We have planned a Courier Company Management System where the cargos can be tracked and stored in a computer system. Therefore, the paths, carriers and specific information on the deliveries can be easily accessed and many reports can be generated for future improvements.

The tables and the relations between them are designed to avoid data redundancy. Thus, the data aimed to be always up to date and one change would be enough for the system to be actual.

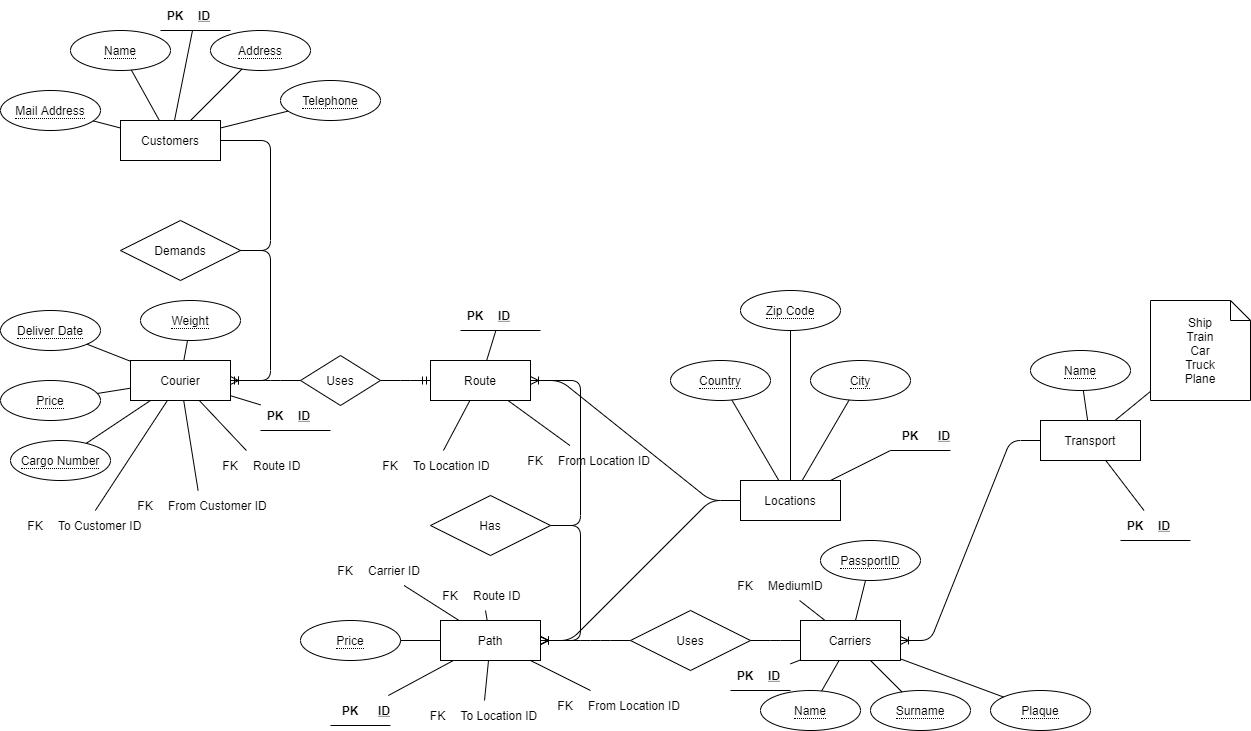


Figure 1-ER Diagram

There are seven entities in our solution.

* Customers: Stores customer information and has relation with Courier entity.
* Courier: Responsible for storing cargo information like; weight, price, deliver date and cargo number. Also has relation with Customers and Route entities.
* Route: Has data for source and destination of cargos. It has relation with Path and Courier and Locations entities. There can be direct transfer of cargos from source to destination or there might be a transfer to other locations before arriving destination.
* Path: Stores paths of cargos. It uses carriers for transfers and gets location info from Locations entity.
* Locations: Stores the data of locations. It has country, city, and zip code attributes. Path and Route entities get location data from this entity.
* Carriers: This entity stores data of carriers like; name, surname, plaque, and passport. It has relation with Transport entity where it gets the vehicle type information.
* Transport: Responsible for storing transport types.

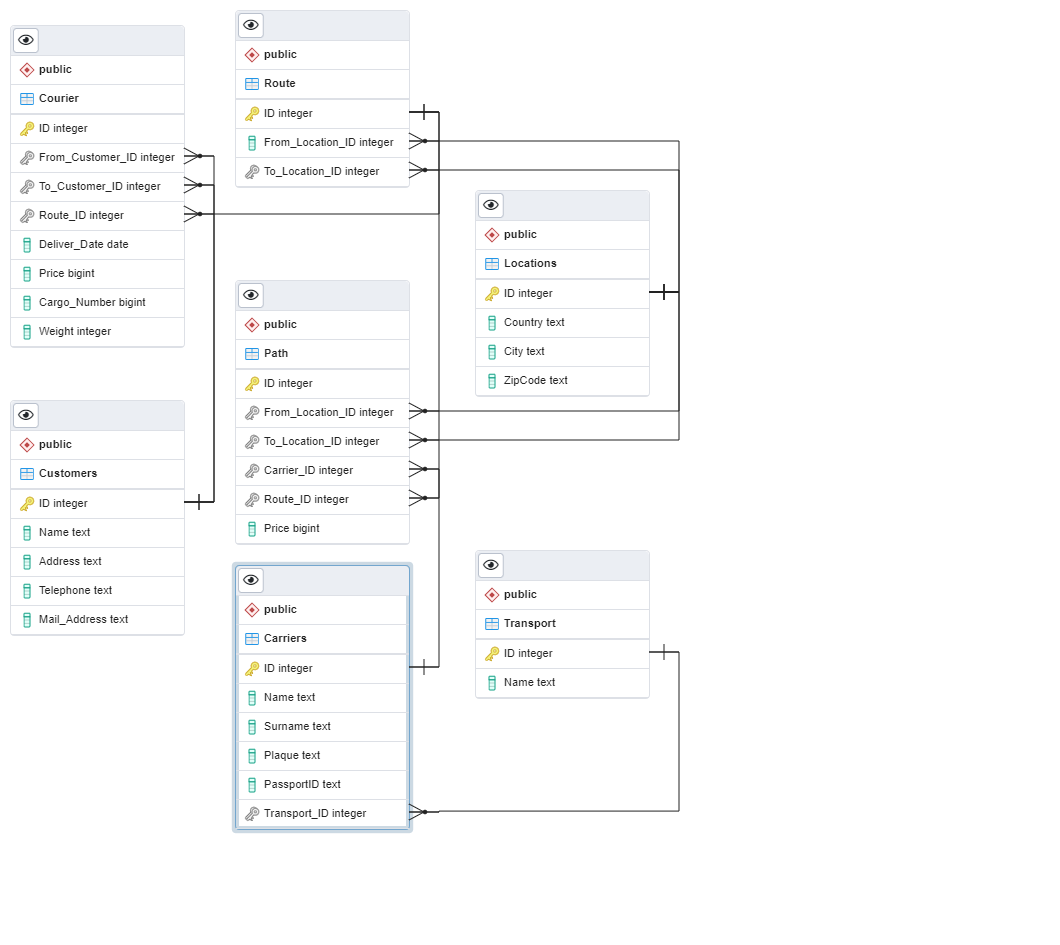


Figure 2- Relational Schema

During the development phase of the lifecycle, we have used PostgreSQL v13 as database server and Eclipse as IDE. To achieve the Java to PostgreSQL connection, *pgJDBC* installed and *postgresql-42.2.18.jar* file added to project as referenced library.

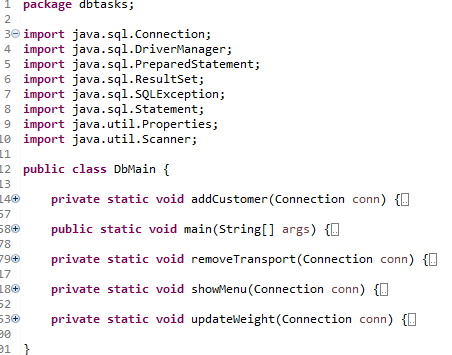


Figure 3- Java source view

All the code is in DbMain.java file. The main method is the entry place for the application. The database connection is implemented and tested here.

String url = "jdbc:postgresql://localhost/CourierCompanyManagementSystem";

Properties props = **new** Properties();

props.setProperty("user", "postgres");

props.setProperty("password", "password");

Connection conn = DriverManager.*getConnection*(url, props);

After a successful connection is established, *showMenu (Connection conn)* is called with connection parameter. This method shows the interface to user for selection.

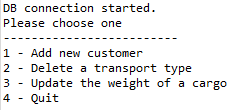


Figure 4- Console UI

The right method selected according to the selction.

**switch** (selection) {

**case** 1:

*addCustomer*(conn);

**case** 2:

*removeTransport*(conn);

**case** 3:

*updateWeight*(conn);

}

AddCustomer method inserts a customer to Customers table. Before the insertion, the UI asks for necessary information for fields and finally insert the newrecord to database. After the insertion the method shows the up to date list of customers.

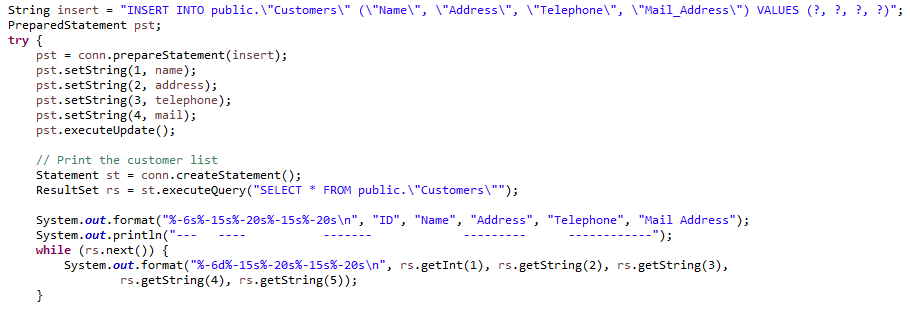


Figure 5- AddCustomer method

RemoveTransport method deletes a transport type from Transport table. This method firstly shows the list of transports and asks the user to enter the ID of the transport to be deleted. After the deletion, it shows the up to date list of transports again.

String delete = "DELETE FROM public.\"Transport\" WHERE \"ID\"=?";

pst = conn.prepareStatement(delete);

pst.setInt(1, id);

pst.executeUpdate();

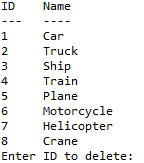


Figure 6- removeTransport menu

3rd choice updates the weight of a cargo entry. The method asks for a cargo number first. If the cargo number is not found, menu keeps on asking the cargo number until a valid is found in the table. When the right cargo number found, the menu asks for the new weight of the record. After the successful update of the record, the current list of the cargos is shown to the user.

String update = "UPDATE public.\"Courier\" SET \"Weight\"=? WHERE \"Cargo\_Number\"=?";

pst = conn.prepareStatement(update);

pst.setInt(1, weight);

pst.setLong(2, cargo\_number);

pst.executeUpdate();

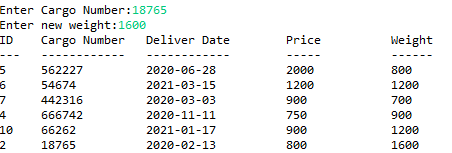


Figure 7- Cargo weight update