НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
Кафедра системного програмування та спеціалізованих комп'ютерних систем

Лабораторна робота №2

з дисципліни

«Бази даних і засоби управління»

Виконав: студент III курсу
ФПМ групи КВ-82
Бікерей Олексій Ігорович
Перевірив(ла):

Київ – 2020

Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL

Метою роботи: є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

Загальне завдання роботи полягає у наступному:

1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

Середовище розробки

Для виконання лабораторної роботи використовувалась мова програмування Java та середовище розробки IntelliJ IDEA.

Для підключення до серверу бази даних PostgreSQL використовувався PostgreSQL JDBC driver.

Логічна модель даних

Для даної лабораторної роботи використовується база даних “ Сервіс з продажу залізничних квитків ”, логічну модель якої наведено на рисунку 1.

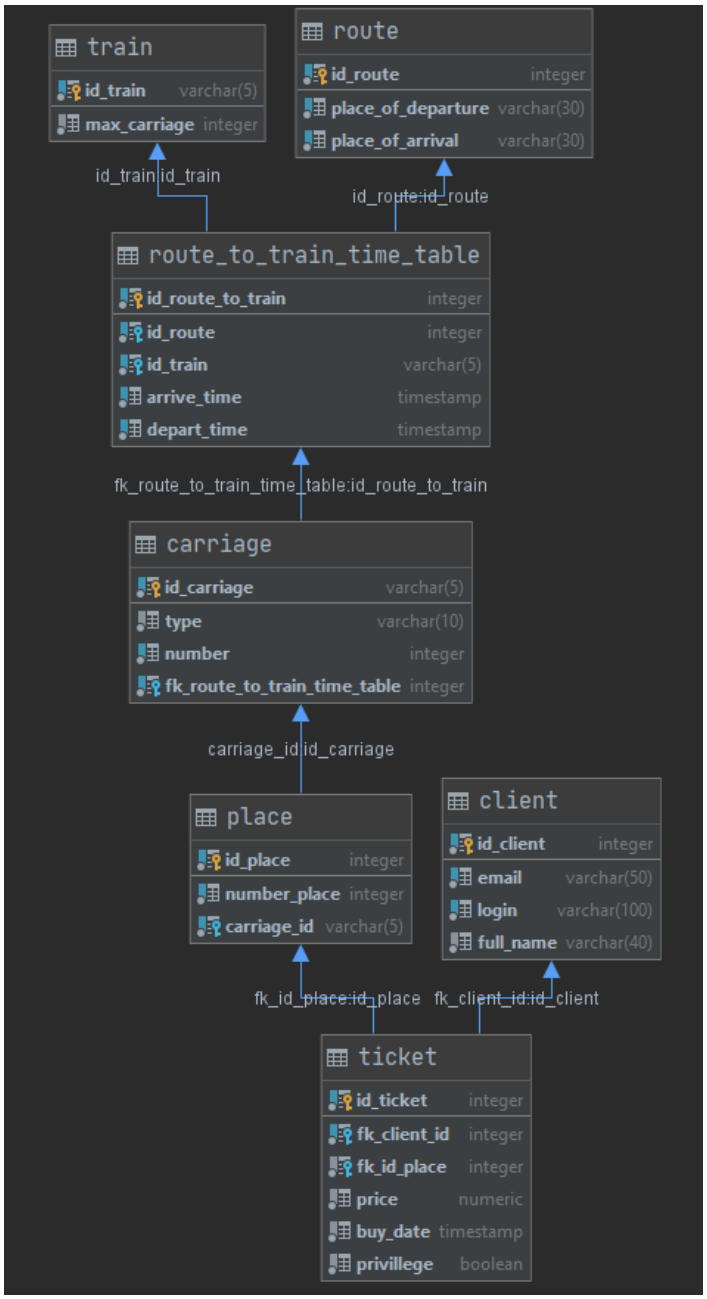

Рисунок 1 – Логічна модель даних

Опис програми

Програма створена за патерном MVC (Model-View-Controller). Складається відповідно з класів Model , View та Controller.

У класі Model реалізовані функції , що здійснюють SQL запити до Бази Даних, а також функція, що виконує з’єднання з БД.

У класі View реалізовані функції, що використовуються для відображення в консоль пунктів меню, виводу даних з таблиць, тобто функції, що відображують певну інформацію в консоль.

У класі Controller реалізовані функції для відповідних меню та допоміжні функції

На рисунку 2 зображена структура програми

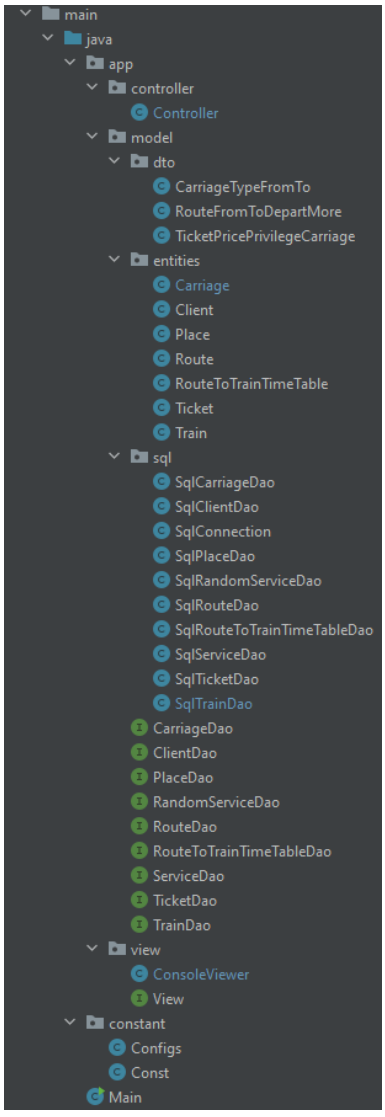

Рисунок 2 – Структура програми

Опис структури меню програми

Меню програми можна розглядати як її концептуальну модель на рисунку

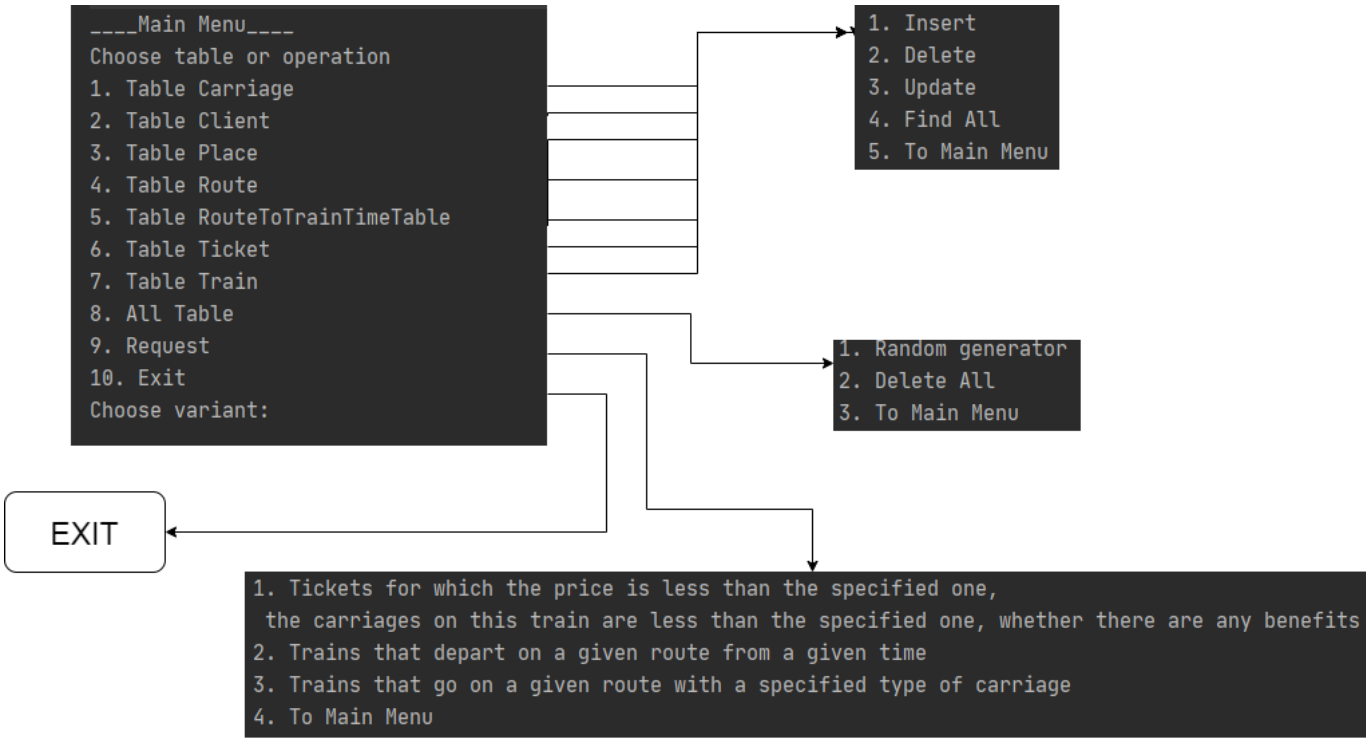

Рисунок 3 - Концептуальну модель меню

Ілюстрації обробки виняткових ситуацій (помилок) при уведенні

Продемонстровано обробка помилки введення некоректних даних. Вводиться не коректний зовнішній ключ який не існує в батьківській таблиці.

```
Insert(number_place(integer), carriage_id(String)):
137
org.postgresql.util.PSQLException: ПОМИЛКА: insert або update в таблиці "place" порушує обмеження зовнішнього ключа "fk_carriage"
  Подробности: Ключ (carriage_id)=() не присутній в таблиці "carriage".
    at org.postgresql.core.v3.QueryExecutorImpl.receiveErrorResponse(QueryExecutorImpl.java:2553)
    at org.postgresql.core.v3.QueryExecutorImpl.processResults(QueryExecutorImpl.java:2285)
    at org.postgresql.core.v3.QueryExecutorImpl.execute(QueryExecutorImpl.java:323)
    at org.postgresql.jdbc.PgStatement.executeInternal(PgStatement.java:473)
    at org.postgresql.jdbc.PgStatement.execute(PgStatement.java:393)
    at org.postgresql.jdbc.PgPreparedStatement.executeWithFlags(PgPreparedStatement.java:164)
    at org.postgresql.jdbc.PgPreparedStatement.executeUpdate(PgPreparedStatement.java:130)
    at app.model.sql.SqlPlaceDao.insertPlace(SqlPlaceDao.java:49)
    at app.controller.Controller.sqlDaoPlace(Controller.java:173)
    at app.controller.Controller.mainMenu(Controller.java:57)
    at app.controller.Controller.appDo(Controller.java:37)
    at Main.main(Main.java:14)
```

Ілюстрації валідації даних при уведенні користувачем

При введені не правильного типу даних до таблиці виводиться помилка про неправильний тип даних.

```
Insert(type(String), number(Int), id(String), idRouteToTrainTimeTable(Int))
afef
af
java.lang.NumberFormatException Create breakpoint : For input string: "af"
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
    at java.lang.Integer.parseInt(Integer.java:580)
    at java.lang.Integer.parseInt(Integer.java:615)
    at app.controller.Controller.sqlDaoCarriage(Controller.java:107)
    at app.controller.Controller.mainMenu(Controller.java:47)
    at app.controller.Controller.appDo(Controller.java:37)
    at Main.main(Main.java:14)
1. Insert
2. Delete
3. Update
4. Find All
5. To Main Menu
```

Копії екрану (ілюстрації) з фрагментами згенерованих даних таблиць

Table Carriage

```
Carriage{type='mxfwtqu', number=89, id='djycf', routeToTrainTimeTable=130041}
Carriage{type='iqhtapw', number=122, id='bspwc', routeToTrainTimeTable=130042}
Carriage{type='xeklmmv', number=178, id='phymt', routeToTrainTimeTable=130043}
Carriage{type='wsroqqq', number=192, id='usqwo', routeToTrainTimeTable=130044}
Carriage{type='ultnmcn', number=197, id='pwikl', routeToTrainTimeTable=130045}
```

Table Client

```
Client{id=132234, e_mail='cldyuxbyt@gmail.com', login='hrcyore', full_name='joeswbm vaakytyr'}
Client{id=132235, e_mail='akqjqbxfw@gmail.com', login='buywrpk', full_name='qviqaeo ghajokrl'}
Client{id=132236, e_mail='pqdnlwbtk@gmail.com', login='pcgkixr', full_name='isxwyvs xysvtibe'}
Client{id=132237, e_mail='pjmlpjgda@gmail.com', login='vhvkjsu', full_name='loyuwsa tmfpqceh'}
```

Table Place

```
Place{id=109525, number=115, carriage='ltbuh'}
Place{id=109526, number=62, carriage='ivuwc'}
Place{id=109527, number=52, carriage='tpgyt'}
Place{id=109528, number=29, carriage='hrjph'}
```

Table Route

```
Route{id=231682, place_of_departure='ubweqjtqyicrneymwrkn', place_of_arrival='mgjfnidkgkrschowddoe'}
Route{id=231683, place_of_departure='ybnhobxkohqxaoirtqxk', place_of_arrival='xveeduostiicacgwduiq'}
Route{id=231684, place_of_departure='ahdonppwljwsmheeywqn', place_of_arrival='tgiekkiwytpempoljrjh'}
Route{id=231685, place_of_departure='tclsnmkbgvnvxrqaqvbd', place_of_arrival='itutsklkmoomkedrhfhp'}
```

Table RouteToTrainTimeTable

```
RouteToTrainTimeTable{id=130138, route=231682, train='jcrnn', arriveTime=2020-09-03 23:10:05.21756, departTime=2020-02-21 10:36:41.3574}
RouteToTrainTimeTable{id=130139, route=231683, train='nrwix', arriveTime=2020-03-17 15:26:25.480886, departTime=2020-03-27 23:09:02.667715}
RouteToTrainTimeTable{id=130140, route=231684, train='dnnyo', arriveTime=2020-04-13 03:51:40.729466, departTime=2020-01-27 05:40:15.142801}
RouteToTrainTimeTable{id=130141, route=231685, train='qiunb', arriveTime=2020-10-01 19:21:41.202019, departTime=2020-02-21 10:36:22.960738}
```

Table Ticket

```
Ticket{id=111816, client=132256, place=109527, price=1031.5659999580585, buy_date=2020-09-09 21:15:27.755461, privilege=true}
Ticket{id=111817, client=132257, place=109528, price=386.51544503586985, buy_date=2020-07-09 04:27:33.375925, privilege=true}
Ticket{id=111818, client=132258, place=109529, price=1076.0155068794807, buy_date=2020-08-21 11:07:05.119653, privilege=false}
```

Table Train

```
Train{id_train='nrwix', max_carriage=94}
Train{id_train='dnnyo', max_carriage=199}
Train{id_train='qiunb', max_carriage=33}
```

Копії SQL-запитів, що ілюструють генерацію при визначених вхідних параметрах

```
public static final String SQL_SELECT_FK_ROUTE = "SELECT r.id_route FROM
route AS r LEFT OUTER JOIN route_to_train_time_table AS rtt " +
        "ON r.id_route = rtt.id_route WHERE rtt.id_route IS NULL";

public static final String SQL_SELECT_FK_ROUTE_TO_TRAIN_TIME_TABLE = "SELECT
rtt.id_route_to_train FROM route_to_train_time_table AS rtt LEFT OUTER JOIN
carriage AS c " +
        "ON rtt.id_route_to_train = c.fk_route_to_train_time_table WHERE
c.fk_route_to_train_time_table IS NULL";

public static final String SQL_SELECT_FK_TRAIN = "SELECT t.id_train FROM
train AS t LEFT OUTER JOIN route_to_train_time_table AS rtt " +
        "ON t.id_train = rtt.id_train WHERE rtt.id_train IS NULL";

public static final String SQL_SELECT_FK_CARRIAGE = "SELECT c.id_carriage
FROM carriage AS c LEFT OUTER JOIN place AS p " +
        "ON c.id_carriage = p.carriage_id WHERE p.carriage_id IS NULL";

public static final String SQL_SELECT_FK_CLIENT = "SELECT c.id_client FROM
client AS c LEFT OUTER JOIN ticket AS t " +
        "ON c.id_client = t.fk_client_id WHERE t.fk_client_id IS NULL";

public static final String SQL_SELECT_FK_PLACE = "SELECT p.id_place FROM
place AS p LEFT OUTER JOIN ticket AS t " +
        "ON p.id_place = t.fk_id_place WHERE t.fk_id_place IS NULL";
```

Ілюстрації уведення пошукового запиту та результатів виконання запитів

Вибираємо пошуковий запит 2 потім вводимо параметри за якими буде відбуватися запит. Запит 2 шукає всі маршрути які відправляються з якогось часу і пізніше.

```
1. Tickets for which the price is less than the specified one,

 the carriages on this train are less than the specified one, whether there are any benefits

2. Trains that depart on a given route from a given time

3. Trains that go on a given route with a specified type of carriage

4. To Main Menu

2

arrive_place(String), depart_time(String), depart_time(format: yyyy-MM-dd HH:mm)

byhgnlwcuuexclgjoqoa

asgqcdvfrjyvxkqgrpnl

2020-06-20 14:39:01.794724

[RouteFromToDepartMore{place_of_departure='asgqcdvfrjyvxkqgrpnl',
place_of_arrive='byhgnlwcuuexclgjoqoa',    arrive_time=2020-06-20    14:39:01.794724,
depart_time=2020-08-27 22:39:41.567563}

]
```

Копії SQL-запитів, що ілюструють пошук з зазначеними початковими параметрами

```
public static final String SQL_TICKET_WITH_PRICELESS_PRIV_CARLESS = "SELECT
\n" +
        "\tt.price, \n" +
        "\tt.privillege,\n" +
        "\tt.buy_date,\n" +
        "\tc.full_name,\n" +
        "\tr.place_of_arrival,\n" +
        "\tr.place_of_departure, \n" +
        "\trtt.arrive_time,\n" +
        "\trtt.depart_time,\n" +
        "\ttr.id_train,\n" +
        "\tcar.number,\n" +
        "\tp.number_place\n" +
        "from ticket as t \n" +
        "inner join client as c \n" +
        "\ton t.fk_client_id = c.id_client\n" +
        "inner join place as p\n" +
        "\ton t.fk_id_place = p.id_place\n" +
        "inner join carriage as car \n" +
        "\ton p.carriage_id = car.id_carriage\n" +
        "inner join route_to_train_time_table as rtt\n" +
        "\ton car.fk_route_to_train_time_table = rtt.id_route_to_train\n" +
        "inner join route as r\n" +
        "\ton rtt.id_route = r.id_route\n" +
        "inner join train as tr\n" +
        "\ton rtt.id_train = tr.id_train\n" +
        "where t.price < ? AND t.privillege = ? AND car.number < ?";

public static final String SQL_ROUTE_FROM_TO_DEPART_MORE = "SELECT
r.place_of_departure, r.place_of_arrival, " +
        "rtt.depart_time, rtt.arrive_time FROM route AS r " +
        "INNER JOIN route_to_train_time_table AS rtt " +
        "ON r.id_route = rtt.id_route " +
        "WHERE r.place_of_departure = ? AND r.place_of_arrival = ? AND
rtt.depart_time >= ?";

public static final String SQL_CARRIAGE_ROUTE_TYPE = "SELECT
r.place_of_departure, r.place_of_arrival, " +
        "car.type, car.id_carriage, rtt.arrive_time, " +
        "rtt.depart_time FROM route AS r INNER JOIN route_to_train_time_table
AS rtt " +
        "ON r.id_route = rtt.id_route " +
        "INNER JOIN carriage AS car ON car.fk_route_to_train_time_table =
rtt.id_route_to_train " +
        "WHERE r.place_of_departure = ? AND r.place_of_arrival = ? AND
car.type = ?";
```

Дослідження режимів обмеження ON DELETE

Дослідження режимів будемо проводити на таблиця route та route_to_train_time_table. Таблиця route – батьківська, а таблиця route_to_train_time_table – дочірня.

	id_route [PK] integer	place_of_departure character varying (30)	place_of_arrival character varying (30)
1	231692	nmpixpyplrdjjllsifog	uictvvhrmhuyafdjraxn
2	231693	rvnmjrlikbbkqdmdoxch	xqptxcjaeauljyagjmme
3	231694	ojrmnuhywlbrwcsnaegx	mbipudmhysnnbctchaak
4	231695	hbksuhhccptddaxtwskm	yhgcwmaoqxkpxuqhbtbq
5	231696	lmnmdwemgdvxiftddyls	npiyuvhjmrunyyjjkepo

Рисунок 4 – вміст таблиці route

	id_route_to_train [PK] integer	id_route integer	id_train character varying (5)	arrive_time timestamp without time zone	depart_time timestamp without time zone
1	130148	231692	ulsio	2020-10-06 15:31:46.729764	2020-03-05 08:49:36.677286
2	130149	231693	oswaf	2020-04-18 12:14:34.146808	2020-05-20 23:23:13.4031
3	130150	231694	nmuks	2020-06-10 15:38:55.793129	2020-06-30 22:27:53.601272
4	130151	231695	ohtxt	2020-07-31 17:15:21.180462	2020-02-28 10:27:37.444063
5	130152	231696	pmtbm	2020-01-20 02:55:19.50464	2020-01-30 02:51:45.80333

Рисунок 5 – вміст таблиці route_to_train_time_table

1. Режим CASCADE

При видаленні запису з таблиці route, запис з таблиці route_to_train_time_table видаляється

2. Режим SET NULL

При видаленні запису з таблиці route, id_route запису з таблиці route_to_train_time_table встановлюється в null. Якщо в налаштуваннях таблиці вказати, що route_to_train_time_table.id_route не може бути null, то перехоплюємо повідомлення про помилку.

```
DeleteById id(Int):
231692
ПОМИЛКА: null значення у стовпці "id_route" порушує not-null обмеження
  Подробности: Помилковий рядок містить (130148, null, ulsio, 2020-10-06 15:31:46.729764, 2020-03-05 08:49:36.677286).
  Где: SQL-оператор "UPDATE ONLY "public"."route_to_train_time_table" SET "id_route" = NULL WHERE $1 OPERATOR(pg_catalog.=) "id_route""
```

3. Режим SET DEFAULT (значення за замовчуванням = 1)

При видаленні запису з таблиці route, перехоплюємо повідомлення про помилку, так як route з id = 1 не існує. Якщо встановити інше значення за замовчуванням, то запис може прив'язатися до route.

```
DeleteById id(Int):
231693
ПОМИЛКА: insert або update в таблиці "route_to_train_time_table" порушує обмеження зовнішнього ключа "fk_route"
  Подробности: Ключ (id_route)=(1) не присутній в таблиці "route".
```

4. Режим NO ACTION

При видаленні запису з таблиці route, запис з таблиці route_to_train_time_table не видаляється.

```
DeleteById id(Int):
231692
ПОМИЛКА: update або delete в таблиці "route" порушує обмеження зовнішнього ключа "fk_route" таблиці "route_to_train_time_table"
  Подробности: На ключ (id_route)=(231692) все ще є посилання в таблиці "route_to_train_time_table".
```

4. Режим RESTRICT

Працює аналогічно з режимом *NO ACTION*

```
DeleteById id(Int):
231692
ПОМИЛКА: update або delete в таблиці "route" порушує обмеження зовнішнього ключа "fk_route" таблиці "route_to_train_time_table"
  Подробности: На ключ (id_route)=(231692) все ще є посилання в таблиці "route_to_train_time_table".
```

Перехопленням помилок (try...except) від сервера PostgreSQL при виконанні відповідної команди SQL;

Перехопленя помилки за допомогою (try...except) выд сервера PostgrsSQL при викрнанні відповідної команди SQL при редагувані даних.

```
  public static final String SQL_UPDATE_CARRIAGE_NUMBER = "UPDATE carriage SET number = ?
WHERE id_carriage LIKE ?";
  public void updateCarriageNumber(String id, Integer number) {
    SqlConnection mySqlConnection = SqlConnection.getInstance();
    Connection connection = mySqlConnection.getConnection();
    try{
      PreparedStatement ps = connection.prepareStatement(SQL_UPDATE_CARRIAGE_NUMBER);
      ps.setInt(1, number);
      ps.setString(2, id);
      ps.executeUpdate();
    }catch (Exception ex){
      System.out.println(ex.getMessage());
    }
  }
```

Контроль наявності відповідного рядка у батьківській таблиці при виконанні внесення до неї нових даних

Контроль наявності відповідного рядка у батьківській таблиці при виконанні внесення до неї нових даних реалізується за допомогою перехоплення помилки (try...except) від сервера PostgreSQL і видачі повідомлення користувачу.

```
  public static final String SQL_INSERT_CARRIAGE = "INSERT INTO carriage (type, number, id_carriage,
fk_route_to_train_time_table) " +
      "VALUES (?, ?, ?, ?) ";

  public void insertCarriage(Carriage carriage) {
    SqlConnection mySqlConnection = SqlConnection.getInstance();
    Connection connection = mySqlConnection.getConnection();
    try{
      PreparedStatement ps = connection.prepareStatement(SQL_INSERT_CARRIAGE);
      ps.setString(1, carriage.getType());
      ps.setInt(2, carriage.getNumber());
      ps.setString(3, carriage.getId());
      ps.setInt(4, carriage.getRouteToTrainTimeTable());
      ps.executeUpdate();
    }catch(Exception ex){
      System.out.println(ex.getMessage());
    }
  }
```

Ілюстрації програмного коду з репозиторію Git

Посилання на github - https://github.com/leshik-xxl/DataBase-Lab2

Посилання для навігації по програмі

1. **Controller**
   - 1.1 **Функція для розпізнання дати**
   - 1.2 **Функція для початку роботи додатку**
   - 1.3 **Функція для головного меню**
   - 1.4 **Функція для роботи з таблицею Carriage**
   - 1.5 **Функція для роботи з таблицею Client**
   - 1.6 **Функція для роботи з таблицею Place**
   - 1.7 **Функція для роботи з таблицею Route**
   - 1.8 **Функція для роботи з таблицею RouteToTrainTimeTable**
   - 1.9 **Функція для роботи з таблицею Ticket**
   - 1.10 **Функція для роботи з таблицею Train**
   - 1.11 **Функція для запитів**
   - 1.12 **Функція для роботи зі всіма таблицями**
   - 1.13 **Функція прослуховування подій**
2. **Model**
   - 2.1 **dto – пакет для реалізуються сутностей запитів**
   - 2.2 **entities – пакет де реалізуються сутності таблиць**
   - 2.3 **sql – пакет де реалізуються робота з базою даних**
     - 2.3.1 – **клас де реалізуються робота з таблицею Carriage**
     - 2.3.2 – **клас де реалізуються робота з таблицею Client**
     - 2.3.3 - **клас де реалізуються робота з таблицею Place**
     - 2.3.4 - **клас де реалізуються робота з таблицею Route**
     - 2.3.5 - **клас де реалізуються робота з таблицею RouteToTrainTimeTable**
     - 2.3.6 - **клас де реалізуються робота з таблицею Ticket**
     - 2.3.7 - **клас де реалізуються робота з таблицею Train**
     - 2.3.7 – **клас де реалізуються з'єднання з базою даних**
     - 2.3.8 – **клас де реалізуються генерація випадкових записів в таблиці**
     - 2.3.8 – **клас де реалізуються пошукові запити**

3. **View – пакет де реалізуються функції для відображення контенту користувачу**

Controller

```
package app.controller;

import app.model.*;
import app.model.entities.*;
import app.model.sql.*;
import app.view.View;

import java.math.BigDecimal;
import java.sql.Timestamp;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Scanner;


public class Controller {
    View view;

    public Controller(View view) {
        this.view = view;
    }

    public static Timestamp parseDate(String date) {
        SimpleDateFormat ft = new SimpleDateFormat("yyyy-MM-dd HH:mm");
        Date parsingDate = null;
        try {
            parsingDate = ft.parse(date);
            //System.out.println(parsingDate);
        } catch (ParseException e) {
            System.out.println("Нераспаршена с помощью " + ft);
        }
        return new Timestamp(parsingDate.getTime());
    }

    public void appDo() {
        while (true) {
            if (mainMenu() == 1) return;
        }

    }

    private int mainMenu() {
        view.mainMenu();
        switch (listener()) {
            case 1:
                while (true) {
                    if (sqlDaoCarriage() == 1) break;
                }
                break;
            case 2:
                while (true) {
                    if (sqlDaoClient() == 1) break;
                }
                break;
```

```
                case 3:
                    while (true) {
                        if (sqlDaoPlace() == 1) break;
                    }
                    break;
                case 4:
                    while (true) {
                        if (sqlDaoRoute() == 1) break;
                    }
                    break;
                case 5:
                    while (true) {
                        if (sqlDaoRouteToTrain() == 1) break;
                    }
                    break;
                case 6:
                    while (true) {
                        if (sqlDaoTicket() == 1) break;
                    }
                    break;
                case 7:
                    while (true) {
                        if (sqlDaoTrain() == 1) break;
                    }
                    break;
                case 8:
                    while (true) {
                        if (allTable() == 1) break;
                    }
                    break;
                case 9:
                    while (true) {
                        if (sqlRequest() == 1) break;
                    }
                    break;
                case 10:
                    return 1;
                default:
                    view.wrong();
                    break;
            }
            return 0;
        }

        public int sqlDaoCarriage() {
            Scanner scan = new Scanner(System.in);
            CarriageDao carriageDao = new SqlCarriageDao();
            view.sqlDao();
            switch (listener()) {
                case 1:
                    view.insertTable(1);
                    try{
                        carriageDao.insertCarriage(new Carriage(scan.nextLine(),
Integer.parseInt(scan.nextLine()), scan.nextLine(),
Integer.parseInt(scan.nextLine())));
                    } catch(Exception ex){
```

```
                    ex.printStackTrace();
                }
                break;
            case 2:
                view.deleteTable(1);
                carriageDao.deleteCarriage(scan.nextLine());
                break;
            case 3:
                view.updateTable(1);
                carriageDao.updateCarriageNumber(scan.nextLine(),
scan.nextInt());
                break;
            case 4:
                view.printCarriage(carriageDao.findAllCarriage());
                break;
            case 5:
                return 1;
            default:
                view.wrong();
                break;
        }
        return 0;
    }

    public int sqlDaoClient() {
        Scanner scan = new Scanner(System.in);
        ClientDao clientDao = new SqlClientDao();
        view.sqlDao();
        switch (listener()) {
            case 1:
                view.insertTable(2);
                try{
                    clientDao.insertClient(new Client(null, scan.nextLine(),
scan.nextLine(), scan.nextLine()));
                } catch(Exception ex){
                    ex.printStackTrace();
                }
                break;
            case 2:
                view.deleteTable(2);
                clientDao.deleteClient(scan.nextInt());
                break;
            case 3:
                view.updateTable(2);
                clientDao.updateClient(scan.nextLine(), scan.nextLine());
                break;
            case 4:
                view.printClient(clientDao.findAllClient());
                break;
            case 5:
                return 1;
            default:
                view.wrong();
                break;
        }
        return 0;
```

```
    }

    public int sqlDaoPlace() {
        Scanner scan = new Scanner(System.in);
        PlaceDao placeDao = new SqlPlaceDao();
        view.sqlDao();
        switch (listener()) {
            case 1:
                view.insertTable(3);
                try{
                    placeDao.insertPlace(new Place(null, scan.nextInt(),
scan.nextLine()));
                } catch(Exception ex){
                    ex.printStackTrace();
                }
                break;
            case 2:
                view.deleteTable(3);
                placeDao.deletePlace(scan.nextInt());
                break;
            case 3:
                view.updateTable(3);
                placeDao.updateNumberPlace(scan.nextInt(), scan.nextInt());
                break;
            case 4:
                view.printPlace(placeDao.findAllPlace());
                break;
            case 5:
                return 1;
            default:
                view.wrong();
                break;
        }
        return 0;
    }

    public int sqlDaoRoute() {
        Scanner scan = new Scanner(System.in);
        RouteDao routeDao = new SqlRouteDao();
        view.sqlDao();
        switch (listener()) {
            case 1:
                view.insertTable(4);
                try{
                    routeDao.insertRoute(new Route(null, scan.nextLine(),
scan.nextLine()));
                } catch(Exception ex){
                    ex.printStackTrace();
                }
                break;
            case 2:
                view.deleteTable(4);
                routeDao.deleteRoute(scan.nextInt());
                break;
            case 3:
                view.updateTable(4);
```

```
                routeDao.updatePlaceOfArrival(scan.nextInt(),
scan.nextLine());
                break;
            case 4:
                view.printRoute(routeDao.findAllRoute());
                break;
            case 5:
                return 1;
            default:
                view.wrong();
                break;
        }
        return 0;
    }

    public int sqlDaoRouteToTrain() {
        Scanner scan = new Scanner(System.in);
        RouteToTrainTimeTableDao routeToTrainTimeTableDao = new
SqlRouteToTrainTimeTableDao();
        view.sqlDao();
        switch (listener()) {
            case 1:
                view.insertTable(5);
                try{
                    routeToTrainTimeTableDao.insertRouteToTrainTimeTable(new
RouteToTrainTimeTable(null, Integer.parseInt(scan.nextLine()),
                            scan.nextLine(), parseDate(scan.nextLine()),
parseDate(scan.nextLine())));
                } catch(Exception ex){
                    ex.printStackTrace();
                }
                break;
            case 2:
                view.deleteTable(5);

routeToTrainTimeTableDao.deleteRouteToTrainTimeTable(scan.nextInt());
                break;
            case 3:
                view.updateTable(5);

routeToTrainTimeTableDao.updateRouteToTrainTimeTableOfDeparture(Integer.parse
Int(scan.nextLine()), parseDate(scan.nextLine()));
                break;
            case 4:

view.printRouteToTrainTimeTable(routeToTrainTimeTableDao.findAllRouteToTrainT
imeTable());
                break;
            case 5:
                return 1;
            default:
                view.wrong();
                break;
        }
        return 0;
    }
```

```
    public int sqlDaoTicket() {
        Scanner scan = new Scanner(System.in);
        TicketDao ticketDao = new SqlTicketDao();
        view.sqlDao();
        switch (listener()) {
            case 1:
                view.insertTable(6);
                try{
                    ticketDao.insertTicket(new Ticket(null,
Integer.parseInt(scan.nextLine()), Integer.parseInt(scan.nextLine()), new
BigDecimal(scan.nextLine()),
                            parseDate(scan.nextLine()),
Boolean.parseBoolean(scan.nextLine())));
                } catch(Exception ex){
                    ex.printStackTrace();
                }
                break;
            case 2:
                view.deleteTable(6);
                ticketDao.deleteTicket(scan.nextInt());
                break;
            case 3:
                view.updateTable(6);
                ticketDao.updateTicketPriceById(scan.nextInt(),
scan.nextBigDecimal());
                break;
            case 4:
                view.printTicket(ticketDao.findAllTicket());
                break;
            case 5:
                return 1;
            default:
                view.wrong();
                break;
        }
        return 0;
    }

    public int sqlDaoTrain() {
        Scanner scan = new Scanner(System.in);
        TrainDao trainDao = new SqlTrainDao();
        view.sqlDao();
        switch (listener()) {
            case 1:
                view.insertTable(7);
                try{
                    trainDao.InsertTrain(new Train(scan.nextLine(),
scan.nextInt()));
                } catch(Exception ex){
                    ex.printStackTrace();
                }
                break;
            case 2:
                view.deleteTable(7);
                trainDao.deleteTrain(scan.nextLine());
```

```
                break;
            case 3:
                view.updateTable(7);
                trainDao.updateTrainMaxCarriage(scan.nextLine(),
scan.nextInt());
                break;
            case 4:
                view.printTrain(trainDao.findAllTrain());
                break;
            case 5:
                return 1;
            default:
                view.wrong();
                break;
        }
        return 0;
    }

    public int sqlRequest() {
        Scanner scan = new Scanner(System.in);
        ServiceDao serviceDao = new SqlServiceDao();
        view.request();
        switch (listener()) {
            case 1:
                view.requestNumber(1);
                try{

view.printTicketPrivilegeCarriage(serviceDao.findTicketPricePrivilegeCarriage
(new BigDecimal(scan.nextLine()), Boolean.parseBoolean(scan.nextLine()),
                            Integer.parseInt(scan.nextLine())));
                } catch(Exception ex){
                    ex.printStackTrace();
                }
                break;
            case 2:
                view.requestNumber(2);

view.printRouteFromToDepartMore(serviceDao.findRouteFromToDepartMore(scan.nex
tLine(), scan.nextLine(), parseDate(scan.nextLine())));
                break;
            case 3:
                view.requestNumber(3);

view.printCarriageTypeFromTo(serviceDao.findRouteWithTypeCarriage(scan.nextLi
ne(), scan.nextLine(), scan.nextLine()));
                break;
            case 4:
                return 1;
            default:
                view.wrong();
                break;

        }
        return 0;
    }
```

```
    public int allTable() {
        Scanner scan = new Scanner(System.in);
        RandomServiceDao randomServiceDao = new SqlRandomServiceDao();
        ServiceDao serviceDao = new SqlServiceDao();
        view.allTable();
        switch (listener()) {
            case 1:
                view.randomGen();
                randomServiceDao.fillRandomTable(scan.nextInt());
                break;
            case 2:
                serviceDao.deleteAll();
                break;
            case 3:
                return 1;
            default:
                view.wrong();
                break;

        }
        return 0;
    }

    private int listener() {
        Scanner scan = new Scanner(System.in);
        return scan.nextInt();
    }
}
```

Model

Dto

CarriageTypeFromTo

```
package app.model.dto;

import java.util.Date;

public class CarriageTypeFromTo {
    private String place_of_depart;
    private String place_of_arrival;
    private String type;
    private String carName;
    private Date arriveTime;
    private Date departTime;

    public CarriageTypeFromTo(String place_of_depart, String
place_of_arrival,
                              String type, String carName, Date arriveTime,
Date departTime) {
        this.place_of_depart = place_of_depart;
        this.place_of_arrival = place_of_arrival;
        this.type = type;
        this.carName = carName;
```

```
        this.arriveTime = arriveTime;
        this.departTime = departTime;
    }

    public String getPlace_of_depart() {
        return place_of_depart;
    }

    public void setPlace_of_depart(String place_of_depart) {
        this.place_of_depart = place_of_depart;
    }

    public String getPlace_of_arrival() {
        return place_of_arrival;
    }

    public void setPlace_of_arrival(String place_of_arrival) {
        this.place_of_arrival = place_of_arrival;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public String getCarName() {
        return carName;
    }

    public void setCarName(String carName) {
        this.carName = carName;
    }

    public Date getArriveTime() {
        return arriveTime;
    }

    public void setArriveTime(Date arriveTime) {
        this.arriveTime = arriveTime;
    }

    public Date getDepartTime() {
        return departTime;
    }

    public void setDepartTime(Date departTime) {
        this.departTime = departTime;
    }

    @Override
    public String toString() {
        return "CarriageTypeFromTo{" +
                "place_of_depart='" + place_of_depart + '\'' +
```

```
                ", place_of_arrival='" + place_of_arrival + '\'' +
                ", type='" + type + '\'' +
                ", carName='" + carName + '\'' +
                ", arriveTime=" + arriveTime +
                ", departTime=" + departTime +
                "\n";
    }
}
```

RouteFromToDepartMore

```
package app.model.dto;

import java.sql.Timestamp;

public class RouteFromToDepartMore {
    private String place_of_departure;
    private String place_of_arrive;
    private Timestamp arrive_time;
    private Timestamp depart_time;

    public RouteFromToDepartMore(String place_of_departure
            , String place_of_arrive, Timestamp arrive_time, Timestamp
depart_time) {
        this.place_of_departure = place_of_departure;
        this.place_of_arrive = place_of_arrive;
        this.arrive_time = arrive_time;
        this.depart_time = depart_time;
    }

    public String getPlace_of_departure() {
        return place_of_departure;
    }

    public void setPlace_of_departure(String place_of_departure) {
        this.place_of_departure = place_of_departure;
    }

    public String getPlace_of_arrive() {
        return place_of_arrive;
    }

    public void setPlace_of_arrive(String place_of_arrive) {
        this.place_of_arrive = place_of_arrive;
    }

    public Timestamp getArrive_time() {
        return arrive_time;
    }

    public void setArrive_time(Timestamp arrive_time) {
        this.arrive_time = arrive_time;
    }

    public Timestamp getDepart_time() {
```

```
        return depart_time;
    }

    public void setDepart_time(Timestamp depart_time) {
        this.depart_time = depart_time;
    }

    @Override
    public String toString() {
        return "RouteFromToDepartMore{" +
                "place_of_departure='" + place_of_departure + '\'' +
                ", place_of_arrive='" + place_of_arrive + '\'' +
                ", arrive_time=" + arrive_time +
                ", depart_time=" + depart_time +
                "}\n";
    }
}
```

TicketPricePrivilegeCarriage

```
package app.model.dto;

import java.math.BigDecimal;
import java.sql.Timestamp;

public class TicketPricePrivilegeCarriage {
    private BigDecimal price;
    private Boolean privilege;
    private Timestamp buyDate;
    private String fullName;
    private String placeOfDepart;
    private String placeOfArrive;
    private Timestamp arriveTime;
    private Timestamp departTime;
    private String train;
    private Integer numberCarriage;
    private Integer numberPlace;

    public TicketPricePrivilegeCarriage(BigDecimal price, Boolean privilege,
                                        Timestamp buyDate, String fullName,
                                        String placeOfDepart, String
placeOfArrive,
                                        Timestamp arriveTime, Timestamp
departTime,
                                        String train, Integer numberCarriage,
Integer numberPlace) {
        this.price = price;
        this.privilege = privilege;
        this.buyDate = buyDate;
        this.fullName = fullName;
        this.placeOfDepart = placeOfDepart;
        this.placeOfArrive = placeOfArrive;
        this.arriveTime = arriveTime;
        this.departTime = departTime;
        this.train = train;
```

```
        this.numberCarriage = numberCarriage;
        this.numberPlace = numberPlace;
    }

    public BigDecimal getPrice() {
        return price;
    }

    public void setPrice(BigDecimal price) {
        this.price = price;
    }

    public Boolean getPrivilege() {
        return privilege;
    }

    public void setPrivilege(Boolean privilege) {
        this.privilege = privilege;
    }

    public Timestamp getBuyDate() {
        return buyDate;
    }

    public void setBuyDate(Timestamp buyDate) {
        this.buyDate = buyDate;
    }

    public String getFullName() {
        return fullName;
    }

    public void setFullName(String fullName) {
        this.fullName = fullName;
    }

    public String getPlaceOfDepart() {
        return placeOfDepart;
    }

    public void setPlaceOfDepart(String placeOfDepart) {
        this.placeOfDepart = placeOfDepart;
    }

    public String getPlaceOfArrive() {
        return placeOfArrive;
    }

    public void setPlaceOfArrive(String placeOfArrive) {
        this.placeOfArrive = placeOfArrive;
    }

    public Timestamp getArriveTime() {
        return arriveTime;
    }
```

```
    public void setArriveTime(Timestamp arriveTime) {
        this.arriveTime = arriveTime;
    }

    public Timestamp getDepartTime() {
        return departTime;
    }

    public void setDepartTime(Timestamp departTime) {
        this.departTime = departTime;
    }

    public String getTrain() {
        return train;
    }

    public void setTrain(String train) {
        this.train = train;
    }

    public Integer getNumberCarriage() {
        return numberCarriage;
    }

    public void setNumberCarriage(Integer numberCarriage) {
        this.numberCarriage = numberCarriage;
    }

    public Integer getNumberPlace() {
        return numberPlace;
    }

    public void setNumberPlace(Integer numberPlace) {
        this.numberPlace = numberPlace;
    }

    @Override
    public String toString() {
        return "TicketPricePrivilegeCarriage{" +
                "price=" + price +
                ", privilege=" + privilege +
                ", buyDate=" + buyDate +
                ", fullName='" + fullName + '\'' +
                ", placeOfDepart='" + placeOfDepart + '\'' +
                ", placeOfArrive='" + placeOfArrive + '\'' +
                ", arriveTime=" + arriveTime +
                ", departTime=" + departTime +
                ", train='" + train + '\'' +
                ", numberCarriage=" + numberCarriage +
                ", numberPlace=" + numberPlace +
                "}\n";
    }
}
```

Entities

Carriage

```
package app.model.entities;

public class Carriage {
    private String type;
    private Integer number;
    private String id;
    private Integer routeToTrainTimeTable;

    public Carriage(String type, Integer number, String id, Integer
routeToTrainTimeTable) {
            this.type = type;
            this.number = number;
            this.id = id;
            this.routeToTrainTimeTable = routeToTrainTimeTable;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public Integer getNumber() {
        return number;
    }

    public void setNumber(Integer number) {
        this.number = number;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public Integer getRouteToTrainTimeTable() {
        return routeToTrainTimeTable;
    }

    public void setRouteToTrainTimeTable(Integer routeToTrainTimeTable) {
        this.routeToTrainTimeTable = routeToTrainTimeTable;
    }

    @Override
    public String toString() {
```

```
        return "Carriage{" +
                "type='" + type + '\'' +
                ", number=" + number +
                ", id='" + id + '\'' +
                ", routeToTrainTimeTable=" + routeToTrainTimeTable +
                "}\n";
    }
}
```

Client

```
package app.model.entities;

public class Client {
    private Integer id;
    private String e_mail;
    private String login;
    private String full_name;

    public Client(Integer id, String e_mail, String login, String full_name)
{
        this.id = id;
        this.e_mail = e_mail;
        this.login = login;
        this.full_name = full_name;
    }

    public String getE_mail() {
        return e_mail;
    }

    public Integer getId() {
        return id;
    }

    public void setE_mail(String e_mail) {
        this.e_mail = e_mail;
    }

    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }

    public String getFull_name() {
        return full_name;
    }

    public void setFull_name(String full_name) {
        this.full_name = full_name;
    }
```

```
    @Override
    public String toString() {
        return "Client{" +
                "id=" + id +
                ", e_mail='" + e_mail + '\'' +
                ", login='" + login + '\'' +
                ", full_name='" + full_name + '\'' +
                "}\n";
    }
}
```

Place

```
package app.model.entities;

import java.util.List;

public class Place {
    private Integer id;
    private Integer number;
    private String carriage;

    public Place(Integer id, Integer number, String carriage) {
        this.id = id;
        this.number = number;
        this.carriage = carriage;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Integer getNumber() {
        return number;
    }

    public void setNumber(Integer number) {
        this.number = number;
    }

    public String getCarriage() {
        return carriage;
    }

    public void setCarriage(String carriage) {
        this.carriage = carriage;
    }

    @Override
```

```
    public String toString() {
        return "Place{" +
                "id=" + id +
                ", number=" + number +
                ", carriage='" + carriage + '\'' +
                "}\n";
    }
}
```

Route

```
package app.model.entities;

public class Route {
    private Integer id;
    private String place_of_departure;
    private String place_of_arrival;

    public Route(Integer id, String place_of_departure, String
place_of_arrival) {
        this.id = id;
        this.place_of_departure = place_of_departure;
        this.place_of_arrival = place_of_arrival;
    }

    public Integer getId() {
        return id;
    }

    public String getPlace_of_departure() {
        return place_of_departure;
    }

    public void setPlace_of_departure(String place_of_departure) {
        this.place_of_departure = place_of_departure;
    }

    public String getPlace_of_arrival() {
        return place_of_arrival;
    }

    public void setPlace_of_arrival(String place_of_arrival) {
        this.place_of_arrival = place_of_arrival;
    }

    @Override
    public String toString() {
        return "Route{" +
                "id=" + id +
                ", place_of_departure='" + place_of_departure + '\'' +
                ", place_of_arrival='" + place_of_arrival + '\'' +
                "}\n";
    }
}
```

RouteToTrainTimeTable

```
package app.model.entities;

import java.util.Date;

public class RouteToTrainTimeTable {
    private Integer id;
    private Integer route;
    private String train;
    private Date arriveTime;
    private Date departTime;

    public RouteToTrainTimeTable(Integer id, Integer route, String train,
Date arriveTime, Date departTime) {
        this.id = id;
        this.route = route;
        this.train = train;
        this.arriveTime = arriveTime;
        this.departTime = departTime;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Integer getRoute() {
        return route;
    }

    public void setRoute(Integer route) {
        this.route = route;
    }

    public String getTrain() {
        return train;
    }

    public void setTrain(String train) {
        this.train = train;
    }

    public Date getArriveTime() {
        return arriveTime;
    }

    public void setArriveTime(Date arriveTime) {
        this.arriveTime = arriveTime;
    }

    public Date getDepartTime() {
```

```
        return departTime;
    }

    public void setDepartTime(Date departTime) {
        this.departTime = departTime;
    }

    @Override
    public String toString() {
        return "RouteToTrainTimeTable{" +
                "id=" + id +
                ", route=" + route +
                ", train='" + train + '\'' +
                ", arriveTime=" + arriveTime +
                ", departTime=" + departTime +
                "}\n";
    }
}
```

Ticket

```
package app.model.entities;

import java.math.BigDecimal;
import java.util.Date;

public class Ticket {
    private Integer id;
    private Integer client;
    private Integer place;
    private BigDecimal price;
    private Date buy_date;
    private Boolean privilege;

    public Ticket(Integer id, Integer client, Integer place, BigDecimal
price, Date buy_date, Boolean privilege) {
        this.id = id;
        this.client = client;
        this.place = place;
        this.price = price;
        this.buy_date = buy_date;
        this.privilege = privilege;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Integer getClient() {
        return client;
```

```
    }

    public void setClient(Integer client) {
        this.client = client;
    }

    public Integer getPlace() {
        return place;
    }

    public void setPlace(Integer place) {
        this.place = place;
    }

    public BigDecimal getPrice() {
        return price;
    }

    public void setPrice(BigDecimal price) {
        this.price = price;
    }

    public Date getBuy_date() {
        return buy_date;
    }

    public void setBuy_date(Date buy_date) {
        this.buy_date = buy_date;
    }

    public Boolean getPrivilege() {
        return privilege;
    }

    public void setPrivilege(Boolean privilege) {
        this.privilege = privilege;
    }

    @Override
    public String toString() {
        return "Ticket{" +
                "id=" + id +
                ", client=" + client +
                ", place=" + place +
                ", price=" + price +
                ", buy_date=" + buy_date +
                ", privilege=" + privilege +
                "}\n";
    }
}
```

Train

```java
package app.model.entities;

public class Train {
    private String id_train;
    private Integer max_carriage;

    public Train(String id_train, Integer max_carriage) {
        this.id_train = id_train;
        this.max_carriage = max_carriage;
    }

    public String getId_train() {
        return id_train;
    }

    public void setId_train(String id_train) {
        this.id_train = id_train;
    }

    public Integer getMax_carriage() {
        return max_carriage;
    }

    public void setMax_carriage(Integer max_carriage) {
        this.max_carriage = max_carriage;
    }

    @Override
    public String toString() {
        return "Train{" +
                "id_train='" + id_train + '\'' +
                ", max_carriage=" + max_carriage +
                "}\n";
    }
}
```

SQL

CarriageDao

```java
package app.model;

import app.model.entities.Carriage;

import java.util.List;

public interface CarriageDao {
    List<Carriage> findAllCarriage();
    void insertCarriage(Carriage carriage);
    void deleteCarriage(String carriageID);
    void updateCarriageNumber(String id, Integer number);
```

```
    void updateCarriageRouteToTrain(String id, Integer routeToTrain);
}
```

```
package app.model.sql;

import app.model.CarriageDao;
import app.model.entities.Carriage;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

public class SqlCarriageDao implements CarriageDao {

    public static final String SQL_FIND_ALL_CARRIAGE = "SELECT * FROM
carriage";
    public static final String SQL_INSERT_CARRIAGE = "INSERT INTO carriage
(type, number, id_carriage, fk_route_to_train_time_table) " +
            "VALUES (?, ?, ?, ?)";
    public static final String SQL_DELETE_CARRIAGE_BY_ID = "DELETE FROM
carriage WHERE id_carriage = ?";
    public static final String SQL_UPDATE_CARRIAGE_NUMBER = "UPDATE carriage
SET number = ? WHERE id_carriage LIKE ?";
    public static final String SQL_UPDATE_CARRIAGE_ROUTE_TO_TRAIN_TIMETABLE =
            "UPDATE carriage SET fk_route_to_train_time_table = ? WHERE
id_carriage LIKE ?";

    @Override
    public List<Carriage> findAllCarriage() {
        List<Carriage> result = new ArrayList<Carriage>();
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            Statement query = connection.createStatement();
            ResultSet rs = query.executeQuery(SQL_FIND_ALL_CARRIAGE);
            while (rs.next()) {
                result.add(new Carriage(rs.getString(1), rs.getInt(2),
                        rs.getString(3), rs.getInt(4)));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
    }

    @Override
    public void insertCarriage(Carriage carriage) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try{
            PreparedStatement ps =
connection.prepareStatement(SQL_INSERT_CARRIAGE);
```

```
            ps.setString(1, carriage.getType());
            ps.setInt(2, carriage.getNumber());
            ps.setString(3, carriage.getId());
            ps.setInt(4, carriage.getRouteToTrainTimeTable());
            ps.executeUpdate();
        }catch(Exception ex){
            ex.printStackTrace();
        }
    }

    @Override
    public void deleteCarriage(String carriageID) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            PreparedStatement ps =
connection.prepareStatement(SQL_DELETE_CARRIAGE_BY_ID);
            ps.setString(1, carriageID);
            ps.executeUpdate();
        }catch (Exception ex){
            System.out.println(ex.getMessage());
        }
    }

    @Override
    public void updateCarriageNumber(String id, Integer number) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try{
            PreparedStatement ps =
connection.prepareStatement(SQL_UPDATE_CARRIAGE_NUMBER);
            ps.setInt(1, number);
            ps.setString(2, id);
            ps.executeUpdate();
        }catch (Exception ex){
            System.out.println(ex.getMessage());
        }
    }

    @Override
    public void updateCarriageRouteToTrain(String id, Integer routeToTrain) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try{
            PreparedStatement ps =
connection.prepareStatement(SQL_UPDATE_CARRIAGE_ROUTE_TO_TRAIN_TIMETABLE);
            ps.setInt(1, routeToTrain);
            ps.setString(2, id);
            ps.executeUpdate();
        }catch (Exception ex){
            System.out.println(ex.getMessage());
        }
    }
}
```

ClientDao

```
package app.model;

import app.model.entities.Client;

import java.util.List;

public interface ClientDao {
    List<Client> findAllClient();
    void insertClient(Client client);
    void deleteClient(Integer id);
    void updateClient(String login, String email);
}

package app.model.sql;

import app.model.ClientDao;
import app.model.entities.Client;
import org.postgresql.util.PSQLException;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

public class SqlClientDao implements ClientDao {
    public static final String SQL_FIND_ALL_CLIENT = "SELECT * FROM client";
    public static final String SQL_INSERT_CLIENT = "INSERT INTO client
(email, login, full_name) " +
            "VALUES (?, ?, ?)";

    public static final String SQL_DELETE_CLIENT_BY_ID = "DELETE FROM client
WHERE id_client = ?";
    public static final String SQL_UPDATE_CLIENT_EMAIL = "UPDATE client SET
email = ? WHERE login LIKE ?";


    @Override
    public List<Client> findAllClient() {
        List<Client> result = new ArrayList<Client>();
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            Statement query = connection.createStatement();
            ResultSet rs = query.executeQuery(SQL_FIND_ALL_CLIENT);
            while (rs.next()) {
                result.add(new Client(rs.getInt(1), rs.getString(2),
                        rs.getString(3), rs.getString(4)));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
```

```
    }

    @Override
    public void insertClient(Client client) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try{
            PreparedStatement ps =
connection.prepareStatement(SQL_INSERT_CLIENT);
            ps.setString(1, client.getE_mail());
            ps.setString(2, client.getLogin());
            ps.setString(3, client.getFull_name());
            ps.executeUpdate();
        }catch(Exception ex){
            ex.printStackTrace();
        }
    }

    @Override
    public void deleteClient(Integer id) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            PreparedStatement ps =
connection.prepareStatement(SQL_DELETE_CLIENT_BY_ID);
            ps.setInt(1, id);
            ps.executeUpdate();
        }catch (Exception ex){
            System.out.println(ex.getMessage());
        }
    }

    @Override
    public void updateClient(String login, String email) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try{
            PreparedStatement ps =
connection.prepareStatement(SQL_UPDATE_CLIENT_EMAIL);
            ps.setString(1, email);
            ps.setString(2, login);
            ps.executeUpdate();
        }catch (Exception ex){
            System.out.println(ex.getMessage());
        }
    }
}
```

PlaceDao

```
package app.model;

import app.model.entities.Carriage;
import app.model.entities.Client;
```

```
import app.model.entities.Place;

import java.util.List;

public interface PlaceDao {
    List<Place> findAllPlace();
    void insertPlace(Place place);
    void deletePlace(Integer id);
    void updateNumberPlace(Integer id_place, Integer number_place);
}


package app.model.sql;

import app.model.PlaceDao;
import app.model.entities.Place;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

public class SqlPlaceDao implements PlaceDao {
    public static final String SQL_FIND_ALL_PLACE = "SELECT * FROM place";
    public static final String SQL_INSERT_PLACE = "INSERT INTO place
(number_place, carriage_id) " +
            "VALUES (?, ?)";

    public static final String SQL_DELETE_PLACE_BY_ID = "DELETE FROM place
WHERE id_place = ?";
    public static final String SQL_UPDATE_PLACE_NUMBER = "UPDATE place SET
number_place = ? WHERE id_place LIKE ?";



    @Override
    public List<Place> findAllPlace() {
        List<Place> result = new ArrayList<Place>();
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            Statement query = connection.createStatement();
            ResultSet rs = query.executeQuery(SQL_FIND_ALL_PLACE);
            while (rs.next()) {
                result.add(new Place(rs.getInt(1), rs.getInt(2),
                        rs.getString(3)));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
    }

    @Override
```

```
    public void insertPlace(Place place) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try{
            PreparedStatement ps =
connection.prepareStatement(SQL_INSERT_PLACE);
            ps.setInt(1, place.getNumber());
            ps.setString(2, place.getCarriage());
            ps.executeUpdate();
        }catch(Exception ex){
            ex.printStackTrace();
        }
    }

    @Override
    public void deletePlace(Integer id) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            PreparedStatement ps =
connection.prepareStatement(SQL_DELETE_PLACE_BY_ID);
            ps.setInt(1, id);
            ps.executeUpdate();
        }catch (Exception ex){
            System.out.println(ex.getMessage());
        }
    }

    @Override
    public void updateNumberPlace(Integer id_place, Integer number_place) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try{
            PreparedStatement ps =
connection.prepareStatement(SQL_UPDATE_PLACE_NUMBER);
            ps.setInt(1, number_place);
            ps.setInt(2, id_place);
            ps.executeUpdate();
        }catch (Exception ex){
            System.out.println(ex.getMessage());
        }
    }
}
```

RouteDao

```
package app.model;

import app.model.entities.Route;

import java.util.List;

public interface RouteDao {
    List<Route> findAllRoute();
    void insertRoute(Route route);
    void deleteRoute(Integer id);
```

```
    void updatePlaceOfDeparture(Integer id, String departure);
    void updatePlaceOfArrival(Integer id, String arrival);
}
```

```
package app.model.sql;

import app.model.RouteDao;
import app.model.entities.Route;
import org.postgresql.util.PSQLException;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

public class SqlRouteDao implements RouteDao {
    public static final String SQL_FIND_ALL_ROUTE = "SELECT * FROM route";
    public static final String SQL_INSERT_ROUTE = "INSERT INTO route
(place_of_departure, place_of_arrival) " +
            "VALUES (?, ?)";
    public static final String SQL_DELETE_ROUTE_BY_ID = "DELETE FROM route
WHERE id_route = ?";
    public static final String SQL_UPDATE_ROUTE_DEPARTURE = "UPDATE route SET
place_of_departure = ? WHERE id_route LIKE ?";
    public static final String SQL_UPDATE_ROUTE_ARRIVAL = "UPDATE route SET
place_of_arrival = ? WHERE id_route LIKE ?";
```

```
    @Override
    public List<Route> findAllRoute() {
        List<Route> result = new ArrayList<Route>();
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            Statement query = connection.createStatement();
            ResultSet rs = query.executeQuery(SQL_FIND_ALL_ROUTE);
            while (rs.next()) {
                result.add(new Route(rs.getInt(1), rs.getString(2),
                        rs.getString(3)));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
    }

    @Override
    public void insertRoute(Route route) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
```

```
            PreparedStatement ps =
connection.prepareStatement(SQL_INSERT_ROUTE);
            ps.setString(1, route.getPlace_of_departure());
            ps.setString(2, route.getPlace_of_arrival());
            ps.executeUpdate();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    @Override
    public void deleteRoute(Integer id) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            PreparedStatement ps =
connection.prepareStatement(SQL_DELETE_ROUTE_BY_ID);
            ps.setInt(1, id);
            ps.executeUpdate();
        }catch (Exception ex){
            System.out.println(ex.getMessage());
        }
    }

    @Override
    public void updatePlaceOfDeparture(Integer id, String departure) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try{
            PreparedStatement ps =
connection.prepareStatement(SQL_UPDATE_ROUTE_DEPARTURE);
            ps.setString(1, departure);
            ps.setInt(2, id);
            ps.executeUpdate();
        }catch (Exception ex){
            System.out.println(ex.getMessage());
        }
    }

    @Override
    public void updatePlaceOfArrival(Integer id, String arrival) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try{
            PreparedStatement ps =
connection.prepareStatement(SQL_UPDATE_ROUTE_ARRIVAL);
            ps.setString(1, arrival);
            ps.setInt(2, id);
            ps.executeUpdate();
        }catch (Exception ex){
            System.out.println(ex.getMessage());
        }
    }
}
```

RouteToTrainTimeTableDao

```
package app.model;

import app.model.entities.Route;
import app.model.entities.RouteToTrainTimeTable;

import java.util.Date;
import java.util.List;

public interface RouteToTrainTimeTableDao {
    List<RouteToTrainTimeTable> findAllRouteToTrainTimeTable();
    void insertRouteToTrainTimeTable(RouteToTrainTimeTable
routeToTrainTimeTable);
    void deleteRouteToTrainTimeTable(Integer id);
    void updateRouteToTrainTimeTableOfDeparture(Integer id, Date departure);
    void updateRouteToTrainTimeTableOfArrival(Integer id, Date arrival);
}


package app.model.sql;

import app.model.RouteToTrainTimeTableDao;
import app.model.entities.RouteToTrainTimeTable;

import java.sql.*;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class SqlRouteToTrainTimeTableDao implements RouteToTrainTimeTableDao
{
    public static final String SQL_FIND_ALL_ROUTE_TO_TRAIN = "SELECT * FROM
route_to_train_time_table";
    public static final String SQL_INSERT_ROUTE_TO_TRAIN_TIMETABLE =
            "INSERT INTO route_to_train_time_table (id_route, id_train,
arrive_time, depart_time) " +
            "VALUES (?, ?, ?, ?)";
    public static final String SQL_DELETE_ROUTE_TO_TRAIN_TIMETABLE_BY_ID =
"DELETE FROM route_to_train_time_table WHERE id_route_to_train = ?";
    public static final String SQL_UPDATE_ROUTE_TO_TRAIN_TIMETABLE_DEPARTURE
=
            "UPDATE route_to_train_time_table SET depart_time = ? WHERE
id_route_to_train = ?";
    public static final String SQL_UPDATE_ROUTE_TO_TRAIN_TIMETABLE_ARRIVAL =
            "UPDATE route_to_train_time_table SET arrive_time = ? WHERE
id_route_to_train = ?";


    @Override
    public List<RouteToTrainTimeTable> findAllRouteToTrainTimeTable() {
        List<RouteToTrainTimeTable> result = new
ArrayList<RouteToTrainTimeTable>();
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
```

```
        try {
            Statement query = connection.createStatement();
            ResultSet rs = query.executeQuery(SQL_FIND_ALL_ROUTE_TO_TRAIN);
            while (rs.next()) {
                result.add(new RouteToTrainTimeTable(rs.getInt(1),
rs.getInt(2),
                        rs.getString(3), rs.getTimestamp(4),
rs.getTimestamp(5)));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
    }

    @Override
    public void insertRouteToTrainTimeTable(RouteToTrainTimeTable
routeToTrainTimeTable) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try{
            PreparedStatement ps =
connection.prepareStatement(SQL_INSERT_ROUTE_TO_TRAIN_TIMETABLE);
            ps.setInt(1, routeToTrainTimeTable.getRoute());
            ps.setString(2, routeToTrainTimeTable.getTrain());
            ps.setTimestamp(3, (Timestamp)
routeToTrainTimeTable.getArriveTime());
            ps.setTimestamp(4, (Timestamp)
routeToTrainTimeTable.getDepartTime());
            ps.executeUpdate();
        }catch(Exception ex){
            ex.printStackTrace();
        }
    }

    @Override
    public void deleteRouteToTrainTimeTable(Integer id) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            PreparedStatement ps =
connection.prepareStatement(SQL_DELETE_ROUTE_TO_TRAIN_TIMETABLE_BY_ID);
            ps.setInt(1, id);
            ps.executeUpdate();
        }catch (Exception ex){
            System.out.println(ex.getMessage());
        }
    }

    @Override
    public void updateRouteToTrainTimeTableOfDeparture(Integer id, Date
departure) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try{
            PreparedStatement ps =
```

```
connection.prepareStatement(SQL_UPDATE_ROUTE_TO_TRAIN_TIMETABLE_DEPARTURE);
            ps.setTimestamp(1, (Timestamp) departure);
            ps.setInt(2, id);
            ps.executeUpdate();
        }catch (Exception ex){
            System.out.println(ex.getMessage());
        }
    }

    @Override
    public void updateRouteToTrainTimeTableOfArrival(Integer id, Date
arrival) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try{
            PreparedStatement ps =
connection.prepareStatement(SQL_UPDATE_ROUTE_TO_TRAIN_TIMETABLE_ARRIVAL);
            ps.setTimestamp(1, (Timestamp) arrival);
            ps.setInt(2, id);
            ps.executeUpdate();
        }catch (Exception ex){
            System.out.println(ex.getMessage());
        }
    }
}
```

ServiceDao

```
package app.model;

import app.model.dto.CarriageTypeFromTo;
import app.model.dto.RouteFromToDepartMore;
import app.model.dto.TicketPricePrivilegeCarriage;

import java.math.BigDecimal;
import java.sql.Timestamp;
import java.util.List;

public interface ServiceDao {
    List<RouteFromToDepartMore> findRouteFromToDepartMore(String arrive,
String depart, Timestamp departTime);
    List<CarriageTypeFromTo> findRouteWithTypeCarriage(String placeDepart,
String placeArrive, String type);
    List<TicketPricePrivilegeCarriage>
findTicketPricePrivilegeCarriage(BigDecimal price, Boolean privilege, Integer
carriageNumber);
    void deleteAll();
}
```

```
package app.model.sql;

import app.model.ServiceDao;
import app.model.dto.CarriageTypeFromTo;
import app.model.dto.RouteFromToDepartMore;
import app.model.dto.TicketPricePrivilegeCarriage;
```

```
import java.math.BigDecimal;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class SqlServiceDao implements ServiceDao {

    public static final String SQL_DELETE_ALL = "DELETE FROM carriage; " +
            "DELETE FROM client; " +
            "DELETE FROM place; " +
            "DELETE FROM route; " +
            "DELETE FROM route_to_train_time_table; " +
            "DELETE FROM ticket; " +
            "DELETE FROM train;";

    public static final String SQL_TICKET_WITH_PRICELESS_PRIV_CARLESS =
"SELECT \n" +
            "\tt.price, \n" +
            "\tt.privillege,\n" +
            "\tt.buy_date,\n" +
            "\tc.full_name,\n" +
            "\tr.place_of_arrival,\n" +
            "\tr.place_of_departure, \n" +
            "\trtt.arrive_time,\n" +
            "\trtt.depart_time,\n" +
            "\ttr.id_train,\n" +
            "\tcar.number,\n" +
            "\tp.number_place\n" +
            "from ticket as t \n" +
            "inner join client as c \n" +
            "\ton t.fk_client_id = c.id_client\n" +
            "inner join place as p\n" +
            "\ton t.fk_id_place = p.id_place\n" +
            "inner join carriage as car \n" +
            "\ton p.carriage_id = car.id_carriage\n" +
            "inner join route_to_train_time_table as rtt\n" +
            "\ton car.fk_route_to_train_time_table = rtt.id_route_to_train\n"
+
            "inner join route as r\n" +
            "\ton rtt.id_route = r.id_route\n" +
            "inner join train as tr\n" +
            "\ton rtt.id_train = tr.id_train\n" +
            "where t.price < ? AND t.privillege = ? AND car.number < ?";

    public static final String SQL_ROUTE_FROM_TO_DEPART_MORE = "SELECT
r.place_of_departure, r.place_of_arrival, " +
            "rtt.depart_time, rtt.arrive_time FROM route AS r " +
            "INNER JOIN route_to_train_time_table AS rtt " +
            "ON r.id_route = rtt.id_route " +
            "WHERE r.place_of_departure = ? AND r.place_of_arrival = ? AND
rtt.depart_time >= ?";

    public static final String SQL_CARRIAGE_ROUTE_TYPE = "SELECT
r.place_of_departure, r.place_of_arrival, " +
            "car.type, car.id_carriage, rtt.arrive_time, " +
```

```
            "rtt.depart_time FROM route AS r INNER JOIN
route_to_train_time_table AS rtt " +
            "ON r.id_route = rtt.id_route " +
            "INNER JOIN carriage AS car ON car.fk_route_to_train_time_table =
rtt.id_route_to_train " +
            "WHERE r.place_of_departure = ? AND r.place_of_arrival = ? AND
car.type = ?";

    @Override
    public List<TicketPricePrivilegeCarriage>
findTicketPricePrivilegeCarriage(BigDecimal price, Boolean privilege, Integer
carriageNumber) {
        List<TicketPricePrivilegeCarriage> result = new
ArrayList<TicketPricePrivilegeCarriage>();
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            PreparedStatement ps =
connection.prepareStatement(SQL_TICKET_WITH_PRICELESS_PRIV_CARLESS);
            ps.setBigDecimal(1, price);
            ps.setBoolean(2, privilege);
            ps.setInt(3, carriageNumber);
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                result.add(new
TicketPricePrivilegeCarriage(rs.getBigDecimal(1), rs.getBoolean(2),
                        rs.getTimestamp(3), rs.getString(4), rs.getString(5),
rs.getString(6),
                        rs.getTimestamp(7), rs.getTimestamp(8),
rs.getString(9),
                        rs.getInt(10), rs.getInt(11)));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
    }

    @Override
    public List<RouteFromToDepartMore> findRouteFromToDepartMore(String
arrive, String depart, Timestamp departTime) {
        List<RouteFromToDepartMore> result = new
ArrayList<RouteFromToDepartMore>();
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            PreparedStatement ps =
connection.prepareStatement(SQL_ROUTE_FROM_TO_DEPART_MORE);
            ps.setString(1, depart);
            ps.setString(2, arrive);
            ps.setTimestamp(3, departTime);
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                result.add(new RouteFromToDepartMore(rs.getString(1),
rs.getString(2),
                        rs.getTimestamp(3), rs.getTimestamp(4)));
```

```
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
    }

    @Override
    public List<CarriageTypeFromTo> findRouteWithTypeCarriage(String
placeDepart, String placeArrive, String type) {
        List<CarriageTypeFromTo> result = new
ArrayList<CarriageTypeFromTo>();
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            PreparedStatement ps =
connection.prepareStatement(SQL_CARRIAGE_ROUTE_TYPE);
            ps.setString(1, placeDepart);
            ps.setString(2, placeArrive);
            ps.setString(3, type);
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                result.add(new CarriageTypeFromTo(rs.getString(1),
rs.getString(2), rs.getString(3),
                        rs.getString(4), rs.getTimestamp(5),
rs.getTimestamp(6)));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
    }

    @Override
    public void deleteAll() {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            Statement query = connection.createStatement();
            query.executeUpdate(SQL_DELETE_ALL);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

TicketDao

```
package app.model;

import app.model.entities.Ticket;

import java.math.BigDecimal;
import java.util.Date;
```

```java
import java.util.List;

public interface TicketDao {
    List<Ticket> findAllTicket();
    void insertTicket(Ticket ticket);
    void deleteTicket(Integer id);
    void updateTicketPriceById(Integer id, BigDecimal price);
    void updateTicketBuyDateById(Integer id, Date buyDate);
    void updateTicketPrivilegeById(Integer id, Boolean privilege);
}
```

```java
package app.model.sql;

import app.model.TicketDao;
import app.model.entities.Ticket;

import java.math.BigDecimal;
import java.sql.*;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class SqlTicketDao implements TicketDao {
    public static final String SQL_FIND_ALL_TICKET = "SELECT * FROM ticket";
    public static final String SQL_INSERT_TICKET = "INSERT INTO ticket
(fk_client_id, fk_id_place, price, buy_date, privillege ) " +
            "VALUES (?, ?, ?, ?, ?)";
    public static final String SQL_DELETE_TICKET_BY_ID = "DELETE FROM ticket
WHERE id_ticket = ?";
    public static final String SQL_UPDATE_TICKET_PRICE =
            "UPDATE ticket SET price = ? WHERE id_ticket LIKE ?";
    public static final String SQL_UPDATE_TICKET_BUY_DATE =
            "UPDATE ticket SET buy_date = ? WHERE id_ticket LIKE ?";
    public static final String SQL_UPDATE_TICKET_PRIVILEGE =
            "UPDATE ticket SET privillege = ? WHERE id_ticket LIKE ?";

    @Override
    public List<Ticket> findAllTicket() {
        List<Ticket> result = new ArrayList<Ticket>();
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            Statement query = connection.createStatement();
            ResultSet rs = query.executeQuery(SQL_FIND_ALL_TICKET);
            while (rs.next()) {
                result.add(new Ticket(rs.getInt(1), rs.getInt(2),
                        rs.getInt(3), rs.getBigDecimal(4),
rs.getTimestamp(5), rs.getBoolean(6)));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
    }
```

```
    @Override
    public void insertTicket(Ticket ticket) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            PreparedStatement ps =
connection.prepareStatement(SQL_INSERT_TICKET);
            ps.setInt(1, ticket.getClient());
            ps.setInt(2, ticket.getPlace());
            ps.setBigDecimal(3, ticket.getPrice());
            ps.setTimestamp(4, (Timestamp) ticket.getBuy_date());
            ps.setBoolean(5, ticket.getPrivilege());
            ps.executeUpdate();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    @Override
    public void deleteTicket(Integer id) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            PreparedStatement ps =
connection.prepareStatement(SQL_DELETE_TICKET_BY_ID);
            ps.setInt(1, id);
            ps.executeUpdate();
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

    @Override
    public void updateTicketPriceById(Integer id, BigDecimal price) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try{
            PreparedStatement ps =
connection.prepareStatement(SQL_UPDATE_TICKET_PRICE);
            ps.setBigDecimal(1, price);
            ps.setInt(2, id);
            ps.executeUpdate();
        }catch (Exception ex){
            System.out.println(ex.getMessage());
        }
    }

    @Override
    public void updateTicketBuyDateById(Integer id, Date buyDate) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try{
            PreparedStatement ps =
connection.prepareStatement(SQL_UPDATE_TICKET_BUY_DATE);
            ps.setTimestamp(1, (Timestamp) buyDate);
```

```
            ps.setInt(2, id);
            ps.executeUpdate();
        }catch (Exception ex){
            System.out.println(ex.getMessage());
        }
    }

    @Override
    public void updateTicketPrivilegeById(Integer id, Boolean privilege) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try{
            PreparedStatement ps =
connection.prepareStatement(SQL_UPDATE_TICKET_PRIVILEGE);
            ps.setBoolean(1, privilege);
            ps.setInt(2, id);
            ps.executeUpdate();
        }catch (Exception ex){
            System.out.println(ex.getMessage());
        }
    }
}
```

TrainDao

```
package app.model;

import app.model.entities.Route;
import app.model.entities.Train;

import java.util.List;

public interface TrainDao {
    List<Train> findAllTrain();
    void InsertTrain(Train train);
    void deleteTrain(String trainId);
    void updateTrainMaxCarriage(String id, Integer maxCarriage);
}
```

```
package app.model.sql;

import app.model.TrainDao;
import app.model.entities.Route;
import app.model.entities.Train;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class SqlTrainDao implements TrainDao{
    public static final String SQL_FIND_ALL_TRAIN = "SELECT * FROM train";
    public static final String SQL_INSERT_TRAIN = "INSERT INTO train
(id_train, max_carriage) " +
            "VALUES (?, ?)";
```

```
    public static final String SQL_DELETE_TRAIN_BY_ID = "DELETE FROM train
WHERE id_train = ?";
    public static final String SQL_UPDATE_TRAIN_MAX_CARRIAGE =
            "UPDATE train SET max_carriage = ? WHERE id_train LIKE ?";

    @Override
    public List<Train> findAllTrain() {
        List<Train> result = new ArrayList<Train>();
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            Statement query = connection.createStatement();
            ResultSet rs = query.executeQuery(SQL_FIND_ALL_TRAIN);
            while (rs.next()) {
                result.add(new Train( rs.getString(1), rs.getInt(2)));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
    }

    @Override
    public void InsertTrain(Train train) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            PreparedStatement ps =
connection.prepareStatement(SQL_INSERT_TRAIN);
            ps.setString(1, train.getId_train());
            ps.setInt(2, train.getMax_carriage());
            ps.executeUpdate();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    @Override
    public void deleteTrain(String trainId) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            PreparedStatement ps =
connection.prepareStatement(SQL_DELETE_TRAIN_BY_ID);
            ps.setString(1, trainId);
            ps.executeUpdate();
        }catch (Exception ex){
            System.out.println(ex.getMessage());
        }
    }

    @Override
    public void updateTrainMaxCarriage(String id, Integer maxCarriage) {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try{
```

```
            PreparedStatement ps =
connection.prepareStatement(SQL_UPDATE_TRAIN_MAX_CARRIAGE);
            ps.setInt(1, maxCarriage);
            ps.setString(2, id);
            ps.executeUpdate();
        }catch (Exception ex){
            System.out.println(ex.getMessage());
        }
    }
}
```

SqlConnection

```
package app.model.sql;

import constant.Configs;

import java.sql.Connection;
import java.sql.DriverManager;

public class SqlConnection extends Configs {

    private static SqlConnection instance;

    private Connection connection;

    private SqlConnection() {
        String connectionString = "jdbc:postgresql://" + dbHost + ":" +
dbPort + "/" + dbName;
        try {
            Class.forName("org.postgresql.Driver").newInstance();
            connection = DriverManager.getConnection(connectionString,
dbUser, dbPass);

        } catch (Exception ignored) {

        }


    }

    public static SqlConnection getInstance() {
        if (instance == null) {
            instance = new SqlConnection();
        }
        return instance;
    }

    public Connection getConnection() {
        return connection;
```

```
    }
}
```

RandomServiceDao

```
package app.model;

public interface RandomServiceDao {
    void fillRandomTable(int numberRow);
}
```

```
package app.model.sql;

import app.model.*;
import app.model.entities.*;

import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.List;

public class SqlRandomServiceDao implements RandomServiceDao {
    //filling procedure
    /*
     * client
     * route
     * train
     * route_to_train_time_table
     * carriage
     * place
     * ticket
     * */
    public static final String SQL_RANDOM_CHAR = "chr(trunc(97 +
random()*25)::int)";
    public static final String SQL_RANDOM_INT = "floor(random()*(200-
1+1))+1";
    public static final String SQL_RANDOM_NUMERIC = "random()*(1300 - 50) +
50";
    public static final String SQL_RANDOM_DATE = "'2020-01-1
20:00:00'::timestamp + " +
            "random() * ('2020-12-30 20:00:00'::timestamp - '2020-01-10
10:00:00'::timestamp)";

    public static final String SQL_SELECT_FK_ROUTE = "SELECT r.id_route FROM
route AS r LEFT OUTER JOIN route_to_train_time_table AS rtt " +
            "ON r.id_route = rtt.id_route WHERE rtt.id_route IS NULL";

    public static final String SQL_SELECT_FK_ROUTE_TO_TRAIN_TIME_TABLE =
"SELECT rtt.id_route_to_train FROM route_to_train_time_table AS rtt LEFT
OUTER JOIN carriage AS c " +
            "ON rtt.id_route_to_train = c.fk_route_to_train_time_table WHERE
```

```
c.fk_route_to_train_time_table IS NULL";

    public static final String SQL_SELECT_FK_TRAIN = "SELECT t.id_train FROM
train AS t LEFT OUTER JOIN route_to_train_time_table AS rtt " +
            "ON t.id_train = rtt.id_train WHERE rtt.id_train IS NULL";

    public static final String SQL_SELECT_FK_CARRIAGE = "SELECT c.id_carriage
FROM carriage AS c LEFT OUTER JOIN place AS p " +
            "ON c.id_carriage = p.carriage_id WHERE p.carriage_id IS NULL";

    public static final String SQL_SELECT_FK_CLIENT = "SELECT c.id_client
FROM client AS c LEFT OUTER JOIN ticket AS t " +
            "ON c.id_client = t.fk_client_id WHERE t.fk_client_id IS NULL";

    public static final String SQL_SELECT_FK_PLACE = "SELECT p.id_place FROM
place AS p LEFT OUTER JOIN ticket AS t " +
            "ON p.id_place = t.fk_id_place WHERE t.fk_id_place IS NULL";


    private String getRandomChar() {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            Statement randomString = connection.createStatement();
            ResultSet rs = randomString.executeQuery("SELECT " +
SQL_RANDOM_CHAR);
            rs.next();

            String res = rs.getString(1);
            return res;
        } catch (Exception throwables) {
            throwables.printStackTrace();
        }
        return null;
    }

    private String getRandomString(int characters) {
        StringBuilder strBuild = new StringBuilder();
        for (int i = 0; i < characters; i++) {
            if (getRandomChar() != null) strBuild.append(getRandomChar());
        }
        return String.valueOf(strBuild);
    }

    private Integer getRandomInteger() {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            Statement randomInt = connection.createStatement();
            ResultSet rs = randomInt.executeQuery("SELECT " +
SQL_RANDOM_INT);
            rs.next();

            Integer res = rs.getInt(1);
            return res;
        } catch (Exception throwables) {
```

```
            throwables.printStackTrace();
        }
        return null;
    }

    private Timestamp getRandomTimestamp() {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            Statement randomInt = connection.createStatement();
            ResultSet rs = randomInt.executeQuery("SELECT " +
SQL_RANDOM_DATE);
            rs.next();

            Timestamp res = rs.getTimestamp(1);
            return res;
        } catch (Exception throwables) {
            throwables.printStackTrace();
        }
        return null;
    }

    private BigDecimal getRandomNumeric(){
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            Statement randomInt = connection.createStatement();
            ResultSet rs = randomInt.executeQuery("SELECT " +
SQL_RANDOM_NUMERIC);
            rs.next();

            BigDecimal res = rs.getBigDecimal(1);
            return res;
        } catch (Exception throwables) {
            throwables.printStackTrace();
        }
        return null;
    }

    private Boolean getRandomBoolean(){
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            Statement randomInt = connection.createStatement();
            ResultSet rs = randomInt.executeQuery("SELECT floor(random() *
2)");
            rs.next();
            int res = rs.getInt(1);
            if(res == 1) return true;
        } catch (Exception throwables) {
            throwables.printStackTrace();
        }
        return false;
    }

    private List<Integer> findAllRouteAvailableRouteId() {
```

```
        List<Integer> result = new ArrayList<Integer>();
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            Statement query = connection.createStatement();
            ResultSet rs = query.executeQuery(SQL_SELECT_FK_ROUTE);
            while (rs.next()) {
                result.add(rs.getInt(1));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
    }

    private List<Integer> findAllCarriageRouteTrainTimeTable() {
        List<Integer> result = new ArrayList<Integer>();
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            Statement query = connection.createStatement();
            ResultSet rs =
query.executeQuery(SQL_SELECT_FK_ROUTE_TO_TRAIN_TIME_TABLE);
            while (rs.next()) {
                result.add(rs.getInt(1));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
    }

    private List<String> findAllCarriage() {
        List<String> result = new ArrayList<String>();
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            Statement query = connection.createStatement();
            ResultSet rs = query.executeQuery(SQL_SELECT_FK_CARRIAGE);
            while (rs.next()) {
                result.add(rs.getString(1));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
    }

    private List<String> findAllRouteAvailableTrainId() {
        List<String> result = new ArrayList<String>();
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            Statement query = connection.createStatement();
            ResultSet rs = query.executeQuery(SQL_SELECT_FK_TRAIN);
            while (rs.next()) {
```

```
                result.add(rs.getString(1));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
    }

    private List<Integer> findAllPlace() {
        List<Integer> result = new ArrayList<Integer>();
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            Statement query = connection.createStatement();
            ResultSet rs = query.executeQuery(SQL_SELECT_FK_PLACE);
            while (rs.next()) {
                result.add(rs.getInt(1));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
    }

    private List<Integer> findAllClient() {
        List<Integer> result = new ArrayList<Integer>();
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            Statement query = connection.createStatement();
            ResultSet rs = query.executeQuery(SQL_SELECT_FK_CLIENT);
            while (rs.next()) {
                result.add(rs.getInt(1));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
    }

    private void fillClient(int numberRow) {
        ClientDao insertClient = new SqlClientDao();
        for (int i = 0; i < numberRow; i++) {
            insertClient.insertClient(new Client(null, getRandomString(9) +
"@gmail.com", getRandomString(7),
                    getRandomString(7) + " " + getRandomString(8)));
        }
    }

    private void fillRoute(int numberRow) {
        RouteDao insertRoute = new SqlRouteDao();
        for (int i = 0; i < numberRow; i++) {
            insertRoute.insertRoute(new Route(null, getRandomString(20),
getRandomString(20)));
        }
    }
```

```
    private void fillTrain(int numberRow) {
        TrainDao insertTrain = new SqlTrainDao();
        for (int i = 0; i < numberRow; i++) {
            insertTrain.InsertTrain(new Train(getRandomString(5),
getRandomInteger()));
        }
    }

    private void fillRouteToTrainTimeTable() {
        RouteToTrainTimeTableDao insertRouteToTrainTimeTable = new
SqlRouteToTrainTimeTableDao();
        List<Integer> id_route = findAllRouteAvailableRouteId();
        List<String> id_train = findAllRouteAvailableTrainId();
        int numberRow = id_route.size();
        if(id_route.size() > id_train.size())
            numberRow = id_train.size();
        for (int i = 0; i < numberRow; i++) {
            insertRouteToTrainTimeTable.insertRouteToTrainTimeTable(new
RouteToTrainTimeTable(null, id_route.get(i),
                    id_train.get(i), getRandomTimestamp(),
getRandomTimestamp()));
        }
    }

    private void fillCarriage() {
        CarriageDao insertCarriage = new SqlCarriageDao();
        List<Integer> fk_id_route_to_train_time_table =
findAllCarriageRouteTrainTimeTable();
        for (Integer integer : fk_id_route_to_train_time_table) {
            insertCarriage.insertCarriage(new Carriage(getRandomString(7),
getRandomInteger(),
                    getRandomString(5), integer));
        }
    }

    private void fillPlace() {
        PlaceDao insertPlace = new SqlPlaceDao();
        List<String> id_carriage = findAllCarriage();
        for (String s : id_carriage) {
            insertPlace.insertPlace(new Place(null, getRandomInteger(), s));
        }

    }

    private void fillTicket() {
        TicketDao insertTicket = new SqlTicketDao();
        List<Integer> fk_client_id = findAllClient();
        List<Integer> fk_id_place = findAllPlace();
        int numberRow = fk_client_id.size();
        if(fk_client_id.size() > fk_id_place.size())
            numberRow = fk_id_place.size();
        for(int i = 0; i < numberRow; i++){
            insertTicket.insertTicket(new Ticket(null, fk_client_id.get(i),
fk_id_place.get(i),
                    getRandomNumeric(), getRandomTimestamp(),
```

```
getRandomBoolean()));
        }
    }


    @Override
    public void fillRandomTable(int numberRow) {
        fillClient(numberRow);
        fillRoute(numberRow);
        fillTrain(numberRow);
        fillRouteToTrainTimeTable();
        fillCarriage();
        fillPlace();
        fillTicket();
    }
}
```

ServiceDao

```
package app.model;

import app.model.dto.CarriageTypeFromTo;
import app.model.dto.RouteFromToDepartMore;
import app.model.dto.TicketPricePrivilegeCarriage;

import java.math.BigDecimal;
import java.sql.Timestamp;
import java.util.List;

public interface ServiceDao {
    List<RouteFromToDepartMore> findRouteFromToDepartMore(String arrive,
String depart, Timestamp departTime);
    List<CarriageTypeFromTo> findRouteWithTypeCarriage(String placeDepart,
String placeArrive, String type);
    List<TicketPricePrivilegeCarriage>
findTicketPricePrivilegeCarriage(BigDecimal price, Boolean privilege, Integer
carriageNumber);
    void deleteAll();
}
package app.model.sql;

import app.model.ServiceDao;
import app.model.dto.CarriageTypeFromTo;
import app.model.dto.RouteFromToDepartMore;
import app.model.dto.TicketPricePrivilegeCarriage;

import java.math.BigDecimal;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class SqlServiceDao implements ServiceDao {
```

```
    public static final String SQL_DELETE_ALL = "DELETE FROM carriage; " +
            "DELETE FROM client; " +
            "DELETE FROM place; " +
            "DELETE FROM route; " +
            "DELETE FROM route_to_train_time_table; " +
            "DELETE FROM ticket; " +
            "DELETE FROM train;";

    public static final String SQL_TICKET_WITH_PRICELESS_PRIV_CARLESS =
"SELECT \n" +
            "\tt.price, \n" +
            "\tt.privillege,\n" +
            "\tt.buy_date,\n" +
            "\tc.full_name,\n" +
            "\tr.place_of_arrival,\n" +
            "\tr.place_of_departure, \n" +
            "\trtt.arrive_time,\n" +
            "\trtt.depart_time,\n" +
            "\ttr.id_train,\n" +
            "\tcar.number,\n" +
            "\tp.number_place\n" +
            "from ticket as t \n" +
            "inner join client as c \n" +
            "\ton t.fk_client_id = c.id_client\n" +
            "inner join place as p\n" +
            "\ton t.fk_id_place = p.id_place\n" +
            "inner join carriage as car \n" +
            "\ton p.carriage_id = car.id_carriage\n" +
            "inner join route_to_train_time_table as rtt\n" +
            "\ton car.fk_route_to_train_time_table = rtt.id_route_to_train\n"
+
            "inner join route as r\n" +
            "\ton rtt.id_route = r.id_route\n" +
            "inner join train as tr\n" +
            "\ton rtt.id_train = tr.id_train\n" +
            "where t.price < ? AND t.privillege = ? AND car.number < ?";

    public static final String SQL_ROUTE_FROM_TO_DEPART_MORE = "SELECT
r.place_of_departure, r.place_of_arrival, " +
            "rtt.depart_time, rtt.arrive_time FROM route AS r " +
            "INNER JOIN route_to_train_time_table AS rtt " +
            "ON r.id_route = rtt.id_route " +
            "WHERE r.place_of_departure = ? AND r.place_of_arrival = ? AND
rtt.depart_time >= ?";

    public static final String SQL_CARRIAGE_ROUTE_TYPE = "SELECT
r.place_of_departure, r.place_of_arrival, " +
            "car.type, car.id_carriage, rtt.arrive_time, " +
            "rtt.depart_time FROM route AS r INNER JOIN
route_to_train_time_table AS rtt " +
            "ON r.id_route = rtt.id_route " +
            "INNER JOIN carriage AS car ON car.fk_route_to_train_time_table =
rtt.id_route_to_train " +
            "WHERE r.place_of_departure = ? AND r.place_of_arrival = ? AND
car.type = ?";
```

```
    @Override
    public List<TicketPricePrivilegeCarriage>
findTicketPricePrivilegeCarriage(BigDecimal price, Boolean privilege, Integer
carriageNumber) {
        List<TicketPricePrivilegeCarriage> result = new
ArrayList<TicketPricePrivilegeCarriage>();
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            PreparedStatement ps =
connection.prepareStatement(SQL_TICKET_WITH_PRICELESS_PRIV_CARLESS);
            ps.setBigDecimal(1, price);
            ps.setBoolean(2, privilege);
            ps.setInt(3, carriageNumber);
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                result.add(new
TicketPricePrivilegeCarriage(rs.getBigDecimal(1), rs.getBoolean(2),
                        rs.getTimestamp(3), rs.getString(4), rs.getString(5),
rs.getString(6),
                        rs.getTimestamp(7), rs.getTimestamp(8),
rs.getString(9),
                        rs.getInt(10), rs.getInt(11)));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
    }

    @Override
    public List<RouteFromToDepartMore> findRouteFromToDepartMore(String
arrive, String depart, Timestamp departTime) {
        List<RouteFromToDepartMore> result = new
ArrayList<RouteFromToDepartMore>();
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            PreparedStatement ps =
connection.prepareStatement(SQL_ROUTE_FROM_TO_DEPART_MORE);
            ps.setString(1, depart);
            ps.setString(2, arrive);
            ps.setTimestamp(3, departTime);
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                result.add(new RouteFromToDepartMore(rs.getString(1),
rs.getString(2),
                        rs.getTimestamp(3), rs.getTimestamp(4)));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
    }
```

```
    @Override
    public List<CarriageTypeFromTo> findRouteWithTypeCarriage(String
placeDepart, String placeArrive, String type) {
        List<CarriageTypeFromTo> result = new
ArrayList<CarriageTypeFromTo>();
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            PreparedStatement ps =
connection.prepareStatement(SQL_CARRIAGE_ROUTE_TYPE);
            ps.setString(1, placeDepart);
            ps.setString(2, placeArrive);
            ps.setString(3, type);
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                result.add(new CarriageTypeFromTo(rs.getString(1),
rs.getString(2), rs.getString(3),
                        rs.getString(4), rs.getTimestamp(5),
rs.getTimestamp(6)));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return result;
    }

    @Override
    public void deleteAll() {
        SqlConnection mySqlConnection = SqlConnection.getInstance();
        Connection connection = mySqlConnection.getConnection();
        try {
            Statement query = connection.createStatement();
            query.executeUpdate(SQL_DELETE_ALL);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

View

```
package app.view;

import app.model.dto.CarriageTypeFromTo;
import app.model.dto.RouteFromToDepartMore;
import app.model.dto.TicketPricePrivilegeCarriage;
import app.model.entities.*;

import java.util.List;

public interface View {
    void mainMenu();

    void randomGen();
```

```
    void request();

    void sqlDao();

    void allTable();

    void insertTable(int table);

    void deleteTable(int table);

    void updateTable(int table);

    void requestNumber(int table);

    void printCarriage(List<Carriage> list);

    void printClient(List<Client> list);

    void printPlace(List<Place> list);

    void printRoute(List<Route> list);

    void printRouteToTrainTimeTable(List<RouteToTrainTimeTable> list);

    void printTicket(List<Ticket> list);

    void printTrain(List<Train> list);

    void printCarriageTypeFromTo(List<CarriageTypeFromTo> list);

    void printRouteFromToDepartMore(List<RouteFromToDepartMore> list);

    void printTicketPrivilegeCarriage(List<TicketPricePrivilegeCarriage>
list);

    void wrong();
}


package app.view;

import app.model.dto.CarriageTypeFromTo;
import app.model.dto.RouteFromToDepartMore;
import app.model.dto.TicketPricePrivilegeCarriage;
import app.model.entities.*;

import java.util.List;

public class ConsoleViewer implements View{
    @Override
    public void mainMenu() {
        System.out.println("____Main Menu____");
        System.out.println("Choose table or operation");
        System.out.println("1. Table Carriage");
        System.out.println("2. Table Client");
        System.out.println("3. Table Place");
```

```
        System.out.println("4. Table Route");
        System.out.println("5. Table RouteToTrainTimeTable");
        System.out.println("6. Table Ticket");
        System.out.println("7. Table Train");
        System.out.println("8. All Table");
        System.out.println("9. Request");
        System.out.println("10. Exit");
        System.out.println("Choose variant: ");
    }

    @Override
    public void randomGen() {
        System.out.println("Enter count of rows to generate random data: ");
    }

    @Override
    public void request() {
        System.out.println("1. Tickets for which the price is less than the
specified one,\n" +
                " the carriages on this train are less than the specified
one, whether there are any benefits");
        System.out.println("2. Trains that depart on a given route from a
given time");
        System.out.println("3. Trains that go on a given route with a
specified type of carriage");
        System.out.println("4. To Main Menu");
    }

    @Override
    public void sqlDao() {
        System.out.println("1. Insert");
        System.out.println("2. Delete");
        System.out.println("3. Update");
        System.out.println("4. Find All");
        System.out.println("5. To Main Menu");
    }


    @Override
    public void allTable() {
        System.out.println("1. Random generator");
        System.out.println("2. Delete All");
        System.out.println("3. To Main Menu");
    }

    @Override
    public void insertTable(int table) {
        switch (table){
            case 1:
                System.out.println("Insert(type(String), number(Int),
id(String), idRouteToTrainTimeTable(Int)) ");
                break;
            case 2:
                System.out.println("Insert(email(String), login(String),
fullName(String)): ");
                break;
```

```
            case 3:
                System.out.println("Insert(number_place(integer),
carriage_id(String)): ");
                break;
            case 4:
                System.out.println("Insert (place_of_departure(String),
place_of_arrival(String)): ");
                break;
            case 5:
                System.out.println("Insert (fk_id_route(Int)),
fk_id_train(String), arrive_time(format: yyyy-MM-dd HH:mm),
depart_time(format: yyyy-MM-dd HH:mm)");
                break;
            case 6:
                System.out.println("Insert (fk_client_id(Int),
fk_id_place(Int), price(BigDecimal), buy_date(format: yyyy-MM-dd HH:mm),
privilege(Boolean))");
                break;
            case 7:
                System.out.println("Insert(id(String), max_number(Int))");
                break;
            default:
                break;
        }
    }

    @Override
    public void deleteTable(int table){
        switch (table){
            case 1:
                System.out.println("DeleteById: (String) ");
                break;
            case 2:
                System.out.println("DeleteById id(Int): ");
                break;
            case 3:
                System.out.println("DeleteById id(Int): ");
                break;
            case 4:
                System.out.println("DeleteById id(Int): ");
                break;
            case 5:
                System.out.println("DeleteById id(Int): ");
                break;
            case 6:
                System.out.println("DeleteById id(Int): ");
                break;
            case 7:
                System.out.println("DeleteById id(Int): ");
                break;
            default:
                break;
        }
    }

    @Override
```

```
    public void updateTable(int table){
        switch (table){
            case 1:
                System.out.println("Update Number Carriage by Id: (String),
(Int) ");
                break;
            case 2:
                System.out.println("Update email by login(login(String),
email(String)): ");
                break;
            case 3:
                System.out.println("Update number_place By Id (id(Integer),
number_place(Integer)): ");
                break;
            case 4:
                System.out.println("Update placeOfArrive By Id(id(Integer),
placeOfArrive(String))");
                break;
            case 5:
                System.out.println("Update departTime By Id(id(Int),
depart_time)(format: yyyy-MM-dd HH:mm)");
                break;
            case 6:
                System.out.println("Update price By Id (id(Integer),
price(BigDecimal))");
                break;
            case 7:
                System.out.println("Update max_carriage By id(id(String),
max_carriage(Int)) ");
                break;
            default:
                break;
        }
    }

    @Override
    public void requestNumber(int table){
        switch (table){
            case 1:
                System.out.println("price(BigDecimal), privilege(Boolean),
carriageNumber(Int)");
                break;
            case 2:
                System.out.println("arrive_place(String),
depart_time(String), depart_time(format: yyyy-MM-dd HH:mm)");
                break;
            case 3:
                System.out.println("arrive_place(String),
depart_time(String), type(String)");
                break;
            default:
                break;
        }
    }

    @Override
```

```
    public void printCarriage(List<Carriage> list) {
        System.out.println(list);
    }

    @Override
    public void printClient(List<Client> list) {
        System.out.println(list);
    }

    @Override
    public void printPlace(List<Place> list) {
        System.out.println(list);
    }

    @Override
    public void printRoute(List<Route> list) {
        System.out.println(list);
    }

    @Override
    public void printRouteToTrainTimeTable(List<RouteToTrainTimeTable> list)
{
        System.out.println(list);
    }

    @Override
    public void printTicket(List<Ticket> list) {
        System.out.println(list);
    }

    @Override
    public void printTrain(List<Train> list) {
        System.out.println(list);
    }

    @Override
    public void printCarriageTypeFromTo(List<CarriageTypeFromTo> list) {
        System.out.println(list);
    }

    @Override
    public void printRouteFromToDepartMore(List<RouteFromToDepartMore> list)
{
        System.out.println(list);
    }

    @Override
    public void
printTicketPrivilegeCarriage(List<TicketPricePrivilegeCarriage> list) {
        System.out.println(list);
    }

    @Override
    public void wrong() {
        System.out.println("Input Error");
```

```
        }
    }
```