Національний технічний університет України
“Київський політехнічний інститут ім. Ігоря Сікорського”

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп’ютерних систем

РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА

З ДИСЦИПЛІНИ “БАЗИ ДАНИХ ТА ЗАСОБИ УПРАВЛІННЯ”

ТЕМА: “Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL”

Виконав: Зорєв М.А
Група: КВ-11
Оцінка:

Київ — 2023

Постановка задачі

Загальне завдання роботи полягає у наступному:

1. Реалізувати функції перегляду, 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).

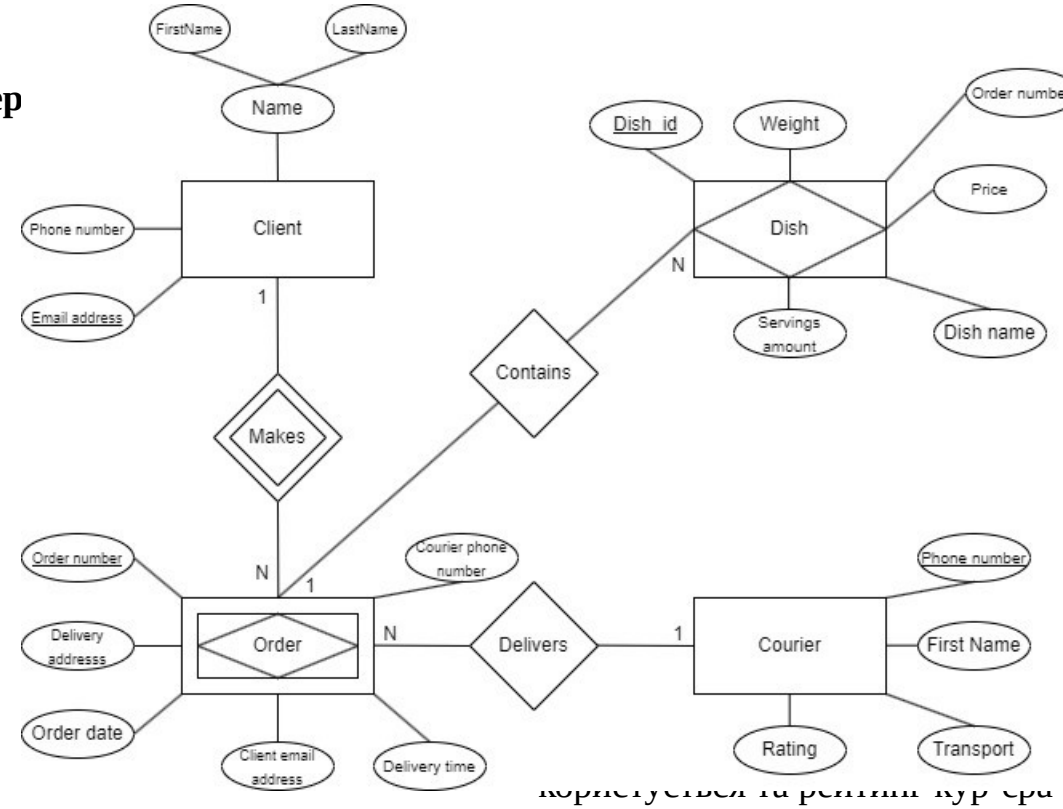
Посилання на репозиторій з вихідним кодом

https://github.com/carcharodon256/DB_LABS.git

Нікнейм у мережі Telegram
@whiteShark816

Діаграма сутність-зв’язок та опис бази даних

Використана нотація: нотація Пітера Чена



Пер **нння**

граву:
номер

ієнт:
брану

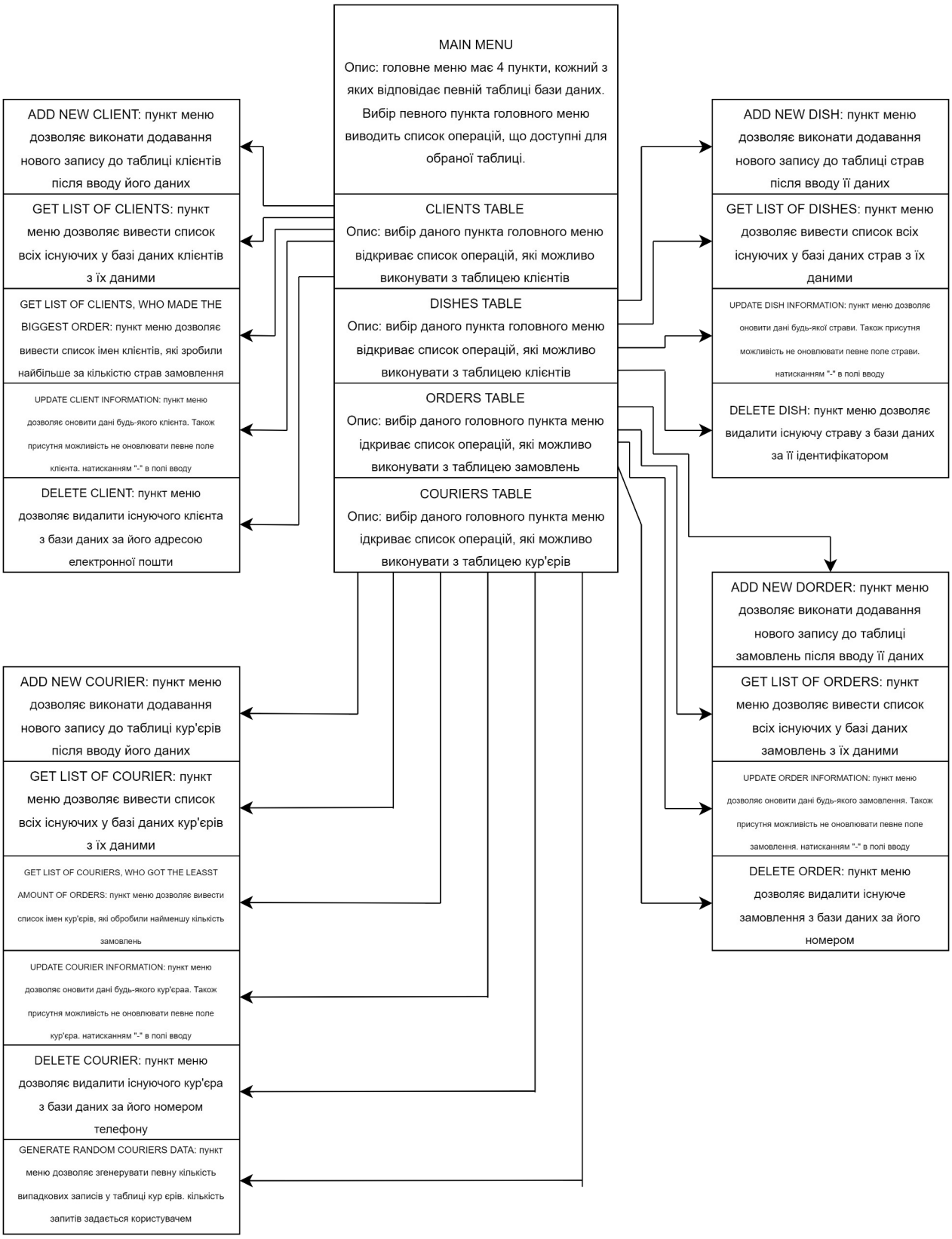
мив
и, дату
а, який
ур’єра,

ляє
ур’єра,
н

Опис зв’язків між сутностями

Опис зв’язку	Вид зв’язку	Пояснення
Order CONTAIN dishes	M : N	Одне замовлення може містити багато страв. Так само одна страва може міститися в багатьох замовленнях
Client MAKES orders	1 : N	Один клієнт може оформити декілька замовлень, але одне замовлення може бути оформлене тільки одним клієнтом
Courier DELIVERS orders	1 : N	Один кур’єр може доставляти багато замовлень, але одне замовлення може бути доставлене лише одним кур’єром

Схема меню користувача



Мова програмування та бібліотеки, що були використані

Мова програмування: Java
Інструментарій: JDBC API для СУБД PostgreSQL

Вставка рядка в дочірню таблицю, коли зовнішній ключ існує в батьківській таблиці

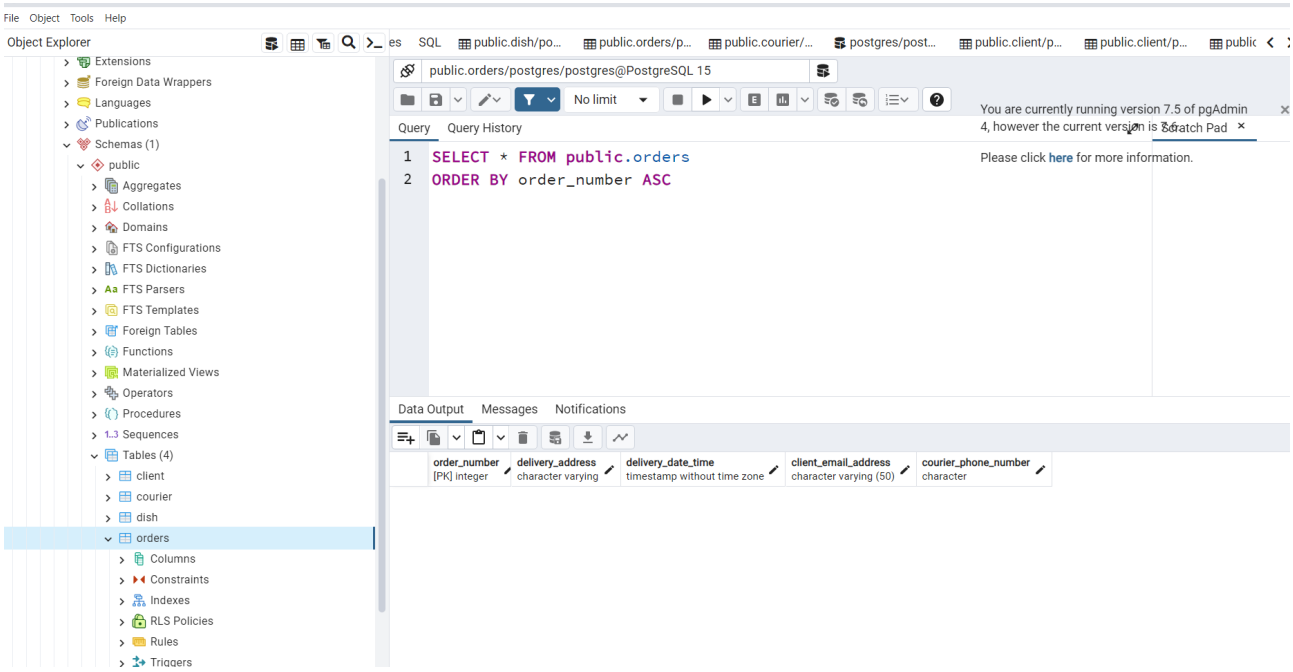
```
Successful database connection.
THIS IS APPLICATION FOR WORK
WITH DELIVERY DATABASE
HAVE A NICE WORK!


MAIN MENU

CHOOSE TABLE, WHAT YOU WANT TO WORK WITH OR OPTION
(PRESS NECESSARY NUMBER ON YOUR KEYBOARD)

1. CLIENTS TABLE
2. DISHES TABLE
3. ORDERS TABLE
4. COURIERS TABLE

3
YOU HAVE SELECTED AN ORDERS TABLE

CHOOSE OPERATION YOU WANT TO DO

(PRESS NECESSARY NUMBER ON YOUR KEYBOARD)
```

Вставка рядка у дочірню таблицю, коли зовнішній ключ відсутній у батьківській таблиці

```
CHOOSE OPERATION YOU WANT TO DO

(PRESS NECESSARY NUMBER ON YOUR KEYBOARD)

1.ADD NEW ORDER
2. GET LIST OF ORDERS
3. UPDATE ORDER INFORMATION
4. DELETE ORDER
1
ADDING NEW ORDER

INPUT ORDER NUMBER (PRESS 'ENTER' AFTER INPUT): 126895
INPUT DELIVERY ADDRESS (PRESS 'ENTER' AFTER INPUT): Balzaka 12, 110
INPUT DELIVERY DATE AND TIME (PRESS 'ENTER' AFTER INPUT): 2023-12-08 10:30:00
INPUT CLIENT EMAIL ADDRESS (PRESS 'ENTER' AFTER INPUT): unexistingEmail@gmail.com
INPUT COURIER PHONE NUMBER (PRESS 'ENTER' AFTER INPUT): 0965866969
New record adding error: ПОМИЛКА: insert або update в таблиці "orders" порушує обмеження зовнішнього ключа "FK_ClientEmailAddress"
  Подробности: Ключ (client_email_address)=(unexistingEmail@gmail.com) не присутній в таблиці "client".

Execution time of this query: 0 milliseconds
Database connection closed.
```

Вилучення рядка з батьківської таблиці та автоматичне вилучення пов’язаних з ним рядків дочірньої таблиці

```
1
YOU HAVE SELECTED A CLIENTS TABLE

CHOOSE OPERATION YOU WANT TO DO

(PRESS NECESSARY NUMBER ON YOUR KEYBOARD)

1. ADD NEW CLIENT
2. GET LIST OF CLIENTS
3. GET LIST OF CLIENTS, WHO MADE THE BIGGEST ORDER
4. UPDATE CLIENT INFORMATION
5. DELETE CLIENT
5
CLIENT DELETION

INPUT EMAIL ADDRESS OF CLIENT, YOU WANT TO DELETE (PRESS 'ENTER' AFTER INPUT): anyEmail@gmail.com
Dish records, orders, and client successfully deleted.

Execution time of this query: 10 milliseconds
Database connection closed.
```



Генерація випадкових даних у таблиці Couriers

SQL-запит для генерації даних:

```
// data generation
String insertQuery = "INSERT INTO courier (courier_phone_number, courier_name, transport_kind, courier_rating) " +
        "SELECT " +
        "  lpad(round(random() * 9999999999)::bigint::text, 10, '0'), " +
        "  md5(random()::text), " +
        "  unnest(ARRAY['Car', 'Motorbike', 'Bike']), " +
        "  floor(random() * 10 + 1)::int " +
        "FROM generate_series(1, ?)";
```

Фрагменти згенерованої таблиці

	courier_phone_number [PK] text	courier_name text	transport_kind character varying (20)	courier_rating numeric
1	0003591695	1073554d4f544431cec0bbfc2ef77880	Bike	2
2	0003751609	44bf1cdfe23adea05635dd6b134814...	Bike	10
3	0005541664	640170175a47cb3f54ac6cb8519c05...	Bike	8
4	0017514101	243041f59ab28e10193be01c1f8f2c1a	Motorbike	8
5	0019779862	5fe865ea2b1fe07d556dfe34671a610a	Bike	5
6	0025545360	339853f65831cd745da175a66ff066d7	Bike	1
7	0025666257	d320b308add22037c2a034dc5f013c...	Bike	9
8	0027812280	222864d7dd166102f52aefb527eb4a...	Car	1
9	0031119539	db6612eb4308f48e0f4c2b67c9a2f221	Motorbike	7
10	0032059845	f33bf9a5a5ca58b0cb530b33d1b896...	Bike	5
11	0052151432	77874489d72a5c64a8d9cfd28e1adc...	Bike	9
12	0052982924	5b84e193fd72a235dd8b53c0b1a373...	Bike	5
13	0057294017	321e8425aad3813639c5bb8c4beba...	Bike	6
14	0059135058	f43903ee652f034c0d5df9393b38e713	Motorbike	8
15	0062086744	0473095e4d35c4d055087ffcf1857fc9	Bike	9
16	0067121496	05d7f980053cde704544d00fcb8b54...	Car	9
17	0075967920	c22ce870221f1c6c95276259417efdaa	Motorbike	6
18	0082845634	e01e0564c3cefa798b7201840dc212...	Bike	9
19	0083562115	e8e5f732385680b58d17f344b932eb...	Car	3
20	0092780872	0188468bec10a2012948990f0e1ecc...	Car	2
21	0101678683	cab1a71de702a5887d3dcd6f7d2bce...	Motorbike	8
22	0102525573	7ef5dd96d71ccf629f02a151364193b2	Bike	4

	courier_phone_number [PK] text	courier_name text	transport_kind character varying (20)	courier_rating numeric
882	5929684379	caaef766612748970f16e5b9f34a6545	Motorbike	2
883	5930196232	1cabc62d55867a39622d094d3b7feff8	Car	3
884	5940114447	c6c6dd57bfd70f133e356f4d6a3ee73f	Car	1
885	5944525561	83dfaad34ae5f942e9d348f15e1150fe	Car	1
886	5948735819	0648f6f91d63675a9d062b3557728f...	Car	3
887	5965897271	d76ccb2411adec84022b952e72ba26...	Car	10
888	5970104885	9f05cd88409e6ac320dd31db414299...	Motorbike	1
889	5979195968	03eab38d9ff287133e656029468a06...	Car	7
890	5979336905	4d7018d9a15aea5db4d05f11702aa4...	Motorbike	6
891	5982603385	e26f33f2cdb190f3fdcf8d6a4646c9aa	Car	1
892	5992109308	fb67ebc1995b589f8153fb7ddf6acf57	Car	6
893	5999759574	461f9cc24969998aa5265bcafb73f474	Bike	4
894	6000393512	b1a82729197970b4094306afaf69d6...	Car	5
895	6001478404	1419340ad36f865b3b73ea54f08b0a...	Bike	2
896	6002968778	6c0344015138e30aa14f9f43f05d964c	Motorbike	5
897	6009350802	33bdd4bde4ce43435704a0690e23b...	Car	10
898	6010772341	64a7beae0ee4077d32bafbdbe92a5a...	Car	10
899	6032490223	733603f88c72aebf1fb63f542a653b80	Car	4
900	6039146840	6d921aa6f01e948dc6ed379df8dc2a...	Bike	1
901	6040153139	1ecdec772ca528ec52ae6f815d974e...	Car	4
902	6049313043	42cd611e8840afc7cbdce9aaab98e0...	Car	1
903	6056702109	427feea9723e29ac97a9d5be2a7b5c...	Car	6

	courier_phone_number [PK] text	courier_name text	transport_kind character varying (20)	courier_rating numeric
1832	4035841282	ad977baf99e38488fcb4b8b54327d3ef	Motorbike	10
1833	4037095034	8141fe86009a7844f24290aee4917e13	Car	6
1834	4039616553	8a188778d7af47b71edc981d9b3d7c...	Bike	5
1835	4039999706	e8c1e8ac5f3944e5aeb9b0d0fa29c8a6	Bike	7
1836	4040085267	7c0500f93eafb98d5aae137ce815eb66	Motorbike	6
1837	4040520492	5739ada0197a9fca52781e7692da30...	Bike	7
1838	4046265678	d7364a759dbd058f64450d3ac07ad2...	Car	1
1839	4046892709	a8cee9a8e4c6711df97c92bf38863a22	Car	6
1840	4049101000	4b005f99d8c01d5bd2d38909a37a65...	Bike	8
1841	4051834797	74a80568a072d4ea981c89637aa253...	Motorbike	1
1842	4052552490	7bde5de69baa75b5af6c72367b066f...	Bike	1
1843	4052764267	3e3fcb32e4c3b618aef82e328a28f026	Bike	1
1844	4055559220	c9a74027c02a92edada6659eed54ad...	Car	6
1845	4056446705	252c7409ea5da709ec73f2b61124dd...	Bike	4
1846	4064033415	ad6c0bce19115db1160db45953cbdc...	Motorbike	6
1847	4065724314	d175b4860fd71757a8ffe4ea60282f29	Bike	3
1848	4071364343	d9818f672e6a5f07bbae0c9165f3f777	Car	9
1849	4073698230	e0b5e4d686a445bc1037d996cebc5a...	Motorbike	4
1850	4075982647	087fa492c4543cea785a525a9125a1...	Motorbike	1
1851	4079549212	bc11372ac9224de0d4c1fc798c4255...	Motorbike	7
1852	4081137182	1b830962b60997a4a6c7184b6d760f...	Bike	1
1853	4084764368	36baa3fdd1135644feec2be14e5bdf8d	Bike	9

Копії SQL-запитів із зазначеними початковими параметрами

```
select client.client_name from client join\n" +
        "(select client_email_address, count(*) as order_count\n"
+
        "from orders group by client_email_address order by \n" +
        "order_count desc limit 1) as s1 on \n" +
        "client.client_email_address = s1.client_email_address;


SELECT courier_name " +
        "FROM courier " +
        "LEFT JOIN orders ON courier.courier_phone_number =
orders.courier_phone_number " +
        "GROUP BY courier_name " +
        "ORDER BY COUNT(orders.order_number) ASC " +
        "LIMIT 1;
```

Результати виконанн запитів

```
1
YOU HAVE SELECTED A CLIENTS TABLE

CHOOSE OPERATION YOU WANT TO DO

(PRESS NECESSARY NUMBER ON YOUR KEYBOARD)

1. ADD NEW CLIENT
2. GET LIST OF CLIENTS
3. GET LIST OF CLIENTS, WHO MADE THE BIGGEST
4. UPDATE CLIENT INFORMATION
5. DELETE CLIENT
3

LIST OF CLIENTS WHO MADE THE BIGGEST ORDERS

Kirilo Bortnik

Execution time of this query: 16 milliseconds
Database connection closed.

Process finished with exit code 0
```

```
4
YOU HAVE SELECTED A COURIERS TABLE

CHOOSE OPERATION YOU WANT TO DO

(PRESS NECESSARY NUMBER ON YOUR KEYBOARD)
1.ADD NEW COURIER
2. GET LIST OF COURIERS
3. GET LIST OF COURIERS, WHO GOT THE LEAST AMOUNT OF ORDERS
4. UPDATE COURIER INFORMATION
5. DELETE COURIER
6. GENERATE RANDOM COURIERS DATA
3

LIST OF COURIERS WHO TOOK THE LEAST AMOUNT OF ORDERS

Roman


Execution time of this query: 23 milliseconds
Database connection closed.
```

Код модуля Model згідно шаблону MVC

```
// DatabaseConnection.java

package model;

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

public class DatabaseConnection {
    private Connection connection;

    public Connection connectToDatabase() {
        if (connection != null)
            return connection;

        try {
            Class.forName("org.postgresql.Driver");

            String url = "jdbc:postgresql://localhost:5432/postgres";
            String user = "postgres";
            String password = "simple";

            connection = DriverManager.getConnection(url, user, password);
            System.out.println("Successful database connection.");

            return connection;

        } catch (ClassNotFoundException | SQLException e){
            System.out.println("PostgreSQL connection error: " +
e.getMessage());
```

```
            throw new RuntimeException("Database connection error. ", e);
        }
    }

    public void disconnectFromDatabase() {
        try {
            if(connection != null){
                connection.close();
                System.out.println("Database connection closed.");
            }
        } catch (SQLException e){
            System.out.println("Closing connection error: " + e.getMessage());
        }
    }
}

// ClientCRUDOperations.java

package model;

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

import entities.Client;
import query_execution_time.*;

import entities.Client;

public class ClientCRUDOperations {
    public static String createRecord(Connection connection, String
clientEmailAddress, String clientName, String clientPhoneNumber){
        // string with query text
        String creationQuery = "INSERT INTO client (client_email_address,
client_name, client_phone_number) VALUES (?, ?, ?)";

        // query result text
        String operationResult;

        // preparing of query object
        try (PreparedStatement queryObject =
connection.prepareStatement(creationQuery)) {
            // putting field values
            queryObject.setString(1, clientEmailAddress);
            queryObject.setString(2, clientName);
            queryObject.setString(3, clientPhoneNumber);

            long queryExecutionStartTime = System.currentTimeMillis();

            // query execution
            int rowsChanged = queryObject.executeUpdate();

            long queryExecutionEndTime = System.currentTimeMillis();

            QueryExecutionTimeMeasuring.queryExecutionTime =
QueryExecutionTimeMeasuring.measureQueryExecutionTime(queryExecutionStartTime,
queryExecutionEndTime);

            // checking query execution correctness
            if (rowsChanged > 0) {
                operationResult = "New record successfully created in 'client'
table.";
            } else {
                operationResult = "New record adding error.";
```

```
            }
        } catch (SQLException e) {
            operationResult = "New record adding error: " + e.getMessage();
        }

        return operationResult;
    }

    public static ArrayList<Client> getAllRecords(Connection connection) throws
SQLException{
        ArrayList<Client> clients = new ArrayList<>(); // list for saving
clients records

        String selectionQuery = "SELECT * FROM client";

        try (PreparedStatement queryObject =
connection.prepareStatement(selectionQuery)) {
            long queryExecutionStartTime = System.currentTimeMillis();

            // object with a set of results
            ResultSet resultSet = queryObject.executeQuery();

            long queryExecutionEndTime = System.currentTimeMillis();

            QueryExecutionTimeMeasuring.queryExecutionTime =
QueryExecutionTimeMeasuring.measureQueryExecutionTime(queryExecutionStartTime,
queryExecutionEndTime);

            while (resultSet.next()) { // getting record fields values
                String clientEmailAddress =
resultSet.getString("client_email_address");
                String clientName = resultSet.getString("client_name");
                String client_phone_number =
resultSet.getString("client_phone_number");

                // client object creation
                Client client = new Client(clientEmailAddress, clientName,
client_phone_number);
                clients.add(client); // adding client objects into list
            }
        } catch (SQLException e){
            throw  e;
        }

        return clients;
    }

    public static ArrayList<String> getClientsWithMostOrders(Connection
connection) throws SQLException{
        ArrayList<String> clients = new ArrayList<>();

        String selectionQuery = "select client.client_name from client join\n" +
                "(select client_email_address, count(*) as order_count\n" +
                "from orders group by client_email_address order by \n" +
                "order_count desc limit 1) as s1 on \n" +
                "client.client_email_address = s1.client_email_address;";

        try (PreparedStatement queryObject =
connection.prepareStatement(selectionQuery)){

            long queryExecutionStartTime = System.currentTimeMillis();

            try (ResultSet resultSet = queryObject.executeQuery()){
```

```
                long queryExecutionEndTime = System.currentTimeMillis();

                QueryExecutionTimeMeasuring.queryExecutionTime =
QueryExecutionTimeMeasuring.measureQueryExecutionTime(queryExecutionStartTime,
queryExecutionEndTime);

                while (resultSet.next()){
                    String clientName = resultSet.getString("client_name");
                    clients.add(clientName);
                }
            }
        } catch (SQLException e){
            throw  e;
        }

        return clients;
    }

    public static String updateRecord(Connection connection, String
clientEmailAddress, String newClientName, String newClientPhoneNumber){
        String updateQuery = "UPDATE client SET client_name = COALESCE(?,
client_name), client_phone_number = COALESCE(?, client_phone_number) WHERE
client_email_address = ?";
        String operationResult;

        long queryExecutionStartTime = System.currentTimeMillis();

        try (PreparedStatement queryObject =
connection.prepareStatement(updateQuery)) {
            long queryExecutionEndTime = System.currentTimeMillis();

            QueryExecutionTimeMeasuring.queryExecutionTime =
QueryExecutionTimeMeasuring.measureQueryExecutionTime(queryExecutionStartTime,
queryExecutionEndTime);

            queryObject.setString(1, newClientName);
            queryObject.setString(2, newClientPhoneNumber);
            queryObject.setString(3, clientEmailAddress);

            int rowsChanged = queryObject.executeUpdate();

            if (rowsChanged > 0) {
                operationResult = "Record successfully updated in 'client'
table.";
            } else {
                operationResult = "Client with email address " +
clientEmailAddress + " not found.";
            }
        } catch (SQLException e) {
            operationResult = "Record update error: " + e.getMessage();
        }

        return operationResult;
    }

    public static String deleteRecord(Connection connection, String
clientEmailAddress){
        String selectOrderNumbersQuery = "SELECT order_number FROM orders WHERE
client_email_address = ?";
        String deleteDishRecordsQuery = "DELETE FROM dish WHERE order_number
= ?";
        String deleteClientQuery = "DELETE FROM client WHERE
client_email_address = ?";
```

```
        String deleteOrdersQuery = "DELETE FROM orders WHERE
client_email_address = ?";

        String operationResult;

        int ordersRowsChanged = 0;
        int dishRowsChanged = 0;

        long queryExecutionStartTime = System.currentTimeMillis();

        try (PreparedStatement selectOrderNumbers =
connection.prepareStatement(selectOrderNumbersQuery);
             PreparedStatement deleteDishRecords =
connection.prepareStatement(deleteDishRecordsQuery);
             PreparedStatement deleteClientRecord =
connection.prepareStatement(deleteClientQuery);
             PreparedStatement deleteOrdersRecord =
connection.prepareStatement(deleteOrdersQuery)) {

            long queryExecutionEndTime = System.currentTimeMillis();

            QueryExecutionTimeMeasuring.queryExecutionTime =
QueryExecutionTimeMeasuring.measureQueryExecutionTime(queryExecutionStartTime,
queryExecutionEndTime);

            connection.setAutoCommit(false);

            selectOrderNumbers.setString(1, clientEmailAddress);
            ResultSet orderNumbersResultSet = selectOrderNumbers.executeQuery();

            while (orderNumbersResultSet.next()) {
                int orderNumber = orderNumbersResultSet.getInt("order_number");

                deleteDishRecords.setInt(1, orderNumber);
                dishRowsChanged = deleteDishRecords.executeUpdate();
            }

            deleteOrdersRecord.setString(1, clientEmailAddress);
            ordersRowsChanged = deleteOrdersRecord.executeUpdate();

            deleteClientRecord.setString(1, clientEmailAddress);
            int clientRowsChanged = deleteClientRecord.executeUpdate();

            if (dishRowsChanged > 0 || ordersRowsChanged > 0 ||
clientRowsChanged > 0) {
                connection.commit();
                operationResult = "Dish records, orders, and client successfully
deleted.";
            } else {
                operationResult = "Client with email address " +
clientEmailAddress + " not found.";
            }
        } catch (SQLException e) {
            try {
                if (connection != null)
                    connection.rollback();
            } catch (SQLException ex) {
                operationResult = "Database connection error: " +
e.getMessage();
            }
            operationResult = "Client deletion error: " + e.getMessage();
        }

        return operationResult;
```

```
    }
}


// DishcRUdOperations.java

package model;

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

import entities.Dish;
import query_execution_time.*;

public class DishCRUDOperations {
    public static String createRecord(Connection connection, int dishId, String
dishName, int weight, int price, int servingsAmount, int orderNumber){
        String creationQuery = "INSERT INTO dish (dish_id, dish_name, weight,
price, servings_amount, order_number) VALUES (?, ?, ?, ?, ?, ?)";

        String operationResult;

        try (PreparedStatement quwryObject =
connection.prepareStatement(creationQuery)){
            quwryObject.setInt(1, dishId);
            quwryObject.setString(2, dishName);
            quwryObject.setInt(3, weight);
            quwryObject.setInt(4, price);
            quwryObject.setInt(5, servingsAmount);
            quwryObject.setInt(6, orderNumber);

            long queryExecutionStartTime = System.currentTimeMillis();

            int rowsChanged = quwryObject.executeUpdate();

            long queryExecutionEndTime = System.currentTimeMillis();

            QueryExecutionTimeMeasuring.queryExecutionTime =
QueryExecutionTimeMeasuring.measureQueryExecutionTime(queryExecutionStartTime,
queryExecutionEndTime);

            if (rowsChanged > 0) {
                operationResult = "New record successfully created in 'dish'
table.";
            } else {
                operationResult = "New record adding error.";
            }
        } catch (SQLException e) {
            operationResult = "New record adding error: " + e.getMessage();
        }

        return operationResult;
    }

    public static ArrayList<Dish> getAllRecords(Connection connection) throws
SQLException{
        ArrayList<Dish> dishes = new ArrayList<>();

        String selectionQuery = "SELECT * FROM dish";

        try (PreparedStatement queryObject =
connection.prepareStatement(selectionQuery)) {

            long queryExecutionStartTime = System.currentTimeMillis();
```

```
            ResultSet resultSet = queryObject.executeQuery();

            long queryExecutionEndTime = System.currentTimeMillis();

            QueryExecutionTimeMeasuring.queryExecutionTime =
QueryExecutionTimeMeasuring.measureQueryExecutionTime(queryExecutionStartTime,
queryExecutionEndTime);

            while (resultSet.next()){
                int dishId = resultSet.getInt("dish_id");
                String dishName = resultSet.getString("dish_name");
                int weight = resultSet.getInt("weight");
                int price = resultSet.getInt("price");
                int servingsAmount = resultSet.getInt("servings_amount");
                int orderNumber = resultSet.getInt("order_number");

                Dish dish = new Dish(dishId, dishName, weight, price,
servingsAmount, orderNumber);
                dishes.add(dish);
            }
        } catch (SQLException e) {
            throw e;
        }

        return dishes;
    }

    public static String updateRecord(Connection connection, int dishId, String
newDishName, int newWeight, int newPrice, int newServingsAmount){
        String updateQuery = "UPDATE dish SET dish_name = COALESCE(?,
dish_name), weight = COALESCE(?, weight), price = COALESCE(?, price),
servings_amount = COALESCE(?, servings_amount) WHERE dish_id = ?";

        String operationResult;

        try (PreparedStatement queryObject =
connection.prepareStatement(updateQuery)) {
            queryObject.setString(1, newDishName);
            queryObject.setInt(2, newWeight);
            queryObject.setInt(3, newPrice);
            queryObject.setInt(4, newServingsAmount);
            queryObject.setInt(5, dishId);

            long queryExecutionStartTime = System.currentTimeMillis();

            int rowsChanged = queryObject.executeUpdate();

            long queryExecutionEndTime = System.currentTimeMillis();

            QueryExecutionTimeMeasuring.queryExecutionTime =
QueryExecutionTimeMeasuring.measureQueryExecutionTime(queryExecutionStartTime,
queryExecutionEndTime);

            if (rowsChanged > 0) {
                operationResult = "Record successfully updated in 'dish'
table.";
            } else {
                operationResult = "Dish with dish ID " + dishId + " not found.";
            }
        } catch (SQLException e) {
            operationResult = "Record update error: " + e.getMessage();
        }
```

```
        return operationResult;
    }

    public static String deleteRecord(Connection connection, int dishId){
        String deletionQuery = "DELETE FROM dish WHERE dish_id = ?";

        String operationResult;

        try(PreparedStatement deleteDishRecord =
connection.prepareStatement(deletionQuery)) {
            deleteDishRecord.setInt(1, dishId);

            long queryExecutionStartTime = System.currentTimeMillis();

            int dishChangedRows = deleteDishRecord.executeUpdate();

            long queryExecutionEndTime = System.currentTimeMillis();

            QueryExecutionTimeMeasuring.queryExecutionTime =
QueryExecutionTimeMeasuring.measureQueryExecutionTime(queryExecutionStartTime,
queryExecutionEndTime);

            if(dishChangedRows > 0)
                operationResult = "Dish successfully deleted.";
            else
                operationResult = "Dish with dish_id " + dishId + " not found.";

        } catch (SQLException e) {
            operationResult = "Dish deletion error: " + e.getMessage();
        }

        return operationResult;
    }
}


// OrderCRUDOperations.java

package model;

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

import entities.Order;
import query_execution_time.*;

public class OrderCRUDOperations {
    public static String createRecord(Connection connection, int orderNumber,
String deliveryAddress, Timestamp deliveryDateTime, String clientEmailAddress,
String courierPhoneNumber){
        String creationQuery = "INSERT INTO orders (order_number,
delivery_address, delivery_date_time, client_email_address,
courier_phone_number) VALUES (?, ?, ?, ?, ?)";

        String operationResult;

        try (PreparedStatement queryObject =
connection.prepareStatement(creationQuery)) {
            queryObject.setInt(1, orderNumber);
            queryObject.setString(2, deliveryAddress);
            queryObject.setTimestamp(3, deliveryDateTime);
            queryObject.setString(4, clientEmailAddress);
            queryObject.setString(5, courierPhoneNumber);
```

```
            long queryExecutionStartTime = System.currentTimeMillis();

            int rowsChanged = queryObject.executeUpdate();

            long queryExecutionEndTime = System.currentTimeMillis();

            QueryExecutionTimeMeasuring.queryExecutionTime =
QueryExecutionTimeMeasuring.measureQueryExecutionTime(queryExecutionStartTime,
queryExecutionEndTime);

            if (rowsChanged > 0) {
                operationResult = "New record successfully created in 'orders'
table.";
            } else {
                operationResult = "New record adding error.";
            }
        } catch (SQLException e) {
            operationResult = "New record adding error: " + e.getMessage();
        }

        return operationResult;
    }

    public static ArrayList<Order> getAllRecords(Connection connection) throws
SQLException{
        ArrayList<Order> orders = new ArrayList<>();

        String selectionQuery = "SELECT * FROM orders";

        try (PreparedStatement queryObject =
connection.prepareStatement(selectionQuery)) {

            long queryExecutionStartTime = System.currentTimeMillis();

            ResultSet resultSet = queryObject.executeQuery();

            long queryExecutionEndTime = System.currentTimeMillis();

            QueryExecutionTimeMeasuring.queryExecutionTime =
QueryExecutionTimeMeasuring.measureQueryExecutionTime(queryExecutionStartTime,
queryExecutionEndTime);

            while (resultSet.next()) {
                int orderNumber = resultSet.getInt("order_number");
                String deliveryAddress =
resultSet.getString("delivery_address");
                Timestamp deliveryDateTime =
resultSet.getTimestamp("delivery_date_time");
                String clientEmailAddress =
resultSet.getString("client_email_address");
                String courierPhoneNumber =
resultSet.getString("courier_phone_number");

                Order order = new Order(orderNumber, deliveryAddress,
deliveryDateTime, clientEmailAddress, courierPhoneNumber);
                orders.add(order);
            }
        } catch (SQLException e) {
            throw e;
        }

        return orders;
    }
```

```
    public static String updateRecord(Connection connection, int orderNumber,
String newDeliveryAddress, Timestamp newDeliveryDateTime){
        String updateQuery = "UPDATE orders SET delivery_address = COALESCE(?,
delivery_address), delivery_date_time = COALESCE(?, delivery_date_time) WHERE
order_number = ?";

        String operationResult;

        try (PreparedStatement queryObject =
connection.prepareStatement(updateQuery)) {
            queryObject.setString(1, newDeliveryAddress);
            queryObject.setTimestamp(2, newDeliveryDateTime);
            queryObject.setInt(3, orderNumber);

            long queryExecutionStartTime = System.currentTimeMillis();

            int rowsChanged = queryObject.executeUpdate();

            long queryExecutionEndTime = System.currentTimeMillis();

            QueryExecutionTimeMeasuring.queryExecutionTime =
QueryExecutionTimeMeasuring.measureQueryExecutionTime(queryExecutionStartTime,
queryExecutionEndTime);

            if (rowsChanged > 0) {
                operationResult = "Record successfully updated in 'orders'
table.";
            } else {
                operationResult = "Order with order number " + orderNumber + "
not found.";
            }
        } catch (SQLException e) {
            operationResult = "Order update error: " + e.getMessage();
        }

        return operationResult;
    }

    public static String deleteRecord(Connection connection, int orderNumber){
        String deletionQuery = "DELETE FROM orders WHERE order_number = ?";
        String cascadeDeletionQuery = "DELETE FROM dish WHERE order_number = ?";

        String operationResult;

        try (PreparedStatement deleteOrderRecord =
connection.prepareStatement(deletionQuery);
             PreparedStatement deleteDishesRecords =
connection.prepareCall(cascadeDeletionQuery)) {
            connection.setAutoCommit(false);

            deleteOrderRecord.setInt(1, orderNumber);

            long queryExecutionStartTime = System.currentTimeMillis();

            int ordersRowsChanged = deleteOrderRecord.executeUpdate();

            long queryExecutionEndTime = System.currentTimeMillis();
            QueryExecutionTimeMeasuring.queryExecutionTime =
QueryExecutionTimeMeasuring.measureQueryExecutionTime(queryExecutionStartTime,
queryExecutionEndTime);

            deleteDishesRecords.setInt(1, orderNumber);
            int dishRowsChanged = deleteDishesRecords.executeUpdate();
```

```
            if (ordersRowsChanged > 0 || dishRowsChanged > 0) {
                operationResult = "Order and its content successfully deleted.";
                connection.commit();
            } else
                operationResult = "Order with number " + orderNumber + " not
found.";
        } catch (SQLException e) {
            try {
                if (connection != null)
                    connection.rollback();
            } catch (SQLException ex) {
                operationResult = "Database connection error: " +
ex.getMessage();
            }
            operationResult = "Order deletion error: " + e.getMessage();
        }

        return operationResult;
    }
}


// CourierCRUdOperations.java

package model;

import java.sql.*;
import java.time.LocalDateTime;
import java.util.ArrayList;

import entities.Courier;
import query_execution_time.*;

public class CourierCRUDOperations {
    public static String createRecord(Connection connection, String
courierPhoneNumber, String courierName, String transportKind, double
courierRating){
        String creationQuery = "INSERT INTO courier (courier_phone_number,
courier_name, transport_kind, courier_rating) VALUES (?, ?, ?, ?)";

        String operationResult;

        try (PreparedStatement queryObject =
connection.prepareStatement(creationQuery)) {
            queryObject.setString(1, courierPhoneNumber);
            queryObject.setString(2, courierName);
            queryObject.setString(3, transportKind);
            queryObject.setDouble(4, courierRating);

            long queryExecutionStartTime = System.currentTimeMillis();

            int rowsChanged = queryObject.executeUpdate();

            long queryExecutionEndTime = System.currentTimeMillis();

            QueryExecutionTimeMeasuring.queryExecutionTime =
QueryExecutionTimeMeasuring.measureQueryExecutionTime(queryExecutionStartTime,
queryExecutionEndTime);

            if (rowsChanged > 0)
                operationResult = "New record successfully created in 'courier'
table.";
            else
                operationResult = "New record adding error.";
```

```
        }
        catch(SQLException e){
            operationResult = "New record adding error: " + e.getMessage();
        }
        return operationResult;
    }

    public static void generateCourierData(Connection connection, int
numberOfRecords) {
        try {
            // data generation
            String insertQuery = "INSERT INTO courier (courier_phone_number,
courier_name, transport_kind, courier_rating) " +
                    "SELECT " +
                    "  lpad(round(random() * 9999999999)::bigint::text, 10,
'0'), " +
                    "  md5(random()::text), " +
                    "  unnest(ARRAY['Car', 'Motorbike', 'Bike']), " +
                    "  floor(random() * 10 + 1)::int " +
                    "FROM generate_series(1, ?)";

            try (PreparedStatement statement =
connection.prepareStatement(insertQuery)) {
                statement.setInt(1, numberOfRecords);

                // query execution
                statement.executeUpdate();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public static ArrayList<Courier> getAllRecords(Connection connection) throws
SQLException{
        ArrayList<Courier> couriers = new ArrayList<>();

        String selectionQuery = "SELECT * FROM courier";

        try (PreparedStatement queryObject =
connection.prepareStatement(selectionQuery)) {

            long queryExecutionStartTime = System.currentTimeMillis();

            ResultSet resultSet = queryObject.executeQuery();

            long queryExecutionEndTime = System.currentTimeMillis();
            QueryExecutionTimeMeasuring.queryExecutionTime =
QueryExecutionTimeMeasuring.measureQueryExecutionTime(queryExecutionStartTime,
queryExecutionEndTime);

            while (resultSet.next()) {
                String courierPhoneNumber =
resultSet.getString("courier_phone_number");
                String courierName = resultSet.getString("courier_name");
                String transportKind = resultSet.getString("transport_kind");
                double courierRating = resultSet.getDouble("courier_rating");

                Courier courier = new Courier(courierPhoneNumber, courierName,
transportKind, courierRating);
                couriers.add(courier);
            }
        } catch (SQLException e) {
            throw e;
```

```
        }

        return couriers;
    }

    public static ArrayList<String> getCouriersWithLessOrders(Connection
connection) throws SQLException{
        ArrayList<String> couriers = new ArrayList<>();

        try {
            String selectionQuery = "SELECT courier_name " +
                    "FROM courier " +
                    "LEFT JOIN orders ON courier.courier_phone_number =
orders.courier_phone_number " +
                    "GROUP BY courier_name " +
                    "ORDER BY COUNT(orders.order_number) ASC " +
                    "LIMIT 1";

            long queryExecutionStartTime = System.currentTimeMillis();
            try (PreparedStatement queryObject =
connection.prepareStatement(selectionQuery);
                 ResultSet resultSet = queryObject.executeQuery()) {

                long queryExecutionEndTime = System.currentTimeMillis();
                QueryExecutionTimeMeasuring.queryExecutionTime =
QueryExecutionTimeMeasuring.measureQueryExecutionTime(queryExecutionStartTime,
queryExecutionEndTime);

                while (resultSet.next()) {
                    String courierName = resultSet.getString("courier_name");
                    couriers.add(courierName);
                }
            }
        } catch (SQLException e) {
            throw e;
        }

        return couriers;
    }

    public static String updateRecord(Connection connection, String
courierPhoneNumber, String newTransportKind, double newCourierRating){
        String updateQuery = "UPDATE courier SET transport_kind = COALESCE(?,
transport_kind), courier_rating = ? WHERE courier_phone_number = ?";
        String operationResult;

        try (PreparedStatement queryObject =
connection.prepareStatement(updateQuery)) {
            queryObject.setString(1, newTransportKind);
            queryObject.setDouble(2, newCourierRating);
            queryObject.setString(3, courierPhoneNumber);

            long queryExecutionStartTime = System.currentTimeMillis();

            int rowsChanged = queryObject.executeUpdate();

            long queryExecutionEndTime = System.currentTimeMillis();
            QueryExecutionTimeMeasuring.queryExecutionTime =
QueryExecutionTimeMeasuring.measureQueryExecutionTime(queryExecutionStartTime,
queryExecutionEndTime);

            if (rowsChanged > 0) {
                operationResult = "Record successfully updated in 'courier'
table.";
```

```
            } else {
                operationResult = "Courier with phone number " +
courierPhoneNumber + " not found.";
            }
        } catch (SQLException e) {
            operationResult = "Courier update error: " + e.getMessage();
        }

        return operationResult;
    }

    public static String deleteRecord(Connection connection, String
courierPhoneNumber){
        String updateOrdersQuery = "UPDATE orders " +
                "SET courier_phone_number = (SELECT courier_phone_number " +
                "FROM courier " +
                "ORDER BY (SELECT COUNT(*) " +
                "FROM orders AS o " +
                "WHERE o.courier_phone_number = courier.courier_phone_number) "
+
                "LIMIT 1) " +
                "WHERE courier_phone_number = ?";

        String deleteCourierQuery = "DELETE FROM courier WHERE
courier_phone_number = ?";

        String operationResult;

        try (PreparedStatement updateOrdersRecords =
connection.prepareStatement(updateOrdersQuery);
             PreparedStatement deleteCourierRecord =
connection.prepareStatement(deleteCourierQuery)) {
            connection.setAutoCommit(false);

            updateOrdersRecords.setString(1, courierPhoneNumber);
            int ordersRowsChanged = updateOrdersRecords.executeUpdate();

            deleteCourierRecord.setString(1, courierPhoneNumber);

            long queryExecutionStartTime = System.currentTimeMillis();

            int courierRowsChanged = deleteCourierRecord.executeUpdate();

            long queryExecutionEndTime = System.currentTimeMillis();
            QueryExecutionTimeMeasuring.queryExecutionTime =
QueryExecutionTimeMeasuring.measureQueryExecutionTime(queryExecutionStartTime,
queryExecutionEndTime);

            if (ordersRowsChanged > 0) {
                operationResult = "Orders successfully updated.";
            } else {
                operationResult = "Orders updating error.";
            }

            if (courierRowsChanged > 0) {
                operationResult = "Courier successfully deleted.";
                connection.commit();
            } else {
                operationResult = "Courier with phone number " +
courierPhoneNumber + " not found.";
            }
        } catch (SQLException e) {
            try {
                if (connection != null)
```

```
                    connection.rollback();
                } catch (SQLException ex) {
                    operationResult = "Database connection error: " +
ex.getMessage();
                }
                operationResult = "Courier deletion error: " + e.getMessage();
            }

            return operationResult;
        }
}
```

Короткий опис методів класів модуля Model

I. Клас DatabaseConnection

1. Метод `public Connection connectToDatabase()` - у методі реалізовано стандартне підключення до бази даних СУБД PostgreSQL. Метод повертає об єкт класу java.sql.Connection.
2. Метод `public void disconnectFromDatabase()` - у методі реалізовано відключення від бази даних. Якщо посилання на об’єкт класу java.sql.Connection має не null значення, викликається метод для закриття з’єднання з базою даних.

II. Клас ClientCRUDOperations

1. Метод `public static String createRecord(Connection connection, String clientEmailAddress, String clientName, String clientPhoneNumber)` – метод приймає в якості параметрів існуюче з’єднання з базою даних та значення полів нового запису таблиці client. Створюється рядок з текстои відповідного sql-запиту та рядкова змінна для зберігання результату виконання запиту. Далі відбувається підготовка об’єкта відповідного запиту, шляхом встановлення об’єкту знаень полів новоствореного запису з використанням плейсхолдерів. Після цього викликається метод для виконання запиту, паралельно вимірюється час його виконання, що зберігається у глобальну змінну. Після цього перевіряється успішніс виконання запиту та повертається повідомлення про неї.

2. Метод `public static ArrayList<Client> getAllRecords(Connection connection) throws SQLException` — метод приймає в якості параметра об єкт існуючого з’єднання з базою даних. Створюються змінні для зберігання тексту запиту, успішності виконання запиту, набору результатів запиту. Після цього отримуються значення полів кожного запису таблиці, з яких створюєтьс об’єкт з полями, ідентичними атрибутам запису. Даний об’єкт додається в колекцію ArrayList. Це відбувається для всіх рядків таблиці. Після цього метод повертає колекцію, яка є списком існуючих записів таблці client.

3. Метод `public static ArrayList<String> getClientsWithMostOrders(Connection connection) throws SQLException -`
аналогічний методу `getAllRecords`, з тією різницею, що даний метод повертає список клієнтів, що відповідає певним параметрам пошуку у базі даних, а саме, список клієнтів,які зробили найбільше за кількістю страв замовлення.

4. Метод `public static String updateRecord(Connection connection, String clientEmailAddress, String newClientName, String newClientPhoneNumber)` — метод приймає в якості параметрів об'єкт існуючого з'єднання з базою даних, значення електронної пошти клієнта, запис якого необхідно оновити, а також значення нових імені та номера телефону клієнта. Створюються змінні з рядком тексту запиту, змінна з результатом успішності виконання запиту. Після цього встановлюються значення значення полів об'єкта запиту, викликається метод виконання запиту, паралельно вимірюється час його виконання і зберігається у глобальну змінну. Далі перевіряється успішність виконання запиту і повертається її результат.

5. Метод `public static String deleteRecord(Connection connection, String clientEmailAddress)` — метод приймає в якості параметрів об'єкт з'єднання з базою даних та рядок з електронною поштою клієнта, запис якого необхідно видалити. Створюються рядки з текстом запитів видалення запису з таблиці клієнтів та записів залежних таблиць. Далі відбувається підготовка об'єктів даних запитів, запити виконуються, перевіряється успішність їх виконання і повертається її результат.

III. Клас DishCRUDOperations

1. Метод `public static String createRecord(Connection connection, int dishId, String dishName, int weight, int price, int servingsAmount, int orderNumber)` - метод приймає в якості параметрів існуюче з'єднання з базою даних та значення полів нового запису таблиці dish. Створюється рядок з текстои відповідного sql-запиту та рядкова змінна для зберігання результату виконання запиту. Далі відбувається підготовка об'єкта відповідного запиту, шляхом встановлення об'єкту знаень полів новоствореного запису з використанням плейсхолдерів. Після цього викликається метод для виконання запиту, паралельно вимірюється час його виконання, що зберігається у глобальну змінну. Після цього перевіряється успішнісь виконання запиту та повертається повідомлення про неї.

2. Метод `public static ArrayList<Dish> getAllRecords(Connection connection) throws SQLException`
- метод приймає в якості параметра об єкт існуючого з'єднання з базою даних. Створюються змінні для зберігання тексту запиту, успішності виконання запиту,

набору результатів запиту. Після цього отримуються значення полів кожного запису таблиці, з яких створюєтьс об’єкт з полями, ідентичними атрибутам запису. Даний об’єкт додається в колекцію ArrayList. Це відбувається для всіх рядків таблиці. Після цього метод повертає колекцію, яка є списком існуючих записів таблці dish.

3. Метод `public static String updateRecord(Connection connection, int dishId, String newDishName, int newWeight, int newPrice, int newServingsAmount)` - метод приймає в якості параметрів об’єкт існуючого з’єднання з базою даних, значення ідентифікатора страви, запис якої необхідно оновити, а також нові значення інших полів страви. Створюються змінні з рядком тексту запиту, змінна з результатом успішності виконання запиту. Після цього встановлюються значення значення полів об’єкта запиту, викликається метод виконання запиту, паралельно вимірюється час його виконання і зберігається у глобальну змінну. Далі перевіряється успішність виконання запиту і повертається її результат.

4. Метод `public static String deleteRecord(Connection connection, int dishId)` - метод приймає в якості параметрів об’єкт з’єднання з базою даних та рядок з ідентифікатором страви, запис якої необхідно видалити. Створюються рядки з текстом запитів видалення запису з таблиці страв. Далі відбувається підготовка об’єктів даних запитів, запити виконуються, перевіряється успішність їх виконання і повертається її результат.

IV. Клас OrderCRUDOperations

1. Метод `public static String createRecord(Connection connection, int orderNumber, String deliveryAddress, Timestamp deliveryDateTime, String clientEmailAddress, String courierPhoneNumber)` - метод приймає в якості параметрів існуюче з’єднання з базою даних та значення полів нового запису таблиці orders. Створюється рядок з текстои відповідного sql-запиту та рядкова змінна для зберігання результату виконання запиту. Далі відбувається підготовка об’єкта відповідного запиту, шляхом встановлення об’єкту знаень полів новоствореного запису з використанням плейсхолдерів. Після цього викликається метод для виконання запиту, паралельно вимірюється час його виконання, що зберігається у глобальну змінну. Після цього перевіряється успішнісь виконання запиту та повертається повідомлення про неї.

2. Метод `public static ArrayList<Order> getAllRecords(Connection connection) throws SQLException` - метод приймає в якості параметра об єкт існуючого з’єднання з базою даних. Створюються змінні для зберігання тексту запиту, успішності виконання запиту, набору результатів запиту. Після цього отримуються значення полів кожного запису таблиці, з яких створюєтьс об’єкт з

полями, ідентичними атрибутам запису. Даний об’єкт додається в колекцію ArrayList. Це відбувається для всіх рядків таблиці. Після цього метод повертає колекцію, яка є списком існуючих записів таблці orders.

3. Метод `public static String updateRecord(Connection connection, int orderNumber, String newDeliveryAddress, Timestamp newDeliveryDateTime)` - метод приймає в якості параметрів об’єкт існуючого з’єднання з базою даних, значення номера замовлення, запис якого необхідно оновити, а також нові значення інших полів замовлення. Створюються змінні з рядком тексту запиту, змінна з результатом успішності виконання запиту. Після цього встановлюються значення значення полів об’єкта запиту, викликається метод виконання запиту, паралельно вимірюється час його виконання і зберігається у глобальну змінну. Далі перевіряється успішність виконання запиту і повертається її результат.

4. Метод `public static String deleteRecord(Connection connection, int orderNumber)` - метод приймає в якості параметрів об’єкт з’єднання з базою даних та рядок з номером замовлення, запис якого необхідно видалити. Створюються рядки з текстом запитів видалення запису з таблиці замовлень. Далі відбувається підготовка об’єктів даних запитів, запити виконуються, перевіряється успішність їх виконання і повертається її результат.

V. Клас CourierCRUDOperations

1. Метод `public static String createRecord(Connection connection, String courierPhoneNumber, String courierName, String transportKind, double courierRating)` - метод приймає в якості параметрів існуюче з’єднання з базою даних та значення полів нового запису таблиці courier. Створюється рядок з текстои відповідного sql-запиту та рядкова змінна для зберігання результату виконання запиту. Далі відбувається підготовка об’єкта відповідного запиту, шляхом встановлення об’єкту знаень полів новоствореного запису з використанням плейсхолдерів. Після цього викликається метод для виконання запиту, паралельно вимірюється час його виконання, що зберігається у глобальну змінну. Після цього перевіряється успішніс виконання запиту та повертається повідомлення про неї.

2. `public static void generateCourierData(Connection connection, int numberOfRecords)` — метод приймає в якості параметрів об’єкт існуючого з’єднання з базою даних і значення кількості рядків таблиці, які необхідно згенерувати. Створюється рядок з текстом запиту та підготовлюється об’єкт запиту. Після цього за допомогою плейсхолдера встановлюється значення кількості рядків, які необхідно створити. Далі викликається метод для виконання запиту.

3. `public static ArrayList<Courier> getAllRecords(Connection connection) throws SQLException` - метод приймає в якості параметра об єкт існуючого з’єднання з базою даних. Створюються змінні для зберігання тексту запиту, успішності виконання запиту, набору результатів запиту. Після цього отримуються значення полів кожного запису таблиці, з яких створюєтьс об’єкт з полями, ідентичними атрибутам запису. Даний об’єкт додається в колекцію ArrayList. Це відбувається для всіх рядків таблиці. Після цього метод повертає колекцію, яка є списком існуючих записів таблці courier.

4. `public static ArrayList<String> getCouriersWithLessOrders(Connection connection) throws SQLException` - аналогічний методу `getAllRecords`, з тією різницею, що даний метод повертає список кур’єрів, що відповідає певним параметрам пошуку у базі даних, а саме, список кур’єрів, які виконали найменшу кількість замовлень.

5. `public static String updateRecord(Connection connection, String courierPhoneNumber, String newTransportKind, double newCourierRating)` - метод приймає в якості параметрів об’єкт існуючого з’єднання з базою даних, значення номера телефону кур’єра, запис якого необхідно оновити, а також нові значення інших полів кур’єра. Створюються змінні з рядком тексту запиту, змінна з результатом успішності виконання запиту. Після цього встановлюються значення значення полів об’єкта запиту, викликається метод виконання запиту, паралельно вимірюється час його виконання і зберігається у глобальну змінну. Далі перевіряється успішність виконання запиту і повертається її результат.

6. `public static String deleteRecord(Connection connection, String courierPhoneNumber)` - метод приймає в якості параметрів об’єкт з’єднання з базою даних та рядок з номером телефону кур’єра, запис якого необхідно видалити. Створюються рядки з текстом запитів видалення запису з таблиці кур’єрів. Далі відбувається підготовка об’єктів даних запитів, запити виконуються, перевіряється успішність їх виконання і повертається її результат.