## INSTRUCTIONS

1.) Open class3 server
2.) Copy 'RailWay.java', 'ParameterizedQueries.java', 'CombinationLeg.java',and the 'run' script to class3
3.) Type './run'
    a.) Application automatically compiles and runs with jar
4.) Use numbers on keyboard to navigate through the menus

## DATABASE ADMINISTRATOR OPERATIONS

1.) Import Database
    a.) How to use: Select the import database option to restore the database from a previous backup. This will first delete all the rows from all the tables then import the backup from the `Database` folder. If the Database folder does not exist or any of the csv files within the database folder are missing the database will not be cleared out and the import will not proceed.
2.) Export Database
    a.) How to use: Select the export database option to store all of the tables as csv files in a `Database` folder located in the same directory as the java program. This will create a `Database` folder if it is not yet created as well as overwrite any prior backups so be sure to move them if you want multiple backups.
3.) Delete Database
    a.) How to use: Select the delete database option to delete all rows from all tables in the database

## DATABASE DESIGN CHOICES

1.) We designed our database to include a 'stop' entity. A stop is basically a composite of two stations. We made this in conjunction with route so that route and stop would have a many to many relationship. This way, many routes could share the same stops, making it easier to select any routes

that share common stops. This is probably our only design feature that differs from the standard implementation.

a.) See Indices

## ADDITIONAL FUNCTIONS

1.) We built a recursive function to sort all of the stops on a route into a linked list esque table. This made testing everything a breeze because we knew what results we should expect from each query.
2.) Our test data comes from a real life transit system - Southeastern Pennsylvania Transportation Authority. We thought this would make it much more intuitive and easy to visualize for both ourselves and the grader. We used 'webscrape.py' to gather all of the different stations and line that run through Southeastern Pennsylvania. We did end up changing train top speeds and distances to account for the large amount of test data needed. SEPTA could never run that many routes in one day.
3.) We decided to implement pagination in the java application instead of the sql application in order to reduce the number of sql operations needed. When a response needs to be paginated, we keep all the results in the java application, while supplying the next 10 as the user asks for them
4.) We added a script to automatically run the application to make everything easier

## DIFFICULTIES

1.) One of the hardest parts of the project for us was understanding what everything meant (routes, rail lines, etc..). There were a lot of words that started with the letter R so it got pretty confusing. Other than that, I think that one of the challenging parts of database design is that you need to make really good choices in the beginning in order to yield good results in the beginning. Since we were also working on two other CS group projects, we had to make executive decisions in terms of database design and stick with them, in the interest of time.
2.) Another difficulty was creating all of the combination route queries as well as processing the results in java.

## SYSTEM CONSTRAINTS

1.) When adding a passenger to a schedule, the application makes an update on schedule. There is a trigger on before update for schedule that checks the trains capacity as well as the current amount of seats taken. If the seats taken is equal to the capacity, the application will not add the passenger to the schedule, and will print “No results - either route doesn't exist or train is full. check availability”.
2.) Trains can’t run on the same track at the same time. We made a trigger in phase1.sql that checks the speed and distance of the trains to make sure they are not on the same track at the same times.

## POSSIBLE IMPROVEMENTS

1.) Usability - the application is not the most user friendly thing. This could be improved by making this application web based, with better scheduling graphics. Also, the route ids mean nothing to a regular person, but we didn’t design this for a regular person to used. We designed it for a teacher to test and grade it. We would’ve loved to spend more time on usability and design if we had more time.
2.) Make train speeds and acceleration more realistic. Our trains go very fast over a short distance to make sure that we could fit 500 routes on 5 lines. If this were to be actually used, there would be more realistic train speeds.
3.) Authentication on DBA operations. This prevents someone from dropping the database without permission.

## INDICES - PREVAILING CASES

1.) "route_stop_pk" PRIMARY KEY, btree (stop_id, route_id)
    a.) Fast access for finding routes with the same stops
        i.) Why: Stops are shared between routes because stops are composites of Station A ID and Station B ID
        ii.) How: DDL

(1) CONSTRAINT Route_Stop_PK

(2) PRIMARY KEY(Stop_ID, Route_ID)

2.) "unique_station_combo" UNIQUE CONSTRAINT, btree (station_a_id, station_b_id)

a.) Fast access to stops table

i.) Why: Stop table index on unique stops consisting of station a and station b. Allows fast access of Stop Table because it is used in mostly every query. This also allows for faster joining on the stop table between routes because routes utilize subsets of stops.

ii.) How: DDL

(1) CONSTRAINT Unique_Station_Combo

(2) UNIQUE (Station_A_ID, Station_B_ID)

3.) "schedule_pk" PRIMARY KEY, btree (schedule_id)

a.) Fast access to schedule table

i.) Why: schedule table index on schedule id. Allows fast access of Schedule Table because it is also used in mostly every query

ii.) How: DDL

(1) CONSTRAINT Schedule_PK

(2) PRIMARY KEY (schedule_id)