CS 1555/2055 – DATABASE MANAGEMENT SYSTEMS (FALL 2022)
DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF PITTSBURGH

Project: Costa Express

Release: Feb 22, 2022
Phase1 - Due: Mar 16, 2022 @ 8:00pm
Phase2 - Due: Apr 12, 2022 @ 8:00pm
Phase3 - Due: Apr 20, 2022 @ 8:00pm

Project Goal

The goal of the semester group project of the database course is to allow groups of three students to become familiar with the development cycle of a database application (i.e., conceptual design, logical design, schema evolution, physical design, and implementation of an application).

In this project, you are given an analysis of the requirements of the supposed travel agency **Costa Express**, which specializes in organizing train journeys. Groups are asked to conceptually model the described miniworld and then implement it using the relational database Postgres. Implementation of the application should be done in Java using JDBC.

Phases

The implementation of the project will be divided into the following three phases:

Phase 1 - Database Design (30%):

- Conceptual Design Using ER Diagram (not EER)
- Specification of the Relational Schema in PostgreSQL (i.e., DDL)

Phase 2 - Database Implementation (40%):

- Data generation in DML
- Implementation of Operations (Queries, Transactions, and Reports) in DML

Phase 3 - Interface and Optimization (30%):

- Implementation of a User Interface for the System (text-based, non-graphical) to access the database functions
- Physical Design to optimize performance
- Documentation

Introduction

Costa Express is a new company in USA that is going to be active in the field of domestic travel within USA. The aim of the company is to establish an information system through which customers can be informed about stations, trains, routes, delays, etc.

Costa Express has created a large IT system that will consist of initially with 100 point of services, which will be managed by the company's users. Workstations will be linked to a PostgreSQL central relational database, which will store all relevant information. Users will be able to connect and interact with the database through a Java application (which can be easily run on different platforms and architectures). Such an application will enable users to perform various operations in a non-technical way (i.e., without using a database language such as SQL).

Data Description:

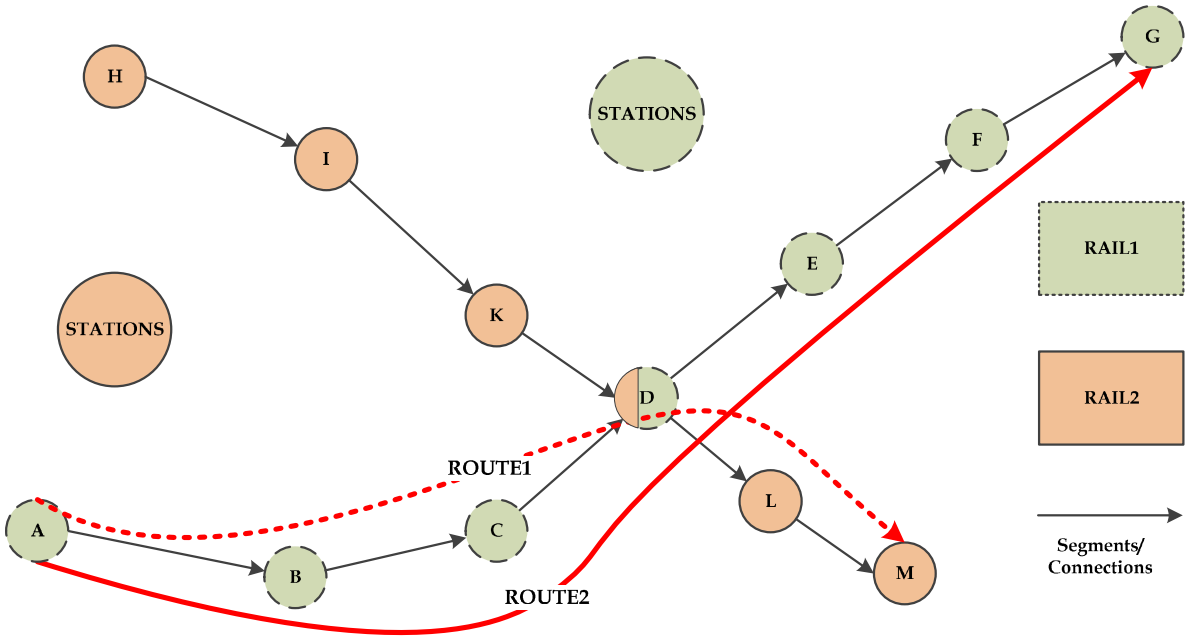

Figure 1: Graphical representation of the rail lines for your database

- Stations
  Costa Express has stations all over USA. For each station, you will need to store the unique station number, address, and hours of operation. The stations have a certain distance from each other (e.g., station A is 3 mi away from station B). Your application should model at least 500 stations.
- Rail Lines
  The rail lines can be considered to be the actual contiguous railway tracks used by the train. A line is a sequence of stations. In Figure 1 you can see how rail lines can be represented (one shown in orange circles with solid outline, one shown in green circles with dashed outlines). Each rail line is defined by the stations on its path, the distance between those stations, and the speed limit on the line. You can assume that each rail line is only a single set of tracks that can be traversed in either direction. Your application should model at least 5 rail lines.
- Routes
  Routes are paths down one or more rail lines. Each route passes through a number of stations, but may not stop at all of them. The red lines in the above graphic illustrate two different routes. Your application should model at least 500 different routes.
- Train Schedules
  A train can run along any route any day of the week (Monday-Sunday) one or more times per day. Each rail line can only be used by 1 train at a time. Your application should model at least 350 different trains. Your application should model at least 5000 different schedules (i.e., different route/day/time combinations).
- Trains
  For each train, you must track its top speed, the number of seats available, and the price per mile to ride that train. Your application should model at least 350 trains.

- Passengers
  Each passenger automatically receives a CustomerID the first time they enter the system. The system must also store the first name, last name, e-mail address, telephone number, and a structured form of the customer's home address. Note that passengers do not interact directly with the system. Instead, the passenger can inform an agent of the company to book the tickets on their behalf. Your application should model at least 300 passengers.
- 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

Description of Required Operations

The application should implement a system of interfaces (non-graphical) that will allow users to connect to the database and perform predefined functions.

- Passenger Service Operations
  1. Update customer list:
     (a) Customer Form:
     - Add / Edit / View client data. When adding each customer, a unique identification number should be assigned. In order to confirm the successful addition of the new customer to the base, the new customer identification number should be displayed on the screen.
  2. Find travel between two stations (4 ways to sort):
     (a) Single Route Trip Search:
     - All routes that stop at the specified Depart Station and then at the specified Destination Station on a specified day of the week.
     (b) Combination Route Trip Search:
     - All route combinations that stop at the specified Depart Station and then at the specified Destination Station on a specified day of the week.
     – Note that all trip searches must account for available seats, and only show results for trains that have available seats.
     – For each of the trip search options listed above, the following sorting options should be allowed. Note that each trip search should produce a paginated list of results (i.e., each trip search show produce 10 results at a time, with the option to grab the next 10).
       * Order based on price
       * Order based on total time
       * Order based on number of stops
       * Order based on number of stations

3. Add Reservation:
   - Book a specified passenger along all legs of the specified route(s) on a given day. The reservation is booked, but not ticketed. The passenger needs to pay in order to get the ticket.
4. Get Ticket (for a Reservation)
   - Ticket a booked reservation when the passenger pays the total amount .
5. Advanced searches
   (a) Find all trains that pass through a specific station at a specific day/time combination
   (b) Find the routes that travel more than one rail line
   (c) Rank the trains that are scheduled for more than one route.
   (d) Find routes that pass through the same stations but don't have the same stops
   (e) Find any stations through which all trains pass through
   (f) Find all the trains that do not stop at a specific station
   (g) Find routes that stop at least at XX% of the Stations they visit:
   - Find routes where they stop at least in XX% (where XX number from 10 to 90) of the stations from which they pass (e.g., if a route passes through 5 stations and stops at at least 3 of them, it will be returned as a result for a 50% search).
   (h) Display the schedule of a route:
   - For a specified route, list the days of departure, departure hours and trains that run it.
   (i) Find the availability of a route at every stop on a specific day and time
6. Other Operations:
   (a) Exit:
   - Exit from the program, which will lead the user back to the certification screen (login)

- Database Administrator
  1. Import Database:
     - Import data to the database
  2. Export Database:
     - Export data from the database
  3. Delete Database:
     - The program should ask the user to confirm the option to delete the database data. After confirmation, the program will delete all rows from all the tables in the database.

In the end, your application should include the individual main menu with all the above operations. For some features, the user will need to provide further information (e.g., for "Import data to Database", the user should specify the name of the file containing the values).

Triggers

In this phase you are asked to develop three triggers. You should feel free to develop more triggers to make your application more robust and efficient.

- You should create a trigger, called `line_disruption`, that adjusts all the tickets to the immediate next line when a line is closed due to an accident or maintenance, unless the customer specified no substitutions/adjustments in which case the ticket is canceled
- You should create a trigger, called `price_change`, that adjusts the reserver routes with pending payment. I assume you have ticketed and reserver routes when the cost per mile of a train is change.

- You should create a trigger, called `reservation_cancel`, that cancels all reservations if they haven't ticketed two hours before departure. This trigger uses the CLOCK table.

Physical Design

Although you have the ability to select whatever indexes you want, you have to declare the three prevailing cases and explain (in the documentation) exactly what indexes you chose, why you chose them and how you did them.

The above specifications may be further clarified, if necessary, during the second phase.

How to submit your project

- Email your Git commit ID and the full web link to your Git repository to cs1555-staff with all team members CC'd. In addition to submitting the Git commit ID via email to **cs1555-staff with all team members CC'd**, you must **add the TAs as collaborators** (Github usernames: ralseghayer, nasrinklt, and costantinos) to the Github repository. Make sure the GitHub repository is **private**. In your GitHub repository you are required to have the **three files specified in What to Submit**.
- Upload a zip file of your project through the Web-based submission interface you have used for previous Assignments. Note: **Only one team member** should submit the files through the Web-base submission interface since this is a group assignment.
- Submit your files by the due date of each phase. **There is no late submission.**
- **It is your responsibility to make sure the project was properly submitted.**

Project Phases and Deliverables

This project will be implemented in groups of three students who are expected to contribute equally in time and substantial work. Under no circumstances any separation of the design or implementation of the database will not be accepted (e.g., having one student do only phase 1 while another did only phase 2 is unacceptable). All team members will have to deal with all stages of the work.

Your project will be collected by the TAs via the GitHub repository that you have shared and as a zip file of your project through the Web-based submission interface you have used for previous assignments. To turn in a project phase, you must do three things by the 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".
2. Push that commit to the GitHub repository that you have shared with the instructor and TAs.
3. Send an email to the instructor and the TAs (cs1555-staff) with all team members CC'd and the title "[CS 1555/2055] Project phase X submission" that includes a link to your repository on GitHub and the Git commit ID for the commit that should be graded. The commit ID is a 40 character string (a SHA-1 hash) and can be found on GitHub or as part of the output of the "git log" command.

What to Submit

1. Phase 1 - **Due Date: (8:00pm, Mar. 16, 2022)**
   (a) Your project git repository and the zip file should include the following deliverables in the following folders:
   (b) er/ - In this folder, you should present a set of file that would comprise a complete ER model to represent the miniworld described above.
   (c) sql/ - In this folder, you should create a series of SQL files that contain the DDL statements needed to create the tables necessary to represent the miniworld as outlined in your DDL files.
   (d) team01.tar or team01.zip:
   A packed or compressed file that contains all project files with the above structure.
2. Phase 2 - **Due Date: (8:00pm, Apr. 13, 2022)**
   (a) Your project git repository and the zip file should include the following deliverables in the following folders:
   (b) sql/ - In this directory, you will need to add the SQL files, which will be text files that will store all commands (SQL-DML, Stored Procedures, etc.) that need to be implemented in the database.
   (c) data/ - In this directory you will need to save the .dat files, which will be text files that will store your application data. Such files should be able to import to your database. You may find it helpful to use scripts to generate this data. You must be sure to generate sufficient data as described above.
   (d) team01.tar or team01.zip:
   A packed or compressed file that contains all project files with the above structure.
3. Phase 3 - **Due Date: (8:00pm, Apr. 20, 2022)**
   (a) Your project git repository and the zip file should include the following deliverables in the following folders:
   (b) src/ - In this directory, save the source code of the Java application and any accompanying libraries required to successfully compile your application. It is recommended to include a Makefile that will compile your application.
   (c) doc/ - Create a user manual for your application (about 2-3 pages in PDF format) that will give you instructions on how to use your application, the basic choices in the design of the database, a description of the additional functions that have been implemented, various difficulties, etc. This document also shows the system constraints and the possibilities for improving it.
   (d) Additionally, you should create whatever indices on the data you feel necessary and helpful to improve the performance of your application. Include the justifications for whatever indices you create in the documentation.
   (e) team01.tar or team01.zip:
   A packed or compressed file that contains all project files with the above structure.

You must verify the login name and password of all users at the beginning. By default, we assume a pre-existing administrator with a login name “admin” and a password “root”. This information can be inserted into your table after you create the database. You can add yourselves as the first customers.

Group Rules

- It is expected that all members of a group will be involved in all aspects of the project development. Division of labor to data engineering (db component) and software engineering (JAVA component) is not acceptable since each member of the group will be evaluated on both components.
- The work in this assignment is to be done *independently* by each group. Discussions with other groups 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.

Grading

The project will be graded on correctness, robustness (error-checking, i.e., it produces user-friendly and meaningful error messages) 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.

Enjoy your class project!