Міністерство освіти та науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет прикладної математики
Кафедра системного програмування і спеціалізованих комп’ютерних систем

Розрахунково-графічна робота

з дисципліни

“Бази даних та засоби управління”

Виконав:
студент 3-го курсу,
групи КВ-23
Литвин Станіслав
Романович

Київ 2024

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).

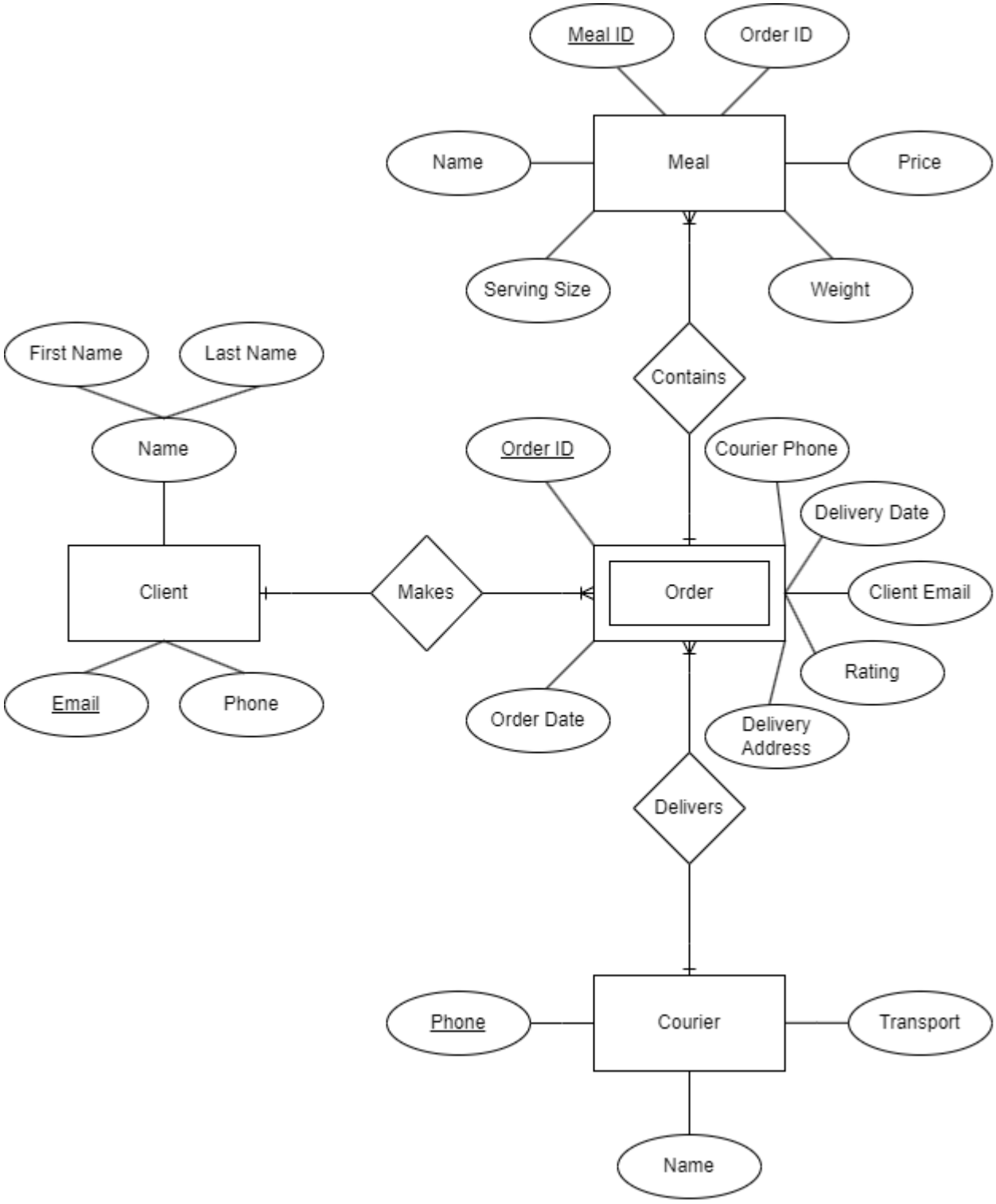
Посилання на репозиторій Github

https://github.com/lynph4/Database.git

Контакт

@lynph4

Діаграма сутність-зв’язок



ER діаграма виконана за нотацією “Пітера Чена”

Опис зв’язків між сутностями:

Зв’язок	**Вид зв’язку**	**Опис**
Order Contains Meals	1 : N	Одне замовлення в ресторані може включати кілька різних страв: суп, салат, напій тощо. Кожна страва прив'язана тільки до одного конкретного замовлення, але одне замовлення може містити кілька страв.
Client Makes Orders	1 : N	Один клієнт може зробити декілька замовлень. Одне замовлення може бути оформлене тільки одним клієнтом.
Courier Delivers Orders	1 : N	Один кур'єр може доставляти багато замовлень одночасно. Одне замовлення може бути доставлене тільки одним кур'єром, оскільки коли клієнт оформлює замовлення, його доставляє конкретний кур'єр.

Короткий опис бази даних:

База даних складається з чотирьох основних сутностей, що представляють різні аспекти процесу замовлення і доставки їжі:

1. **Client –** відображає інформацію про осіб, які здійснюють замовлення. Містить такі атрибути: ім'я, прізвище, електронну адресу для зв'язку та номер телефону клієнта.
2. **Order –** представляє транзакції, зроблені клієнтами. Включає такі дані, як унікальний ідентифікатор замовлення, дата і час замовлення, адреса доставки, дата і час доставки, номер телефону кур'єра для координації, рейтинг доставки та електронну адресу клієнта.
3. **Meal –** описує страви, які пов’язані з замовленням. Містить унікальний ідентифікатор страви, назву, ціну, вагу та кількість на одну порцію.
4. **Courier –** характеризує осіб, відповідальних за доставку замовлень. Включає ім'я кур'єра, його номер телефону для зв'язку та опис транспортного засобу, який використовується для доставки.

Схема меню користувача

Мова програмування та бібліотеки

Мова програмування: Java 23

Бібліотеки:

- PostgreSQL JDBC Driver
- Apache Commons Lang 3.17.0

Вилучення запису батьківської таблиці та виведення вмісту дочірньої таблиці

Вміст таблиці Order:

```
═══════════════════════════════════
            ORDERS TABLE MANAGEMENT
═══════════════════════════════════
  [1] Add Order
  [2] View Orders
  [3] Update Order
  [4] Delete Order
  [5] Fetch Client Analytics
  [6] Fetch Courier Analytics
  [7] Back to Main Menu
>> 2
+----------+---------------------+---------------+---------------------+-------------------------------+--------+---------------------+
| Order ID | Order Date          | Courier Phone | Delivery Date       | Client Email                  | Rating | Delivery Address    |
+----------+---------------------+---------------+---------------------+-------------------------------+--------+---------------------+
| 1001     | 2024-10-04 14:20:00.0 | 8765432109  | 2024-10-05 18:45:00.0 | jane.smith@yahoo.com        | 3      | Nova 12             |
| 1003     | 2024-10-04 14:31:00.0 | 9876543210  | 2024-11-06 15:20:00.0 | john.doe@gmail.com          | 3      | Nova 16             |
| 1002     | 2024-10-04 14:30:00.0 | 9876543210  | 2024-10-05 16:20:00.0 | john.doe@gmail.com          | 4      | Nova 14             |
| 1004     | 2024-10-04 14:37:00.0 | 6314414270  | 2024-11-06 15:25:00.0 | frank.miller@example.com    | 4      | Nova 33             |
| 1005     | 2024-10-04 14:39:00.0 | 6314414270  | 2024-11-06 15:45:00.0 | frank.miller@example.com    | 5      | Nova 33             |
+----------+---------------------+---------------+---------------------+-------------------------------+--------+---------------------+
Press any key to continue.
```

Вміст таблиці Meal:

```
═══════════════════════════════
            MEALS TABLE MANAGEMENT
═══════════════════════════════
  [1] Add Meal
  [2] View Meals
  [3] Update Meal
  [4] Delete Meal
  [5] Back to Main Menu
>> 2
+----------+----------+----------------------+----------+----------+--------------+
| Meal ID  | Order ID | Name                 | Price    | Weight   | Serving Size |
+----------+----------+----------------------+----------+----------+--------------+
| 10       | 1001     | Pizza                | 50       | 300      | 1            |
| 11       | 1001     | Pierogi              | 50       | 300      | 1            |
| 12       | 1002     | Burger               | 150      | 200      | 5            |
| 13       | 1003     | Ramen                | 75       | 400      | 1            |
| 15       | 1005     | Paella               | 400      | 400      | 1            |
| 14       | 1004     | Moussaka             | 175      | 400      | 1            |
+----------+----------+----------------------+----------+----------+--------------+
Press any key to continue.
```

Каскадне видалення клієнта із таблиці Client:

```
                CLIENTS TABLE MANAGEMENT

  [1] Add Client
  [2] View Clients
  [3] Update Client
  [4] Delete Client
  [5] Get Client with Most Orders
  [6] Generate Random Clients Data
  [7] Back to Main Menu
>> 4
Enter the client's email: frank.miller@example.com
Client successfully deleted.
Press any key to continue.
```

Вміст таблиці Order після видалення:

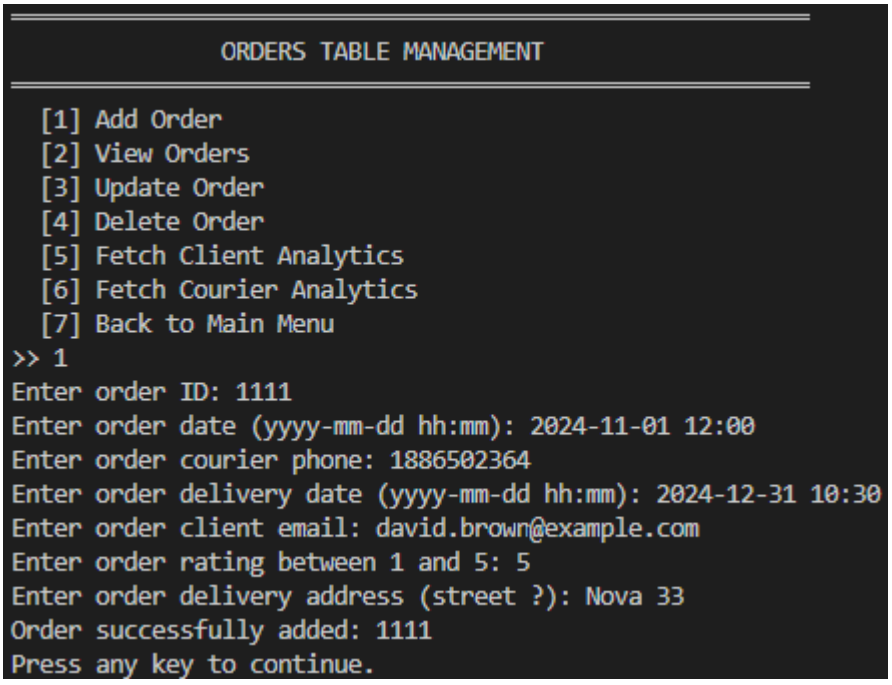
```
              ORDERS TABLE MANAGEMENT

  [1] Add Order
  [2] View Orders
  [3] Update Order
  [4] Delete Order
  [5] Fetch Client Analytics
  [6] Fetch Courier Analytics
  [7] Back to Main Menu
>> 2
+------------+---------------------+---------------+---------------------+-------------------------------+--------+---------------------+
| Order ID   | Order Date          | Courier Phone | Delivery Date       | Client Email                  | Rating | Delivery Address    |
+------------+---------------------+---------------+---------------------+-------------------------------+--------+---------------------+
| 1001       | 2024-10-04 14:20:00.0 | 8765432109    | 2024-10-05 18:45:00.0 | jane.smith@yahoo.com          | 3      | Nova 12             |
| 1003       | 2024-10-04 14:31:00.0 | 9876543210    | 2024-11-06 15:20:00.0 | john.doe@gmail.com            | 3      | Nova 16             |
| 1002       | 2024-10-04 14:30:00.0 | 9876543210    | 2024-10-05 16:20:00.0 | john.doe@gmail.com            | 4      | Nova 14             |
+------------+---------------------+---------------+---------------------+-------------------------------+--------+---------------------+
Press any key to continue.
```

Вміст таблиці Meal після видалення:

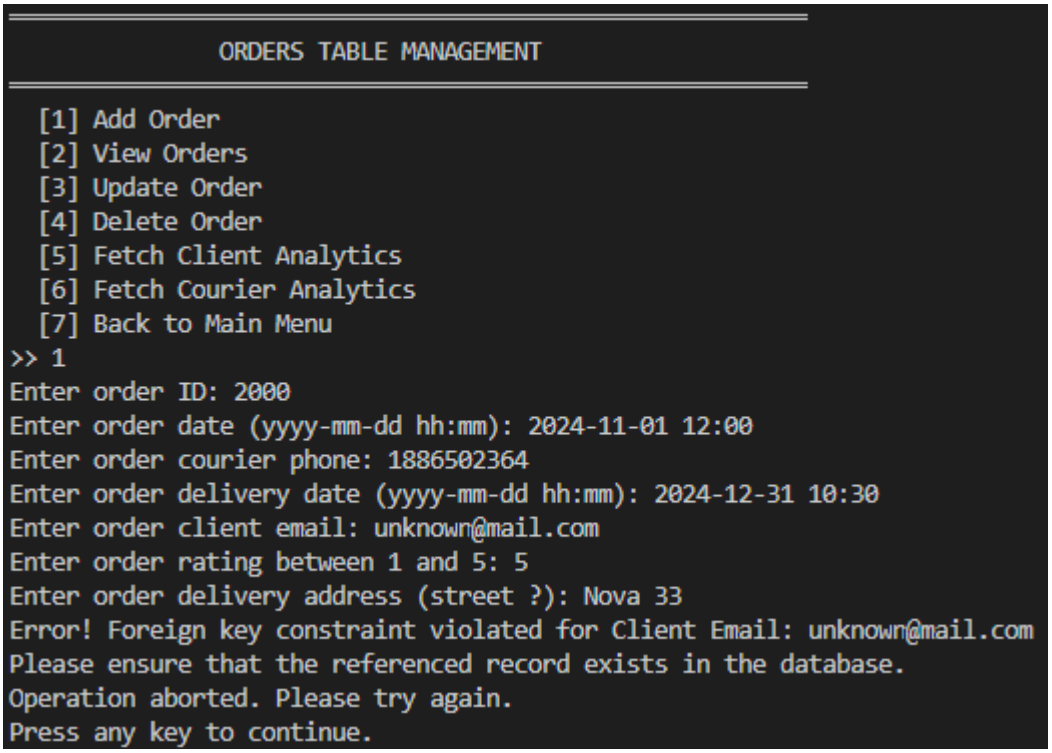
```
              MEALS TABLE MANAGEMENT

  [1] Add Meal
  [2] View Meals
  [3] Update Meal
  [4] Delete Meal
  [5] Back to Main Menu
>> 2
+------------+------------+----------------------+------------+------------+-----------------+
| Meal ID    | Order ID   | Name                 | Price      | Weight     | Serving Size    |
+------------+------------+----------------------+------------+------------+-----------------+
| 10         | 1001       | Pizza                | 50         | 300        | 1               |
| 11         | 1001       | Pierogi              | 50         | 300        | 1               |
| 12         | 1002       | Burger               | 150        | 200        | 5               |
| 13         | 1003       | Ramen                | 75         | 400        | 1               |
+------------+------------+----------------------+------------+------------+-----------------+
Press any key to continue.
```

Вставка запису в дочірню таблицю та виведення повідомлення про її неможливість

Вставка запису, коли зовнішній ключ існує в батьківській таблиці:



Вставка запису, коли зовнішній ключ відсутній у батьківській таблиці:



Генерація випадкових даних

Генерація 100 000 працівників для таблиці Client:

```
═══════════════════════════════════
            CLIENTS TABLE MANAGEMENT
═══════════════════════════════════
  [1] Add Client
  [2] View Clients
  [3] Update Client
  [4] Delete Client
  [5] Get Client with Most Orders
  [6] Generate Random Clients Data
  [7] Back to Main Menu
>> 6
Enter the number of records: 100000
Successfully generated 100000 clients.
Press any key to continue.
```

Фрагменти згенерованої таблиці:

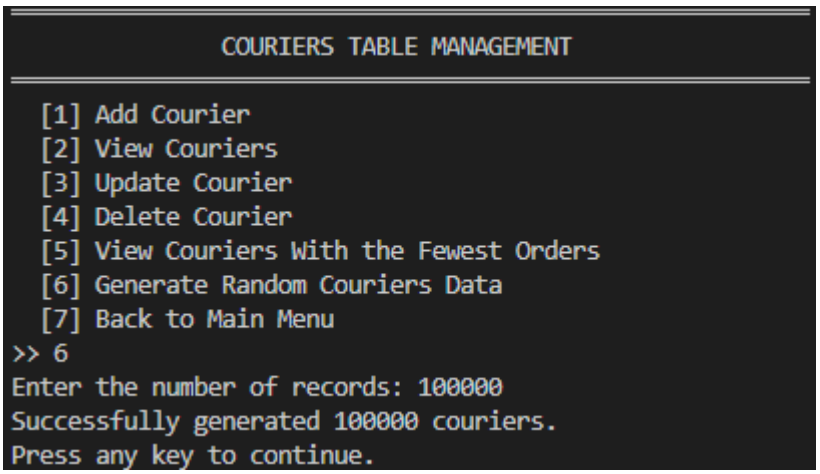
Email	Name	Phone
gus.liu4491247@svc.com	Gus Liu	0674811784
fin.liu1069504@sm.com	Fin Liu	1508701121
ava.bai264623@w.com	Ava Bai	2680743601
ava.gar9368706@u.com	Ava Gar	0805407440
nia.bai609032@svc.com	Nia Bai	1085591079
sky.kim5785826@ex.com	Sky Kim	6017494002
sky.zhu6624336@d.com	Sky Zhu	8310718374
oli.kim5798238@dm.com	Oli Kim	1538632536
jane.smith@yahoo.com	Jane Smith	0987654321
john.doe@gmail.com	John Doe	1234567890
fin.ali6068466@w.com	Fin Ali	0719017892
sky.lin9136350@sm.com	Sky Lin	6261431200
ava.lin956514@rnd.com	Ava Lin	9739945644
mia.liu9146793@ex.com	Mia Liu	7629091165
eli.ali835003@dm.com	Eli Ali	6910061043
oli.gar5357540@ex.com	Oli Gar	8928222004
cal.gar7316431@u.com	Cal Gar	0796990230
jax.lin1785156@ex.com	Jax Lin	4082396789
cal.lee4541161@d.com	Cal Lee	4376023825
david.brown@example.com	David Brown	1427005488
pax.roy3080791@u.com	Pax Roy	5477194876
frank.williams@example.com	Frank Williams	5210982605
bob.johnson@example.com	Bob Johnson	4526397289
grace.smith@example.com	Grace Smith	9638086097
cal.lin5944437@svc.com	Cal Lin	9001513288
kirby@outlook.com	Anthony Kirby	0965550011
fin.ali2972118@tm.com	Fin Ali	6939755372
ivy.yin482890@d.com	Ivy Yin	4390824431

ben.gar3550689@ex.com	Ben Gar	8345061811
eli.lin8053656@svc.com	Eli Lin	0654399354
gus.yin9487052@rnd.com	Gus Yin	0911593775
jax.roy9699658@ex.com	Jax Roy	5890475041
eli.roy6979546@d.com	Eli Roy	1290988128
nia.doe8407821@sm.com	Nia Doe	1733849164
ava.kim9533459@sm.com	Ava Kim	5097116613
fin.wang9516115@svc.com	Fin Wang	4847450894
mia.joy2107113@tm.com	Mia Joy	4457104922
ray.joy181226@u.com	Ray Joy	0716509218
kai.wang3230456@svc.com	Kai Wang	5046906636
gus.lin5290707@sm.com	Gus Lin	2348798683
sky.tan3152639@sm.com	Sky Tan	0693590907
ivy.roy730016@ex.com	Ivy Roy	3091094384
oli.bai9108203@w.com	Oli Bai	5750644179
gus.doe3844326@rnd.com	Gus Doe	3217009330
frank.johnson@example.com	Frank Johnson	8164926723
cal.bai3696082@ml.com	Cal Bai	5378355618
dan.tan9772121@sm.com	Dan Tan	3641756275
dan.doe688002@svc.com	Dan Doe	3426311192
ray.hsu4855205@rnd.com	Ray Hsu	3902315091
oli.roy8447217@rnd.com	Oli Roy	7878375075
kai.hsu3104417@dm.com	Kai Hsu	2802017453
eli.doe5636611@tm.com	Eli Doe	0008694782
jax.zhu7425616@ex.com	Jax Zhu	7855289117
pax.gar2593040@ex.com	Pax Gar	3590308130
leo.bai181763@tm.com	Leo Bai	4741538585
hal.doe6055610@tm.com	Hal Doe	5209322900
ben.wang7550368@svc.com	Ben Wang	3545498953
leo.lin2001333@ml.com	Leo Lin	3245223453
ava.hsu9723172@d.com	Ava Hsu	0764292884
hal.yin2734575@d.com	Hal Yin	2582613517
jack.white164@example.com	Jack White	6479899488
ben.kim162245@d.com	Ben Kim	6131699507
dan.tan4591623@d.com	Dan Tan	5796056902
zoe.liu9054664@u.com	Zoe Liu	0499120833
jax.hsu2159596@ex.com	Jax Hsu	4981846992
cal.liu5426194@d.com	Cal Liu	5001611615
gus.hsu7993060@tm.com	Gus Hsu	6639831327
tia.doe2480676@svc.com	Tia Doe	7169246545
hal.joy7326341@ex.com	Hal Joy	9144716127
zoe.ali1918800@rnd.com	Zoe Ali	0071016971
kai.yin6458068@ex.com	Kai Yin	8223170821
gus.gar7813245@tm.com	Gus Gar	4591351988
gus.bai8739907@sm.com	Gus Bai	1021241917
kai.ali140937@tm.com	Kai Ali	4448879476
hal.liu8521588@d.com	Hal Liu	0723217471
oli.wang1212376@u.com	Oli Wang	2847744852

SQL-запит для генерації даних:

```
INSERT INTO "Client" ("Email", "Name", "Phone")
SELECT
    LOWER(first_name || '.' || last_name || FLOOR(RANDOM() *
10000000)::text || '@' || domain) AS "Email",
    first_name || ' ' || last_name AS "Name",
    LPAD(FLOOR(RANDOM() * 10000000000)::text, 10, '0') AS \"Phone\"
FROM (
    SELECT
        (ARRAY[
            'Ava', 'Ben', 'Cal', 'Dan', 'Eli', 'Fin', 'Gus',
             'Hal', 'Ivy', 'Jax', 'Kai', 'Leo', 'Mia', 'Nia',
             'Oli', 'Pax', 'Ray', 'Sky', 'Tia', 'Zoe'
        ])[FLOOR(RANDOM() * 20) + 1] AS first_name,

        (ARRAY[
             'Doe', 'Lee', 'Kim', 'Zhu', 'Wang', 'Liu',
              'Gar', 'Ali', 'Bai', 'Hsu', 'Roy', 'Joy',
              'Lin', 'Tan', 'Yin'
        ])[FLOOR(RANDOM() * 15) + 1] AS last_name,

        (ARRAY[
            'ex.com', 'tm.com', 'sm.com', 'dm.com', 'rnd.com',
             'ml.com', 'd.com', 'svc.com', 'w.com', 'u.com'
        ])[FLOOR(RANDOM() * 10) + 1] AS domain
    FROM generate_series(1, 100000) AS s
) AS names;
```

Генерація 100 000 працівників для таблиці Courier:



Фрагменти згенерованої таблиці:

```
═══════════════════════════════════════
            COURIERS TABLE MANAGEMENT
═══════════════════════════════════════
  [1] Add Courier
  [2] View Couriers
  [3] Update Courier
  [4] Delete Courier
  [5] View Couriers With the Fewest Orders
  [6] Generate Random Couriers Data
  [7] Back to Main Menu
>> 2
+------------+----------------------+----------------------+
| Phone      | Name                 | Transport            |
+------------+----------------------+----------------------+
| 7654321098 | Liam Miller          | Motorbike            |
| 9876543210 | Mike Johnson         | Bicycle              |
| 8765432109 | Emma Brown           | Motorbike            |
| 7879392303 | Charlie Brown        | Motorbike            |
| 5997457432 | Eve Johnson          | Van                  |
| 9432872324 | Charlie Jones        | Bicycle              |
| 4751325734 | Grace Miller         | Motorbike            |
| 9438129987 | Charlie Smith        | Van                  |
| 6314414270 | Bob Williams         | Bicycle              |
| 6327257236 | Grace Garcia         | Scooter              |
| 2882424404 | Grace Miller         | Bicycle              |
| 1139691935 | Charlie Jones        | Motorbike            |
| 4322074430 | Alice Smith          | Truck                |
| 6613397074 | Charlie Williams     | Motorbike            |
| 9002750535 | Eve Garcia           | Truck                |
| 1886502364 | Charlie Brown        | Scooter              |
| 8256519523 | Bob Williams         | Truck                |
| 0165352992 | Frank Miller         | Truck                |
| 4886718446 | Grace Smith          | Motorbike            |
| 7967873210 | Eve Garcia           | Scooter              |
| 4377128978 | Frank Johnson        | Truck                |
| 5373025681 | Alice Johnson        | Bicycle              |
| 0872180718 | Bob Williams         | Bicycle              |
| 6251236049 | Alice Williams       | Bicycle              |
| 8690207885 | Charlie Garcia       | Scooter              |
| 1902240646 | Alice Smith          | Truck                |
| 7273332008 | David Smith          | Truck                |
| 2105287871 | Alice Smith          | Bicycle              |
| 1082304578 | Charlie Jones        | Motorbike            |
| 5291920533 | Eve Johnson          | Scooter              |
| 6930642780 | Eve Williams         | Van                  |
| 7150170122 | Bob Brown            | Bicycle              |
| 0125114869 | David Garcia         | Van                  |
| 3038380291 | Grace Miller         | Bicycle              |
| 9991294805 | Frank Williams       | Motorbike            |
```

SQL-запит для генерації даних:

```
INSERT INTO "Courier" ("Phone", "Name", "Transport")
SELECT
    LPAD(FLOOR(RANDOM() * 10000000000)::text, 10, '0') AS \"Phone\",
    (ARRAY['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank',
'Grace'])[FLOOR(RANDOM() * 7) + 1] || ' ' ||
    (ARRAY['Smith', 'Johnson', 'Williams', 'Brown', 'Jones', 'Garcia',
'Miller'])[FLOOR(RANDOM() * 7) + 1] AS \"Name\",
    (ARRAY['Bicycle', 'Motorbike', 'Van', 'Truck',
'Scooter'])[FLOOR(RANDOM() * 5) + 1] AS \"Transport\"
FROM generate_series(1, 100000) AS s;
```

Пошукові запити

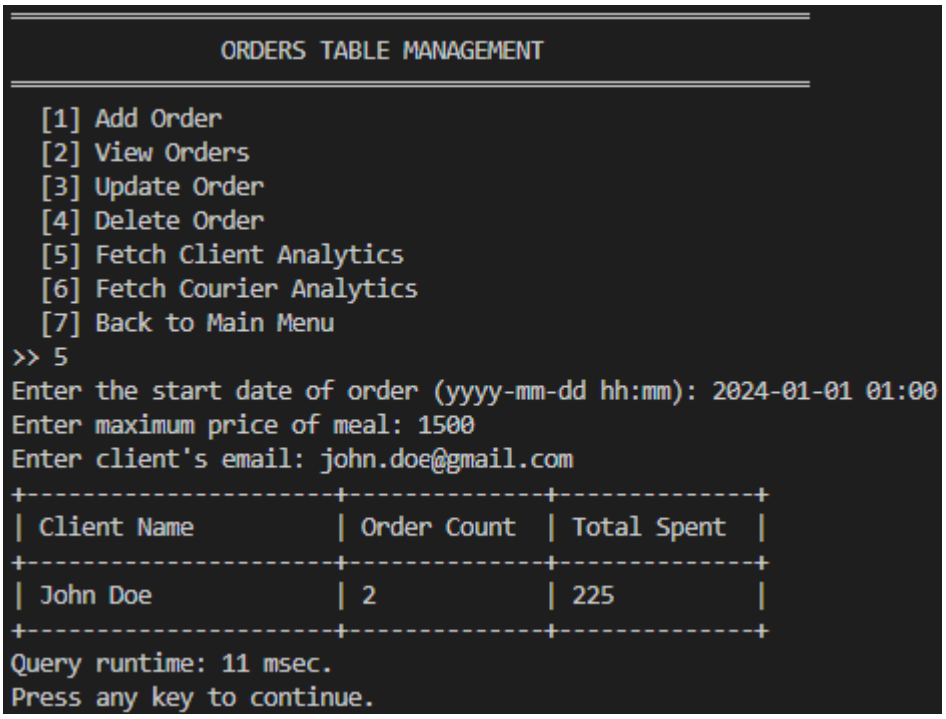
Пошуковий запит №1 (пошук кур’єрів із найбільшою кількістю замовлень):

```
==========================================================
              COURIERS TABLE MANAGEMENT
==========================================================
  [1] Add Courier
  [2] View Couriers
  [3] Update Courier
  [4] Delete Courier
  [5] View Couriers With the Most Orders
  [6] Generate Random Couriers Data
  [7] Back to Main Menu
>> 5
Enter the number of records: 15

**********************************************************
 TOP 15 COURIERS WITH THE MOST ORDERS
+--------------------+------------+-------------+
| Name               | Phone      | Order Count |
| Mike Johnson       | 9876543210 | 2           |
| Emma Brown         | 8765432109 | 1           |
| Charlie Brown      | 1886502364 | 1           |
| Grace Brown        | 0000324163 | 0           |
| Frank Williams     | 0000765325 | 0           |
| Frank Jones        | 0000774039 | 0           |
| Alice Smith        | 0000407106 | 0           |
| Alice Jones        | 0001005008 | 0           |
| Frank Jones        | 0001265114 | 0           |
| Eve Miller         | 0001455230 | 0           |
| David Jones        | 0001469266 | 0           |
| David Miller       | 0001483441 | 0           |
| Frank Jones        | 0001522327 | 0           |
| Eve Johnson        | 0001575848 | 0           |
| Charlie Johnson    | 0000847097 | 0           |
+--------------------+------------+-------------+
Query runtime: 102 msec.
Press any key to continue.
```

SQL-запит, що ілюструє пошук №1:

```
SELECT
    "Courier"."Name",
    "Courier"."Phone",
    "Courier"."Transport",
     COUNT("Order"."Order ID") AS "Order Count"
FROM "Courier"
LEFT JOIN "Order" ON "Courier"."Phone" = "Order"."Courier Phone"
GROUP BY "Courier"."Name", "Courier"."Phone"
ORDER BY "Order Count" DESC
LIMIT 15;
```

Пошуковий запит №2 (обчислення загальних витрат окремим клієнтом):

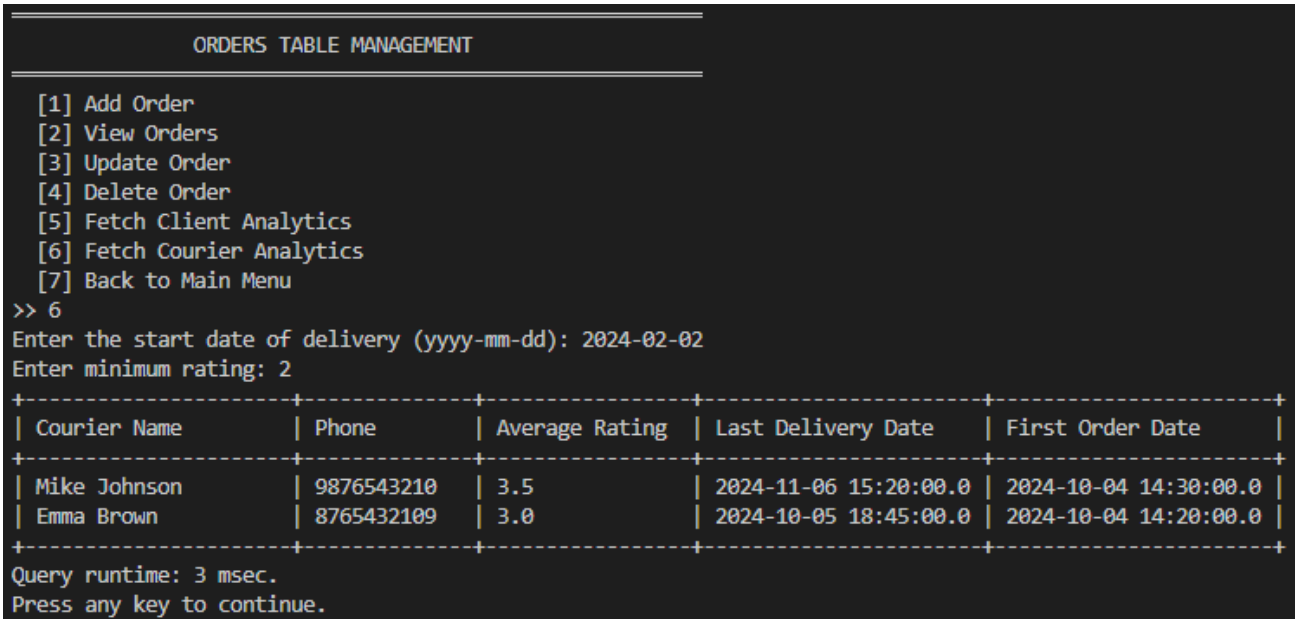


SQL-запит, що ілюструє пошук №2:

```
SELECT
    c."Name" AS client_name,
    COUNT(DISTINCT o."Order ID") AS order_count,
    SUM(m."Price") AS total_spent
FROM "Client" c
JOIN "Order" o ON c."Email" = o."Client Email"
JOIN "Meal" m ON o."Order ID" = m."Order ID"
WHERE o."Order Date" >= '2024-01-01 01:00'
AND m."Price" <= 1500
AND c."Email" LIKE 'john.doe@gmail.com'
GROUP BY c."Email", c."Name"
```

```
ORDER BY total_spent DESC;
```

Пошуковий запит №3 (середня оцінка за кур’єрами):



SQL-запит, що ілюструє пошук №3:

```
SELECT
    co."Name" AS courier_name,
    co."Phone",
    AVG(o."Rating") AS average_rating,
    MAX(o."Delivery Date") AS last_delivery_date,
    MIN(o."Order Date") AS first_order_date
FROM "Courier" co
JOIN "Order" o ON co."Phone" = o."Courier Phone"
JOIN "Meal" m ON o."Order ID" = m."Order ID"
WHERE
    o."Delivery Date" >= '2024-02-02' AND
    o."Rating" >= 2
GROUP BY
    co."Name",
    co."Phone",
    co."Transport"
ORDER BY average_rating DESC;
```

Програмний код модуля “Model”, згідно із шаблоном MVC

```
package model;

import java.sql.Timestamp;
import java.sql.Date;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Optional;
import java.util.function.Predicate;
import org.apache.commons.lang3.tuple.Pair;

import common.*;
import entities.*;
import model.validation.*;
import util.Error;
import util.Result;
import util.SQLQueryRuntime;

public class Model {
    private Connection connection;

    public Model(Connection connection) {
        this.connection = connection;
    }

    public Optional<Error> addClient(Client client) {
        if (!NameValidator.isValidName(client.name())) {
            return Optional.of(new Error.ValidationError("Wrong name."));
        }

        if (!EmailValidator.isValidEmail(client.email())) {
            return Optional.of(new Error.ValidationError("Wrong email."));
        }

        if (!PhoneNumberValidator.isValidPhoneNumber(client.phone())) {
            return Optional.of(new Error.ValidationError("Wrong phone number."));
        }

        Predicate<String> isClientExists = (String email) -> {
            final String sql = "SELECT EXISTS(SELECT 1 FROM \"Client\" WHERE \"Email\"
= ?)";

            try (PreparedStatement pstmt = connection.prepareStatement(sql,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY)) {
                pstmt.setString(1, email);
                try (ResultSet resultSet = pstmt.executeQuery()) {
                    if (resultSet.next()) {
                        return resultSet.getBoolean(1);
                    }
                }
            } catch (SQLException e) {
                assert false;
            }
            return false;
        };
```

```
        if (isClientExists.test(client.email())) {
            return Optional.of(new Error.DuplicateKeyError(client.email()));
        }

        final String sql = "INSERT INTO \"Client\"(\"Email\", \"Name\", \"Phone\")
VALUES(?, ?, ?)";

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setString(1, client.email());
            pstmt.setString(2, client.name());
            pstmt.setString(3, client.phone());
            pstmt.executeUpdate();
        } catch (SQLException e) {
            throw new IllegalStateException("An unexpected error occurred while adding
a client to the database.");
        }

        return Optional.empty();
    }

    public ArrayList<Client> getAllClients() throws IllegalStateException {
        ArrayList<Client> clients = new ArrayList<>();
        final String sql = "SELECT * FROM \"Client\"";

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            ResultSet resultSet = pstmt.executeQuery();

            while (resultSet.next()) {
                String email = resultSet.getString("Email");
                String name = resultSet.getString("Name");
                String phone = resultSet.getString("Phone");

                Client client = new Client(email, name, phone);
                clients.add(client);
            }
        } catch (SQLException e){
            throw new IllegalStateException("An unexpected error occurred while
collecting clients from the database.");
        }

        return clients;
    }

    public Result<Error, Client> getClient(String email) throws IllegalStateException {
        final String sql = "SELECT \"Name\", \"Phone\" FROM \"Client\" WHERE \"Email\"
= ?";

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setString(1, email);
            ResultSet resultSet = pstmt.executeQuery();

            if (resultSet.next()) {
                String name = resultSet.getString("Name");
                String phone = resultSet.getString("Phone");

                Client client = new Client(email, name, phone);
                return new Result.Success<>(client);
            }
        } catch (SQLException e){
```

```
            throw new IllegalStateException("An unexpected error occurred while
collecting client from the database.");
        }

        return new Result.Failure<>(new Error.RecordNotFound(email));
    }

    public Optional<Error> updateClient(Client client) throws IllegalStateException {
        if (!NameValidator.isValidName(client.name())) {
            return Optional.of(new Error.ValidationError("Wrong name."));
        }

        if (!EmailValidator.isValidEmail(client.email())) {
            return Optional.of(new Error.ValidationError("Wrong email."));
        }

        if (!PhoneNumberValidator.isValidPhoneNumber(client.phone())) {
            return Optional.of(new Error.ValidationError("Wrong phone number."));
        }

        final String sql = "UPDATE \"Client\" SET \"Name\" = COALESCE(?, \"Name\"),
\"Phone\" = COALESCE(?, \"Phone\") WHERE \"Email\" = ?";

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setString(1, client.name());
            pstmt.setString(2, client.phone());
            pstmt.setString(3, client.email());

            int rowCount = pstmt.executeUpdate();
            if (rowCount == 0) {
                return Optional.of(new Error.RecordNotFound(client.email()));
            }
        } catch (SQLException e) {
            throw new IllegalStateException("An unexpected error occurred while
updating the client with email '" +
                client.email() + "' from the database.");
        }

        return Optional.empty();
    }

    public Optional<Error> deleteClient(String email) {
        final String sql = "DELETE FROM \"Client\" WHERE \"Email\" = ?";

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setString(1, email);

            int rowCount = pstmt.executeUpdate();
            if (rowCount == 0) {
                return Optional.of(new Error.RecordNotFound(email));
            }
        } catch (SQLException e) {
            throw new IllegalStateException("An unexpected error occurred while
deleting the client with email '" +
                email + "' from the database.");
        }

        return Optional.empty();
    }

    public Optional<Pair<Client, Integer>> getClientWithMostOrders() {
```

```
        final String sql = """
                    WITH MaxOrderClient AS (
                    SELECT \"Client Email\", COUNT(*) AS OrderCount
                    FROM \"Order\"
                    GROUP BY \"Client Email\"
                    ORDER BY OrderCount DESC
                    LIMIT 1
                )
                SELECT c.*, moc.OrderCount
                FROM \"Client\" c
                JOIN MaxOrderClient moc ON c."Email" = moc.\"Client Email\";
                """;

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            SQLQueryRuntime.beginScope();
            ResultSet resultSet = pstmt.executeQuery();
            SQLQueryRuntime.endScope();

            if (resultSet.next()) {
                String email = resultSet.getString("Email");
                String name = resultSet.getString("Name");
                String phone = resultSet.getString("Phone");
                int orderCount = resultSet.getInt("OrderCount");

                Client client = new Client(email, name, phone);
                return Optional.of(Pair.of(client, Integer.valueOf(orderCount)));
            }
        } catch (SQLException e) {
            throw new IllegalStateException("An unexpected error occurred while
fetching a client with most orders from the database.");
        }

        return Optional.empty();
    }

    public void generateRandomClients(int numberOfRecords) {
        final String sql = """
                INSERT INTO \"Client\" (\"Email\", \"Name\", \"Phone\")
                SELECT
                    LOWER(first_name || '.' || last_name || FLOOR(RANDOM() *
10000000)::text || '@' || domain) AS \"Email\",
                    first_name || ' ' || last_name AS "Name",
                    LPAD(FLOOR(RANDOM() * 10000000000)::text, 10, '0') AS \"Phone\"
                FROM (
                    SELECT
                        (ARRAY[
                            'Ava', 'Ben', 'Cal', 'Dan', 'Eli', 'Fin', 'Gus',
                            'Hal', 'Ivy', 'Jax', 'Kai', 'Leo', 'Mia', 'Nia',
                            'Oli', 'Pax', 'Ray', 'Sky', 'Tia', 'Zoe'
                        ])[FLOOR(RANDOM() * 20) + 1] AS first_name,

                        (ARRAY[
                            'Doe', 'Lee', 'Kim', 'Zhu', 'Wang', 'Liu',
                            'Gar', 'Ali', 'Bai', 'Hsu', 'Roy', 'Joy',
                            'Lin', 'Tan', 'Yin'
                        ])[FLOOR(RANDOM() * 15) + 1] AS last_name,

                        (ARRAY[
                            'ex.com', 'tm.com', 'sm.com', 'dm.com', 'rnd.com',
                            'ml.com', 'd.com', 'svc.com', 'w.com', 'u.com'
                        ])[FLOOR(RANDOM() * 10) + 1] AS domain
```

```
                    FROM generate_series(1, ?) AS s
                ) AS names;
                """;

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setInt(1, numberOfRecords);
            pstmt.executeUpdate();
        } catch (SQLException e) {
            throw new IllegalStateException("An unexpected error occurred while
generating random clients in the database.");
        }
    }

    public Optional<Error> addCourier(Courier courier) {
        if (!PhoneNumberValidator.isValidPhoneNumber(courier.phone())) {
            return Optional.of(new Error.ValidationError("Wrong phone number."));
        }

        if (!NameValidator.isValidName(courier.name())) {
            return Optional.of(new Error.ValidationError("Wrong name."));
        }

        Predicate<String> isCourierExists = (String phone) -> {
            final String sql = "SELECT EXISTS(SELECT 1 FROM \"Courier\" WHERE \"Phone\"
= ?)";

            try (PreparedStatement pstmt = connection.prepareStatement(sql,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY)) {
                pstmt.setString(1, phone);
                try (ResultSet resultSet = pstmt.executeQuery()) {
                    if (resultSet.next()) {
                        return resultSet.getBoolean(1);
                    }
                }
            } catch (SQLException e) {
                throw new IllegalStateException("An unexpected error occurred while
checking the availability of the courier in the database.");
            }
            return false;
        };

        if (isCourierExists.test(courier.phone())) {
            return Optional.of(new Error.DuplicateKeyError(courier.phone()));
        }

        final String sql = "INSERT INTO \"Courier\"(\"Phone\", \"Name\", \"Transport\")
VALUES(?, ?, ?)";

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setString(1, courier.phone());
            pstmt.setString(2, courier.name());
            pstmt.setString(3, courier.transport());

            int rowCount = pstmt.executeUpdate();
            System.out.println("" + rowCount);
        } catch (SQLException e) {
            throw new IllegalStateException("An unexpected error occurred while adding
a courier to the database.");
        }

        return Optional.empty();
```

```
    }

    public ArrayList<Courier> getAllCouriers() throws IllegalStateException {
        ArrayList<Courier> couriers = new ArrayList<>();
        final String sql = "SELECT * FROM \"Courier\"";

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            ResultSet resultSet = pstmt.executeQuery();

            while (resultSet.next()) {
                String phone = resultSet.getString("Phone");
                String name = resultSet.getString("Name");
                String transport = resultSet.getString("Transport");

                Courier client = new Courier(phone, name, transport);
                couriers.add(client);
            }
        } catch (SQLException e){
            throw new IllegalStateException("An unexpected error occurred while
collecting couriers from the database.");
        }

        return couriers;
    }

    public Result<Error, Courier> getCourier(String phone) throws IllegalStateException
{
        final String sql = """
                SELECT * FROM \"Courier\"
                WHERE \"Phone\" = ?
                """;

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setString(1, phone);
            ResultSet resultSet = pstmt.executeQuery();

            if (resultSet.next()) {
                String name = resultSet.getString("Name");
                String transport = resultSet.getString("Transport");

                Courier courier = new Courier(phone, name, transport);
                return new Result.Success<>(courier);
            }
        } catch (SQLException e){
            throw new IllegalStateException("An unexpected error occurred while
collecting courier from the database.");
        }

        return new Result.Failure<>(new Error.RecordNotFound(phone));
    }

    public Optional<Error> updateCourier(Courier courier) throws IllegalStateException
{
        if (!PhoneNumberValidator.isValidPhoneNumber(courier.phone())) {
            return Optional.of(new Error.ValidationError("Wrong phone number."));
        }

        if (!NameValidator.isValidName(courier.name())) {
            return Optional.of(new Error.ValidationError("Wrong name."));
        }
```

```
        final String sql = "UPDATE \"Courier\" SET \"Name\" = COALESCE(?, \"Name\"),
\"Transport\" = COALESCE(?, \"Transport\") WHERE \"Phone\" = ?";

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setString(1, courier.name());
            pstmt.setString(2, courier.transport());
            pstmt.setString(3, courier.phone());

            int rowCount = pstmt.executeUpdate();
            if (rowCount == 0) {
                return Optional.of(new Error.RecordNotFound(courier.phone()));
            }
        } catch (SQLException e) {
            throw new IllegalStateException("An unexpected error occurred while
updating the courier with phone number '" +
                courier.phone() + "' from the database.");
        }

        return Optional.empty();
    }

    public Optional<Error> deleteCourier(String phone) {
        final String sql = """
                DELETE FROM \"Courier\"
                WHERE \"Phone\" = ?
                """;

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setString(1, phone);

            int rowCount = pstmt.executeUpdate();
            if (rowCount == 0) {
                return Optional.of(new Error.RecordNotFound(phone));
            }
        } catch (SQLException e) {
            throw new IllegalStateException("An unexpected error occurred while
deleting the courier with phone number '" +
                phone + "' from the database.");
        }

        return Optional.empty();
    }

    public ArrayList<Pair<Courier, Integer>> getCouriersWithMostOrders(int
numberOfRecords) {
        final String sql = """
                SELECT
                    \"Courier\".\"Name\",
                    \"Courier\".\"Phone\",
                    \"Courier\".\"Transport\",
                    COUNT(\"Order\".\"Order ID\") AS "Order Count"
                FROM \"Courier\"
                LEFT JOIN "Order" ON \"Courier\".\"Phone\" = \"Order\".\"Courier
Phone\"
                GROUP BY \"Courier\".\"Name\", \"Courier\".\"Phone\"
                ORDER BY "Order Count" DESC
                LIMIT ?;
                """;

        ArrayList<Pair<Courier, Integer>> couriers = new ArrayList<>();
        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
```

```
            pstmt.setInt(1, numberOfRecords);

            SQLQueryRuntime.beginScope();
            ResultSet resultSet = pstmt.executeQuery();

            while (resultSet.next()) {
                String name = resultSet.getString("Name");
                String phone = resultSet.getString("Phone");
                int orderCount = resultSet.getInt("Order Count");

                Courier courier = new Courier(phone, name, null);
                couriers.add(Pair.of(courier, Integer.valueOf(orderCount)));
            }

            SQLQueryRuntime.endScope();

        } catch (SQLException e) {
            throw new IllegalStateException("An unexpected error occurred while
fetching couriers with most orders from the database.");
        }

        return couriers;
    }

    public void generateRandomCouriers(int numberOfRecords) {
        final String sql = """
                INSERT INTO \"Courier\" (\"Phone\", \"Name\", \"Transport\")
                SELECT
                    LPAD(FLOOR(RANDOM() * 10000000000)::text, 10, '0') AS \"Phone\",
                    (ARRAY['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank',
'Grace'])[FLOOR(RANDOM() * 7) + 1] || ' ' ||
                    (ARRAY['Smith', 'Johnson', 'Williams', 'Brown', 'Jones', 'Garcia',
'Miller'])[FLOOR(RANDOM() * 7) + 1] AS \"Name\",
                    (ARRAY['Bicycle', 'Motorbike', 'Van', 'Truck',
'Scooter'])[FLOOR(RANDOM() * 5) + 1] AS \"Transport\"
                FROM generate_series(1, ?) AS s;
                """;

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setInt(1, numberOfRecords);
            pstmt.executeUpdate();
        } catch (SQLException e) {
            throw new IllegalStateException("An unexpected error occurred while
generating random clients in the database.");
        }
    }

    public Optional<Error> addMeal(Meal meal) {
        if (!MealNameValidator.isValidName(meal.name())) {
            return Optional.of(new Error.ValidationError("Wrong name."));
        }

        switch (getOrder(meal.orderID())) {
            case Result.Failure<Error, ?> failure -> {
                if (failure.error() instanceof Error.RecordNotFound _) {
                    return Optional.of(new Error.ForeignKeyConstraintError("Order ID",
String.valueOf(meal.orderID())));
                }
                else {
                    return Optional.of(new Error.UnknownError());
                }
```

```
            }
            case Result.Success<?,Order> success -> {
                break;
            }
        }

        Predicate<Integer> isMealExists = (Integer ID) -> {
            final String sql = "SELECT EXISTS(SELECT 1 FROM \"Meal\" WHERE \"Meal ID\"
= ?)";

            try (PreparedStatement pstmt = connection.prepareStatement(sql,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY)) {
                pstmt.setInt(1, ID.intValue());
                try (ResultSet resultSet = pstmt.executeQuery()) {
                    if (resultSet.next()) {
                        return resultSet.getBoolean(1);
                    }
                }
            } catch (SQLException e) {
                throw new IllegalStateException("An unexpected error occurred while
checking the availability of the meal in the database.");
            }
            return false;
        };

        if (isMealExists.test(Integer.valueOf(meal.mealID()))) {
            return Optional.of(new
Error.DuplicateKeyError(String.valueOf(meal.mealID())));
        }

        final String sql = "INSERT INTO \"Meal\"(\"Meal ID\", \"Order ID\", \"Name\",
\"Price\", \"Weight\", \"Serving Size\") VALUES(?, ?, ?, ?, ?, ?)";

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setInt(1, meal.mealID());
            pstmt.setInt(2, meal.orderID());
            pstmt.setString(3, meal.name());
            pstmt.setInt(4, meal.price());
            pstmt.setInt(5, meal.weight());
            pstmt.setInt(6, meal.servingSize());

            if (pstmt.executeUpdate() == 0) {
                Optional.of(new Error.InsertError("Meal"));
            }
        } catch (SQLException e) {
            throw new IllegalStateException("An unexpected error occurred while adding
a meal to the database.");
        }

        return Optional.empty();
    }

    public ArrayList<Meal> getAllMeals() throws IllegalStateException {
        ArrayList<Meal> meals = new ArrayList<>();
        final String sql = "SELECT * FROM \"Meal\"";

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            ResultSet resultSet = pstmt.executeQuery();

            while (resultSet.next()) {
                int mealID = resultSet.getInt("Meal ID");
```

```
                int orderID = resultSet.getInt("Order ID");
                String name = resultSet.getString("Name");
                int price = resultSet.getInt("Price");
                int weight = resultSet.getInt("Weight");
                int servingSize = resultSet.getInt("Serving Size");

                Meal meal = new Meal(mealID, orderID, name, price, weight,
servingSize);
                meals.add(meal);
            }
        } catch (SQLException e){
            throw new IllegalStateException("An unexpected error occurred while
collecting meals from the database.");
        }

        return meals;
    }

    public Optional<Error> updateMeal(Meal meal) throws IllegalStateException {
        if (!MealNameValidator.isValidName(meal.name())) {
            return Optional.of(new Error.ValidationError("Wrong name."));
        }

        final String sql = "UPDATE \"Meal\" SET \"Name\" = COALESCE(?, \"Name\"),
\"Price\" = COALESCE(?, \"Price\"), \"Weight\" = COALESCE(?, \"Weight\"), \"Serving
Size\" = COALESCE(?, \"Serving Size\") WHERE \"Meal ID\" = ?";

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setString(1, meal.name());
            pstmt.setInt(2, meal.price());
            pstmt.setInt(3, meal.weight());
            pstmt.setInt(4, meal.servingSize());
            pstmt.setInt(5, meal.mealID());

            int rowCount = pstmt.executeUpdate();
            if (rowCount == 0) {
                return Optional.of(new
Error.RecordNotFound(String.valueOf(meal.mealID())));
            }
        } catch (SQLException e) {
            throw new IllegalStateException("An unexpected error occurred while
updating the meal with ID '" +
                meal.mealID() + "' from the database.");
        }

        return Optional.empty();
    }

    public Result<Error, Meal> getMeal(int mealID) throws IllegalStateException {
        final String sql = """
                SELECT * FROM \"Meal\"
                WHERE \"Meal ID\" = ?
                """;

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setInt(1, mealID);
            ResultSet resultSet = pstmt.executeQuery();

            if (resultSet.next()) {
                int orderID = resultSet.getInt("Order ID");
                String name = resultSet.getString("Name");
```

```
                int price = resultSet.getInt("Price");
                int weight = resultSet.getInt("Weight");
                int servingSize = resultSet.getInt("Serving Size");

                Meal meal = new Meal(mealID, orderID, name, price, weight,
servingSize);
                return new Result.Success<>(meal);
            }
        } catch (SQLException e){
            throw new IllegalStateException("An unexpected error occurred while
collecting meal from the database.");
        }

        return new Result.Failure<>(new Error.RecordNotFound(String.valueOf(mealID)));
    }

    public Optional<Error> deleteMeal(int mealID) {
        final String sql = """
                DELETE FROM \"Meal\"
                WHERE \"Meal ID\" = ?
                """;

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setInt(1, mealID);

            int rowCount = pstmt.executeUpdate();
            if (rowCount == 0) {
                return Optional.of(new Error.RecordNotFound(String.valueOf(mealID)));
            }
        } catch (SQLException e) {
            throw new IllegalStateException("An unexpected error occurred while
deleting the meal with ID '" +
                mealID + "' from the database.");
        }

        return Optional.empty();
    }

    public Result<Error, Order> getOrder(int orderID) throws IllegalStateException {
        final String sql = """
                SELECT * FROM \"Order\"
                WHERE \"Order ID\" = ?
                """;

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setInt(1, orderID);
            ResultSet resultSet = pstmt.executeQuery();

            if (resultSet.next()) {
                String orderDate = resultSet.getTimestamp("Order Date").toString();
                String courierPhone = resultSet.getString("Courier Phone");
                String deliveryDate = resultSet.getTimestamp("Delivery
Date").toString();
                String clientEmail = resultSet.getString("Client Email");
                int rating = resultSet.getInt("Rating");
                String deliveryAddress = resultSet.getString("Delivery Address");

                Order order = new Order(orderID, orderDate, courierPhone, deliveryDate,
clientEmail, rating, deliveryAddress);
                return new Result.Success<>(order);
            }
```

```
        } catch (SQLException e){
            throw new IllegalStateException("An unexpected error occurred while
collecting order from the database.");
        }

        return new Result.Failure<>(new Error.RecordNotFound(String.valueOf(orderID)));
    }

    public Optional<Error> addOrder(Order order) {
        Timestamp orderDate, deliveryDate;
        try {
            orderDate = Timestamp.valueOf(order.orderDate());
            deliveryDate = Timestamp.valueOf(order.deliveryDate());
        } catch (IllegalArgumentException e) {
            return Optional.of(new Error.ValidationError("Wrong date format."));
        }

        if (order.rating() < 1 || order.rating() > 5) {
            return Optional.of(new Error.ValidationError("Wrong delivery address."));
        }

        if (!AddressValidator.isValidAddress(order.deliveryAddress())) {
            return Optional.of(new Error.ValidationError("Wrong delivery address."));
        }

        Predicate<Integer> isOrderExists = (Integer ID) -> {
            final String sql = "SELECT EXISTS(SELECT 1 FROM \"Order\" WHERE \"Order
ID\" = ?)";

            try (PreparedStatement pstmt = connection.prepareStatement(sql,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY)) {
                pstmt.setInt(1, ID.intValue());
                try (ResultSet resultSet = pstmt.executeQuery()) {
                    if (resultSet.next()) {
                        return resultSet.getBoolean(1);
                    }
                }
            } catch (SQLException e) {
                throw new IllegalStateException("An unexpected error occurred while
checking the availability of the order in the database.");
            }
            return false;
        };

        if (isOrderExists.test(Integer.valueOf(order.orderID()))) {
            return Optional.of(new
Error.DuplicateKeyError(String.valueOf(order.orderID())));
        }

        switch (getCourier(order.courierPhone())) {
            case Result.Failure<Error, ?> failure -> {
                if (failure.error() instanceof Error.RecordNotFound _) {
                    return Optional.of(new Error.ForeignKeyConstraintError("Courier
Phone", order.courierPhone()));
                }
                else {
                    return Optional.of(new Error.UnknownError());
                }
            }
            case Result.Success<?,Courier> success -> {
                break;
```

```
            }
        }

        switch (getClient(order.clientEmail())) {
            case Result.Failure<Error, ?> failure -> {
                if (failure.error() instanceof Error.RecordNotFound _) {
                    return Optional.of(new Error.ForeignKeyConstraintError("Client
Email", order.clientEmail()));
                }
                else {
                    return Optional.of(new Error.UnknownError());
                }
            }
            case Result.Success<?,Client> success -> {
                break;
            }
        }

        final String sql = "INSERT INTO \"Order\"(\"Order ID\", \"Order Date\",
\"Courier Phone\", \"Delivery Date\", \"Client Email\", \"Rating\", \"Delivery
Address\") VALUES(?, ?, ?, ?, ?, ?, ?)";

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setInt(1, order.orderID());
            pstmt.setTimestamp(2, orderDate);
            pstmt.setString(3, order.courierPhone());
            pstmt.setTimestamp(4, deliveryDate);
            pstmt.setString(5, order.clientEmail());
            pstmt.setInt(6, order.rating());
            pstmt.setString(7, order.deliveryAddress());

            if (pstmt.executeUpdate() == 0) {
                Optional.of(new Error.InsertError("Meal"));
            }
        } catch (SQLException e) {
            throw new IllegalStateException("An unexpected error occurred while adding
an order to the database.");
        }

        return Optional.empty();
    }

    public ArrayList<Order> getAllOrders() {
        ArrayList<Order> orders = new ArrayList<>();
        final String sql = "SELECT * FROM \"Order\"";

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            ResultSet resultSet = pstmt.executeQuery();

            while (resultSet.next()) {
                int orderID = resultSet.getInt("Order ID");
                String orderDate = resultSet.getTimestamp("Order Date").toString();
                String courierPhone = resultSet.getString("Courier Phone");
                String deliveryDate = resultSet.getTimestamp("Delivery
Date").toString();
                String clientEmail = resultSet.getString("Client Email");
                int rating = resultSet.getInt("Rating");
                String deliveryAddress = resultSet.getString("Delivery Address");

                Order order = new Order(orderID, orderDate, courierPhone, deliveryDate,
clientEmail, rating, deliveryAddress);
```

```
                orders.add(order);
            }
        } catch (SQLException e){
            throw new IllegalStateException("An unexpected error occurred while
collecting orders from the database.");
        }

        return orders;
    }

    public Optional<Error> updateOrder(Order order) {
        Timestamp deliveryDate;
        try {
            deliveryDate = Timestamp.valueOf(order.deliveryDate());
        } catch (IllegalArgumentException e) {
            return Optional.of(new Error.ValidationError("Wrong date format."));
        }

        if (order.rating() < 1 || order.rating() > 5) {
            return Optional.of(new Error.ValidationError("Wrong rating."));
        }

        if (!AddressValidator.isValidAddress(order.deliveryAddress())) {
            return Optional.of(new Error.ValidationError("Wrong delivery address."));
        }

        final String sql = "UPDATE \"Order\" SET \"Delivery Date\" = COALESCE(?,
\"Delivery Date\"), \"Rating\" = COALESCE(?, \"Rating\"), \"Delivery Address\" =
COALESCE(?, \"Delivery Address\") WHERE \"Order ID\" = ?";

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setTimestamp(1, deliveryDate);
            pstmt.setInt(2, order.rating());
            pstmt.setString(3, order.deliveryAddress());
            pstmt.setInt(4, order.orderID());

            int rowCount = pstmt.executeUpdate();
            if (rowCount == 0) {
                return Optional.of(new
Error.RecordNotFound(String.valueOf(order.orderID())));
            }
        } catch (SQLException e) {
            throw new IllegalStateException("An unexpected error occurred while
updating the order with ID '" +
                order.orderID() + "' from the database.");
        }

        return Optional.empty();
    }

    public Optional<Error> deleteOrder(int orderID) {
        final String sql = """
                DELETE FROM \"Order\"
                WHERE \"Order ID\" = ?
                """;

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setInt(1, orderID);

            int rowCount = pstmt.executeUpdate();
            if (rowCount == 0) {
```

```
                return Optional.of(new Error.RecordNotFound(String.valueOf(orderID)));
            }
        } catch (SQLException e) {
            throw new IllegalStateException("An unexpected error occurred while
deleting the order with ID '" +
                orderID + "' from the database.");
        }

        return Optional.empty();
    }

    public Result<Error, ClientAnalytics> fetchClientAnalytics(ClientFilterParameters
parameters) {
        Timestamp startOrderDate;
        try {
            startOrderDate = Timestamp.valueOf(parameters.getOrderStartDate());
        } catch (IllegalArgumentException _) {
            return new Result.Failure<>(new Error.ValidationError("Wrong date
format."));
        }

        final String sql = """
                SELECT
                    c."Name" AS client_name,
                    COUNT(DISTINCT o."Order ID") AS order_count,
                    SUM(m."Price") AS total_spent
                FROM "Client" c
                JOIN "Order" o ON c."Email" = o."Client Email"
                JOIN "Meal" m ON o."Order ID" = m."Order ID"
                WHERE o."Order Date" >= ?
                AND m."Price" <= ?
                AND c."Email" LIKE ?
                GROUP BY c."Email", c."Name"
                ORDER BY total_spent DESC;
                """;

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setTimestamp(1, startOrderDate);
            pstmt.setInt(2, parameters.getMaxMealPrice());
            pstmt.setString(3, parameters.getEmail());

            SQLQueryRuntime.beginScope();
            ResultSet resultSet = pstmt.executeQuery();

            if (resultSet.next()) {
                String clientName = resultSet.getString("client_name");
                int orderCount = resultSet.getInt("order_count");
                int totalSpent = resultSet.getInt("total_spent");
                SQLQueryRuntime.endScope();

                ClientAnalytics analytics = new ClientAnalytics(clientName, orderCount,
totalSpent);
                return new Result.Success<>(analytics);
            }
        } catch (SQLException e) {
            throw new IllegalStateException("An unexpected error occurred while
fetching client analytics from the database.");
        }

        return new Result.Failure<>(new Error.RecordNotFound(""));
    }
```

```
    public Result<Error, ArrayList<CourierAnalytics>>
fetchCourierAnalytics(CourierFilterParameters parameters) {
        Date startDeliveryDate;
        try {
            startDeliveryDate = Date.valueOf(parameters.getStartDeliveryDate());
        } catch (IllegalArgumentException _) {
            return new Result.Failure<>(new Error.ValidationError("Wrong date
format."));
        }

        if (parameters.getMinRating() < 1 || parameters.getMinRating() > 5) {
            return new Result.Failure<>(new Error.ValidationError("Wrong rating."));
        }

        final String sql = """
                SELECT
                    co."Name" AS courier_name,
                    co."Phone",
                    AVG(o."Rating") AS average_rating,
                    MAX(o."Delivery Date") AS last_delivery_date,
                    MIN(o."Order Date") AS first_order_date
                FROM "Courier" co
                JOIN "Order" o ON co."Phone" = o."Courier Phone"
                JOIN "Meal" m ON o."Order ID" = m."Order ID"
                WHERE
                    o."Delivery Date" >= ? AND
                    o."Rating" >= ?
                GROUP BY
                    co."Name",
                    co."Phone",
                    co."Transport"
                ORDER BY average_rating DESC;
                """;

        ArrayList<CourierAnalytics> couriers = new ArrayList<>();
        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setDate(1, startDeliveryDate);
            pstmt.setInt(2, parameters.getMinRating());

            SQLQueryRuntime.beginScope();
            ResultSet resultSet = pstmt.executeQuery();

            while (resultSet.next()) {
                String courierName = resultSet.getString("courier_name");
                String phone = resultSet.getString("Phone");
                float averageRating = resultSet.getFloat("average_rating");
                String lastDeliveryDate =
resultSet.getTimestamp("last_delivery_date").toString();
                String firstOrderDate =
resultSet.getTimestamp("first_order_date").toString();

                CourierAnalytics analytics = new CourierAnalytics(courierName, phone,
averageRating, lastDeliveryDate, firstOrderDate);
                couriers.add(analytics);
            }

            SQLQueryRuntime.endScope();

        } catch (SQLException e) {
```

```
            throw new IllegalStateException("An unexpected error occurred while
fetching courier analytics from the database.");
        }

        return new Result.Success<>(couriers);
    }
}
```

```
package model.connector;

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

class ConnectionProperties {
    public static final String USER = "postgres";
    public static final String PASSWORD = "root";
}

public class DatabaseConnector {
    private Connection psqlConnection;

    public Connection connect() throws IllegalStateException
    {
        if (psqlConnection != null)
            return psqlConnection;

        try {
            DriverManager.registerDriver(new org.postgresql.Driver());

            Properties properties = new Properties();
            properties.setProperty("user", ConnectionProperties.USER);
            properties.setProperty("password", ConnectionProperties.PASSWORD);

            String url = "jdbc:postgresql://localhost:5432/postgres";
            psqlConnection = DriverManager.getConnection(url, properties);
            System.out.println("Successfully connected to the database.");

            return psqlConnection;

        } catch (NullPointerException e) {
            throw new IllegalStateException("Failed to load PostgreSQL JDBC driver.");
        } catch (SQLException e) {
            throw new IllegalStateException("Failed to connect to the database.");
        }
    }
}
```

```
package model.validation;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class AddressValidator {
    private static final String ADDRESS_REGEX = "^[A-Za-z]+(?: [A-Za-z]+)* \\d+$";
    private static final Pattern pattern = Pattern.compile(ADDRESS_REGEX);

    public static boolean isValidAddress(String address) {
        if (address == null) {
            return false;
        }
```

```
        Matcher matcher = pattern.matcher(address);
        return matcher.matches();
    }
}
```

```
package model.validation;

import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class EmailValidator {
    private static final String EMAIL_REGEX = "^[\\w\\.\\-]+@[a-zA-Z\\d\\-]+\\.[a-zA-
Z]{2,}(?:\\.[a-zA-Z]{2,})?$";
    private static final Pattern pattern = Pattern.compile(EMAIL_REGEX);

    public static boolean isValidEmail(String email) {
        if (email == null) {
            return false;
        }
        Matcher matcher = pattern.matcher(email);
        return matcher.matches();
    }
}
```

```
package model.validation;

import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class MealNameValidator {
    private static final String NAME_REGEX = "^[A-Za-z]+([- ][A-Za-z]+)*$";
    private static final Pattern pattern = Pattern.compile(NAME_REGEX);

    public static boolean isValidName(String name) {
        if (name == null) {
            return false;
        }
        Matcher matcher = pattern.matcher(name);
        return matcher.matches();
    }
}
```

```
package model.validation;

import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class NameValidator {
    private static final String NAME_REGEX = "^[A-Z][a-z]+(?:-[A-Z][a-z]+)?\\s[A-Z][a-
z]+(?:-[A-Z][a-z]+)?$";
    private static final Pattern pattern = Pattern.compile(NAME_REGEX);

    public static boolean isValidName(String name) {
        if (name == null) {
            return false;
        }
        Matcher matcher = pattern.matcher(name);
        return matcher.matches();
    }
}
```

```
package model.validation;
```

```java
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class PhoneNumberValidator {
    private static final String PHONE_REGEX = "^\\d{10}$";
    private static final Pattern pattern = Pattern.compile(PHONE_REGEX);

    public static boolean isValidPhoneNumber(String phoneNumber) {
        if (phoneNumber == null) {
            return false;
        }
        Matcher matcher = pattern.matcher(phoneNumber);
        return matcher.matches();
    }
}
```

Короткий опис функцій модуля

Модуль “Model” є частиною архітектури MVC і забезпечує взаємодію з базою даних. Він містить методи для управління записами клієнтів, кур'єрів, замовлень та страв, включаючи додавання, отримання, оновлення та видалення даних. Основна мета — забезпечити безпечний доступ до даних, включаючи валідацію введених даних і обробку помилок.

- **addClient()** - додає нового клієнта в базу даних після перевірки правильності його даних, таких як ім'я, електронна пошта та номер телефону.
- **getAllClients()** - повертає список усіх клієнтів з бази даних, зібраних із таблиці Client.
- **getClient()** - шукає клієнта в базі за електронною поштою та повертає відповідний запис або помилку, якщо клієнт не знайдений.
- **updateClient()** - оновлює інформацію про клієнта у базі даних, якщо його дані успішно пройшли валідацію.
- **deleteClient()** - видаляє клієнта з бази за його електронною поштою, якщо такий клієнт існує.
- **getClientWithMostOrders()** - повертає клієнта, який зробив найбільшу кількість замовлень, та кількість його замовлень.
- **generateRandomClients()** - генерує випадкових клієнтів і додає їх у базу даних у зазначеній кількості.
- **addCourier()** - додає нового кур'єра в базу даних після перевірки правильності його даних.
- **getAllCouriers()** - повертає список усіх кур'єрів з бази даних.
- **getCourier()** - шукає кур'єра за номером телефону та повертає відповідний запис або помилку.

- **updateCourier()** - оновлює інформацію про кур'єра у базі, якщо його дані успішно пройшли валідацію.
- **deleteCourier()** - видаляє кур'єра з бази даних за номером телефону, якщо запис існує.
- **getCouriersWithMostOrders()** - повертає список кур'єрів, які мають найбільшу кількість доставлених замовлень, обмежений зазначеною кількістю записів.
- **generateRandomCouriers()** - генерує випадкових кур'єрів і додає їх у базу даних у кількості, вказаній параметром. Імена, прізвища та транспортні засоби кур'єрів вибираються випадковим чином з попередньо визначених масивів, а номери телефонів генеруються автоматично.
- **addMeal()** - додає нову страву в базу даних, після перевірки її унікальності та валідності назви. У випадку, якщо така страва вже існує або не знайдено відповідне замовлення, повертається відповідна помилка.
- **getAllMeals()** - повертає список усіх страв з бази даних, витягуючи інформацію про кожну страву, таку як ID страви, ID замовлення, назва, ціна, вага та розмір порції.
- **updateMeal()** - оновлює існуючу страву у базі даних. Якщо страва не знайдена або надані некоректні дані (наприклад, невірна назва), повертається відповідна помилка.
- **getMeal()** - повертає інформацію про страву за її ID. Якщо страва не знайдена, повертається помилка про відсутність запису.
- **deleteMeal()** - видаляє страву з бази даних за її ID. Якщо страва не знайдена, повертається помилка про відсутність запису.
- **getOrder()** - повертає інформацію про замовлення за його ID, включаючи дату замовлення, номер телефону кур'єра, дату доставки, електронну пошту клієнта, оцінку та адресу доставки.
- **addOrder()** - додає нове замовлення до бази даних після перевірки валідності дат замовлення і доставки, рейтингу, правильності адреси доставки та наявності замовлення з таким же ID. Перевіряє існування кур'єра та клієнта. Якщо валідація пройдена, додає замовлення.
- **getAllOrders()** - повертає список усіх замовлень з бази даних, витягуючи інформацію про ID замовлення, дати замовлення та доставки, номер телефону кур'єра, електронну пошту клієнта, рейтинг та адресу доставки.
- **updateOrder()** - оновлює замовлення в базі даних за його ID, змінюючи дату доставки, рейтинг або адресу доставки. Повертає помилку, якщо замовлення не знайдено або виникають проблеми з валідацією даних.
- **deleteOrder()** - видаляє замовлення з бази даних за його ID. Якщо замовлення не знайдено, повертає відповідну помилку.
- **fetchClientAnalytics()** - повертає аналітику по клієнтам, які здійснили замовлення після вказаної дати, з урахуванням максимальної ціни страви і

фільтрації за email. Витягує кількість замовлень та загальну суму витрат кожного клієнта.

- **fetchCourierAnalytics()** - повертає аналітику по кур'єрам, які виконували доставку після вказаної дати і мають рейтинг не нижче заданого. Витягує середній рейтинг, дату останньої доставки і дату першого замовлення для кожного кур'єра.
- **connect()** - встановлює з'єднання з базою даних PostgreSQL. Якщо з'єднання вже існує, повертає його. У разі невдачі під час завантаження драйвера або підключення до бази даних, генерує відповідні помилки. При успішному підключенні виводить повідомлення про успішне з'єднання.
- **isValidAddress()** - перевіряє, чи є передана адреса валідною, використовуючи регулярний вираз. Метод повертає true, якщо адреса відповідає формату (слова, за якими йде номер будинку), і false, якщо адреса є null або не відповідає вказаному шаблону.
- **isValidEmail()** - перевіряє, чи є передана електронна адреса валідною, використовуючи регулярний вираз. Метод повертає true, якщо адреса відповідає формату (локальна частина, символ "@" та домен), і false, якщо адреса є null або не відповідає вказаному шаблону.
- **isValidName()** – у випадку валідації страви, перевіряє, чи є передане ім'я страви валідним, у випадку імені клієнта – перевіряє, чи передане ім’я людини є валідним, використовуючи регулярний вираз. Метод повертає true, якщо ім'я відповідає формату і false, якщо ім'я є null або не відповідає вказаному шаблону.
- **isValidPhoneNumber()** - перевіряє, чи є переданий номер телефону валідним, використовуючи регулярний вираз. Метод повертає true, якщо номер телефону складається з 10 цифр, і false, якщо номер є null або не відповідає вказаному шаблону.