CS 1555
Database Management Systems

Group Term Project

Project Goal

The goal of the semester group project of the database course is to allow groups of **2** people 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 **ExpressRailway**, 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.

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)
- Implementation of the Relational Schema in Postgres (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

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

ExpressRailway has created a large IT system that will be manned initially with 100 workstations, which will be managed by the company's users (database administrators). Workstations will be linked to a Postgres 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).

After your successful graduation from the University of Pittsburgh, **ExpressRailway** has decided to contract your team to design and implement the database that will support the above application. Therefore, you will work both as Database Administrators (DBA) and Database Application Developers.

Data Description

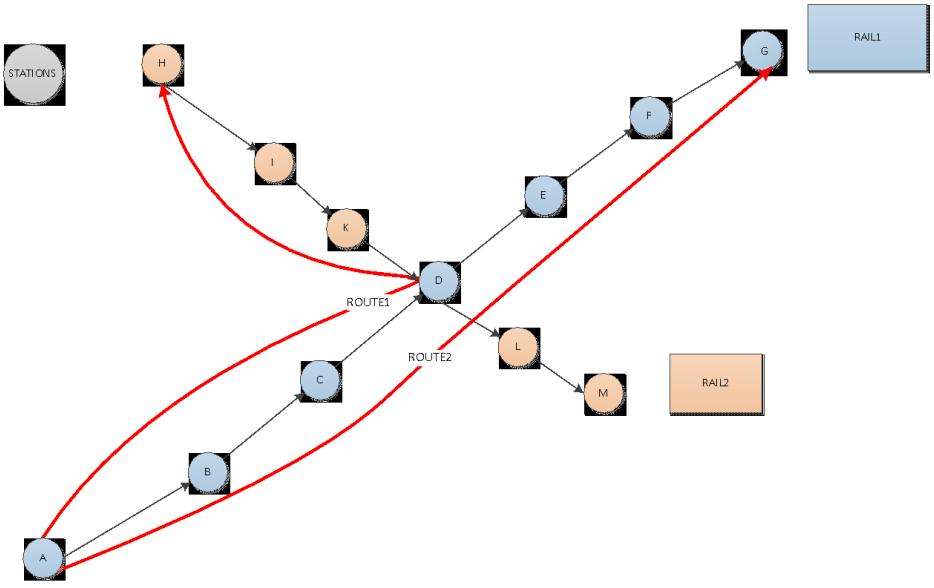

Figure 1: Graphical representation of rail lines

1. **Stations**
   ExpressRailway has stations all over America. 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 50 stations.

2. 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 we can see how rail lines can be represented (one shown in orange, one shown in blue). 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.

3. 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.

4. 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 300 different trains. Your application should model at least 2000 different schedules (i.e., different route/day/time combinations).

5. 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.

6. 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 the agent of the company to book the tickets on their behalf. Your application should model at least 300 passengers.

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.

1. Passenger Service Operations

1.1. Update customer list:

1.1.1. **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.

1.2. Finding a trip between two stations:

1.2.1. **Single Route Trip Search:** All routes that stop at the specified *Arrival Station* and then at the specified *Destination Station* on a specified day of the week.

1.2.2. **Combination Route Trip Search:** All route combinations that stop at the specified *Arrival Station* and then at the specified *Destination Station* on a specified day of the week.

1.2.3. Note that all trip searches must account for available seats, and only show results for trains that have available seats.

1.2.4. 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 if needed).

- 1.2.4.1. Fewest *stops*
- 1.2.4.2. Run through most *stations*
- 1.2.4.3. Lowest price
- 1.2.4.4. Highest price
- 1.2.4.5. Least total time
- 1.2.4.6. Most total time
- 1.2.4.7. Least total distance
- 1.2.4.8. Most total distance

1.2.5. **Add Reservation:** Book a specified *passenger* along all legs of the specified *route(s)* on a given day.

1.3. Advanced searches

1.3.1. **Find all trains that pass through a specific station at a specific day/time combination:** Find the trains that pass through a specific station on a specific day and time.

1.3.2. **Find the routes that travel more than one rail line:** Find all routes that travel more than one rail line.

1.3.3. **Find routes that pass through the same stations but don't have the same stops:** Find seemingly similar routes that differ by at least 1 stop.

1.3.4. **Find any stations through which all trains pass through:** Find any stations that all the trains (that are in the system) pass at any time during an entire week.

1.3.5. **Find all the trains that do not stop at a specific station:** Find all trains that do not stop at a specified station at any time during an entire week.

1.3.6. **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).

1.3.7. **Display the schedule of a route:** For a specified route, list the days of departure, departure hours and trains that run it.

1.3.8. **Find the availability of a route at every stop on a specific day and time:** Find the number of available seats at each stop of a route for the day and time given as parameters.

1.4. Other Operations:

1.4.1. **Exit:** Exit from the program, which will lead the user back to the certification screen (login)

2. Database Administrator

2.1. **Import Database:** Import data to the database

2.2. **Export Database:** Export data from the database

2.3. **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).

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.

Project Phases and Deliverables

This project will be implemented in groups of 2 people 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 TA via the GitHub repository that you have shared. 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 TA.
3. Send an email to the instructor and the TA with the title "[CS1555] 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.

1. **Phase 1**
    a. Your project git repository should include the following deliverables in the following folders:
        i. **er/** - In this folder, you should present a set of file that would comprise a complete ER model to represent the miniworld described above.
        ii. **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

2. **Phase 2**
    a. Your project git repository be augmented to include the following deliverables in the following folders:
        i. **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.
        ii. **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.

3. **Phase 3**
    a. Your project git repository be augmented to include the following deliverables in the following folders:
        i. **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.
        ii. **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.
    b. 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.

Note that your TA will evaluate the results of each phase and provide feedback that should be addressed in the following phases. The deadlines for each project phase will be posted to the class website.

GOOD LUCK!