

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО"

Кафедра програмного забезпечення комп'ютерних систем

Лабораторна робота №3
із дисципліни «Бази даних»
на тему «Засоби оптимізації роботи СУБД PostgreSQL»

Виконала
студентка 3 курсу
групи КП-82
Джергалова Рената Олександрівна

Перевірів
“ _____ ” “ _____ ” 2020 р.
Радченко К.О.

Київ 2020

Мета роботи: здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL

Варіант: 5

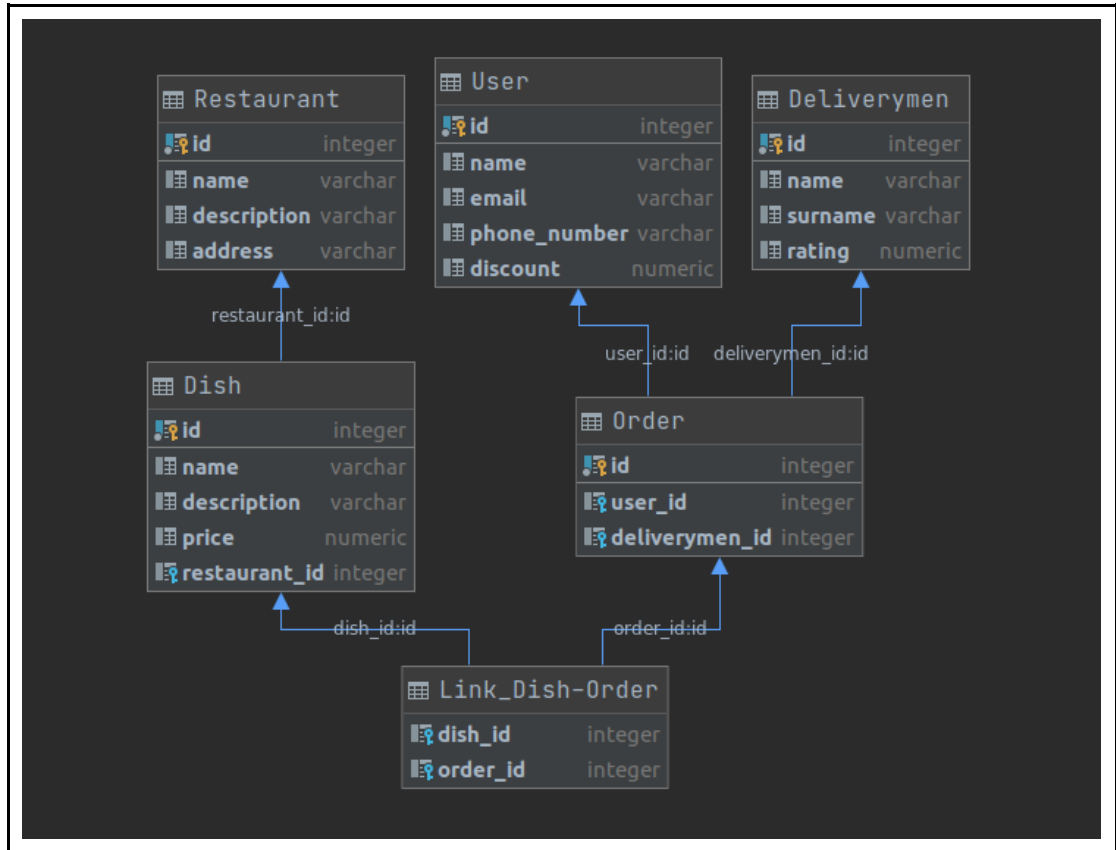
Завдання:

1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.

URL репозиторію з вихідним кодом: <https://github.com/le-kalmique/db-course>

Завдання 1

- Схеми бази даних у вигляді таблиць і зв'язків між ними



- Класи ORM, що відповідають таблицям бази даних

```
class Deliverymen(Base):
    __tablename__ = 'Deliverymen'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    surname = Column(String)
    rating = Column(Numeric)

    def __repr__(self):
        return "<Deliverymen(name='%s', " \
            " surname='%s', " \
            " rating='%i')>" % \
            (self.name,
             self.surname,
             self.rating)

    def __init__(self,
                  name: str,
                  surname: str,
                  rating: int):
        self.name = name
        self.surname = surname
        self.rating = rating
```

```

class Dish(Base):
    __tablename__ = 'Dish'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    description = Column(String)
    price = Column(Numeric)
    restaurant_id = Column(Integer, ForeignKey('Restaurant.id', ondelete='CASCADE'))

    Orders = relationship("Order", secondary=link_dish_order, cascade="all,delete")
    Restaurant = relationship("Restaurant", backref=backref("Dish", uselist=False, cascade="all,delete"))

    def __repr__(self):
        return "<Dish(name='%s'," \
            " description='%s'," \
            " price='%i'," \
            " restaurant_id='%i')>" % \
            (self.name,
             self.description,
             self.price,
             self.restaurant_id)

    def __init__(self,
        name: str,
        description: str,
        price: int,
        restaurant_id: int):
        self.name = name

```

```

from sqlalchemy import Column, Integer, Table, ForeignKey
from db import Base

link_dish_order = Table(
    'Link_Dish-Order', Base.metadata,
    Column('dish_id', Integer, ForeignKey('Dish.id')),
    Column('order_id', Integer, ForeignKey('Order.id'))
)

```

```

class Order(Base):
    __tablename__ = 'Order'

    id = Column(Integer, primary_key=True)
    user_id = Column(Integer, ForeignKey('User.id', ondelete='CASCADE'))
    deliverymen_id = Column(Integer, ForeignKey('Deliverymen.id', ondelete='CASCADE'))
    User = relationship("User", backref=backref("Order", uselist=False, cascade="all,delete"))
    Deliverymen = relationship("Deliverymen", backref=backref("Order", uselist=False, cascade="all,delete"))

    def __repr__(self):
        return "<Order(user_id='%i', deliverymen_id='%i')>" % \
            (self.user_id, self.deliverymen_id)

    def __init__(self, user_id: int, deliverymen_id: int):
        self.user_id = user_id
        self.deliverymen_id = deliverymen_id

```

```

class Order(Base):
    __tablename__ = 'Order'

    id = Column(Integer, primary_key=True)
    user_id = Column(Integer, ForeignKey('User.id', ondelete='CASCADE'))
    deliverymen_id = Column(Integer, ForeignKey('Deliverymen.id', ondelete='CASCADE'))
    User = relationship("User", backref=backref("Order", uselist=False, cascade="all,delete"))
    Deliverymen = relationship("Deliverymen", backref=backref("Order", uselist=False, cascade="all,delete"))

    def __repr__(self):
        return "<Order(user_id='%i', deliverymen_id='%i')>" % \
            (self.user_id, self.deliverymen_id)

    def __init__(self, user_id: int, deliverymen_id: int):
        self.user_id = user_id
        self.deliverymen_id = deliverymen_id

```

```
class User(Base):
    __tablename__ = 'User'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    email = Column(String)
    phone_number = Column(String)
    discount = Column(Numeric)

    def __repr__(self):
        return "<User(name='%s'," \
            " email='%s'," \
            " phone_number='%s'," \
            " discount='%i')>" % \
            (self.name,
             self.email,
             self.phone_number,
             self.discount)

    def __init__(self,
                  name: str,
                  email: str,
                  phone_number: str,
                  discount: int):
        self.name = name
        self.email = email
        self.phone_number = phone_number
        self.discount = discount
```

- Навести приклади запитів у вигляді ORM

```
items = session.query(self.instance) \
    .order_by(self.instance.id.asc()) \
    .offset(page * per_page) \
    .limit(per_page) \
    .all()
```

```
session.add(item)
session.commit()
session.refresh(item)
return item.id
```

```
return session.query(self.instance).get(itemId)
```

```
session.query(self.instance).filter(self.instance.id == itemId).delete()
session.commit()
return deletedItem
```

```
session.query(self.instance) \
    .filter(self.instance.id == item.id) \
    .update(self.getModelEntityMappedKeys(item))
session.commit()
```

```
session.execute(select([func.count()]).select_from(self.instance)).scalar()
```


Завдання 2

- команди створення індексів, тексти, результати і час виконання запитів SQL

1

Створення індексації

```
Create index gin_idx
on "Deliverymen" using gin(name gin_trgm_ops, surname gin_trgm_ops)
```

Запит

```
explain analyze select rating, array_agg(name), array_agg(surname) from "Deliverymen"
where name like 'a%' and surname like 'a%'
group by rating
order by rating desc
```

Результат без індексації (3,2 ms)

```
■ QUERY PLAN
1 GroupAggregate (cost=244.78..245.06 rows=8 width=69) (actual time=3.193..3.208 rows=9 loops=1)
2   Group Key: rating
3   -> Sort (cost=244.78..244.82 rows=16 width=37) (actual time=3.155..3.158 rows=20 loops=1)
4     Sort Key: rating DESC
5     Sort Method: quicksort Memory: 26kB
6     -> Seq Scan on "Deliverymen" (cost=0.00..244.47 rows=16 width=37) (actual time=0.184..3.110 rows=20 loops=1)
7       Filter: (((name)::text ~ 'a'::text) AND ((surname)::text ~ 'a'::text))
8       Rows Removed by Filter: 10011
9 Planning Time: 0.175 ms
10 Execution Time: 3.269 ms
```

Результат з індексацією (0,2 ms)

```
■ QUERY PLAN
1 GroupAggregate (cost=62.75..63.03 rows=8 width=69) (actual time=0.134..0.153 rows=9 loops=1)
2   Group Key: rating
3   -> Sort (cost=62.75..62.79 rows=16 width=37) (actual time=0.125..0.128 rows=20 loops=1)
4     Sort Key: rating DESC
5     Sort Method: quicksort Memory: 26kB
6     -> Bitmap Heap Scan on "Deliverymen" (cost=20.17..62.43 rows=16 width=37) (actual time=0.077..0.109 rows=20 loops=1)
7       Recheck Cond: (((name)::text ~ 'a'::text) AND ((surname)::text ~ 'a'::text))
8       Heap Blocks: exact=16
9       -> Bitmap Index Scan on gin_idx (cost=0.00..20.16 rows=16 width=0) (actual time=0.066..0.067 rows=20 loops=1)
10         Index Cond: (((name)::text ~ 'a'::text) AND ((surname)::text ~ 'a'::text))
11 Planning Time: 0.128 ms
12 Execution Time: 0.193 ms
```

Висновок: Оскільки ми маємо повнотекстовий пошук та використовуємо для пошуку декілька атрибутів, то доречно використати gin індексування, щоб значно поліпшити час виконання запиту.

2

Створення індексації

```
Create index rest_gin_idx
on "Restaurant" using gin(name gin_trgm_ops, address gin_trgm_ops)
```

Запит

```
explain analyze Select array_agg(name), description, array_agg(address) from "Restaurant"
where description like 'a%' and address like 'a%'
group by description
order by description desc
```

Результат без індексації (25 ms)

```
QUERY PLAN
1  GroupAggregate  (cost=3042.15..3044.45 rows=92 width=115) (actual time=24.608..24.798...
2    Group Key: description
3    -> Sort  (cost=3042.15..3042.38 rows=92 width=88) (actual time=24.586..24.599 rows=92)
4        Sort Key: description DESC
5        Sort Method: quicksort  Memory: 51kB
6        -> Seq Scan on "Restaurant"  (cost=0.00..3039.15 rows=92 width=88) (actual time=0.014..0.014 rows=92)
7            Filter: (((description)::text ~ 'a% '::text) AND ((address)::text ~ 'a% '::text))
8            Rows Removed by Filter: 99825
9  Planning Time: 0.223 ms
10 Execution Time: 24.857 ms
```

Результат з індексацією (3,8 ms)

```
QUERY PLAN
1  GroupAggregate  (cost=1635.63..1637.93 rows=92 width=115) (actual time=3.355..3.772 rows=185)
2    Group Key: description
3    -> Sort  (cost=1635.63..1635.86 rows=92 width=88) (actual time=3.333..3.363 rows=185)
4        Sort Key: description DESC
5        Sort Method: quicksort  Memory: 51kB
6        -> Bitmap Heap Scan on "Restaurant"  (cost=42.75..1632.63 rows=92 width=88) (actual time=0.014..0.014 rows=92)
7            Recheck Cond: ((address)::text ~ 'a% '::text)
8            Filter: ((description)::text ~ 'a% '::text)
9            Rows Removed by Filter: 3940
10           Heap Blocks: exact=1443
11         -> Bitmap Index Scan on rest_gin_idx  (cost=0.00..42.73 rows=3031 width=0) (actual time=0.014..0.014 rows=92)
12             Index Cond: ((address)::text ~ 'a% '::text)
13 Planning Time: 0.288 ms
14 Execution Time: 3.836 ms
```

Висновок: Оскільки ми маємо повнотекстовий пошук та використовуємо для пошуку декілька атрибутів, то доречно використати gin індексування, щоб значно поліпшити час виконання запиту.

Створення індексації

```
CREATE INDEX btree_idx ON "Order" USING btree (user_id, deliverymen_id);
```

Запит

```
explain analyse SELECT user_id, deliverymen_id from "Order"  
where user_id between 1980 and 1989 and deliverymen_id between 1980 and 1989
```

Результат без індексації (12,3ms)

```
QUERY PLAN  
Seq Scan on "Order" (cost=0.00..1525.12 rows=1 width=8) (actual time=12.303..12.305 rows=0 loops=1)  
  Filter: ((user_id >= 1980) AND (user_id <= 1989) AND (deliverymen_id >= 1980) AND (deliverymen_id <= 1989))  
  Rows Removed by Filter: 60006  
Planning Time: 0.131 ms  
Execution Time: 12.346 ms
```

Результат з індексацією (0,049ms)

```
QUERY PLAN  
1  Index Only Scan using btree_idx on "Order" (cost=0.29..8.52 rows=1 width=8) (actual time=0.026..0.026 rows=0 loops=1)  
2    Index Cond: ((user_id >= 1980) AND (user_id <= 1989) AND (deliverymen_id >= 1980) AND (deliverymen_id <= 1989))  
3    Heap Fetches: 0  
4  Planning Time: 0.384 ms  
5  Execution Time: 0.049 ms
```

Висновок: оскільки ми використовуємо селектор, де використовуємо діапазон чисел, доцільно проіндексувати нашу таблицю btree деревом

Створення індексації

```
CREATE INDEX btree_dish_idx ON "Dish" USING btree (price);
```

Запит

```
explain analyse SELECT name, description from "Dish"  
where price between 1760 and 1780
```

Результат без індексації (20ms)

```
QUERY PLAN  
1  Seq Scan on "Dish" (cost=0.00..1461.16 rows=680 width=30) (actual time=0.040..19.737 rows=599 loops=1)  
2    Filter: ((price >= '1760'::numeric) AND (price <= '1780'::numeric))  
3    Rows Removed by Filter: 59412  
4  Planning Time: 0.162 ms  
5  Execution Time: 19.846 ms
```

Результат з індексацією (0,6ms)

```
QUERY PLAN
1 Bitmap Heap Scan on "Dish" (cost=15.26..615.63 rows=680 width=30) (actual time=0.134..0.610 rows=599 loops=1)
2   Recheck Cond: ((price >= '1760'::numeric) AND (price <= '1780'::numeric))
3   Heap Blocks: exact=387
4   -> Bitmap Index Scan on btree_dish_idx (cost=0.00..15.09 rows=680 width=0) (actual time=0.097..0.097 rows=599 loops=1)
5       Index Cond: ((price >= '1760'::numeric) AND (price <= '1780'::numeric))
6 Planning Time: 0.086 ms
7 Execution Time: 0.644 ms
```

Висновок: оскільки ми використовуємо селектор, де використовуємо діапазон чисел, доцільно проіндексувати нашу таблицю btree деревом

Завдання 3

- команди, що ініціюють виконання тригера, текст тригера та скріншоти зі змінами у таблицях бази даних

Опис роботи тригера: При зміні чи видаленні запису в таблиці Restaurant, тригер ставить всім записам з таблиці Dish, що містили як FK змінений, або видалений запис з Restaurant, загальний Restaurant id (restaurant_id = 0).

Текст тригера

```
CREATE OR REPLACE FUNCTION trigger() RETURNS trigger AS
$$
    DECLARE
        ref_cursor_update NO SCROLL CURSOR FOR Select id,
        restaurant_id from "Dish"
                                                where
        restaurant_id = New.id;

        ref_cursor_delete NO SCROLL CURSOR FOR Select id,
        restaurant_id from "Dish"
                                                where
        restaurant_id = Old.id;
    record record;
    BEGIN
        if (TG_OP = 'DELETE') then
            if old.id = 0 then raise EXCEPTION 'You cannot
delete this';
            end if;

            OPEN ref_cursor_delete;
            LOOP
                FETCH ref_cursor_delete INTO record;
```

```

        if not found then
            exit;
        end if;
        UPDATE "Dish" SET restaurant_id = 0 WHERE
record.restaurant_id = old.id;
    END LOOP;
    RETURN old;
end if;

if (TG_OP = 'UPDATE') then
    OPEN ref_cursor_update;
    LOOP
        FETCH ref_cursor_update INTO record;
        if not found then
            exit;
        end if;
        UPDATE "Dish" SET restaurant_id = 0 WHERE
record.restaurant_id = new.id;
    END LOOP;
    RETURN new;
end if;
END;
$$ LANGUAGE plpgsql;

```

Create TRIGGER trigger BEFORE UPDATE OR DELETE ON
"Restaurant" FOR EACH ROW EXECUTE PROCEDURE trigger();

Команди, що ініціюють виконання тригера

```

Delete from "Restaurant" where id = 1





Update "Restaurant" set id = 2 where id = 1

delete from "Restaurant" where id = 0






```





Записи до змін:

	id	name	description	price	restaurant_id
1	1	borsh	toze godnota	200	1
2	2	pizza	tasty pizza	300	1
3	3	drgfd	gdxfgd	32	1

	 id	÷	 name	÷	 description	÷	 address	÷
1	0		default		general		world	
2	1		Il Molino		Smth		oishrf	

Записи після змін:

	 id	÷	 name	÷	 description	÷	 price	÷	 restaurant_id	÷
1	1		borsh		toze godnota		200		0	
2	2		pizza		tasty pizza		300		0	
3	3		drgfd		gdxfgd		32		0	

	 id	÷	 name	÷	 description	÷	 address	÷
1	0		default		general		world	