CS 2550 – Principles of Database Systems (Spring 2024)
Dept. of Computer Science, University of Pittsburgh

Assignment #2: SQL Review

Release: Thursday, Jan. 25, 2024 Due: 8:00 PM, Feb. 7, 2024

Goal

The objective of this assignment is to refresh or gain familiarity with SQL and PL/SQL. You will create and manipulate the database of a coffee chain brand that is called *CoffeeDB* using SQL in the PostgreSQL server. You will also *have to use* transactions appropriately, create views, and create triggers using SQL and PL/pgSQL in the PostgreSQL server.

Database Schema

Assume the following relational database schema that records information related to the *CoffeeDB* system given that PK denotes a primary key, FK denotes a foreign key, and UQ denotes an alternative key:

- STORE (store_id, name, store_type, gps_lon, gps_lat)
  PK(store_id)
  UQ(name)
- COFFEE (coffee_id, name, description, country_of_origin, intensity, price)
  PK(customer_id)
- CUSTOMER (customer_id, fname, lname, email, month_of_birth, day_of_birth)
  PK(customer_id)
  UQ(email)
- SALE (sale_id, customer_id, store_id, coffee_id, quantity, time_of_purchase)
  PK(sale_id)
  FK(customer_id)→CUSTOMER(customer_id)
  FK(store_id)→STORE(store_id)
  FK(coffee_id)→COFFEE(coffee_id)
- PROMOTION (promotion_id, name, start_date, end_date)
  PK(promotion_id)
- PROMOTES (promotion_id, coffee_id)
  PK(promotion_id, coffee_id)
  FK(promotion_id)→PROMOTION(promotion_id)
  FK(coffee_id)→COFFEE(coffee_id)
- CARRIES (promotion_id, store_id)
  PK(promotion_id, store_id)
  FK(promotion_id)→PROMOTION(promotion_id)
  FK(store_id)→STORE(store_id)

Assumptions

Here are some basic assumptions:

- The fields ***fname*** and ***lname*** mean first name and last name respectively in any table.
- The ***store_type*** field in the table STORE is an enumeration, the store could be a: *sitting*, *kiosk*, or *drive-through* store. All types of stores support the full coffee menu maintained by CoffeeDB.
- Each store in CoffeeDB has a location which is identified by its GPS(lat, lon) coordinates.
- The attribute ***description*** in the COFFEE table is a free text type that has a string length limit of 280 characters.
- The attribute ***country_of_origin*** in the COFFEE table is the name of the country where the coffee originated as a string with a length limit of 60 characters.
- The attribute ***intensity*** in the COFFEE table represents the flavor profile of the coffee blend. The intensity should only range from values of 1 (weak coffee profile) to 12 (strong coffee profile).
- The field ***price*** in the COFFEE table reflects the amount the customer should be charged when purchasing a single quantity of the specific coffee.
- The fields ***month_of_birth*** and ***day_of_birth*** in the CUSTOMER table represent the birthday month and day of the customer. The ***month_of_birth*** is a string with a length limit of 3 characters that stores the month abbreviation (e.g., 'Jan', 'Feb', 'Mar', etc.) while the ***day_of_birth*** is a string of length 2 characters that stores the day (e.g., '01', '02', '12', '23', etc.). Note that the birth year of a customer is ommitted for privacy reasons.
- The attribute ***quantity*** in the SALE table represents the quantity purchased of a specific coffee_id. For instance, if Dr. Chrysanthis purchased one espresso (assume coffee_id is 0) from a STORE, the quantity on the sale would be 1 for coffee_id 0. If Brian purchased two caramel macchiatos (assume coffee_id is 7) from a STORE, the quantity on the sale would be 2 for coffee_id 7. Finally, if Brian bought one espresso and two caramel macchiatos, this would be stored as two separate rows in the SALE table for each coffee_id.
- The attribute ***time_of_purchase*** in the SALE table is a timestamp that represents when the SALE was completed. In the scenario where Brian bought an espresso and two caramel macchiatos, the time_of_purchase should be the same for both sale records (i.e., for each coffee_id).
- The attribute ***start_date*** in the PROMOTION table should always be strictly before the ***end_date***.
- State any additional assumption(s) that you have clearly, as long as they do not contradict with the provided ones.

Datatypes

We assume the following data types:

- ***store_id***, ***coffee_id***, ***intensity***, ***customer_id***, ***sale_id***, ***quantity***, ***promotion_id***: integer
- ***name***, ***country_of_origin***, ***fname***, ***lname***, ***email***: varchar(60)
- ***store_type***: varchar(13)
- ***month_of_birth***: char(3)
- ***day_of_birth***: char(2)
- ***description***: varchar(280)
- ***gps_lon***, ***gps_lat***, ***price***: numeric(7, 2) or decimal(7, 2)
- ***start_date***, ***end_date***: date
- ***time_of_purchase***: timestamp

Questions

Answer the following questions in a manner that at no point in time the database is in inconsistent state (i.e., use transactions whenever necessary) [for a total of 100 points]:

1. [24 points] Create a database for the above seven relations. You need to fully define the primary keys, foreign keys, and unique constraints in your statements. You need to make all primary keys **NOT DEFERRABLE**, all foreign key to be **DEFERRABLE INITIALLY IMMEDIATE**, and all unique constraints **DEFERRABLE INITIALLY DEFERRED**. Also, add semantic constraints based on the assumptions above, that could be achieved by check constraints, and the triggers in Q9 and Q10.
2. [6 points] After creating the database using your SQL statements, populate the database according to the data in `sample-data.sql`.
3. [25 points] Express the following queries in SQL and answer them using the database you have created above. **Do not use any views or PL/SQL constructs**.
   (a) List the first name, last name, emails, and number of purchases of all customers who have made more than one coffee purchase. List them in a descending order based on the number of purchases.
   (b) List the store_id, store name, store type, and number of promotions of the store(s) who carried the highest number of promotions. List them in an ascending order of their store name.
   (c) List the store_id, store name and store type of all stores of *CoffeeDB* along with the year in which the store sold the highest total quantity of coffee.
   (d) Rank coffees based on the most quantity of the coffee that was sold by *CoffeeDB*. List: coffee_id, coffee name, intensity, and rank.
   (e) List the top-3 customers with the highest number of coffee purchases. List customer_id, fname, lname, email, and number of coffee purchases.
4. [5 points] Create a view named LAST_QUARTER_PERFORMACE that lists the number of customers that each store sold coffee to during the fourth quarter of 2023 (i.e., September 1st, 2023 to December 31, 2023). The view schema is: store_id, store name, store_type, and number_of_customers.

5. [5 points] Create a function called *avg_customers_per_store* that utilizes the (created above) view called LAST_QUARTER_PERFORMACE that calculates and returns the average number of customers that purchased coffee across all stores. The function has no input.
6. [5 points] Create a function called *avg_customers_per_kiosk_store* that utilizes the (created above) view called LAST_QUARTER_PERFORMACE that only takes into consideration the kiosk stores. The function calculates and returns the average number of customers that purchased coffee across all kiosk stores. The function has no input.
7. [10 points] Create a function called *customer_coffee_intensity* that for a particular customer with a given ID, it finds out whether their most recent coffee purchase had a strong coffee profile (intensity greater than 6) or not. The function has the following input:
   - customer_to_query: the id of the customer to be looked up.

   The function should output a boolean value of *true* if the most recent coffee purchase had a strong coffee profile and *false* otherwise.
8. [5 points] Create a stored procedure called *monthly_coffee_promotion_proc* that creates a promotion with the start_date set to the day of the procedure call with the end_date set to 30 days after the day of the procedure call. In addition, the procedure should promote a specific coffee. The procedure has the following input:
   - beverage_id: the id of the coffee to be promoted
   - ad_id: the promotion_id to be created and used to promote the coffee_id.
   - ad_name: the name of the promotion being created.
9. [5 points] Create a trigger *check_daily_inventory_trig*, that will enforce a semantic integrity constraint. That is when a store goes to sell a coffee (i.e., a new sale is added to the SALE table), the trigger checks whether the store's remaining daily inventory is enough to handle the current sale. We will assume that the inventory size for each type of coffee is 200, which is reset on a daily basis. If the store does not have enough remaining quantity of a coffee (i.e., the store has already sold 200 cups of coffee #7 or the new SALE quantity plus prior purchases would exceed 200 cups of coffee #7), the trigger should raise an exception.
10. [10 points] Create a trigger *coffee_birthday_sale_trig*, that when a coffee is sold on a customer's birthday (i.e., a new sale is added to the SALE table and the current date is the customer's birthday), it updates the field ***quantity*** to reflect a BOGO (buy one get one free) offer. If the SALE quantity is greater than 1, i.e., the customer is purchasing at least two coffees of the same type, the quantity of the SALE should be decreased by 1 to reflect the free beverage. If the SALE quantity is not greater than 1, the trigger should not apply the BOGO offer (i.e., the quantity will not be changed) and should raise the message 'Two drinks must be purchased as part of this sale in order to apply your birthday offer.' Note that this offer can be applied multiple times for each SALE on the customer's birthday.

What to submit

Each student is required to submit **exactly two** files under their **pitt_user_name** (e.g, pitt01).

- **<pitt_user_name>-db.sql**
  In this SQL script file, please submit the answers to Q1 and Q2. In addition to providing the answers, you are expected to:

  - **include your name and pitt-user name at the top of the SQL script as an SQL comment**, and
  - identify the question number before each answer using SQL comments (see below).
  - you must use SQL **DROP TABLE IF EXISTS <table_name>;** statements at the beginning of this script so that you can make sure your database does not have pre-existing tables, which have the same name as the tables described above in this assignment.

  The entire text file should be composed of **valid SQL statements**.

- **<pitt_user_name>-query.sql**
  In this SQL script file, please submit the answers to [Q3-Q10]. In addition to providing the answers, you are expected to:
  - **include your name and pitt-user name at the top of the SQL script as an SQL comment**, and
  - identify the question number before each answer using SQL comments (see below).
  - at the beginning of this script, you must write simple queries to list the content of the tables described above (e.g., SELECT * FROM CUSTOMER;)

  The entire SQL script file should be composed of **valid SQL statements**.

How to submit your assignment

1. After preparing your solution, submit your files (i.e., two SQL script files) that contain your solution to the class' Gradescope by either navigating to https://www.gradescope.com/ and selecting the course CS 2550 from the Course Dashboard or by clicking the Gradescope Navigation option under our course Canvas page.
2. Submit your assignment by the due date (**Feb. ???, 2024 at 8:00pm**). **No late submissions will be accepted**.

Academic Honesty

The work in this assignment is to be done *independently*. Discussions with other students on the assignment should be limited to understanding the statement of the problem. Cheating in any way, including giving your work to someone else will result in an **F** for the course and a report to the appropriate University authority.

Comments and Suggestion

- For an official reference to PostgreSQL, please refer to the PostgreSQL Documentation.
- The list of PostgreSQL errors are available in the PostgreSQL documentation at Appendix A.
- Providing additional information in submission files (see **What to submit**) should be done by using the comment feature of SQL: any text after a double dash is considered a comment until the end of the line. For example, add the lines

```
--------------------------------------------
-- Question #3.a:
--------------------------------------------
```

  before the SQL query that is your answer to Question #3, bullet (a). Your are also encouraged to leave a couple of empty lines between answers, for better clarity.