CS 1555/2055 – Database Management Systems (Summer 2019)
Dept. of Computer Science, University of Pittsburgh

Project: BoutiqueCoffee

Release Date:	Monday, Jun. 24, 2019
Team Formation Due:	8:00 PM, Friday, Jun. 21, 2019
Phase 1 Due:	8:00 PM, Monday, Jul. 08, 2019
Phase 2 Due:	8:00 PM, Monday, Jul. 15, 2019
Phase 3 Due:	8:00 PM, Wednesday, Jul. 24, 2019

Purpose of the project

The primary goal of this project is to implement a single Java application program that will operate **BoutiqueCoffee**, a database management system for the coffee chain brand. The secondary goal is to learn how to work as a member of a team which designs and develops a relatively large, real database application.

You must implement your application program using Java, PostgreSQL, and JDBC. The assignment focuses on the database component and not on the user interface. Hence, no user interface is required. All functions will be called directly.

Phase 1: The BoutiqueCoffee database schema and example data

The database system stores information about the coffee chain and its loyalty program. You are required to define all of the structural and semantic integrity constraints and their modes of evaluation. For both structural and semantic integrity constraints, you must state your assumptions as comments in your database creation script. Semantic integrity constraints involving multiple relations should be specified using triggers. It will have the following relations. **Please follow the table and attribute names strictly.**

- Store (Store_ID, Name, Address, Store_Type, GPS_Long, GPS_Lat)
  PK(Store_ID)
  Datatype:
  - Store_ID: int
  - GPS_Long, GPS_Lat: float
  - Other attributes: varchar(20)

- Coffee (Coffee_ID, Name, Description, Intensity, Price, Reward_Points, Redeem_Points)
  PK(Coffee_ID)
  Datatype:
  - Coffee_ID, Intensity: int
  - Price, Reward_Points, Redeem_Points: float
  - Other attributes: varchar(20)
  Notes:
  - Reward points are points that a customer earns when purchasing a coffee.
  - Redeem points are points that a customer spends to redeem a coffee for free.
  - A customer doesn't earn reward points for a coffee if it is being redeemed.

- Promotion (Promotion_ID, Name, Start_Date, End_Date)
  PK(Promotion_ID)

Datatype:
- Promotion_ID: int
- Name: varchar(20)
- Other attributes: Date

- MemberLevel (MemberLevel_ID, Name, Booster_Factor)
  PK(MemberLevel_ID)
  Datatype:
  - MemberLevel_ID: int
  - Name: varchar(20)
  - Booster_Factor: float

- Customer (Customer_ID, First_Name, Last_Name, Email, MemberLevel_ID, Total_Points)
  PK(Customer_ID)
  FK(MemberLevel_ID) → MemberLevel(MemberLevel_ID)
  Datatype:
  - Customer_ID, MemberLevel_ID: int
  - Total_Points: float
  - Other attributes: varchar(20)
  Notes:
  - Total_Points is expected to be kept up to date at all time. (Hint: Use a trigger.)

- Purchase (Purchase_ID, Customer_ID, Store_ID, Purchase_Time)
  PK(Purchase_ID)
  FK(Customer_ID) → Customer(Customer_ID)
  FK(Store_ID) → Store(Store_ID)
  Datatype:
  - Purchase_ID, Customer_ID, Store_ID: int
  - Purchase_Time: Date
  Notes:
  - The final reward points of a coffee is the reward points of the coffee times the booster factor of the customer's member level. If the coffee is being promoted, the reward points will further be doubled. The reward points can be doubled at most once by promotion.

- OfferCoffee (Store_ID, Coffee_ID)
  PK(Store_ID, Coffee_ID)
  FK(Store_ID) → Store(Store_ID)
  FK(Coffee_ID) → Coffee(Coffee_ID)
  Datatype:
  - Store_ID, Coffee_ID: int

- HasPromotion (Store_ID, Promotion_ID)
  PK(Store_ID, Promotion_ID)
  FK(Store_ID) → Store(Store_ID)
  FK(Promotion_ID) → Promotion(Promotion_ID)
  Datatype:
  - Store_ID, Promotion_ID: int

- PromoteFor (Promotion_ID, Coffee_ID)
  PK(Promotion_ID, Coffee_ID)
  FK(Promotion_ID) → Promotion(Promotion_ID)

FK(Coffee_ID) → Coffee(Coffee_ID)
Datatype:
- Promotion_ID, Coffee_ID: int

- BuyCoffee (Purchase_ID, Coffee_ID, Purchase_Quantity, Redeem_Quantity)
  PK(Purchase_ID, Coffee_ID)
  FK(Purchase_ID) → Purchase(Purchase_ID)
  FK(Coffee_ID) → Coffee(Coffee_ID)
  Datatype:
  - Purchase_ID, Coffee_ID, Purchase_Quantity, Redeem_Quantity: int
  Notes:
  - Purchase_Quantity is the quantity of coffee, that the customer buys with money in this purchase.
  - Redeem_Quantity is the quantity of coffee, that the customer redeems with points in this purchase.

The ID field in table Store, Coffee, Promotion, MemberLevel, Cutomer and Purchase should be generated using SERIALs, whose documentation can be found at:
`https://www.postgresql.org/docs/10/datatype-numeric.html`
Once you have created a schema and integrity constraints for storing all of this information, you should generate sample data to insert into your tables. Generate the data to represent at least 3 stores, 20 customer and 50 purchases.

Phase 2: A JDBC application to manage BoutiqueCoffee

You are expected to do your project in Java interfacing PostgreSQL server using JDBC. For all tasks, you are expected to check for a properly react to any errors reported by the DBMS (PostgreSQL), and provide appropriate success or failure feedback to user. Further, be sure that your application carefully checks the input data from the user and avoids SQL injection.

Attention must be paid in defining transactions appropriately. Specifically, you need to design the **SQL transactions** appropriately and when necessary, use the concurrency control mechanism supported by PostgreSQL(e.g., isolation level, locking models) to make sure that inconsistent states will not occur. For example, if a customer makes two purchases concurrently on two terminals, then either one transaction fails or if no transaction fails, the total points she receives after the two purchases should be correct.

The objective of this project is to familiarize yourself with all the powerful features of SQL/PL which include functions, procedures and triggers. Recall that triggers and stored procedures can be used to make your code more efficient besides enforcing integrity constraints. All tasks can be implemented in database approaches using triggers, stored procedures and functions. You will lose points if you implement them using Java approaches.

You application should implement at least the following functions. The code skeleton with function signatures will be provided. However, if you decide to write the functions by your own, then **please follow the function signatures strictly.** (For grading purpose, the functions are defined as public. However, in real production environment, they shoulde be defined as private.)

- public int addStore(String name, String address, String storeType, double gpsLong, double gpsLat)
  @return the auto-generated ID of this store or -1 if the operation is not possible or failed

- public int addCoffee(String name, String description, int intensity, double price, double rewardPoints, double redeemPoints)

@return the auto-generated ID of this coffee or -1 if the operation is not possible or failed

- public int offerCoffee(int storeId, int coffeeId)
  @return 1 if the operation succeeded or -1 if the operation is not possible or failed

- public int addPromotion(String name, Date startDate, Date endDate)
  @return the auto-generated ID of this promotion or -1 if the operation is not possible or failed

- public int promoteFor(int promotionId, int coffeeId)
  @return 1 if the operation succeeded or -1 if the operation is not possible or failed

- public int hasPromotion(int storeId, int promotionId)
  @return 1 if the operation succeeded or -1 if the operation is not possible or failed

- public int addMemberLevel(String name, double boosterFactor)
  @return the auto-generated ID of this member level or -1 if the operation is not possible or failed

- public int addCustomer(String firstName, String lastName, String email, int memberLevelId, double totalPoints)
  @return the auto-generated ID of this customer or -1 if the operation is not possible or failed

- public int addPurchase(int customerId, int storeId, Date purchaseTime, List<Integer> coffeeIds, List<Integer> purchaseQuantities, List<Integer> redeemQuantities)
  @param coffeeIds - ID's of the coffees being bought by this purchase
  @param purchaseQuantities - Mapping 1-to-1 to coffeeIds, indicating the purchase quantity of each coffee in this purchase.
  @param redeemQuantities - Mapping 1-to-1 to coffeeIds, indicating the redeem quantity of each coffee in this purchase.
  @return the auto-generated ID of this purchase or -1 if the operation is not possible or failed
  Notes:
  - Examples of failures: not having enough points to redeem coffees, certain coffee not being sold in the store, etc.

- public List<Integer> getCoffees()
  @return a list of ID of all coffees in the database. It returns an empty list if no coffee is in the database or the operation failed.

- public List<Integer> getCoffeesByKeywords(String keyword1, String keyword2)
  @return a list of ID of all coffees, each of which has both keywords in its name, in the database. It returns an empty list if no coffee satisfying the conditions is in the database or the operation failed.

- public double getPointsByCustomerId(int customerId)
  @return the total points of the customer identified by the customerId or -1 if the operation is not possible or failed

- public List<Integer> getTopKStoresInPastXMonth(int k, int x)
  @param k - the K in top K
  @param x - the timespan in month
  @return a list of ID of top k stores having the highest revenue in the past x month
  Notes:
  - Revenue is defined as sum of money that the customers pay for the coffees.

- 1 month is defined as 30 days counting back starting from the current date time.
- The returned list should be sorted, with ID of the store having the highest revenue at its head.
- If multiple stores have the same revenue, the order of their ID's in the returned list can be arbitrary.
- If multiple stores have the same revenue in the Kth highest revenue position, their ID's should all be returned.
- It returns an empty list if no store is in the database or the operation failed.

- public List<Integer> getTopKCustomersInPastXMonth(int k, int x)
  @param k - the K in top K
  @param x - the timespan in month
  @return a list of ID of top k customers having spent most money in buying coffee in the past x month
  Notes:
  - 1 month is defined as 30 days counting back starting from the current date time.
  - The returned list should be sorted, with ID of the customer having spent most money at its head.
  - If multiple customers have the same spending, the order of their ID's in the returned list can be arbitrary.
  - If multiple customers have the same spending in the Kth highest spending position, their ID's should all be returned.
  - It returns an empty list if no customer is in the database or the operation failed.

Phase 3: Bringing it all together

The primary task for this phase is to create a Java driver program to demonstrate the correctness of your BoutiqueCoffee backend by calling all of the above functions and display corresponding results. It may prove quite handy to write this driver as you develop the functions as a way to test them. Your should also include a benchmark program to stress test the stability of your system buy calling each function automatically for multiple time with reasonably large amount of data and display corresponding results each time a function is being called.

Now this may not seem like a lot for a third phase (especially since it is stated that you may want to do this work alongside Phase 2). The reasoning for this is to allow you to focus on incorporating feedback from the TA regarding Phase 1 and 2 into your project as part of Phase 3. This will be the primary focus of Phase 3.

Project Submission

Phase 1 The first phase should contain only the SQL part of the project. Specifically,

- **schema.sql** the script to create the database schema
- **trigger.sql** the script containing definition of the triggers, and any stored procedures or functions that you designed
- **query.sql** the script containing all insert queries. If you wish to receive more feedback, include your additional SQL queries in this file, too.

Note that after the first phase submission, you should continue working on your project without waiting for our feedback. Furthermore, you should feel free to correct and enhance your SQL part with new views, functions, procedures etc.

Phase 2 The second phase should contain, in addition to the SQL part, the Java code. Specifically,

- **BoutiqueCoffee.java** the file containing your main class and all the functions

Phase 3 The third phase should contain, in addition to the second phase, the driver and benchmark. Specifically,

- **BCDriver.java** the driver file to show correctness of your functions
- **BCBenchmark.java** the benchmark file to stress test your functions

The project will be collected by the TA via the GitHub repository. Therefore, at the beginning of the project, you need to do two things:

1. Create one common private repository as a team, where all team members are contributing and use it to develop the project.
2. Give full permission of the project repository to your TA (GitHub ID: NannanWen) and your instructor (GitHub ID: constantinos).

To turn in your code, you must do three things by each deadline:

1. Make a commit to your project repository that represents what should be graded as your group's submission for that phase. The message for this commit should be "Phase X submission" where X is 1,2 or 3.
2. Push that commit to the GitHub repository that you have shared with the instructor and TA
3. Send an email to cs1555-staff@cs.pitt.edu with the title "[CS1555] Project Phase X submission" that includes a link to your GitHub repository and Git commit ID for the commit that should be graded. Commit ID can be found on GitHub or as part of output of "gitlog" command.

Multiple submissions are allowed for each phase for each team. The last submission before the corresponding deadline will be graded. *NO late submission is allowed.*

Group Rules

- You are asked to form groups of two persons. You need to notify the instructor and the TA by emailing group information to cs1555-staff@cs.pitt.edu (Remember that you need to send the email from your pitt email address, otherwise the email will not go through). The email should include the names of the team members and should be CC-ed to all team members (otherwise the team will not be accepted).
- The deadline to declare teams was **8:00 PM, Friday, Jun. 21, 2019** . If no email was sent before this time, you will be assigned a team member by the instructor. Each group will be assigned a unique number which will be sent by email upon receiving the notification email.
- It is expected that all members of a group will be involved in all aspects of the project development and contribute equally. Division of labor in any way (e.g. with regard to phases) is not acceptable since each member of the group will be evaluated on all phases.

Grading

The project will be graded on correctness (e.g. coping with violation of integrity constraints), robustness (e.g. coping with failed transactions) and readability. You will not be graded on efficient code with respect to speed although bad programming will certainly lead to incorrect programs. Programs that fail to compile or run or connect to the database server earn zero and *no partial points.*

Academic Honesty

The work in this assignment is to be done *independently* by each team. Discussions with other students or teams on the project 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.

Enjoy your class project!