `CARS-modify.py` which performs the following actions.

1. Keep in the table storing the technical characteristics about the cars (we refer to this table as "the car data table"), ONLY the records that satisfy *at least one* of the following conditions:
 - (a) vehicles made in 1978 or 1979 with MPG of 20 or above.
 - (b) vehicles that have MPG of 26 miles per gallon or better what have an engine with more than 110 horsepowers.
 - (c) vehicles that have 8 cylinders and accelerate to 60 mph in less than 10 seconds.
2. Run the following SQL query:

```
SELECT *  
FROM <car-data-table>  
ORDER BY <year-column>, <car-Id>;
```

where `<car-data-table>` is the name of the car data table in your **CARS** database and `<year-column>` is the column in that table storing the year in which a vehicle was made and `<car-id>` is the unique Id of each tuple in the car data table.

3. Remove from the car data table all attributes except car id, car year, acceleration, MPG number of cylinders.
4. Remove from the car data table information about all cars with 5 cylinders or fewer.
5. Run the

```
SELECT *  
FROM <car-data-table>  
ORDER BY <year-column>, <car-Id>;
```

query again.

[**AIRLINES dataset.**] Create a Python script `AIRLINES-modify.py` 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**)², you will look up the 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. First, some context on how the data in the **Flights** table is organized. All flights in this 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.

²As opposed to similar questions in future labs.

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.)

Place the appropriate SQL statements that achieve these results into your Python script.

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

Also, you can perform any *temporary* operations you want with the **Flights** table 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 Python script `BAKERY-modify.py` which performs the actions below.

(please note, attempts to delete tuples from the **Goods** table will result in foreign key constraint violations, so your Python code needs to include SQL statements that alleviate that issue.)

1. Remove from the table containing the listing of the pastries, information about all pastries that are cheaper than \$5.
2. Increase the prices of the **hocolate**-flavored items by 35%.
3. Reduce the prices of **Lemon**-flavored items by \$2.
4. Reduce the price of all other cakes by 10%.
5. Make the price of pie items equal to the price of the least expensive cake (**for this question** you can look up what that number is.
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 Python script `CSU-modify.py` 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, Cal Poly Pomona, and Humboldt State University.
 - San Jose State enrollements for disciplines with more than 500 graduate students.
 - All enrollments in LA County CSUs (look up which ones are in LA County) where graduate enrollment exceeds undergraduate enrollment for the same discipline.
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 Diego 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.

[**KATZENJAMMER dataset.**] Create a Pytho script `KATZENJAMMER-modify.py` 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 `'awesome bass balalaika'` players and the information about all instruments **Anne-Marit** (her band member id is 3 - 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 grade will be assigned for conformance to the assignment specifications, **including the submission 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 authenticated 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 nine dataset directories: AIRLINES, CARS, CSU, BAKERY, INN, MARATHON, STUDENTS, WINE, and KATZENJAMMER. Each directory shall contain new versions of all your .sql files for creating and testing your datasets. In addition, the eight directories for the datasets used in Lab 3 assignment shall contain your Python scripts.

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 365-lab03 <file>
```