How to run this application?

If this is the first time running, go to **sql/** folder:

1. run **init_db.sql** to create all schemas and triggers
2. run **init_func.sql** to create all commands
3. run **insert_data.sql** to create all initial data

Then go to **src/** folder:

1. To compile java files, enter in command line prompt
   make app
   or
   javac -cp postgresql-42.2.5.jar ExpressRailwayDriver.java
   Both are equaivalent.

2. To run java applications, enter in command line prompt
   java -cp .:postgresql-42.2.5.jar ExpressRailwayDriver

How does the program look like?

There are three main screens: login screen, data administrator screen, and passenger service screen. The login screen provides the options to switch to the two other screens, and **exit the program**. The other two provide corresponding functions, and **the option to go back to login screen.**

This means that if you want to exit the program in the correct way, you need to return to login screen first, and then quit the program.

Other implementation details

Most operations in this project can be concluded as the following three frequent cases:

1) to find route/routes that stop/pass through at certain station/stations
2) to search for some schedules with time/distance constraints
3) to find station/stations that some routes stop/pass through.

Consider two examples:
finding similar routes is case 3 + 1, where we are checking if stations two pass are the same, while they stop at different stations.

Searching for route combo is case 1 + 3 + 2 where we want one route to stop at the origin, the other at the destination. We also need to know what rail line the train is currently on. **This is important because different rail lines have different speed limit. We need to make sure that whenever the train switch to another rail, use the new rail's speed and distance info.** This requires us to know 1) where the transfer station is between rails 2) based on the station history, figure out the direction so that we know what previous station is and what distance value in the database to use. Finally, such combo must exist in schedule and timely possible.

Therefore, to optimize for the above three cases,
I create index for the following:

```
create index idx_leg on express_railway.legs(route_id, sid);
create index idx_schedule on express_railway.schedules(route_id);
create index idx_station on express_railway.stations(sid);
create index idx_train on express_railway.trains(tid);
create index idx_rail on express_railway.trails(rid, sid);
```

Most operations consider routes finding, so route id is indexed. Station id is also indexed to provide faster look up for matching stations. When calculating time and distance, they are related to specific rails and trains. Therefore, I also created index on rails and trains.