CS 1555/2055 – Database Management Systems (Fall 2022)
Dept. of Computer Science, University of Pittsburgh

Project: BoutiqueCoffee

Release Date:	Friday, Oct. 14, 2022
Team Formation Due:	8:00 PM, Friday, Oct. 21, 2022
Phase 1 Due:	8:00 PM, Wednesday, Nov. 02, 2022
Phase 2 Due:	8:00 PM, Monday, Dec. 5, 2022
Phase 3 Due:	8:00 PM, Thursday, Dec. 15, 2022

Purpose of the project

The primary goal of this project is to realize a database system that supports the operations of **BoutiqueCoffee**, a coffee chain brand, by following the development cycle of a database application (i.e., conceptual design, logical design, schema evolution, physical design, and implementation of an application).

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. Undergraduate students (CS1555) will work in teams of three whereas graduate students (CS2055) in teams of two.

You must implement your application program using Java, PostgreSQL, and JDBC. The assignment focuses on the database component and not on the user interface. A simple two-level user interface is sufficient, where the first level lets the user choose functions and second level prompts for field values in the function.

Team Declaration

- You are asked to form groups of three members, if you are enrolled in CS1555 and of two members if you are enrolled in CS2055. You will have to declare the composition and team name (if you wish) of your team by emailing your 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 your team name, as well as the name and pittID of each team member and should be CC-ed to all team members (otherwise the team will not be accepted).
- The deadline to declare teams is **8:00 PM, Wednesday, Oct. 21, 2022**. If no email is 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.

Phase 1: The BoutiqueCoffee database design

BoutiqueCoffee is a chain of premium coffee stores. The BoutiqueCoffee database system stores information about the coffee chain and its loyalty program. Specifically,

- BoutiqueCoffee maintains a number of stores, each of which has a unique name (i.e., BoutiqueCoffeeOne, BoutiqueCoffeeTwo, etc.) and is recognized by its establishment number. There are two different kinds of stores: sitting stores and kiosks. Both kinds support full menu and rewards. The store has a location and is identified on smartphone apps by their GPS(lat, lon).

- BoutiqueCoffee features a variety of coffees, each of which has a unique coffee ID, name, description, country of origin, and intensity 1 to 12. Furthermore, each coffee has a price, and associated rewards points and redeem points for loyal customers. 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 does not earn reward points for a coffee if it is being redeemed.

- For each of its customers, BoutiqueCoffee assigns them an ID and wants to store their name (first, last, and middle initial), birth day and month (the year is omitted for privacy reasons), and a phone number (home, mobile, work, other). If the customer is a member of BoutiqueCoffee's loyalty program, BoutiqueCoffee wishes to store their loyalty level and total number of points earned. On the day of the birth day of a customer, the total number of points earned is increased by 10% as a gift.

- The loyalty levels are basic, bronze, silver, gold, platinum, and diamond and each are associated with a booster factor. The final reward points of a coffee is the reward points of the coffee times the booster factor of the customer's loyalty level.

- For each sale, BoutiqueCoffee records the unique purchase ID, the customer, the store, purchase time, the coffee, purchase portion, and redeem portion. Purchase portion is the cost of the coffee that the customer buys with money in this purchase. Redeem portion is the cost of the coffee that the customer redeems with points in this purchase.

- BoutiqueCoffee as any company during celebrations or major events offers promotions. Each promotion has a unique number, a name, a period (start and end date), the promoted coffees, and the list of stores participating in the promotion. When a promotion expires it should be removed from the database.

In this phase, you are required first to produce a complete conceptual design of the BoutiqueCoffee database using the ER Model and second to produce and define the corresponding relational model in PostgreSQL.

– As part of your conceptual design you must state your assumptions about both structural and semantic integrity constraints.
– As part of your relational model, you are expected to define all of the structural and semantic integrity constraints on individual tables, referential integrity triggering actions, and default values.
– Semantic integrity constraints involving multiple relations should be specified using triggers and you will be required to specify them as part of Phase 2.

Datatypes are as follows:

- IDs, intensity: integer
- price, reward points, booster, GPS location: float
- names: varchar(50)
- store_type is a DOMAIN of varchar(7) with values { 'sitting', 'kiosk' }
- description: varchar(250)

- country of origin: varchar(60)
- middle_initial: char(1)
- day of birth: char(2)
- month of birth: char(3)
- phone_number: varchar(16)
- phone_type is a DOMAIN of varchar(6) with values { 'home', 'mobile', 'work', 'other' }
- loyalty_level is a DOMAIN of varchar(10) with values { 'basic', 'bronze', 'silver', 'gold', 'platinum', 'diamond' }
- purchase_time: timestamp
- start and end dates: date

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, 50 purchases, 12 coffees, and 3 promotions. Also initialize the clock to be Sep. 1, 2022.

Phase 2: A JDBC application to manage BoutiqueCoffee

The objective of this phase of the 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. As stated above, integrity constraints involving multiple relations should be specified using triggers in this phase. *You are expected to write at least three triggers.* 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.

Attention must be paid in defining transactions appropriately. Specifically, you need to specify *the modes of evaluation* of each integrity constraint, 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 application should implement a system of interfaces (non-graphical) that will allow users to connect to the database and perform predefined functions accomplishing a task. You are expected to use Java interfacing PostgreSQL server using JDBC.

To facilitate time travel, you are expected to implement a *Clock*. You must maintain a "pseudo" date (not the real system date) in the auxiliary table CLOCK. The reason for making such date, and not use system one is to make it easy to generate scenarios (time traveling) to debug and test your project. CLOCK has only one tuple, inserted as part of initialization and is updated during time traveling. The the schema of this relation is the following.

- CLOCK (p_date)
  PK (p_date)
  Datatype: p_date: date

For all tasks, you are expected to check for and properly react to any errors reported by the DBMS (PostgreSQL), and provide appropriate success or failure feedback to the user. For example, function return 1 if the operation succeeded or -1 if the operation is not possible or failed. Further, be sure that your application carefully checks the input data from the user and avoids SQL injection.

Your application should implement at least the following tasks.

Task #1: Add a new store
Ask the user to supply all the necessary fields for the new store: *name*, *storeType*, *gpsLong*, and *gpsLat*. Your program must print the appropriate prompts so that the user supplies the information one field at a time.

Produce an error message if a store with the same name already exists.

Assign a unique store number for the new store.

Insert all the supplied information and the store number into the database.

Display the store number as a confirmation of successfully adding the new store in the database

Task #2: Add a new coffee to the BoutiqueCoffee menu/catalog
Ask the user to supply the *name*, *description*, *country of origin*, *intensity*, *price*, *award points*, and *redeem points*.

Assign a unique coffee ID for the new coffee.

Insert all the supplied information and the coffee ID into the database.

Display the coffee ID as a confirmation of successfully adding the new coffee in the database.

Task #3: Schedule a promotion for a coffee for a given period
Ask the user to supply the *promotion name*, *start date*, *end date*, and *coffee ID*.

Assign a unique promotion ID for the new promotion.

Insert all the supplied information and the promotion ID into the database.

Display the promotion ID as a confirmation of successfully scheduling the new promotion in the database.

Task #4: Add an offer/promotion to a store
Ask the user to supply the *promotion ID* and *store ID*.

Insert all the supplied information into the database.

Display the store ID as a confirmation of successfully offering the promotion in the database.

Task #5: List all the stores with promotions
Ask the user to specify if the query should list the stores promoting any coffee or for a specific coffee. In the latter case, the user should supply the *coffee ID*.

Display the stores offering the promotion or a message (e.g., 'No stores are currently offering any promotions' or 'No stores are currently promoting this coffee') if no stores have the specified promotions.

Task #6: Check if a given store has promotions
Ask the user to specify if the query should list any promotions at the specified store or only the offers for a specific coffee. In the first case, the user should supply the *store ID*. In the second case, the user should supply the *store ID* and *coffee ID*.

Display the promotions offered at the specified store or a message (e.g., ‘No promotions are currently offered at this store’ or ‘No promotions for this coffee are currently offered at this store’) if the provided store does not have the specified promotions.

Task #7: Find the closest store(s) for a given location (GPS)
Ask the user to specify if the query should list any closest store or the closest stores offering a specific promotion. In the first case, the user should supply a *GPS location (lat, lon).* In the second case, the user should supply a *GPS location (lat, lon)* and *promotion ID.*

In order to compute the closest store(s), you must use the Euclidean Distance:

$d(p, q) = \sqrt{(q_{lat} - p_{lat})^2 + (q_{lon} - p_{lon})^2}$

Display the closest store(s) for the provided *GPS location.* In the event of a tie, equidistant stores, all of the stores that are equidistantly the closest should be displayed.

Task #8: Add/Set a loyalty level
Ask the user to specify the required fields for creating or updating a loyalty level: *loyalty level name* and *booster factor.*

If the member level doesn't exist, the member level and its booster factor are inserted. If the member level exists, the booster factor is updated.

Display the name of this loyalty level as a confirmation of successfully inserting or updating the loyalty level in the database.

Task #9: Add a new customer
Ask the user to provide the required fields for creating a new customer: *first name*, *last name*, *middle initial*, *day of birth*, *month of birth*, *phone number* and *phone type.* The new user's loyalty level should be set to ‘basic’ since no reward points have been earned yet.

Display the customer ID as a confirmationof successfully adding the new customer in the database.

Task #10: Show the loyalty points of a customer
Ask the user to supply the *customer ID* to display the total loyalty points for.

Task #11: Produce a ranked list of the most loyal customers

Task #12: Add a purchase.
Ask the user to supply the *customer ID*, *store ID*, *time of purchase*, *purchased coffee ID(s)*, *purchase quantitie(s)*, and *redeem quantitie(s).*

If the purchase uses points, the purchase should fail if the customer does not have enough.

Display the purchase ID of this purchase to as a confirmation of successfully completing the purchase in the database.

Task #13: List the full coffee information in the BoutiqueCoffee menu/catalog Display the full coffee information offered in the BoutiqueCoffee menu/catalog or a message (e.g., ‘No items are currently offered by BoutiqueCoffee’) if no coffees are currently offered.

Task #14: List the IDs and names of all coffees of a specified intensity and each of which has two specified keywords in their names (i.e., both keywords in its name)
Ask the user to supply the *intensity* and both *keywords.*

Display coffee IDs of the provided intensity, each of which has both keywords in the coffee's name, or a message (e.g., ‘No coffees satisfied these conditions’) if no coffees satisfy the provided intensity or contain both keywords in the coffee's name.

Task #15: List the IDs of the top K stores having the highest revenue for the past X months
Ask the user to supply: k the K in top K, and x the timespan of month(s).

Revenue is defined as the sum of money that the customer paid for the coffees.

1 month is defined as 30 days counting back starting from the current date time.

Display the top K stores in the past X months in sorted order, with the store ID of the store having the highest revenue at the head.

If multiple stores have the same revenue, their order in the returned result can be arbitrary.

If multiple stores have the same revenue in the K^{th} highest revenue position, their store IDs should all be returned.

Task #16: List the IDs of the top K customers having spent the most money buying coffee in the past X months
Ask the user to supply: k the K in top K, and x the timespan of month(s).

1 month is defined as 30 days counting back starting from the current date time.

Display the top K customers in the past X months in sorted order, with the customer ID of the customer having spent the most money at the head.

If multiple customers have the same spending, their order in the returned result can be arbitrary.

If multiple customers have the same spending in the K^{th} highest spending position, their customer IDs should all be returned.

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 a pdf for the ER Diagram and an SQL component for the SQL DDL and INSERT statements of the project. Specifically,

- **er-model.pdf** the conceptual database design of the BoutiqueCoffee database (similar to HW2).
- **schema.sql** the script to create the database schema
- **sample-data.sql** the script containing all insert statements. 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,

• **schema2.sql**	the script to create the enhanced database schema with integrity constraint evaluation modes.
• **trigger.sql**	the script containing definition of the triggers, and any stored procedures or functions that you designed
• **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
• **README**	a README file that elaborates (fully) on how to use your *BoutiqueCoffee* client application, driver program, and benchmark tests.

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

1. Create one common private GitHub 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 TAs (GitHub ID: nixonb91, GitHub ID: yoshita120).

To turn in your ER diagram, **er-model.pdf**, for Phase 1:

1. Submit your **er-model.pdf** file to Gradescope under the Project ER Phase 1 assignment link.

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 TAs
3. Submit your GitHub repository (including all necessary SQL files) to Gradescope under the Project Code Phase X assignment link, where X is 1, 2, or 3.

   To submit to Gradescope, you will need to:

   - Select the appropriate assignment submission link (as you've previously done with homework assignments).

- On the “Submit Programming Assignment” window that appears, choose “GitHub.” If this is your first time submitting to Gradescope via GitHub, you will be prompted to link your GitHub account and authenticate.
- Select your team’s GitHub repository in the dropdown (searching will filter the repositories listed).
- Select the branch of your GitHub repository with the code your team wishes to submit (typically just the main branch). Then click the green upload button in the bottom left of the window.
- After uploading your team’s submission, you will be taken to the submission results page. The next step is to add each team member to the assignment to allow for a single linked submission. Note: **There should only be one submission per team, i.e., every team member does not need to submit**.
- To link team members, click “Add Group Member” in the top right corner of this page.
- On the new window, add all of your corresponding team members to the group, then hit the green “Save” button in the bottom left of this window.
- If done correctly, every team member should receive a confirmation email from Gradescope.

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.*

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!