Courier Management System - Group BM

**Document History**

| **Version Number** | **Date** | **Author/Owner** | **Description of Change** |
| --- | --- | --- | --- |
| 1 | 2/3/2022 | Valli Priya M | Inital RS Document |
| 2 | 2/3/2022 | Valli Priya M | Added Problem Statement |
| 3 | 5/3/2022 | Vaishnavi S | Add EER Diagram |
| 4 | 29/3/2022 | Vaishnavi S | Added Use Case Diagram |
| 5 | 2/4/2022 | Vaishnavi S | Updated ER Diagram Based On Feedback From Team |
| 6 | 22/4/2022 | Vedant S | Updated RS Document To Match Specified Format |
| 7 | 23/4/2022 | Vedant S | Updated RS Document To Add SQL Queries, Procedures, Updated Problem Statement, Updated ER Diagram and Relational Model |
| 8 | 26/4/2022 | Varenya Varshney | Added Demo Document; Object Model Diagram; Final Review |

# Team Members and their Responsibilities

| **BITS ID** | **Team Members** | **Responsibility** |
| --- | --- | --- |
| 2021MT13019 | Varenya Varshney | Problem Statement, Requirement Documentation, Relational Schema, Java Application development, SQL Queries, Stored Procedures |
| 2021MT93254 | Valli Priya M | Initial RS Document, Contributed to Forward Shipment Module |
| 2021MT13185 | Vaishnavi S | Entity-Relationship Model, Use Case Diagram, Stored Procedure, Contributed to Manage Agent module |
| 2021MT93065 | Vedant Shetty | Requirement Documentation, RS Document, Frontend Application, Contributed to Manage Employee module |
| 2021HT66007 | Velidi Rahul | Relational Schema, Contributed to Book Shipment module |
| 2021MT13057 | Varun Mediratta | Object Model, Contributed to Deliver Shipment module |
| 2021MT13083 | Varsha Basavaraj | Relational Schema, Contributed to Book Shipment module |
| 2021MT93061 | Veera Rajesh Kommoju | Object Model, Contributed to Deliver Shipment module |

# Problem Statement & Requirements Definition

## Courier Management System

We have decided to create a database for a courier delivery company such as Delhivery or Blue Dart. Customers are billed by a rate card.

The purpose of the database is to manage the complete lifecycle of a courier shipments, track its history and store the rate card to calculate delivery costs within the application.

Customer gets a quote by entering source pincode, destination pincode, shipment dimensions (length, width, height and weight). He likes the quote and goes to the nearest agency to book his shipment. Agent verifies the route and price details, receives booking amount from the customer and books the shipment. He shares the receipt with the customer which has consignment number in it which can be used for all tracking purposes. Shipments from across all the agencies are moved to the local branches at regular interval. Shipments can also arrive at an intermediate branch which falls in the pre-determined route between a given source and a destination pincode. Employee at a branch receives the shipment and based upon the route, he either forwards it to another branch via one of the transportation media viz. Air, Railway, Road or if it is the home branch of the destination address, then he forwards it into the bucket of delivery agents. Agents at every branch timely check their buckets for any fresh or undelivered shipments. Agent picks the shipment from his bucket and attempts its delivery at the destination address. Finally, a shipment either gets delivered or left undelivered. Agent updates the final status of the shipment along with optional remarks.

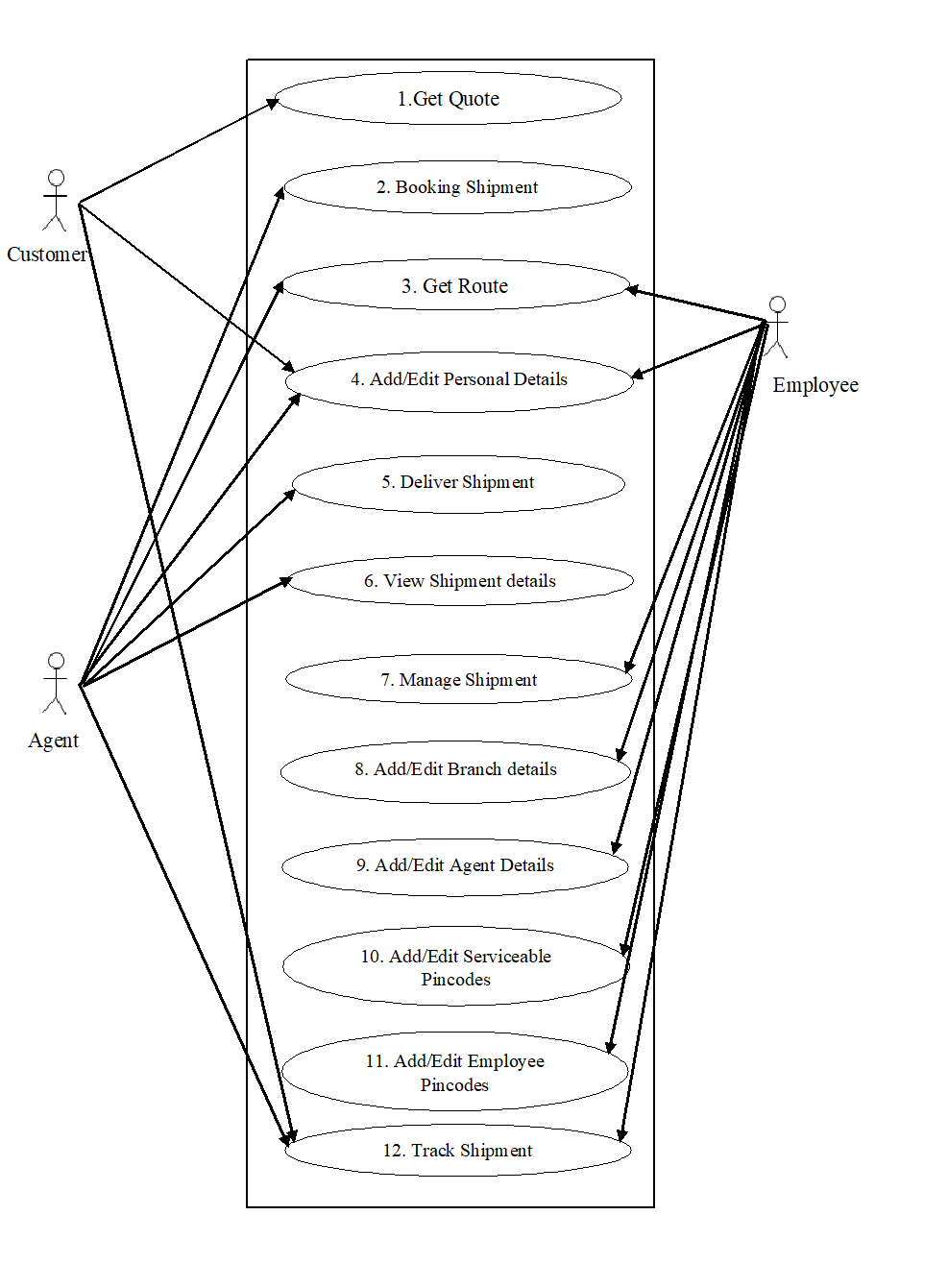
### Assumptions

* Distance between any two pincodes is generated randomly.
* Route between any two pincodes is generated randomly.
* To calculate the booking amount, a rate card is chosen which looks like this:

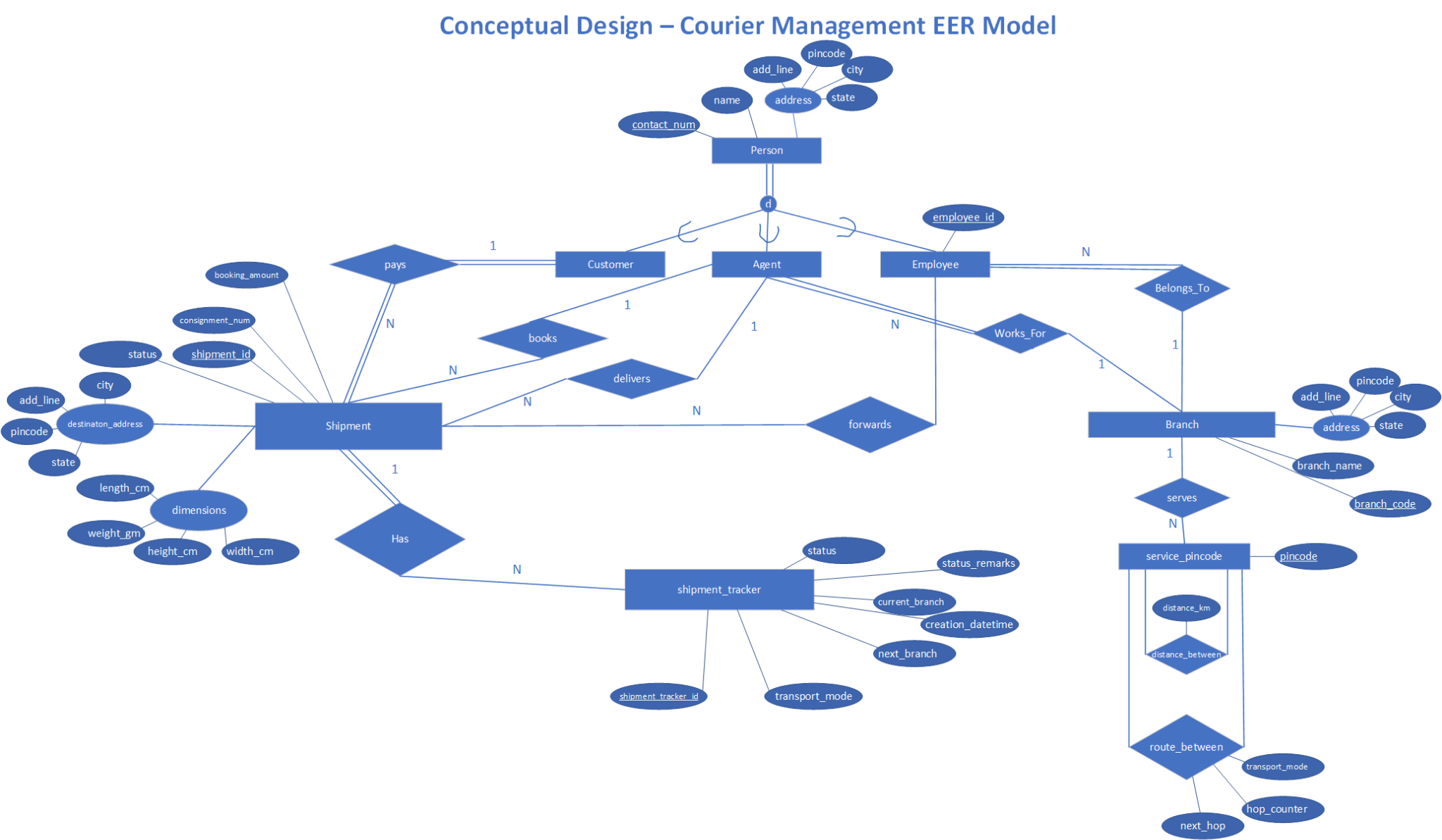
|  |  |  |
| --- | --- | --- |
| **Rate Card** | | |
| **Distance** | **Base Rate** | **Extra Weight Factor** |
| 0 - 50Km | 50 | ₹ 10 every 1KG |
| 51-100Km | 100 | ₹ 10 every 1KG |
| 101-500Km | 150 | ₹ 10 every 1KG |
| >500 Km | 200 | ₹ 20 every 1KG |

Metric Shipping Factor (MSF) = 5000 cm3/kg; DIM Weight = Package Volume/MSF; Weight = Max (Actual Weight, DIM Weight)

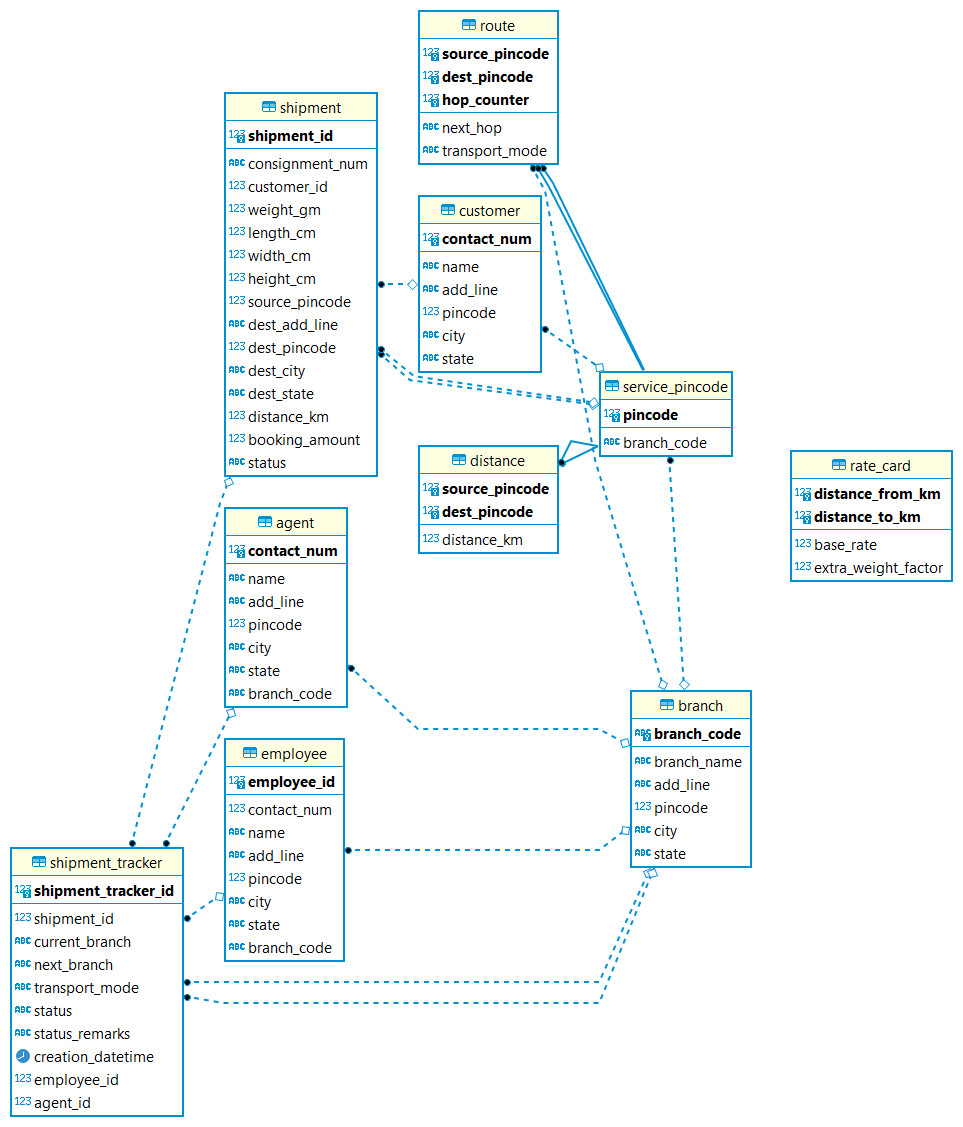
## Use Case Diagram



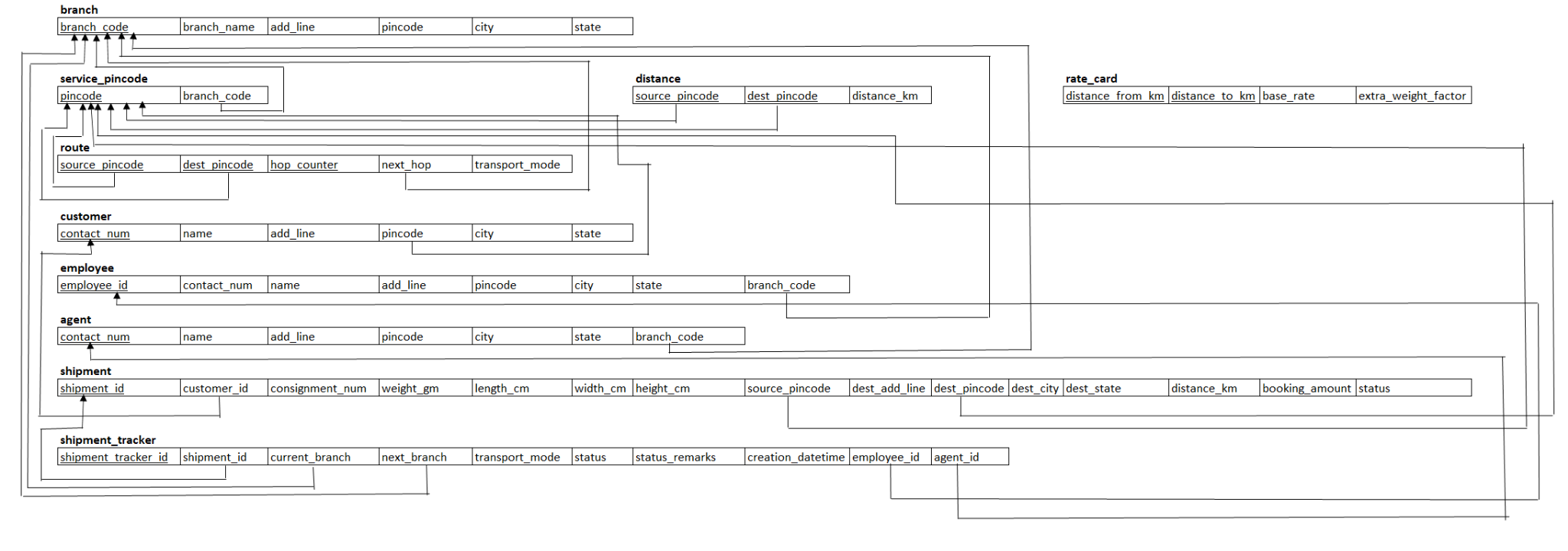
# Entity Relationship (EER Model)



# Object Model Diagram



# Relational Database Schema



# Normalization

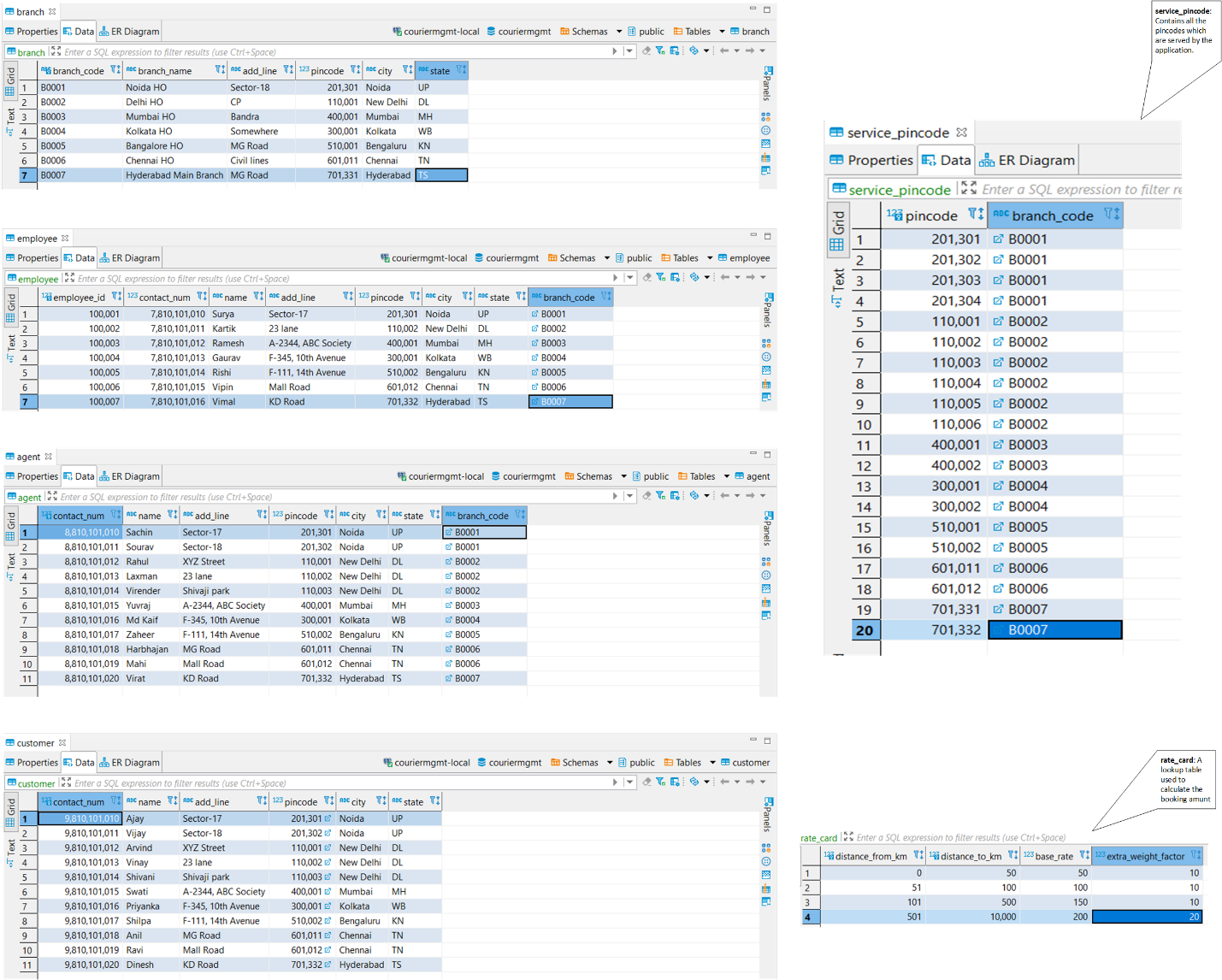
|  |  |
| --- | --- |
| Table Name | branch, service\_pincode, distance, route, customer, agent, employee, agent, shipment, shipment\_tracker, rate\_card |
| State | 3NF |
| Reason | No multivalued attributes, No partial dependencies, No transitive dependency |

# Table Definitions and Data Contents

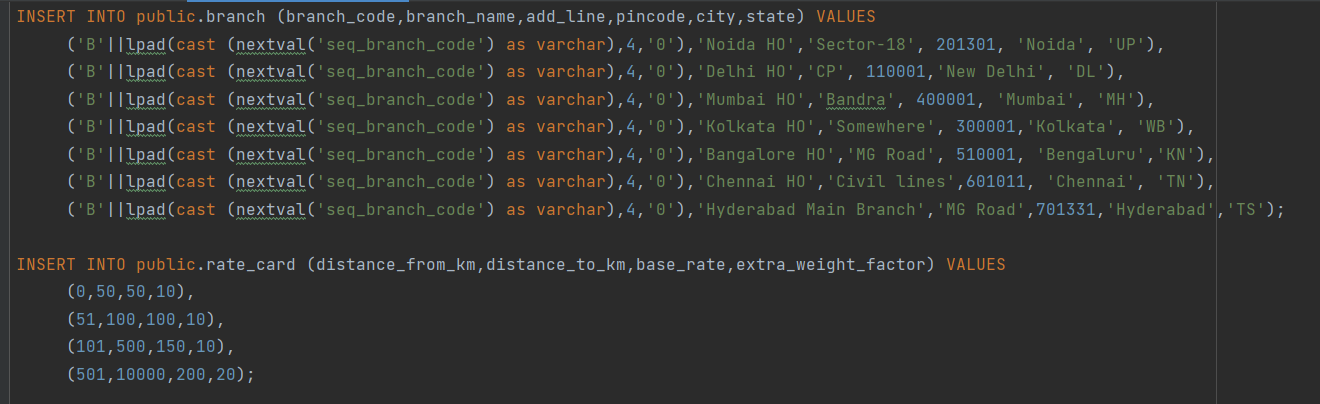
## Table Definitions

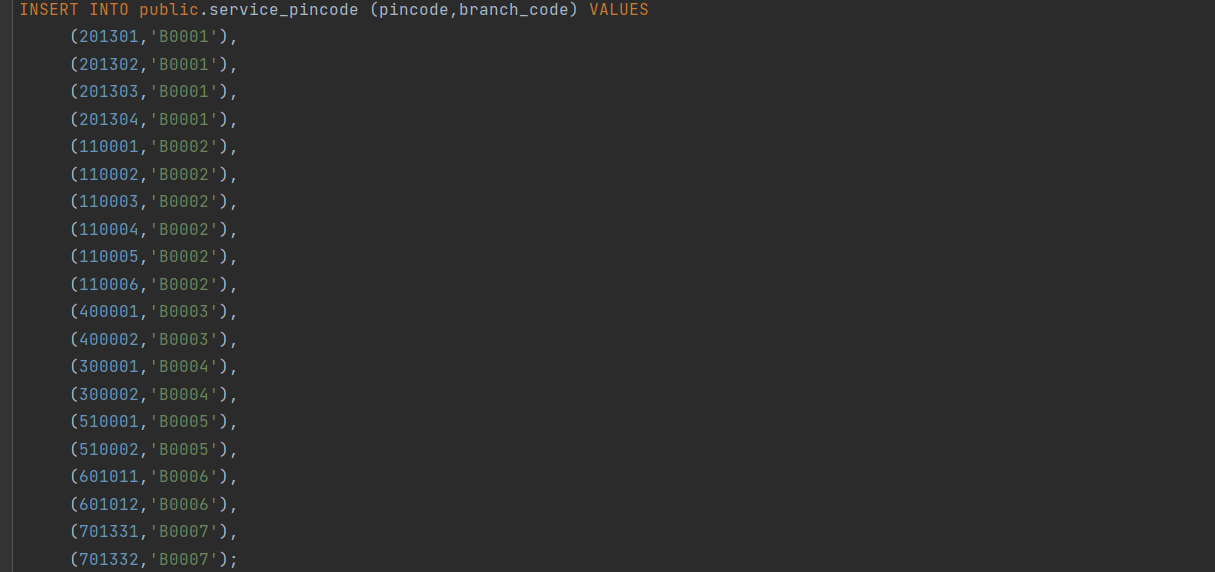
[Table Definitions are listed under [SQL Statements >> DDL Statements](#_DDL_Statements) section]

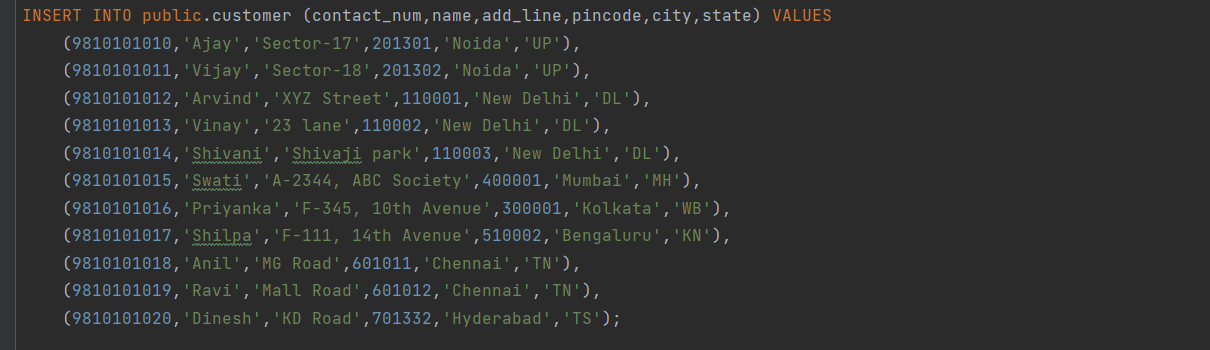
## Data Contents

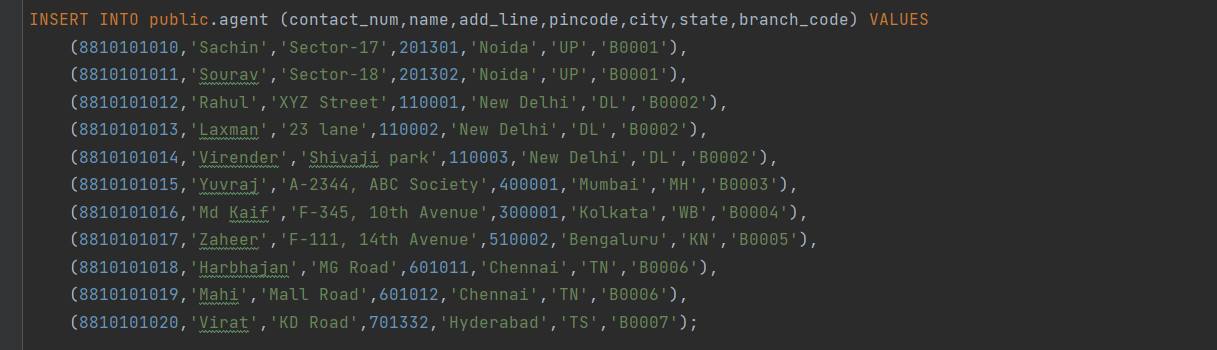


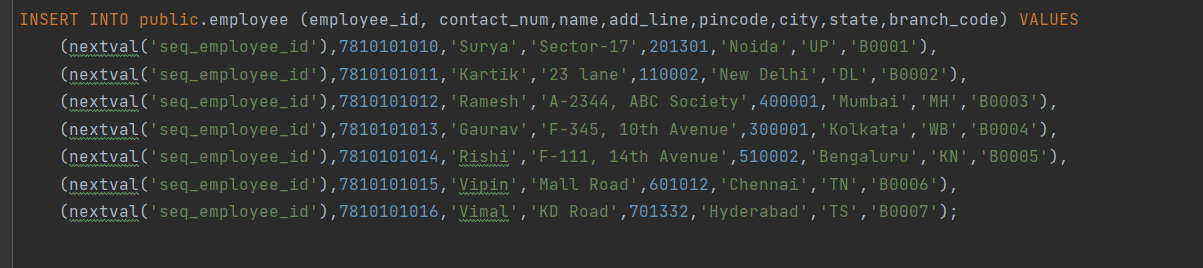
**This data is pre-loaded before the application starts-up using below Insert Queries:**











# SQL Statements

## DDL Statements

### Enums

|  |
| --- |
|  |

### Branch

|  |
| --- |
|  |

### Route

|  |
| --- |
|  |

### Actors

|  |
| --- |
|  |

|  |
| --- |
|  |

### Shipment

|  |
| --- |
|  |

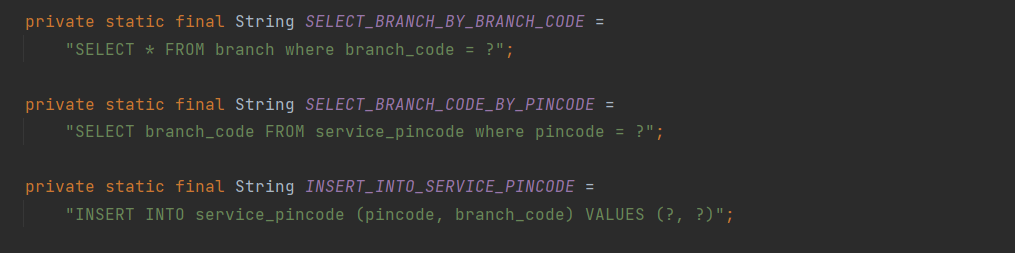
|  |
| --- |
|  |

|  |
| --- |
|  |

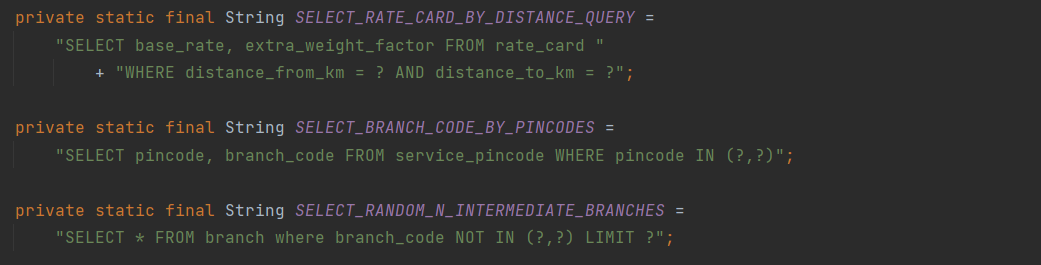
## DML Statements

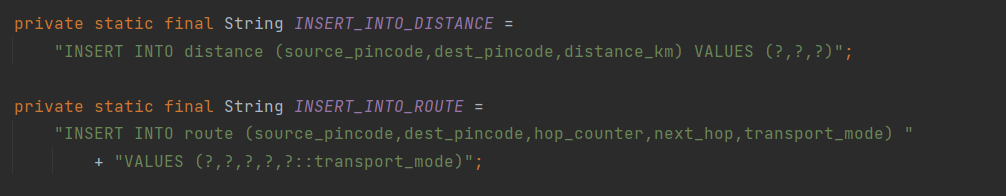
Depending upon the use-cases, this application includes various DML SQL Statements for manipulating application data. These SQLs are called from the Java REST APIs which is discussed in detail under [User Interface](#_User_Interface_and) section.

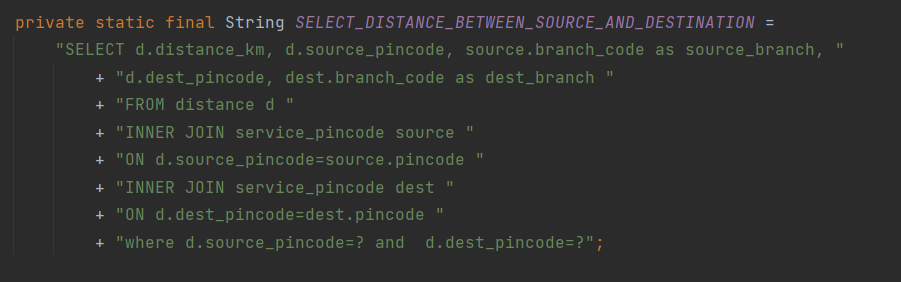
### Branch

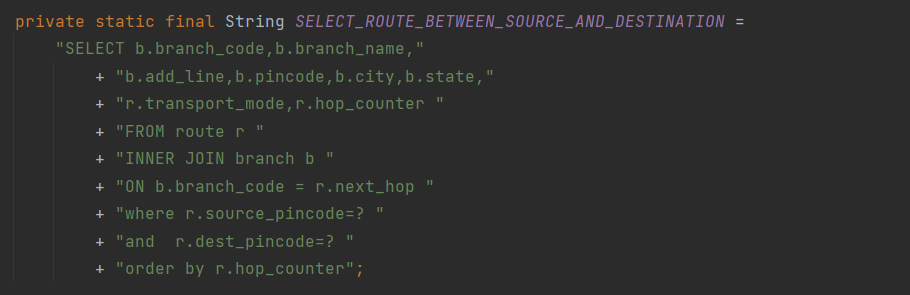


### Route

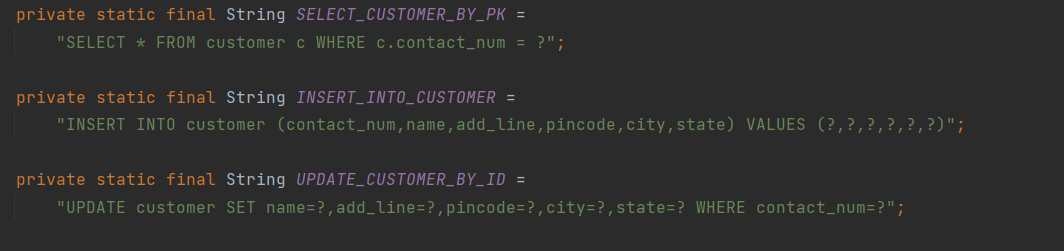


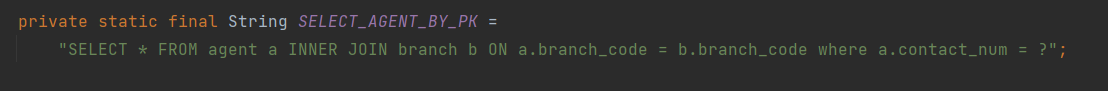


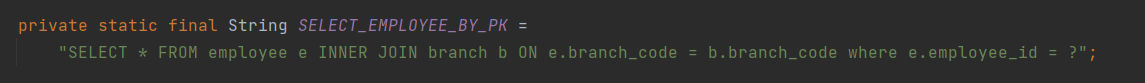




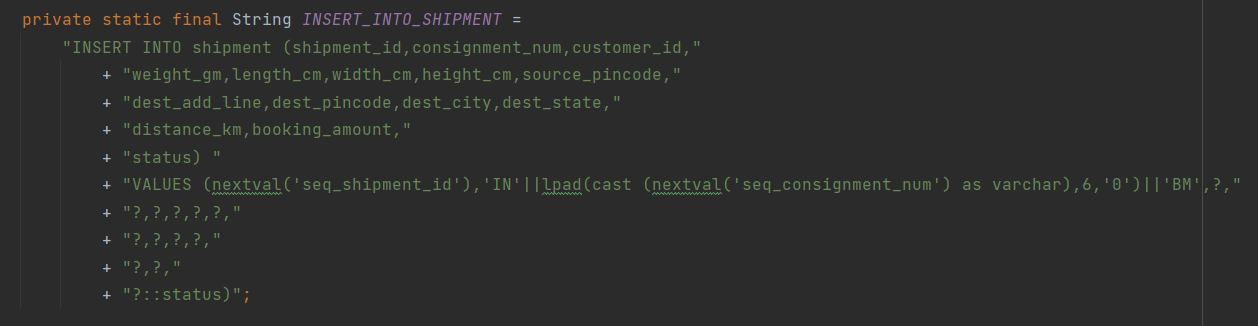
### Actors

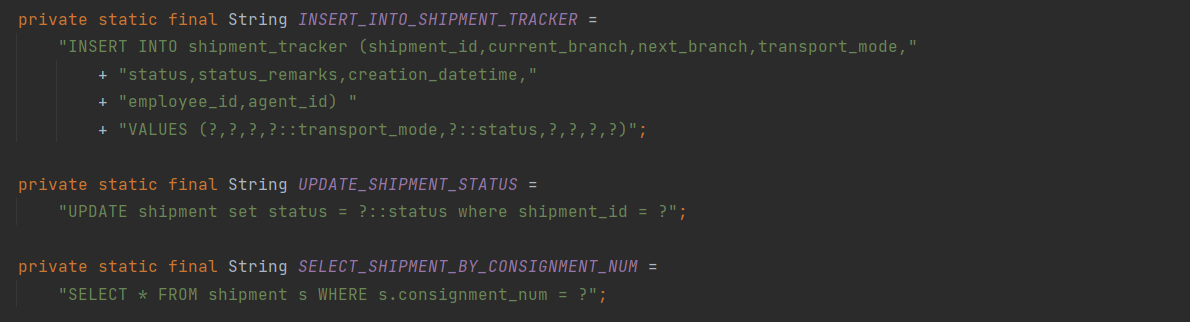


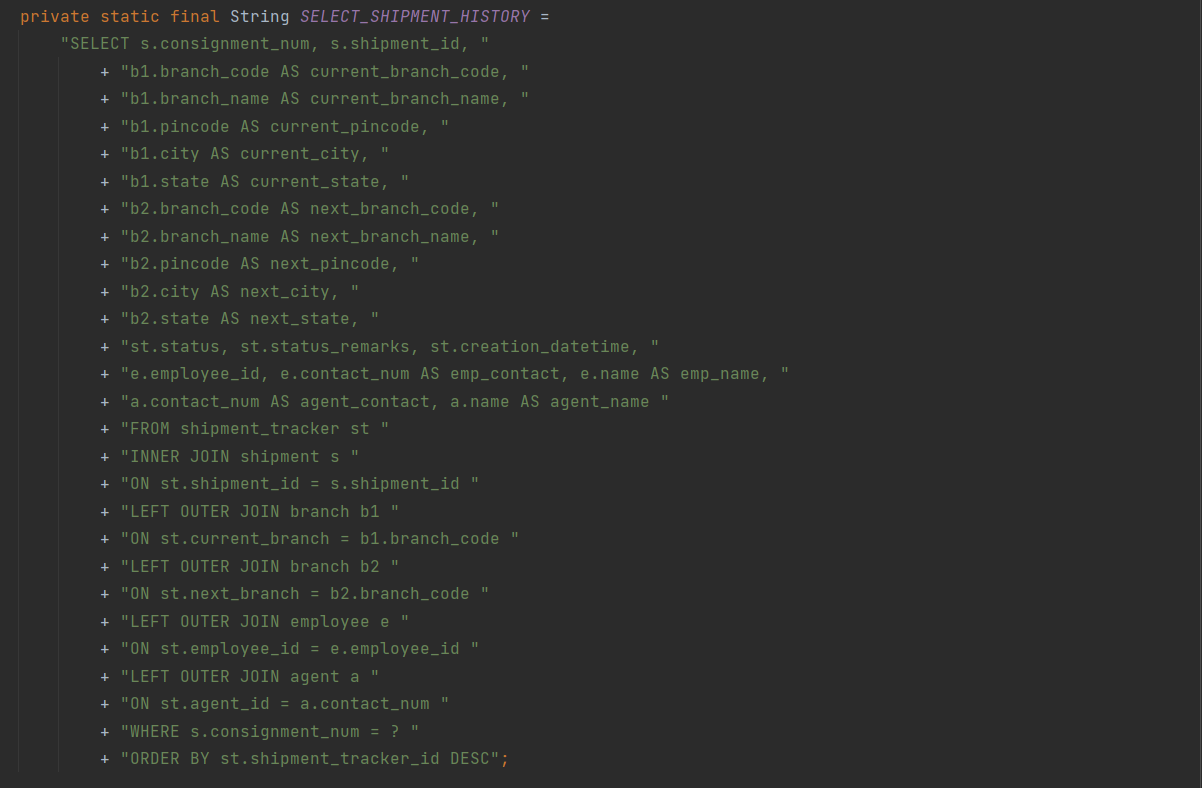




### Shipment







# Stored Procedures/ Triggers

## Procedure to Insert or Update Branch

|  |
| --- |
| create or replace  procedure upsert\_branch  (  p\_error inout VARCHAR,  p\_is\_insert in BOOLEAN,  p\_branch\_code inout branch.branch\_code%type,  p\_branch\_name in branch.branch\_name%type,  p\_add\_line in branch.add\_line%type,  p\_pincode in branch.pincode%type,  p\_city in branch.city%type,  p\_state in branch.state%type  )  language plpgsql  as  $$  declare  -- No variable declarations at this time.  begin  if p\_is\_insert  then  p\_branch\_code = 'B' || lpad(cast (nextval('seq\_branch\_code') as varchar), 4, '0');  insert  into  branch  (  branch\_code,  branch\_name,  add\_line,  pincode,  city,  state  )  values (  p\_branch\_code,  p\_branch\_name,  p\_add\_line,  p\_pincode,  p\_city,  upper(p\_state));  insert into service\_pincode (pincode, branch\_code) values (p\_pincode,p\_branch\_code);  else  update  branch  set  branch\_name = p\_branch\_name,  add\_line = p\_add\_line,  pincode = p\_pincode,  city = p\_city,  state = upper(p\_state)  where  branch\_code = p\_branch\_code;  update service\_pincode set pincode = p\_pincode where branch\_code = p\_branch\_code;  end if;  exception  when others then  p\_error := sqlerrm;  end  $$; |

## Procedure to Insert Or Update Agent

|  |
| --- |
| create or replace  procedure upsert\_agent  (  p\_error inout VARCHAR,  p\_is\_insert in BOOLEAN,  p\_contact\_num in agent.contact\_num%type,  p\_name in agent.name%type,  p\_branch\_code in agent.branch\_code%type,  p\_add\_line in agent.add\_line%type,  p\_pincode in agent.pincode%type,  p\_city in agent.city%type,  p\_state in agent.state%type  )  language plpgsql  as  $$  declare  -- No variable declarations at this time.  begin  if p\_is\_insert  then  insert  into  agent  (  contact\_num,  name,  branch\_code,  add\_line,  pincode,  city,  state  )  values (  p\_contact\_num,  p\_name,  p\_branch\_code,  p\_add\_line,  p\_pincode,  p\_city,  upper(p\_state));  else  update  agent  set  branch\_code = p\_branch\_code,  name = p\_name,  add\_line = p\_add\_line,  pincode = p\_pincode,  city = p\_city,  state = upper(p\_state)  where  contact\_num = p\_contact\_num;  end if;  exception  when others then  p\_error := sqlerrm;  end  $$; |

## Procedure to Insert Or Update Employee

|  |
| --- |
| create or replace  procedure upsert\_employee  (  p\_error inout VARCHAR,  p\_is\_insert in BOOLEAN,  p\_employee\_id inout employee.employee\_id%type,  p\_contact\_num in employee.contact\_num%type,  p\_name in employee.name%type,  p\_branch\_code in employee.branch\_code%type,  p\_add\_line in employee.add\_line%type,  p\_pincode in employee.pincode%type,  p\_city in employee.city%type,  p\_state in employee.state%type  )  language plpgsql  as  $$  declare  -- No variable declarations at this time.  begin  if p\_is\_insert  then  p\_employee\_id = nextval('seq\_employee\_id');  insert  into  employee  (  employee\_id,  contact\_num,  name,  branch\_code,  add\_line,  pincode,  city,  state  )  values (  p\_employee\_id,  p\_contact\_num,  p\_name,  p\_branch\_code,  p\_add\_line,  p\_pincode,  p\_city,  upper(p\_state));  else  update  employee  set  contact\_num = p\_contact\_num,  branch\_code = p\_branch\_code,  name = p\_name,  add\_line = p\_add\_line,  pincode = p\_pincode,  city = p\_city,  state = upper(p\_state)  where  employee\_id = p\_employee\_id;  end if;  exception  when others then  p\_error := sqlerrm;  end  $$; |

# User Interface and Database connectivity

Project is designed and implemented as a web application which exposes the functionalities into set of different RESTful APIs. These APIs are built using Java/Spring MVC framework. Primary user interface to access these APIs is a simple OpenAPI REST client. Using this rest client, users can build and execute the HTTP request which internally accesses the database and then return an HTTP response.

This REST client is well-documented so that any user can easily access these APIs and get the desired response.

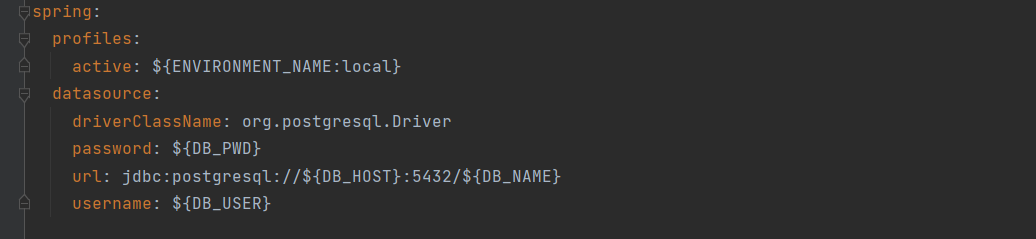
Other than this, a front-end application is also being implemented which is being built using Angular framework. It is consuming the REST APIs.

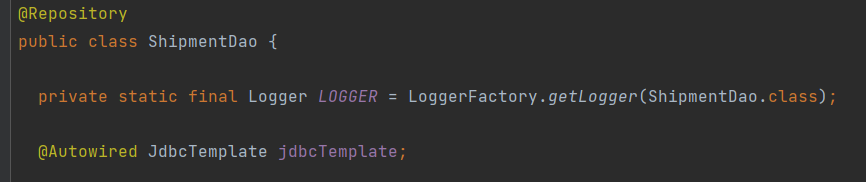
## Database Connectivity

Since it is a Java/Spring application, Spring JDBC is used to setup the database connectivity.

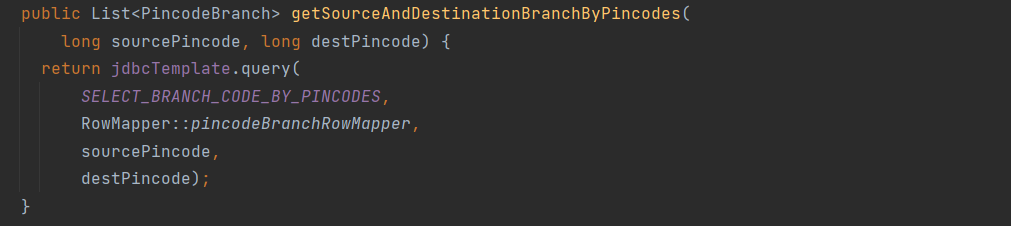
### DB Connection Details

Using these connection details, JdbcTemplate bean is instantiated which holds the DB session. This template is then used to execute SQL statements.



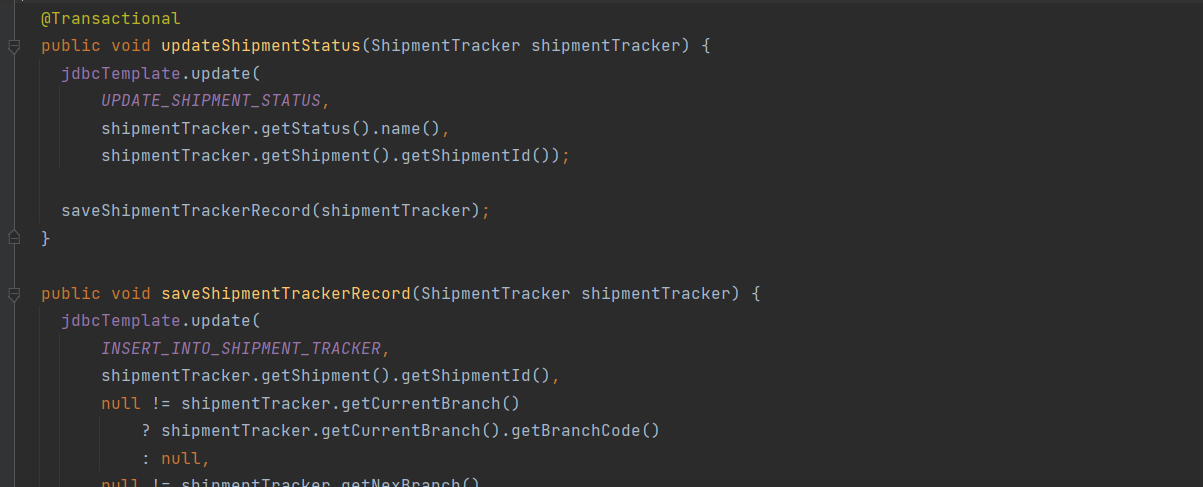


### SQL Read Operations

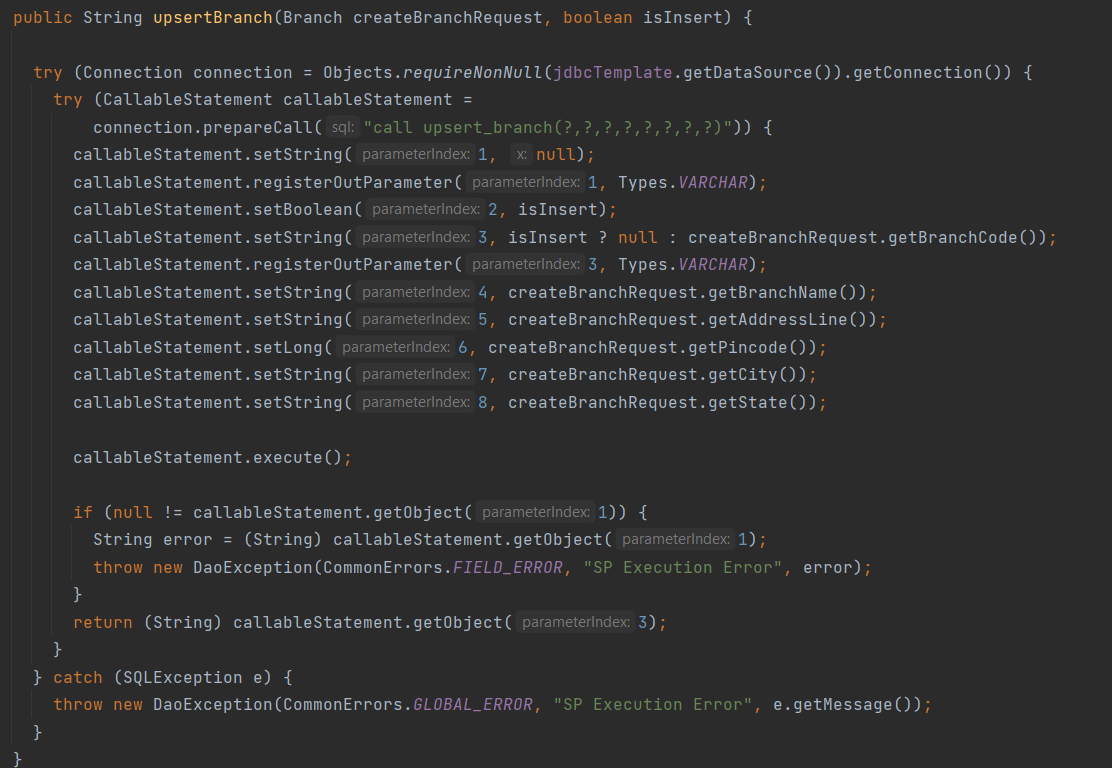


### SQL Write Operations | Transactions

**@Transactional** annotation makes sure that any DB operation wrapped inside it will be considered as an atomic operation. If any runtime exception is thrown during the execution of the code lying within @Transactional boundary then the changes will be rolled back to the previous consistent state otherwise the changes will be committed to the DB.



### Call Stored Procedure



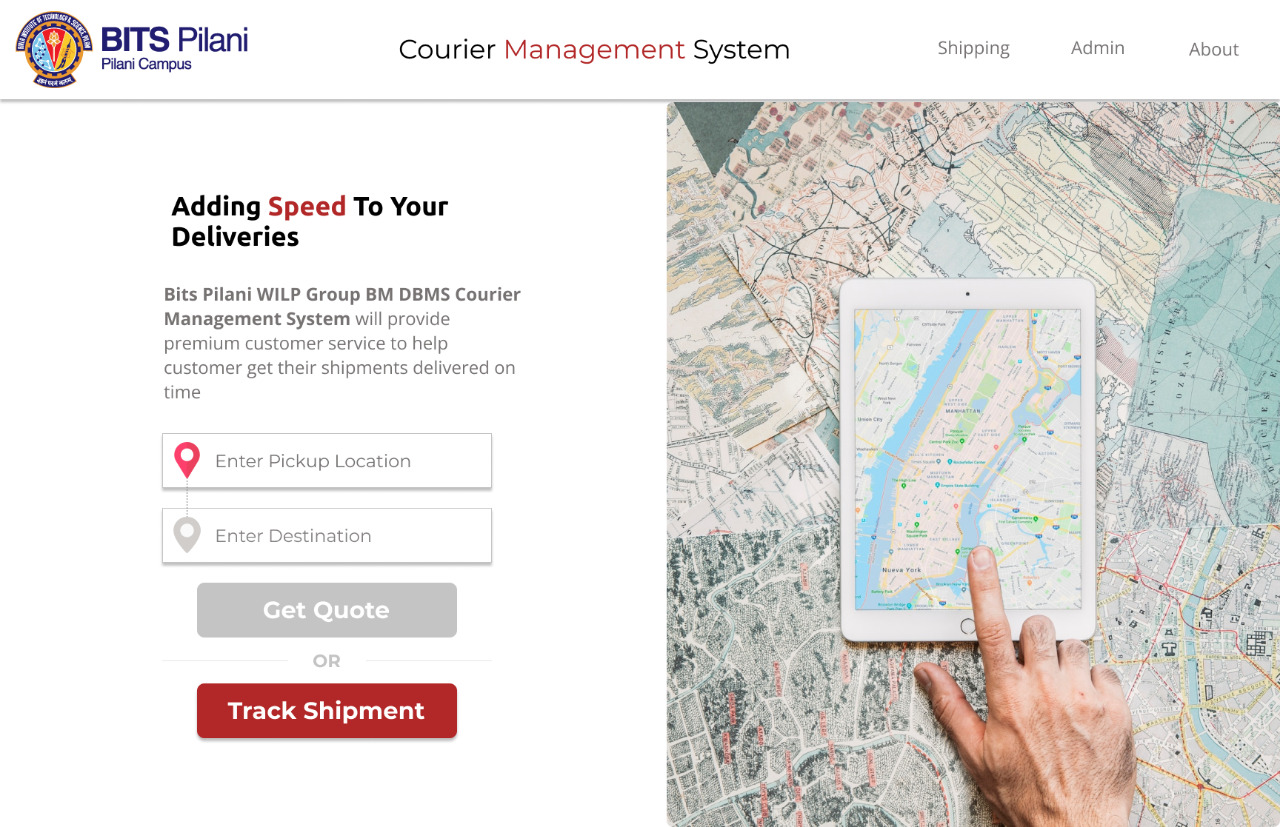
## Application Demo | User Interface

### API Documentation

Attached is the Demo Documentation which showcases the overall flow of this courier management application.



### Frontend Application Screenshot



# Reference

For complete code including SQLs and documentation, please refer to the following links:

* GitHub repository:
  + <https://github.com/varenyavv/courier-mgmt-group-bm>
* DB Scripts:
  + <https://github.com/varenyavv/courier-mgmt-group-bm/tree/main/src/main/resources/db/migration>
* Documentation:
  + <https://github.com/varenyavv/courier-mgmt-group-bm/tree/main/Documentation/Final>