Setup:

1. ddl
2. customer_form
3. is_multi_line
4. views
5. triggers
6. speed_train
7. common_line
8. dist_same_line
9. find_transfer_to_line
10. find_dist_same_route
11. Is_on_line
12. distance_table
13. has_seats
14. book_trip
15. find_dist from_same_line
16. hours_to_stop
17. num_stations_num_stops
18. price
19. single_route_trip_search
20. multi_route _search
21. dml

How to run:

First, move the postgresql-42.2.18.jar to the same folder and add it to the library if needed. Make sure you change the username and password fields to be the same ones you used to connect to the dataGrip server.

Windows

- ./makeWindows.sh

Linux

- ./makeLinux.sh

All Commands:

When the program is first run, you will need to either create an account or login to one. We do not have any default users created. User account passwords are hashed through Java, so the default administrator account must be created through the Java Interface. For the purpose of the demo, when creating accounts, administrator status can be set using True/False. To create the default administrator account, run the following inputs.

- ‘create’
- admin root True

You will need to input ‘list’ in order to see what commands are available. Below are more in-depth explanations as to what each command does and its inputs. All inputs have a single space between them and all multi-word inputs are surrounded by double quotes.

1. ‘add’ - Allows the passenger to book a reservation on a route at a day and time listed on the schedule. The expected inputs are as follows:
    a. Customer_no: The customer’s unique customer number as given to them when added to the database
    b. Dep_no: The station number that the customer is departing from
    c. Arrival: The station number that the customer is getting off the train at
    d. Route: The route number assigned to the train
    e. Day: The day of the week the trip is to be scheduled for
    f. Tiime: The time at which the train is expected to depart
    g. ifSubstitution: Whether or not the customer wants a substitution in the case that the current line is closed due to an accident or maintenance.
        i. True: the ticket will be automatically moved to the next line if the current line is canceled.
        ii. False: if a line is closed, the ticket will be canceled.
2. ‘advanced’ - Takes the user to a list of commands that contain more advanced searches
    a. 'a' - Find all trains that pass through a specific station at a day/time combination
        i. Station_no: the station number
        ii. Day: day of the week
        iii. Time: the time it passes through the station
    b. 'b' - Find the routes that travel more than one rail line
        i. No input required
        ii. It lists up to 10 routes at a time. In order to see more, input next,
        iii. ‘Exit’ will take you back to the main menu
    c. 'c' - Rank the trains that are scheduled for more than one route
        i. No input required
        ii. It lists up to 10 trains at a time. In order to see more, input next,
        iii. ‘Exit’ will take you back to the main menu
    d. 'd' - Find routes that pass through the same stations but don't have the same stops
        i. No input required

    e. 'e' - Find any stations through which all trains pass through
        i. No input required
    f. 'f' - Find all the trains that do not stop at a specific station
        i. Station_no: the station number
        ii. It lists up to 10 trains at a time. In order to see more, input next,
        iii. ‘Exit’ will take you back to the main menu
    g. 'g' - Find routes that stop at least at XX% of the stations they visit
        i.
    h. 'h' - Display the schedule of a route
        i. Route_no: the route number that you want the schedule for
    i. 'i' - Find the availability of a route at every stop on a specific day/time
        i. Day: day of the week
        ii. Time: time at which the train passing through departs
        iii. It lists up to 10 trains at a time. In order to see more, input next,
        iv. ‘Exit’ will take you back to the main menu
3. ‘customer’ - Allows the user to add passengers, edit information about passengers, or view information about a customer
    a. ‘add’ - add a customer
        i. fName - customer’s first name
        ii. lName - customer’s last name
        iii. street - street address (place in double quotes if it is multiple words)
        iv. city - city the customer resides in
        v. Zipcode - zipcode of the area the customer lives in
    b. ‘edit’ - update customer information. If specific pieces of information have not changed, type them in as they already are.
        i. Customer_no - the number assigned to the customer (returned by the customer add command)
        ii. fName - customer’s first name
        iii. lName - customer’s last name
        iv. street - street address (place in double quotes if it is multiple words)
        v. city - city the customer resides in
        vi. Zipcode - zipcode of the area the customer lives in
    c. ‘view’ - displays all stored information about a customer
        i. Customer_no - the number assigned to the customer (returned by the customer add command)
4. ‘Exit’ - exits out of the program
5. ‘Routesearch’ - . Finds travel between two stations
    a. 'type' - the type of route search to be performed. Will be either
        i. ‘Single’ - All routes that stop at the specified Depart Station and then at the specified Destination Station on a specified day of the week.
        ii. ‘Combination” - All route combinations that stop at the specified Depart Station and then at the specified Destination Station on a specified day of the week.
    b. 'depart_station_no' - the starting station

   c. 'dest_station_no' - the destination station
   d. 'Day_of_week' - the day of week
6. 'Ticket" - Ticket a booked reservation when the passenger pays the total amount
   a. 'Resv_no' - the number returned when the reservation was booked.
7. 'Admin' - administrative actions
   - If current user logged in is an administrator, (specifies when you log in), they have the option to execute these functions
     - 'Delete' - delete the entire database, will ask user to verify beforehand

Assumptions

- When marking a ticket as paid, we assumed that the person had paid the full price since we did not have them type in how much they were paying
- When booking, it is assumed that the data is actually present
  - For example, the system says success after trying to book a ticket with '1 1 6 22 Sunday 02:28:00 false' even if there are no trips that occur on that route at the day and time, but the ticket will not actually be booked
- Customers are not allowed allowed to create stops or stations
- Trains are never delayed and always arrive on time
- If a line is down for construction, if a customer choose not to substitute, they automatically get their money back

Improvements:

We would have improved the interface a bit more as well as condensing some of the sql files down. When we were writing the sql, the separate files allowed us to organize our thoughts better but as we were determining the order to run files in, it became a hassle to sift through the files to find the precise order.

A similar issue occurs with the seats on a train. We did not track seats between stations but rather on the train as a whole. Booking a ticket for a specific day and time subtracts from the total number of tickets for a train and not for the individual day as it should.

The user account took a lot of set up and the only administrative function we could successfully implement on time was the delete. We did not set up the import/export.