

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

**Лабораторна робота №2**  
з дисципліни  
**“Бази даних та засоби управління”**

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

**Київ 2024**

## Постановка задачі

Завдання роботи полягає у наступному:

1. Перетворити модуль “Модель” з шаблону MVC РГР у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.
4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

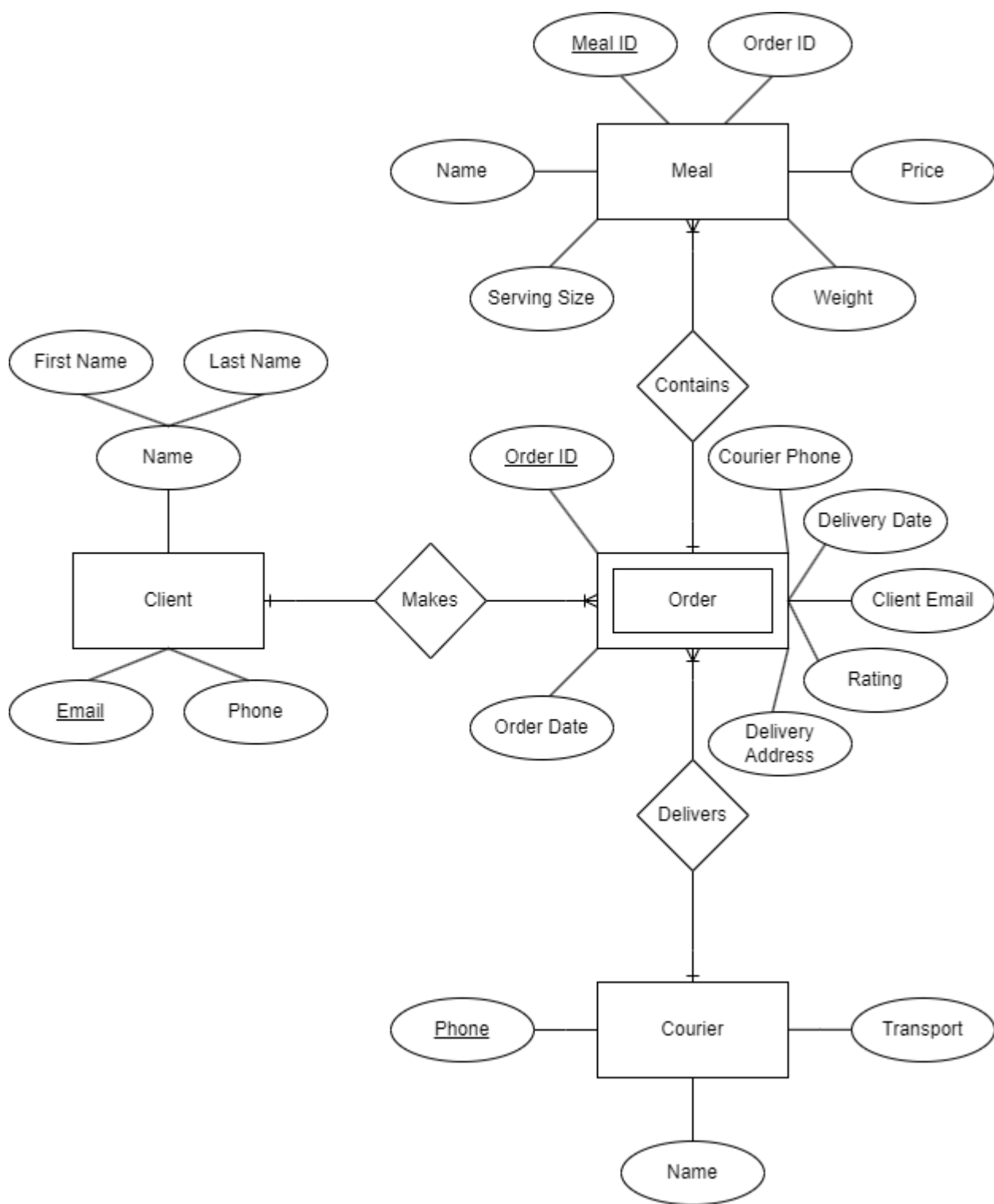
Варіант:

15	Hash, BRIN	before delete, update
----	------------	-----------------------

Посилання на репозиторій GitHub:

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

## 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



## Завдання 1

**Об'єктно-реляційна проекція (ORM)** — це технологія, яка забезпечує автоматичне відображення об'єктів програмного коду на реляційні таблиці бази даних та навпаки. Її головною метою є усунення необхідності безпосереднього написання SQL-запитів для роботи з базами даних, що спрощує розробку додатків та забезпечує прозору інтеграцію об'єктно-орієнтованого програмування (ООП) із реляційними базами даних.

Вибір Jakarta Persistence як стандарту ORM був логічним кроком, оскільки він надає потужні інструменти для роботи з реляційними базами даних у Java, дозволяючи зручно інтегрувати об'єктно-орієнтовані моделі з реляційними базами даних, зберігаючи при цьому чистоту та зручність коду.

*Код класу **Client** для відображення сутності:*

```
@Entity
@Table(name = "\"Client\"")
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Data
public final class Client {

    @Id
    @Column(name = "\"Email\"", nullable = false, unique = true,
length = 32)
    private String email;

    @Column(name = "\"Name\"", nullable = false, length = 255)
    private String name;

    @Column(name = "\"Phone\"", nullable = false, length = 10)
    private String phone;

}
```

*Код класу **Courier** для відображення сутності:*

```
@Entity
@Table(name = "\"Courier\"")
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Data
public final class Courier {
```

```

    @Id
    @Column(name = "\"Phone\"", nullable = false, unique = true,
length = 10)
    private String phone;

    @Column(name = "\"Name\"", nullable = false, length = 25)
    private String name;

    @Column(name = "\"Transport\"", nullable = false, length = 25)
    private String transport;
}

```

*Код класу **Order** для відображення сутності:*

```

@Entity
@Table(name = "\"Order\"")
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Data
public final class Order {

    @Id
    @Column(name = "\"Order ID\"")
    private Long orderID;

    @ManyToOne
    @JoinColumn(name = "\"Courier Phone\"", referencedColumnName =
 "\"Phone\"", nullable = false)
    private Courier courier;

    @ManyToOne
    @JoinColumn(name = "\"Client Email\"", referencedColumnName =
 "\"Email\"", nullable = false)
    private Client client;

    @Column(name = "\"Order Date\"", nullable = false)
    private LocalDateTime orderDate;

    @Column(name = "\"Delivery Date\"", nullable = false)
    private LocalDateTime deliveryDate;

    @Column(name = "\"Rating\"", nullable = false)
    private Integer rating;
}

```

```

        @Column(name = "\"Delivery Address\"", nullable = false, length
= 50)
        private String deliveryAddress;
    }

```

Код класу *Meal* для відображення сутності:

```

@Entity
@Table(name = "\"Meal\"")
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Data
public final class Meal {

    @Id
    @Column(name = "\"Meal ID\"")
    private Long mealID;

    @ManyToOne
    @JoinColumn(name = "\"Order ID\"", referencedColumnName =
"\"Order ID\"", nullable = false)
    private Order order;

    @Column(name = "\"Name\"", nullable = false, length = 25)
    private String name;

    @Column(name = "\"Price\"", nullable = false)
    private Integer price;

    @Column(name = "\"Weight\"", nullable = false)
    private Integer weight;

    @Column(name = "\"Serving Size\"", nullable = false)
    private Integer servingSize;
}

```



*Додавання клієнта:*

```
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
>> 1
Enter client first name: Jayden
Enter client last name: Ward
Enter client email: jaydrd@example.com
Enter client phone number: 0841834942
Client successfully added: Jayden Ward
Press any key to continue.
```

*Перегляд інформації про всіх клієнтів:*

```
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
>> 2
+-----+-----+-----+
| 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        |
```

*Оновлення інформації про клієнта:*

```
=====
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
>> 3
Enter the client's email: jaydrd@example.com
Updating details for client: Jayden Ward
Current Name: Jayden Ward
Enter new name (or press Enter to keep current): Calvin Nelson
Current Phone: 0841834942
Enter new phone (or press Enter to keep current): 2537052768
Client details updated successfully.
Press any key to continue.
```

*Видалення клієнта:*

```
=====
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: jaydrd@example.com
Client successfully deleted.
Press any key to continue.
```

*Отримання клієнта із найбільшою кількістю замовлень:*

```
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
>> 5

*****
CLIENT WITH MOST ORDERS
+-----+-----+-----+-----+
| Order Count | Email                               | Name           | Phone         |
| 10000       | ivy.wang1776813@rnd.com           | Ivy Wang       | 5609235039    |
+-----+-----+-----+-----+
Query runtime: 1028 msec.
Press any key to continue.
```

*Генерування рандомізованих даних:*

```
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: 10000
Successfully generated 10000 clients.
Press any key to continue.
```

*Додавання кур'єра:*

```
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
>> 1
Enter courier phone: 5209389643
Enter courier first name: Lincoln
Enter courier last name: Gray
Enter courier transport: Bicycle
Courier successfully added: Lincoln Gray
Press any key to continue.
```

*Перегляд інформації про всіх кур'єрів:*

```
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
>> 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   |
+-----+-----+-----+
```

*Оновлення інформації про кур'єра:*

```
=====
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
>> 3
Enter the courier's phone: 5209389643
Updating details for courier: Lincoln Gray
Current Name: Lincoln Gray
Enter new name (or press Enter to keep current): Charles Miller
Current Transport: Bicycle
Enter new transport (or press Enter to keep current): Motorbike
Courier details updated successfully.
Press any key to continue.
```

*Видалення кур'єра:*

```
=====
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
>> 4
Enter the courier's phone: 5209389643
Courier successfully deleted.
Press any key to continue.
```

*Отримання 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: 10

*****
TOP 10 COURIERS WITH THE MOST ORDERS
+-----+-----+-----+
| Name           | Phone       | Order Count |
| David Johnson  | 1358140310  | 10000       |
| Bob Garcia     | 7248934309  | 3198        |
| Bob Garcia     | 1509098423  | 2001        |
| Bob Brown      | 9008835074  | 1001        |
| Bob Smith      | 9551962819  | 1000        |
| David Williams | 5335068056  | 801         |
| Mike Johnson   | 9876543210  | 613         |
| Liam Miller    | 7654321098  | 612         |
| Emma Brown     | 8765432109  | 612         |
| Charlie Brown  | 1886502364  | 611         |
+-----+-----+-----+
Query runtime: 1214 msec.
Press any key to continue.
```

*Генерування рандомізованих даних:*

```
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
>> 6
Enter the number of records: 50000
Successfully generated 50000 couriers.
Press any key to continue.
```

Додавання страви:

```
MEALS TABLE MANAGEMENT

[1] Add Meal
[2] View Meals
[3] Update Meal
[4] Delete Meal
[5] Back to Main Menu
>> 1
Enter meal ID: 0
Enter meal order ID: 0
Enter meal name: Churros
Enter meal price: 450
Enter meal weight: 500
Enter meal serving size: 1
Meal successfully added: 0
Press any key to continue.
```

Перегляд інформації про всі страви:

```
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
30	31	Frittata	40	40	1
31	32	Ceviche	41	41	2
32	33	Clams Casino	42	42	3
33	34	Lobster Roll	43	43	4
34	35	Risotto	44	44	5
35	36	Mac and Cheese	45	45	1
36	37	Chili	46	46	2
37	38	Beef Wellington	47	47	3
38	39	Tiramisu	48	48	4
39	40	Cheesecake	49	49	5
40	41	Chocolate Cake	50	50	1
41	42	Ice Cream	51	51	2
42	43	Panna Cotta	52	52	3
43	44	Baklava	53	53	4
44	45	Mousse	54	54	5
45	46	Creme Brulee	55	55	1

*Оновлення інформації про страву:*

```
=====
MEALS TABLE MANAGEMENT
=====
[1] Add Meal
[2] View Meals
[3] Update Meal
[4] Delete Meal
[5] Back to Main Menu
>> 3
Please enter the meal's ID: 0
Updating details for meal: Churros
Current Name: Churros
Enter new name (or press Enter to keep current): Ice Cream
Current Price: 450
Enter new price (or press Enter to keep current): 30
Current Weight: 450
Enter new weight (or press Enter to keep current): 100
Current Serving Size: 1
Enter new serving size (or press Enter to keep current): 1
Meal details updated successfully.
Press any key to continue.
```

*Видалення страви:*

```
=====
MEALS TABLE MANAGEMENT
=====
[1] Add Meal
[2] View Meals
[3] Update Meal
[4] Delete Meal
[5] Back to Main Menu
>> 4
Please enter the meal's ID: 0
Meal successfully deleted.
Press any key to continue.
```



Додавання замовлення:

```
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
>> 1
Enter order ID: 711007
Enter order date (yyyy-mm-dd hh:mm): 2024-11-01 11:00
Enter order courier phone: 8181995442
Enter order delivery date (yyyy-mm-dd hh:mm): 2024-12-30 10:00
Enter order client email: john.doe@gmail.com
Enter order rating between 1 and 5: 5
Enter order delivery address (street ?): Address 1
Order successfully added: 711007
Press any key to continue.
```

Перегляд інформації про всі замовлення:

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	8765432109	2024-10-05 18:45	jane.smith@yahoo.com	3	Nova 12
1003	2024-10-04 14:31	9876543210	2024-11-06 15:20	john.doe@gmail.com	3	Nova 16
1002	2024-10-04 14:30	9876543210	2024-10-05 16:20	john.doe@gmail.com	4	Nova 14
1111	2024-11-01 12:00	1886502364	2024-12-31 10:30	david.brown@example.com	5	Nova 33
5001	2024-10-29 20:56	1509098423	2024-10-30 20:56	zoe.ali7105635@d.com	2	Address 5001
2	2024-11-17 20:34	9551962819	2024-11-18 20:34	ava.zhu6241195@svc.com	3	Address 2
3	2024-11-16 20:34	9551962819	2024-11-17 20:34	ava.zhu6241195@svc.com	4	Address 3
4	2024-11-15 20:34	9551962819	2024-11-16 20:34	ava.zhu6241195@svc.com	5	Address 4
5	2024-11-14 20:34	9551962819	2024-11-15 20:34	ava.zhu6241195@svc.com	1	Address 5
6	2024-11-13 20:34	9551962819	2024-11-14 20:34	ava.zhu6241195@svc.com	2	Address 6
7	2024-11-12 20:34	9551962819	2024-11-13 20:34	ava.zhu6241195@svc.com	3	Address 7
8	2024-11-11 20:34	9551962819	2024-11-12 20:34	ava.zhu6241195@svc.com	4	Address 8
9	2024-11-10 20:34	9551962819	2024-11-11 20:34	ava.zhu6241195@svc.com	5	Address 9
10	2024-11-09 20:34	9551962819	2024-11-10 20:34	ava.zhu6241195@svc.com	1	Address 10
11	2024-11-08 20:34	9551962819	2024-11-09 20:34	ava.zhu6241195@svc.com	2	Address 11
12	2024-11-07 20:34	9551962819	2024-11-08 20:34	ava.zhu6241195@svc.com	3	Address 12
13	2024-11-06 20:34	9551962819	2024-11-07 20:34	ava.zhu6241195@svc.com	4	Address 13
14	2024-11-05 20:34	9551962819	2024-11-06 20:34	ava.zhu6241195@svc.com	5	Address 14
15	2024-11-04 20:34	9551962819	2024-11-05 20:34	ava.zhu6241195@svc.com	1	Address 15
16	2024-11-03 20:34	9551962819	2024-11-04 20:34	ava.zhu6241195@svc.com	2	Address 16
17	2024-11-02 20:34	9551962819	2024-11-03 20:34	ava.zhu6241195@svc.com	3	Address 17
18	2024-11-01 20:34	9551962819	2024-11-02 20:34	ava.zhu6241195@svc.com	4	Address 18
19	2024-10-31 20:34	9551962819	2024-11-01 20:34	ava.zhu6241195@svc.com	5	Address 19
20	2024-10-30 20:34	9551962819	2024-10-31 20:34	ava.zhu6241195@svc.com	1	Address 20
21	2024-10-29 20:34	9551962819	2024-10-30 20:34	ava.zhu6241195@svc.com	2	Address 21
22	2024-10-28 20:34	9551962819	2024-10-29 20:34	ava.zhu6241195@svc.com	3	Address 22
23	2024-10-27 20:34	9551962819	2024-10-28 20:34	ava.zhu6241195@svc.com	4	Address 23
24	2024-10-26 20:34	9551962819	2024-10-27 20:34	ava.zhu6241195@svc.com	5	Address 24
25	2024-10-25 20:34	9551962819	2024-10-26 20:34	ava.zhu6241195@svc.com	1	Address 25

### Оновлення інформації про замовлення:

```
=====
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
>> 3
Please enter the order's ID: 711007
Updating details for order: 711007
Current Delivery Date: 2024-12-30 10:00
Enter new delivery date (yyyy-mm-dd hh:mm) (or press Enter to keep current): 2024-12-31 14:30
Current Rating: 5
Enter new rating between 1 and 5 (or press Enter to keep current): 4
Current Delivery Address: Address 1
Enter order delivery address (street ?) (or press Enter to keep current): Address 2
Order details updated successfully.
Press any key to continue.
```

### Видалення замовлення:

```
=====
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
>> 4
Please enter the order's ID: 711007
Order successfully deleted.
Press any key to continue.
```

### Отримання клієнтської аналітики:

```
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
>> 5
Enter the start date of order (yyyy-mm-dd hh:mm): 2024-01-01 01:00
Enter maximum price of meal: 1500
Enter client's email: john.doe@gmail.com
+-----+-----+-----+
| Client Name      | Order Count | Total Spent |
+-----+-----+-----+
| John Doe         | 2           | 693         |
+-----+-----+-----+
Query runtime: 202 msec.
Press any key to continue.
```

### Отримання кур'єрської аналітики:

```
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
>> 6
Enter the start date of delivery (yyyy-mm-dd): 2042-12-31
Enter minimum rating: 5
+-----+-----+-----+-----+-----+
| Courier Name      | Phone       | Average Rating | Last Delivery Date | First Order Date |
+-----+-----+-----+-----+-----+
| Grace Williams   | 0440386399 | 5.0            | 2042-12-31 00:00  | 2037-01-08 00:00 |
| David Jones      | 2993336282 | 5.0            | 2042-12-31 00:00  | 2034-03-31 00:00 |
| Eve Miller       | 3084054160 | 5.0            | 2042-12-31 00:00  | 2031-12-10 00:00 |
| Eve Jones        | 3786350483 | 5.0            | 2043-01-01 00:00  | 2042-01-09 00:00 |
| Grace Miller     | 3834849930 | 5.0            | 2042-12-31 00:00  | 2027-08-07 00:00 |
| Grace Garcia     | 3873844878 | 5.0            | 2042-12-31 00:00  | 2040-06-09 00:00 |
| Charlie Garcia   | 5556960891 | 5.0            | 2042-12-31 00:00  | 2027-10-15 00:00 |
| Eve Williams     | 6115764574 | 5.0            | 2042-12-31 00:00  | 2041-08-05 00:00 |
| Grace Garcia     | 6627027884 | 5.0            | 2042-12-31 00:00  | 2025-11-12 00:00 |
| Eve Garcia       | 7141968337 | 5.0            | 2042-12-31 00:00  | 2024-11-11 00:00 |
| Bob Smith        | 7151757415 | 5.0            | 2042-12-31 00:00  | 2024-01-14 00:00 |
| Alice Brown      | 7164478717 | 5.0            | 2042-12-31 00:00  | 2033-07-20 00:00 |
| Alice Jones      | 7460277679 | 5.0            | 2042-12-31 00:00  | 2030-02-19 00:00 |
| David Miller     | 8076883911 | 5.0            | 2043-01-01 00:00  | 2024-12-09 00:00 |
| Charlie Johnson  | 8902922642 | 5.0            | 2042-12-31 00:00  | 2040-08-23 00:00 |
+-----+-----+-----+-----+-----+
Query runtime: 602 msec.
Press any key to continue.
```

## Завдання 2

Індекси в PostgreSQL дозволяють значно покращити продуктивність запитів, особливо при обробці великих обсягів даних. Індекс **HASH** генерує хеш-код для кожного значення стовпця, для якого створюється індекс. Потім ці хеш-коди використовуються для розподілу значень по різних слотах індексу. Коли виконується запит, PostgreSQL використовує хеш-функцію для обчислення хеш-коду значення, яке шукається, і порівнює його з хешами, що зберігаються в індексі. Це дозволяє здійснювати дуже швидкий пошук за точними значеннями, оскільки порівняння хеш-кодів є набагато швидшим за порівняння самих значень.

### *Переваги індексу HASH:*

1. Hash індекси дуже ефективні при виконанні запитів з точними порівняннями (оператор =), оскільки пошук хеш-коду є дуже швидким.
2. Для великих таблиць, де часто виконуються запити на точну відповідність значень, Hash індекси можуть значно покращити швидкість виконання запитів.

### *Недоліки індексу HASH:*

1. Hash індекси не можна використовувати для запитів, що включають діапазони значень (наприклад, BETWEEN, >, <), оскільки вони працюють лише з точними значеннями.
2. У порівнянні з індексами B-tree, Hash індекси не підтримують сортування даних.
3. Для хеш-індексів PostgreSQL не підтримує забезпечення унікальності значень в індексі так, як це робиться в B-tree індексах.
4. Hash індекси можуть працювати повільніше, якщо дані в таблиці мають нерівномірний розподіл, оскільки можуть виникати колізії хешів.

Індекс **BRIN** зберігає мінімальні та максимальні значення стовпців для блоків таблиці, що зменшує кількість даних, які потрібно сканувати під час виконання запитів. При цьому кожен блок таблиці зазвичай містить багато рядків, і індекс зберігає лише кілька значень для кожного блоку, а не для кожного окремого рядка.

### *Переваги індексу BRIN:*

1. BRIN індекси займають значно менше пам'яті, оскільки вони зберігають лише мінімальні та максимальні значення для блоків, а не для кожного рядка.

2. Запити з діапазовими умовами (BETWEEN, >=, <=, тощо) можуть бути значно швидшими за рахунок того, що BRIN індекс допомагає пропустити багато блоків даних, які не відповідають умові запиту.
3. BRIN особливо корисний для великих таблиць, де інші типи індексів, такі як B-tree, можуть бути неефективними через великі витрати пам'яті та часу на створення.


### Недоліки індексу BRIN:

1. BRIN індекси не підходять для запитів, які використовують точні значення (=), оскільки індекс працює з діапазонами значень блоків, а не з конкретними значеннями.
2. BRIN індекси працюють найкраще для впорядкованих даних, і їх ефективність значно знижується, якщо дані в таблиці не мають певної впорядкованості.

### Запит №1.

```
EXPLAIN ANALYZE
SELECT
```

```
    DATE_TRUNC('month', "Order Date") AS order_month,
    COUNT(*) AS total_orders,
    AVG("Rating") AS avg_rating,
    SUM(EXTRACT(EPOCH FROM "Delivery Date" - "Order Date") / 3600) AS
avg_delivery_time
FROM
    "Order"
WHERE
    "Order Date" BETWEEN '2024-01-01' AND '2030-12-31'
    AND "Rating" BETWEEN 1 AND 5
GROUP BY
    order_month
ORDER BY
    order_month ASC;
```

	QUERY PLAN	
	text	
1	GroupAggregate (cost=10000061977.79..10000068484.10 rows=7108 width=80) (actual time=149.697..282.175 rows=84 loops=1)	
2	Group Key: (date_trunc('month':text, "Order Date"))	
3	-> Sort (cost=10000061977.79..10000062775.53 rows=319096 width=28) (actual time=148.745..177.362 rows=319324 loops=1)	
4	Sort Key: (date_trunc('month':text, "Order Date"))	
5	Sort Method: external merge Disk: 13128kB	
6	-> Seq Scan on "Order" (cost=10000000000.00..10000025169.62 rows=319096 width=28) (actual time=0.010..86.021 rows=319324 loops=1)	
7	Filter: (("Order Date" >= '2024-01-01 00:00:00':timestamp without time zone) AND ("Order Date" <= '2030-12-31 00:00:00':timestamp without time zone) AND ("Rating" >= 1) AND ("Rating" ...	
8	Rows Removed by Filter: 391620	
9	Planning Time: 0.474 ms	
10	Execution Time: 283.830 ms	

*Примітка: аналіз виконання запити №1*

Створимо BRIN індекс для стовпця "Order Date": **CREATE INDEX idx\_order\_date\_brin ON "Order" USING BRIN ("Order Date");**

	QUERY PLAN
	text
6	-> Sort (cost=19385.20..19402.95 rows=7101 width=80) (actual time=80.047..80.051 rows=84 loops=3)
7	Sort Key: (date_trunc('month'::text, "Order Date"))
8	Sort Method: quicksort Memory: 36kB
9	Worker 0: Sort Method: quicksort Memory: 36kB
10	Worker 1: Sort Method: quicksort Memory: 36kB
11	-> Partial HashAggregate (cost=18824.44..18930.95 rows=7101 width=80) (actual time=79.989..80.019 rows=84 loops=3)
12	Group Key: date_trunc('month'::text, "Order Date")
13	Batches: 1 Memory Usage: 273kB
14	Worker 0: Batches: 1 Memory Usage: 273kB
15	Worker 1: Batches: 1 Memory Usage: 273kB
16	-> Parallel Bitmap Heap Scan on "Order" (cost=94.14..16503.27 rows=132638 width=28) (actual time=0.113..42.496 rows=106441 loops=3)
17	Recheck Cond: (("Order Date" >= '2024-01-01 00:00:00'::timestamp without time zone) AND ("Order Date" <= '2030-12-31 00:00:00'::timestamp without time z...
18	Rows Removed by Index Recheck: 130540
19	Filter: (("Rating" >= 1) AND ("Rating" <= 5))
20	Rows Removed by Filter: 0
21	Heap Blocks: lossy=3857
22	-> Bitmap Index Scan on idx_order_date_brin (cost=0.00..14.56 rows=710944 width=0) (actual time=0.265..0.265 rows=101530 loops=1)
23	Index Cond: (("Order Date" >= '2024-01-01 00:00:00'::timestamp without time zone) AND ("Order Date" <= '2030-12-31 00:00:00'::timestamp without time z...
24	Planning Time: 0.135 ms
25	Execution Time: 101.907 ms

*Примітка: аналіз виконання запиту №1 із використанням індексу*

В SQL запиті використовуються групування і агрегація за місяцями `DATE_TRUNC('month', "Order Date")`, що вимагає обробки великих обсягів даних та умова `"Order Date" BETWEEN '2024-01-01' AND '2030-12-31'`, яка дозволяє PostgreSQL використовувати BRIN для швидкої оцінки, які блоки даних підпадають під цей діапазон дат. Зниження часу виконання з 283 мс до 101 мс свідчить про високу ефективність BRIN індексу при роботі з впорядкованими даними.

## Запит №2.

```
EXPLAIN ANALYZE
SELECT
    DATE_TRUNC('week', "Delivery Date") AS delivery_week,
    COUNT(*) AS deliveries_count,
    AVG(EXTRACT(EPOCH FROM "Delivery Date" - "Order Date") / 3600) AS
avg_delivery_time,
    MIN("Rating") AS min_rating,
    MAX("Rating") AS max_rating
FROM
    "Order"
WHERE
    "Delivery Date" BETWEEN '2024-01-01' AND '2042-12-31'
    AND EXTRACT(DOW FROM "Delivery Date") IN (1, 2, 3, 4, 5)
GROUP BY
```

```

delivery_week
ORDER BY
delivery_week DESC;

```

	QUERY PLAN text
1	Finalize GroupAggregate (cost=10000018682.78..10000020753.27 rows=6546 width=56) (actual time=177.397..264.821 rows=992 loops=1)
2	Group Key: (date_trunc('week'::text, 'Delivery Date'))
3	-> Gather Merge (cost=10000018682.78..10000020458.70 rows=13092 width=56) (actual time=177.311..263.217 rows=2976 loops=1)
4	Workers Planned: 2
5	Workers Launched: 2
6	-> Partial GroupAggregate (cost=10000017682.75..10000017947.53 rows=6546 width=56) (actual time=149.840..221.862 rows=992 loops=3)
7	Group Key: (date_trunc('week'::text, 'Delivery Date'))
8	-> Sort (cost=10000017682.75..10000017701.26 rows=7404 width=28) (actual time=149.778..164.858 rows=169775 loops=3)
9	Sort Key: (date_trunc('week'::text, 'Delivery Date')) DESC
10	Sort Method: external merge Disk: 7448kB
11	Worker 0: Sort Method: external merge Disk: 5448kB
12	Worker 1: Sort Method: external merge Disk: 6080kB
13	-> Parallel Seq Scan on "Order" (cost=10000000000.00..10000017206.89 rows=7404 width=28) (actual time=0.015..100.318 rows=169775 loops=3)
14	Filter: (("Delivery Date" >= '2024-01-01 00:00:00'::timestamp without time zone) AND ("Delivery Date" <= '2042-12-31 00:00:00'::timestamp without time zone) AND (EXTRACT(dow FROM "Delivery Date") = ANY
15	Rows Removed by Filter: 67206
16	Planning Time: 0.116 ms
17	Execution Time: 266.423 ms

*Примітка: аналіз виконання запиту №2*

```

Створимо BRIN індекс для стовпця "Delivery Date": CREATE INDEX
idx_order_delivery_date_brin ON "Order" USING BRIN ("Delivery Date");

```

	QUERY PLAN text	
6	-> Sort (cost=17887.66..17904.05 rows=6553 width=56) (actual time=164.872..164.914 rows=992 loops=3)	
7	Sort Key: (date_trunc('week'::text, 'Delivery Date')) DESC	
8	Sort Method: quicksort Memory: 141kB	
9	Worker 0: Sort Method: quicksort Memory: 141kB	
10	Worker 1: Sort Method: quicksort Memory: 141kB	
11	-> Partial HashAggregate (cost=17373.98..17472.27 rows=6553 width=56) (actual time=164.426..164.665 rows=992 loops=3)	
12	Group Key: date_trunc('week'::text, 'Delivery Date')	
13	Batches: 1 Memory Usage: 721kB	
14	Worker 0: Batches: 1 Memory Usage: 721kB	
15	Worker 1: Batches: 1 Memory Usage: 721kB	
16	-> Parallel Bitmap Heap Scan on "Order" (cost=19.00..17225.90 rows=7404 width=28) (actual time=0.083..101.755 rows=169775 loops=3)	
17	Recheck Cond: (("Delivery Date" >= '2024-01-01 00:00:00'::timestamp without time zone) AND ("Delivery Date" <= '2042-12-31 00:00:00'::timestamp without time z...	
18	Rows Removed by Index Recheck: 3	
19	Filter: (EXTRACT(dow FROM "Delivery Date") = ANY ({1,2,3,4,5}::numeric[]))	
20	Rows Removed by Filter: 67204	
21	Heap Blocks: lossy=3892	
22	-> Bitmap Index Scan on idx_order_delivery_date_brin (cost=0.00..14.56 rows=710944 width=0) (actual time=0.164..0.164 rows=101530 loops=1)	
23	Index Cond: (("Delivery Date" >= '2024-01-01 00:00:00'::timestamp without time zone) AND ("Delivery Date" <= '2042-12-31 00:00:00'::timestamp without time ...	
24	Planning Time: 0.726 ms	
25	Execution Time: 187.364 ms	

*Примітка: аналіз виконання запиту №2 із використанням індексу*

Індекс BRIN прискорює виконання запиту, оскільки оптимізує пошук по діапазону дат у колонці "Delivery Date", працюючи з блоками даних замість окремих рядків. Це дозволяє швидко визначати блоки, що містять необхідні записи, що значно знижує час виконання. Оскільки запит використовує функції для групування по тижнях та фільтрації за днями тижня, BRIN індекс допомагає



ефективно відсіяти непотрібні дані, зменшуючи навантаження на систему. Це дозволяє знизити час виконання запиту на 79 мс (з 266 мс до 187 мс).

### Запит №3.


```
EXPLAIN ANALYZE
SELECT
    "Phone",
    COUNT(*) AS total_orders,
    SUM(LENGTH("Name")) AS total_name_length
FROM
    "Client"
WHERE
    "Phone" IN (
        '3801234567', '3802345678', '3803456789', '3804567890',
        '3805678901')
GROUP BY
    "Phone"
ORDER BY
    total_orders DESC;
```

	QUERY PLAN	
	text	
1	Sort (cost=10000008850.68..10000008850.69 rows=5 width=27) (actual time=51.676..71.541 rows=0 loops=1)	
2	Sort Key: (count(*)) DESC	
3	Sort Method: quicksort Memory: 25kB	
4	-> GroupAggregate (cost=10000008849.94..10000008850.62 rows=5 width=27) (actual time=51.672..71.536 rows=0 loops=1)	
5	Group Key: "Phone"	
6	-> Gather Merge (cost=10000008849.94..10000008850.52 rows=5 width=19) (actual time=51.671..71.534 rows=0 loops=1)	
7	Workers Planned: 2	
8	Workers Launched: 2	
9	-> Sort (cost=10000007849.92..10000007849.92 rows=2 width=19) (actual time=34.686..34.686 rows=0 loops=3)	
10	Sort Key: "Phone"	
11	Sort Method: quicksort Memory: 25kB	
12	Worker 0: Sort Method: quicksort Memory: 25kB	
13	Worker 1: Sort Method: quicksort Memory: 25kB	
14	-> Parallel Seq Scan on "Client" (cost=10000000000.00..10000007849.91 rows=2 width=19) (actual time=34.624..34.624 rows=0 loop...	
15	Filter: (("Phone")::text = ANY ('{3801234567,3802345678,3803456789,3804567890,3805678901}'::text[]))	
16	Rows Removed by Filter: 170007	
17	Planning Time: 0.137 ms	
18	Execution Time: 71.579 ms	

*Примітка: аналіз виконання запиту №3*



Створимо HASH індекс для стовпця "Phone": `CREATE INDEX idx_client_phone_hash ON "Client" USING HASH ("Phone");`

	QUERY PLAN	
	text	
1	Sort (cost=39.86..39.87 rows=5 width=27) (actual time=0.058..0.058 rows=0 loops=1)	
2	Sort Key: (count(*)) DESC	
3	Sort Method: quicksort Memory: 25kB	
4	-> GroupAggregate (cost=39.69..39.80 rows=5 width=27) (actual time=0.049..0.049 rows=0 loops=1)	
5	Group Key: "Phone"	
6	-> Sort (cost=39.69..39.70 rows=5 width=19) (actual time=0.048..0.049 rows=0 loops=1)	
7	Sort Key: "Phone"	
8	Sort Method: quicksort Memory: 25kB	
9	-> Bitmap Heap Scan on "Client" (cost=20.04..39.63 rows=5 width=19) (actual time=0.040..0.040 rows=0 loops=1)	
10	Recheck Cond: (("Phone")::text = ANY ('{3801234567,3802345678,3803456789,3804567890,3805678901}'::text[]))	
11	-> Bitmap Index Scan on idx_client_phone_hash (cost=0.00..20.04 rows=5 width=0) (actual time=0.038..0.038 rows=0 loop...	
12	Index Cond: (("Phone")::text = ANY ('{3801234567,3802345678,3803456789,3804567890,3805678901}'::text[]))	
13	Planning Time: 0.600 ms	
14	Execution Time: 0.100 ms	

*Примітка: аналіз виконання запиту №3 із використанням індексу*

Індекс типу HASH прискорив виконання запиту, оскільки він дозволяє PostgreSQL швидко знаходити точні значення в колонці "Phone" без необхідності повного сканування таблиці. В даному запиті використовується умова `WHERE "Phone" IN (...)`, що означає точний пошук кількох конкретних значень. Індекс HASH ефективно оптимізує цей пошук, дозволяючи PostgreSQL миттєво знаходити потрібні записи, замість того щоб перевіряти кожен рядок таблиці. Це значно знижує час виконання запиту, особливо коли таблиця містить велику кількість рядків.


#### Запит №4.

```
EXPLAIN ANALYZE
SELECT
    c."Name",
    COUNT(o."Order ID") AS total_orders,
    AVG(o."Rating") AS avg_rating,
    SUM(EXTRACT(EPOCH FROM o."Delivery Date" - o."Order Date") / 3600) AS
total_delivery_time,
    COUNT(o."Order ID") FILTER (WHERE o."Rating" > 3) AS high_rated_orders
FROM
    "Courier" c
JOIN
    "Order" o ON c."Phone" = o."Courier Phone"
WHERE
    c."Transport" = 'Motorbike'
```

```

AND o."Order Date" BETWEEN '2024-01-01' AND '2025-12-31'
GROUP BY
c."Name"
HAVING
COUNT(o."Order ID") > 5
ORDER BY
total_orders DESC;

```

	QUERY PLAN	
	text	
13	Sort Key: c."Name"	
14	Sort Method: quicksort Memory: 1387kB	
15	Worker 0: Sort Method: quicksort Memory: 1071kB	
16	Worker 1: Sort Method: quicksort Memory: 1064kB	
17	-> Nested Loop (cost=10000000000.43..10000018942.46 rows=12778 width=40) (actual time=0.070..253.670 rows=11207 loops=3)	
18	-> Parallel Seq Scan on "Order" o (cost=10000000000.00..10000014596.40 rows=64297 width=39) (actual time=0.009..26.474 rows=52020 loops=3)	
19	Filter: (("Order Date" >= '2024-01-01 00:00:00'::timestamp without time zone) AND ("Order Date" <= '2025-12-31 00:00:00'::timestamp without time z...	
20	Rows Removed by Filter: 184961	
21	-> Memoize (cost=0.43..0.55 rows=1 width=23) (actual time=0.004..0.004 rows=0 loops=156061)	
22	Cache Key: o."Courier Phone"	
23	Cache Mode: logical	
24	Hits: 32709 Misses: 31266 Evictions: 0 Overflows: 0 Memory Usage: 2629kB	
25	Worker 0: Hits: 18898 Misses: 27420 Evictions: 0 Overflows: 0 Memory Usage: 2309kB	
26	Worker 1: Hits: 18446 Misses: 27322 Evictions: 0 Overflows: 0 Memory Usage: 2297kB	
27	-> Index Scan using "Courier_pkey" on "Courier" c (cost=0.42..0.54 rows=1 width=23) (actual time=0.007..0.007 rows=0 loops=86008)	
28	Index Cond: (("Phone")::text = (o."Courier Phone")::text)	
29	Filter: (("Transport")::text = 'Motorbike')::text)	
30	Rows Removed by Filter: 1	
31	Planning Time: 0.388 ms	
32	Execution Time: 296.286 ms	

*Примітка: аналіз виконання запиту №4*

Створимо HASH індекс для стовпця "Phone": **CREATE INDEX**  
**idx\_courier\_phone\_hash ON "Courier" USING HASH ("Phone");**

	QUERY PLAN	
	text	
14	Sort Method: quicksort Memory: 1434kB	
15	Worker 0: Sort Method: quicksort Memory: 1039kB	
16	Worker 1: Sort Method: quicksort Memory: 1049kB	
17	-> Nested Loop (cost=10000000000.01..10000016732.17 rows=12600 width=40) (actual time=0.036..83.377 rows=11207 loops=3)	
18	-> Parallel Seq Scan on "Order" o (cost=10000000000.00..10000014596.40 rows=64297 width=39) (actual time=0.008..21.614 rows=52020 loops=3)	
19	Filter: ((("Order Date" >= '2024-01-01 00:00:00'::timestamp without time zone) AND ("Order Date" <= '2025-12-31 00:00:00'::timestamp without time z...	
20	Rows Removed by Filter: 184961	
21	-> Memoize (cost=0.01..0.11 rows=1 width=23) (actual time=0.001..0.001 rows=0 loops=156061)	
22	Cache Key: o."Courier Phone"	
23	Cache Mode: logical	
24	Hits: 31680 Misses: 35467 Evictions: 0 Overflows: 0 Memory Usage: 2983kB	
25	Worker 0: Hits: 18658 Misses: 25642 Evictions: 0 Overflows: 0 Memory Usage: 2157kB	
26	Worker 1: Hits: 19715 Misses: 24899 Evictions: 0 Overflows: 0 Memory Usage: 2096kB	
27	-> Index Scan using idx_courier_phone_hash on "Courier" c (cost=0.00..0.10 rows=1 width=23) (actual time=0.001..0.001 rows=0 loops=86008)	
28	Index Cond: (("Phone")::text = (o."Courier Phone")::text)	
29	Rows Removed by Index Recheck: 0	
30	Filter: (("Transport")::text = 'Motorbike')::text)	
31	Rows Removed by Filter: 1	
32	Planning Time: 0.749 ms	
33	Execution Time: 124.740 ms	

*Примітка: аналіз виконання запиту №4 із використанням індексу*

Індекс HASH для колонки "Phone" у таблиці "Courier" прискорив запит завдяки тому, що він дозволяє PostgreSQL швидко знаходити відповідні записи кур'єрів за телефоном при виконанні операції JOIN з таблицею "Order". У запиті використовується з'єднання JOIN по телефонному номеру кур'єра ("Phone" = "Courier Phone"), і для кожного кур'єра необхідно знайти всі відповідні замовлення. Без індексу, PostgreSQL довелося б сканувати таблицю "Courier" та перевіряти кожен рядок на наявність співпадінь за телефоном. Однак завдяки індексу HASH для колонки "Phone", пошук кур'єрів з конкретним телефоном значно прискорюється, що дозволяє зменшити час виконання запиту. Індекс допомагає ефективно обробляти великі обсяги даних, коли шукаються точні значення, що знижує час витрат на пошук і з'єднання даних.

### Завдання №3

Тригери у PostgreSQL використовуються для автоматичного виконання певних операцій на базі даних при виникненні визначених подій, таких як вставка, оновлення чи видалення записів. Вони дозволяють реалізувати додаткову бізнес-логіку, перевірки даних або забезпечення цілісності інформації без необхідності ручного втручання.

Функція для тригера:

```
CREATE OR REPLACE FUNCTION order_trigger_function()
RETURNS trigger AS $$
DECLARE
    order_cursor CURSOR FOR
        SELECT "Order ID", "Order Date", "Delivery Date", "Rating"
        FROM "Order"
        WHERE "Order ID" = OLD."Order ID";

    v_order_id INT;
    v_order_date TIMESTAMP;
    v_delivery_date TIMESTAMP;
    v_rating NUMERIC;
BEGIN
    IF TG_OP = 'DELETE' THEN
        OPEN order_cursor;
        LOOP
            BEGIN
                FETCH order_cursor INTO v_order_id, v_order_date,
v_delivery_date, v_rating;
                EXIT WHEN NOT FOUND;

                IF v_order_id < 1000 THEN
                    RAISE EXCEPTION 'ID замовлення % занадто малий для
видалення', v_order_id;
                ELSE
                    RAISE NOTICE 'Видаляється запис: ID = %, дата
замовлення = %, дата доставки = %, рейтинг = %',
v_order_id, v_order_date, v_delivery_date,
v_rating;
                END IF;
            EXCEPTION
                WHEN OTHERS THEN
                    RAISE NOTICE 'Помилка при видаленні запису ID %: %',
v_order_id, SQLERRM;
                    RETURN NULL;
            END;
        END LOOP;
        CLOSE order_cursor;
    END IF;

    IF TG_OP = 'UPDATE' THEN
        BEGIN
```

```

        IF NEW."Delivery Date" < NEW."Order Date" THEN
            RAISE EXCEPTION 'Дата доставки не може передувати даті
замовлення для ID: %', OLD."Order ID";
        END IF;

        IF NEW."Rating" <= 0 THEN
            RAISE EXCEPTION 'Рейтинг повинен бути більше 0 для ID: %',
OLD."Order ID";
        END IF;

        RAISE NOTICE 'Оновлення запису: ID = %, стара дата замовлення =
%, нова дата замовлення = %, стара дата доставки = %, нова дата доставки =
%, старий рейтинг = %, новий рейтинг = %',
        OLD."Order ID", OLD."Order Date", NEW."Order Date",
OLD."Delivery Date", NEW."Delivery Date", OLD."Rating", NEW."Rating";

    EXCEPTION
        WHEN OTHERS THEN
            RAISE NOTICE 'Помилка при оновленні запису %:', SQLERRM;
            RETURN NULL;

    END;
END IF;

IF TG_OP = 'UPDATE' THEN
    RETURN NEW;
ELSE
    RETURN OLD;
END IF;
END;
$$ LANGUAGE plpgsql;

```

Тригер для функції:

```

CREATE TRIGGER order_trigger
BEFORE DELETE OR UPDATE
ON "Order"
FOR EACH ROW
EXECUTE FUNCTION order_trigger_function();

```

*Видалення кількох записів (демонстрація курсора):*

[Query](#)
[Query History](#)

---

1 DELETE FROM "Order"  
2 WHERE "Order ID" IN (1001, 999, 1002);

---

[Data Output](#)
[Messages](#)
[Notifications](#)

---

NOTICE: Помилка при видаленні запису ID 999: ID замовлення 999 занадто малий для видалення  
NOTICE: Видаляється запис: ID = 1001, дата замовлення = 2024-11-02 20:55:52.517992, дата доставки = 2024-11-03 20:55:52.517992, рейтинг = 3  
NOTICE: Видаляється запис: ID = 1002, дата замовлення = 2024-11-01 20:55:52.517992, дата доставки = 2024-11-02 20:55:52.517992, рейтинг = 4  
DELETE 2

---

Query returned successfully in 495 msec.

Зміни у таблиці після видалення:

Query Query History

1 SELECT \* FROM public."Order"

2 WHERE "Order ID" BETWEEN 999 AND 1002

3 ORDER BY "Order ID" ASC

Data Output Messages Notifications

Order ID [PK] bigint

Order Date timestamp without time zone

Courier Phone character varying (10)

Delivery Date timestamp without time zone

Client Email character varying (32)

Rating integer

Delivery Address character varying (50)

1

999

2024-11-11 20:34:46.32684

9551962819

2024-11-12 20:34:46.32684

ava.zhu6241195@svc.com

4

Address 998

Оновлення з некоректною датою доставки:

Query Query History

1 UPDATE "Order"

2 SET "Order Date" = '2024-12-31', "Delivery Date" = '2024-01-01'

3 WHERE "Order ID" = 1010;

Data Output Messages Notifications

NOTICE: Помилка при оновленні запису: Дата доставки не може передувати даті замовлення для ID: 1010

UPDATE 0

Query returned successfully in 48 msec.

Оновлення з некоректним рейтингом:

Query Query History

1 UPDATE "Order"

2 SET "Rating" = -1

3 WHERE "Order ID" = 1010;

Data Output Messages Notifications

NOTICE: Помилка при оновленні запису: Рейтинг повинен бути більше 0 для ID: 1010

UPDATE 0

Query returned successfully in 52 msec.

Успішне оновлення запису:

Query Query History

1 UPDATE "Order"

2 SET "Order Date" = '2024-12-30', "Delivery Date" = '2024-12-31', "Rating" = 5

3 WHERE "Order ID" = 1020;

Data Output Messages Notifications

NOTICE: Оновлення запису: ID = 1020, стара дата замовлення = 2024-11-19 20:55:52.517992, нова дата замовлення = 2024-12-30 00:00:00, стара дата доставки = 2024-11-20 20:55:52.517992, нова дата доставки = 2024-12-31 00:00:00, старий рейтинг = 1, новий рейтинг = 5

UPDATE 1

Query returned successfully in 50 msec.

Зміни у таблиці після оновлення:

Query Query History

1 SELECT \* FROM public."Order"

2 WHERE "Order ID" = 1020

Data Output Messages Notifications

SQL

Order ID [PK] bigint

Order Date timestamp without time zone

Courier Phone character varying (10)

Delivery Date timestamp without time zone

Client Email character varying (32)

Rating integer

Delivery Address character varying (50)

1	1020	2024-12-30 00:00:00	7248934309	2024-12-31 00:00:00	hal.lin4508638@w.com	5	Address 1020
---	------	---------------------	------------	---------------------	----------------------	---	--------------

У тригері реалізована перевірка даних перед операцією видалення та оновлення. Використано курсор для поетапної обробки записів, що дозволяє здійснювати перевірки та виключення помилок на кожному етапі. Якщо умови не виконуються, операція скасовується, що допомагає підтримувати цілісність даних. Така реалізація дозволяє ефективно контролювати зміни в таблиці.

## Завдання №4

**READ COMMITTED** гарантує, що кожен запит в межах транзакції бачить лише підтверджені зміни, які були здійснені до моменту виконання запиту. Це означає, що якщо одна транзакція змінює значення в таблиці, інша транзакція може побачити ці зміни тільки після того, як вони будуть зафіксовані.










Неповторюване читання (*non-repeatable read*) відбувається, коли транзакція читає одні й ті ж дані кілька разів, і між читаннями ці дані змінюються іншою транзакцією.

Транзакція 1:

[Query](#) [Query History](#)

```
1 BEGIN;
2 SELECT * FROM public."Meal"
3 ORDER BY "Meal ID" ASC;
```

[Data Output](#) [Messages](#) [Notifications](#)



SQL

	Meal ID [PK] bigint	Order ID bigint	Name character varying (25)	Price integer	Weight integer	Serving Size integer
1	0	1	Tiramisu	400	150	1
2	1	100	Burger	15	200	1
3	30	31	Frittata	0	130	1
4	31	32	Ceviche	41	131	2



## Транзакція 2:

Query Query History

1

BEGIN;

2

UPDATE "Meal"

3

SET "Price" = 999

4

WHERE "Meal ID" = 30;

5

COMMIT;

Data Output Messages Notifications

COMMIT

Query returned successfully in 55 msec.

## Транзакція 1 (після зміни у транзакції 2):

Query Query History

1

SELECT \* FROM public."Meal"

2

ORDER BY "Meal ID" ASC;

Data Output Messages Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

📥

📈

SQL

	Meal ID [PK] bigint	Order ID bigint	Name character varying (25)	Price integer	Weight integer	Serving Size integer
1	0	1	Tiramisu	400	150	1
2	1	100	Burger	15	200	1
3	30	31	Frittata	999	130	1
4	31	32	Ceviche	41	131	2
5	32	33	Clams Casino	42	132	3

В першому запиті транзакції 1 переглядаємо вміст таблиці “Meal”. В другому запиті транзакції 1, після того як транзакція 2 змінила дані і підтвердила їх, значення зміняться. READ COMMITTED не забезпечує консистентність даних протягом всієї транзакції, тому цей феномен може відбуватися.











Фантомне читання (*phantom reads*) відбувається, коли між двома запитами в одній транзакції з'являються нові рядки, які відповідають умовам запиту, через виконання інших транзакцій.


Транзакція 1:

[Query](#) [Query History](#)

```
1 BEGIN;  
2 SELECT COUNT(*) FROM "Meal" WHERE "Price" = 100;
```

[Data Output](#) [Messages](#) [Notifications](#)



	count bigint 
1	32935

Транзакція 2:

[Query](#) [Query History](#)

```
1 BEGIN;  
2 INSERT INTO "Meal" ("Meal ID", "Order ID", "Name", "Price", "Weight", "Serving Size")  
3 VALUES (3, 1, 'Pizza', 100, 250, 1);  
4 COMMIT;
```

[Data Output](#) [Messages](#) [Notifications](#)

COMMIT

Query returned successfully in 48 msec.

Транзакція 1 (після зміни у транзакції 2):

Query

Query History










1


SELECT COUNT(\*) FROM "Meal" WHERE "Price" = 100;

Data Output

Messages

Notifications



	count	
	bigint	
1	32936	

В першому запиті транзакції 1 буде повернено кількість страв з ціною 100. У другому запиті транзакції 1 кількість страв буде іншою, оскільки транзакція 2 додала новий запис, що відповідає умовам запиту.

На рівні **REPEATABLE READ**, PostgreSQL використовує механізм, що дозволяє транзакціям бачити лише ті дані, які були доступні на момент початку транзакції, або ж ті, які вже були підтверджені до початку транзакції. Це дозволяє вирішити такі проблеми, як неповторюване та фантомне читання.

Транзакція 1:

Query

Query History

1 BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;

2 SELECT "Price" FROM "Meal" WHERE "Meal ID" = 1;

Data Output

Messages

Notifications

## Транзакція 2:

Query Query History

```
1 BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
2 UPDATE "Meal" SET "Price" = 1000 WHERE "Meal ID" = 1;
3 COMMIT;
```

Data Output Messages Notifications

COMMIT

Query returned successfully in 166 msec.

## Транзакція 1 (після зміни у транзакції 2):

Query Query History

```
1 SELECT "Price" FROM "Meal" WHERE "Meal ID" = 1;
```

Data Output Messages Notifications

≡+

📄

▼

📋

▼


🗑️

🗄️

⬇️

📈

SQL

	Price integer 
1	500










На рівні ізоляції REPEATABLE READ, транзакція, яка читає рядок, бачить те саме значення кожного разу, навіть якщо інша транзакція в цей час намагається змінити ці дані.

## Транзакція 1:

[Query](#) [Query History](#)

```
1 BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
2 SELECT * FROM "Meal" WHERE "Price" > 100;
```

[Data Output](#) [Messages](#) [Notifications](#)



SQL

	Meal ID [PK] bigint	Order ID bigint	Name character varying (25)	Price integer	Weight integer	Serving Size integer
1	30	31	Frittata	2000	130	1
2	0	1	Tiramisu	400	150	1
3	1	100	Burger	500	200	1

## Транзакція 2:

[Query](#) [Query History](#)

```
1 BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
2 INSERT INTO "Meal" ("Meal ID", "Order ID", "Name", "Price", "Weight", "Serving Size")
3   VALUES (10, 1, 'New Dish', 150, 200, 1);
4 COMMIT;
```

[Data Output](#) [Messages](#) [Notifications](#)

COMMIT

Query returned successfully in 50 msec.

Транзакція 1 (після зміни у транзакції 2):

Query Query History

1 SELECT \* FROM "Meal" WHERE "Price" > 100;

Data Output Messages Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	Meal ID [PK] bigint	Order ID bigint	Name character varying (25)	Price integer	Weight integer	Serving Size integer
1	30	31	Frittata	2000	130	1
2	0	1	Tiramisu	400	150	1
3	1	100	Burger	500	200	1

На рівні ізоляції **REPEATABLE READ**, транзакція «заморожує» набір рядків, який вона отримала на момент виконання першого запиту. Якщо між запитами транзакції інші транзакції додають або видаляють рядки, які відповідають умовам запиту, транзакція не побачить цих змін. Вона побачить однаковий набір рядків протягом всього свого виконання.

Рівень ізоляції **SERIALIZABLE** забезпечує найвищий рівень ізоляції для транзакцій у PostgreSQL. На цьому рівні ізоляції всі транзакції працюють так, ніби вони виконуються по черзі, а не паралельно. Це означає, що **SERIALIZABLE** дозволяє усунути всі основні феномени паралельних транзакцій.

Транзакція 1:

Query Query History

1 BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
2 SELECT "Price" FROM "Meal" WHERE "Meal ID" = 1;

Data Output Messages Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	Price integer
1	1000

Транзакція 1 (після зміни у транзакції 2):

Query Query History

1

BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;

2

UPDATE "Meal" SET "Price" = 5000 WHERE "Meal ID" = 1;

3

COMMIT;

Data Output Messages Notifications

COMMIT

Query returned successfully in 48 msec.

Транзакція 2:

Query Query History

1

SELECT "Price" FROM "Meal" WHERE "Meal ID" = 1;

Data Output Messages Notifications

≡+

▼

▼

SQL

Price

integer

1	1000
---	------

Ситуація, коли при повторному читанні в рамках однієї транзакції, раніше прочитані дані виявляються зміненими (неповторюване читання) не відбувається на рівні SERIALIZABLE, оскільки транзакція не може побачити незавершених (незафіксованих) змін іншої транзакції.

## Транзакція 1:

[Query](#) [Query History](#)

```
1 BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
2 SELECT * FROM "Meal" WHERE "Price" > 500;
```

[Data Output](#) [Messages](#) [Notifications](#)

	Meal ID [PK] bigint	Order ID bigint	Name character varying (25)	Price integer	Weight integer	Serving Size integer
1	30	31	Frittata	2000	130	1
2	1	100	Burger	5000	200	1

## Транзакція 2:

[Query](#) [Query History](#)

```
1 BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
2 INSERT INTO "Meal" ("Meal ID", "Order ID", "Name", "Price", "Weight", "Serving Size")  
3 VALUES (15, 1, 'New Dish', 550, 200, 1);  
4 COMMIT;
```

[Data Output](#) [Messages](#) [Notifications](#)

COMMIT

Query returned successfully in 48 msec.

## Транзакція 1 (після зміни у транзакції 2):

[Query](#) [Query History](#)

```
1 SELECT * FROM "Meal" WHERE "Price" > 500;
```

[Data Output](#) [Messages](#) [Notifications](#)

	Meal ID [PK] bigint	Order ID bigint	Name character varying (25)	Price integer	Weight integer	Serving Size integer
1	30	31	Frittata	2000	130	1
2	1	100	Burger	5000	200	1



Фантомне читання не відбувається на рівні **SERIALIZABLE**, оскільки транзакція отримує незмінений набір рядків, навіть якщо інші транзакції додають або видаляють рядки, які відповідають умовам запиту.

Аномалія серіалізації (*serialization anomaly*) відбувається, коли паралельно виконувані транзакції порушують цілісність і узгодженість даних.

#### Транзакція 1:

[Query](#) [Query History](#)

---

```
1 BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
2 UPDATE "Meal" SET "Price" = "Price" + 50 WHERE "Price" > 500;
```

[Data Output](#) [Messages](#) [Notifications](#)

---

UPDATE 4

Query returned successfully in 279 msec.

#### Транзакція 2:

[Query](#) [Query History](#)

---

```
1 BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
2 ✓ INSERT INTO "Meal" ("Meal ID", "Order ID", "Name", "Price", "Weight", "Serving Size")
3   VALUES (27, 1, 'Dish 2', 501, 200, 1);
4 COMMIT;
```

[Data Output](#) [Messages](#) [Notifications](#)

---

COMMIT

Query returned successfully in 52 msec.

Транзакція 1 (після зміни у транзакції 2):

```
Query  Query History
1  UPDATE "Meal" SET "Price" = "Price" + 50 WHERE "Price" > 500;
2  COMMIT;
```

---

Data Output Messages Notifications

---

COMMIT

Query returned successfully in 386 msec.

Аномалія серіалізації також не відбувається на рівні **SERIALIZABLE**, оскільки всі транзакції виконуються таким чином, ніби вони виконуються по черзі, а не паралельно, що дозволяє уникнути аномалій серіалізації.

✚ Також варто зауважити, що існує поняття “брудного читання”. Це означає, що в теорії рівень ізоляції **READ UNCOMMITTED** дозволяє “брудні” читання, однак, у PostgreSQL цей рівень ізоляції фактично не підтримується, і він автоматично працює як **READ COMMITTED**. Брудне читання (*dirty read*) відбувається, коли одна транзакція читає незафіксовані зміни іншої транзакції, які можуть бути скасовані після того, як друга транзакція скасує свої зміни.

Рівні ізоляції транзакцій у PostgreSQL забезпечують різні ступені захисту від феноменів некоректної роботи з даними. **READ COMMITTED** усуває брудне читання, але дозволяє неповторюване читання, фантомне читання та аномалії серіалізації. **REPEATABLE READ** усуває брудне та неповторюване читання, але допускає фантомне читання (хоча в PostgreSQL воно відсутнє). Найвищий рівень ізоляції, **SERIALIZABLE**, усуває всі феномени, забезпечуючи максимально можливу консистентність даних.