CS1555 - Project Report

Jacob Shaji[†], Aidan Walsh[†], Lichang Chen[†]

Abstract. Our project report will include 6 parts: user guide, assumptions and basic choices in the design of database, important functions in implementation, physical design, difficulties, constraints and possibilities for improving.

1. User Guide (ReadMe)

There are 18 choices in our user interface and the first thing user need to do is choosing what they want to do and type in a number. Next, we will elaborate the functions we have and how to use it.

1. Update the customer list: there are 3 choices in this function: **A**. a new customer can be added. Type in all the information(First Name, Last Name, Street Address, City, State and Zip) as the instructions show then you can add a new customers. **B**. Update the customers' information. In this part, you will need type in the customer id to do the update. **C**. View a customers' entire data. Use the id or (First Name, Last Name, Street Address, State and Zip) to view the entire data of the customer.
2. Single route trip: you need to enter 3 information including the day of the week(Monday, Tuesday etc.), departure stop, and the destination stop. You can choose the format where the data ordered, by station, by stops etc.
3. Combination 2 route trip: works the same as single-trip, but finds all routes to destination from departure that can use two routes.
4. Make reservation: make sure you have the customer ID and the train schedule ID to book a reservation. If not, you can go to the previous and search the route you want.
5. Get ticket. You will need reservation ID to get the ticket of your reservation.
6. Find all trains that pass through a specific station. You need to enter station ID and the day of the week to get the results.
7. Find the routes that travel more than one rail line.
8. Rank the trains that are scheduled for more than one route. We rank by the number of routes.
9. Find routes that pass through the same stations but don't have the same stops. You will need to input the route ID to do the search.
10. Find any stations through which all trains pass through.
11. Find all the trains that do not stop at a specific station.
12. Find routes that stop at least at XX of the Stations they visit.

[†] equal contribution.
Email: `jas675@pitt.edu`, `ajw137@pitt.edu`, `lic138@pitt.edu`.

13. Display the schedule of a route.
14. Find the availability of a route at every stop on a specific day and time
15. Exit. You will enter the login screen.
16. Export Database. You will need the administrator access to use it. You just need to type in the schema you want to export.
17. Delete Database. You will need the administrator access to use it.
18. Update Clock. You will need the administrator access to use it.
19. Trigger 2 Test. Testing Feature for testing line disruptions trigger. You will need the administrator access to use it.
20. Import Data. Creates Tables, Adds Data, Defines Functions by reading the SQL files. The positioning of the SQL file in relation to the java file is important. You will need the administrator access to use it.

2. Basic Choices in the Design of DB (Assumptions)

We will split the assumption of phase 1 and phase 2. About the phase 1, we have the following assumptions:

- All non pseudo tables has a primary ID.
- A station has a station ID.
- A stations hours of operation is in 24 hours format eg. ”23:59-23:59”.
- Rail Line is defined as a line segment with two stations. One as start and one as end station.
- If a Rail Line contains more than two stations, the other subset of rail line can be referenced to itself.
- A Route is made up of a rail line.
- A reservation can only be held for a certain amount of time.
- A string data type has appropriate length limitations.
- The Rail Line follows a recursive relation with itself.

We decided to redo significant parts of our schema for the second phase of the project. In using intermediate tables rather than arrays, we sought to make our database more in line with the normal forms; our relation should be in 3NF now (or at close to it). In order to generate data for the intermediate tables, we created a python script that transformed the seemingly array-based data into a large sequence of insert statements, which can now be found in new_schema.sql. Additionally, we changed some attributes of our schema to be more in line with the data given to us.

The following shows some changes of assumptions we make in phase 2:

- There will be no alerts when each trigger is triggered.

- Disruptions happen to a train schedule. In that case, a the original reservation is changed to the new train schedule. If paid, it will stay paid and there is no change in reservation ID.
- In the case of disruptions, the trigger will find a new train schedule with following the same route(s). If no such route is found, the reservation is cancelled and the price is refunded implicitly.
- Since no data was given about the exact dates on the days of the week, hard coded date values were used to align with the pseudo-clock
- A flat-rate is calculated for each route not matter where the passengers' destination is. The price is calculated using the per kilometer rate attribute in train table.
- If stations aren't directly next to each other in the route's list of stations provided, they are at least on the same line, or directly connecting lines.
- For counting the number of stops in the route searches, we count the departure and destination stops.
- Sorting by 'total time' means the total time travelling, not the time waiting at the stops.
- A route traversing multiple lines means that at least one 'segment' (not station) of the line must be on another line
- Only credentials for the Employees and Administers were implemented. The Employees and Administers are though to be reasonably educated about the program and will not make any input errors.

3. Important Functions in Implementation

We implement *route_time* functions to compute the time for a specific train going from station A to station B which helps a lot in finding the trains at the station on a specific day and time. *Line_segment* is also very useful, which can get the rail-line ID and distance between two stations (which are next to each other). From Line_segment, we can get *segments*, which returns an array of Line segments.

4. Physical Design

Most of the data is searched using their primary keys which helps reduce the resulting table which in turn helps with memory and performance. Because it is unlikely that adding a index will change the performance of this database by a high margin, only basic Physical Design techniques like memory usage reduction is implemented.

5. Difficulties

A lot of the route stuff was done before simplifications were announced. Rather than fixing the data to a large degree, we wrote SQL function that dealt with path finding stuff, which was challenging.

6. Constraints and Possibilities for Improving

Our function *find_availability* execute slowly(about 10 seconds). There are two ways to improve this. One way is that in *find_availability* function we choose another implementation of find the trains passing the specific station in the specific day and time. Another way is to improve the *trains_at_station* function. (We use this function in our *find_availability* function. Actually, for each station, we use this function once. Thus, if we can optimize this, then the speed will be enhanced a lot)

Acknowledgements

All 3 group members participate 3 phases work from database design, dml and sql code writing to the java code writing.