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»**

Виконав
студент 2 курсу
групи КП-92
Фенченко Ігор Юрійович

Перевірив
“____” “_____” 20___ р.
к. т. н. доцент
Петрашенко Андрій Васильович

Київ 2020

Мета роботи: здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL

Варіант: 23

Завдання:

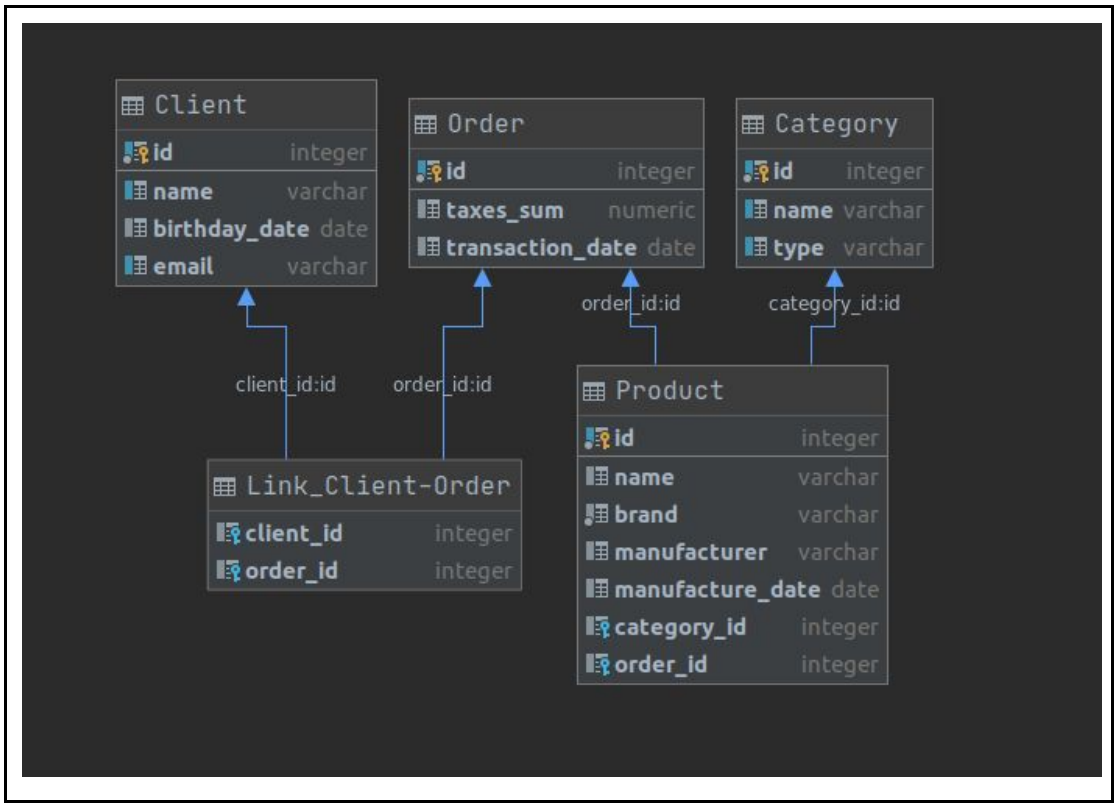
1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.

URL репозиторію з вихідним кодом:

https://github.com/bujhmt/DB_labs/tree/master/lab3

Завдання 1

- Схему бази даних у вигляді таблиць і зв’язків між ними



- Класи ORM, що відповідають таблицям бази даних

```python
from sqlalchemy import Column, Integer, String
from db import Base

class Category(Base):
    __tablename__ = 'Category'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    type = Column(String)

    def __repr__(self):
      return "<Category(name='%s', type='%s')>" % \
             (self.name, self.type)

    def __init__(self, name: str, type: str):
        self.name = name
        self.type = type
```

```python
from sqlalchemy import Column, Integer, String, Date, func
from sqlalchemy.orm import relationship
from models.links import links_orders_association
from db import Base

class Client(Base):
    __tablename__ = 'Client'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    birthday_date = Column(Date, default=func.now())
    email = Column(String)

    Orders = relationship("Order", secondary=links_orders_association)

    def __repr__(self):
      return "<Client(name='%s', birthday_date='%s', email='%s')>" % \
             (self.name, self.birthday_date, self.email)

    def __init__(self, name: str, birthday_date: str, email: str):
        self.name = name
        self.birthday_date = birthday_date
        self.email = email
```

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

links_orders_association = Table(
    'Link_Client-Order', Base.metadata,
    Column('client_id', Integer, ForeignKey('Client.id')),
    Column('order_id', Integer, ForeignKey('Order.id'))
)
```

```python
from sqlalchemy import Column, Integer, Numeric, Date, func
from sqlalchemy.orm import relationship
from models.links import links_orders_association
from db import Base


class Order(Base):
    __tablename__ = 'Order'

    id = Column(Integer, primary_key=True)
    taxes_sum = Column(Numeric)
    transaction_date = Column(Date, default=func.now())
    Clients = relationship("Client", secondary=links_orders_association

    def __repr__(self):
        return "<Order(taxes_sum='%i', transaction_date='%s')>" % \
               (self.taxes_sum, self.transaction_date)

    def __init__(self, transaction_date: str, taxes_sum: int):
        self.taxes_sum = taxes_sum
        self.transaction_date = transaction_date
```

```python
class Product(Base):
    __tablename__ = 'Product'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    brand = Column(String)
    manufacturer = Column(String)
    manufacture_date = Column(Date, default=func.now())
    category_id = Column(Integer, ForeignKey('Category.id'))
    order_id = Column(Integer, ForeignKey('Order.id'))
    Order = relationship("Order", backref=backref("Product", uselist=False))
    Category = relationship("Category", backref=backref("Product", uselist=False))

    def __repr__(self):
      return "<Category(name='%s'," \
             " brand='%s'," \
             " manufacturer='%s'," \
             " manufacture_date='%s'," \
             " category_id='%i'," \
             " order_id='%i')>" % \
             (self.name,
              self.brand,
              self.manufacturer,
              self.manufacture_date,
              self.category_id,
              self.order_id)
```

- Навести приклади запитів у вигляді ORM

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

```python
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_all_inx ON "Client" using gin(name gin_trgm_ops, email gin_trgm_ops)`
Запит
`explain analyse select birthday_date, array_agg(name), array_agg(email) from "Client" where name like 'a%' and email like 'a%' group by birthday_date order by birthday_date`
Результат без індексації (18 ms)

```
GroupAggregate  (cost=2340.08..2344.12 rows=161 width=68) (actual time=17.720..17.875 rows=155 loop
  Group Key: birthday_date
  ->  Sort  (cost=2340.08..2340.49 rows=163 width=29) (actual time=17.685..17.714 rows=157 loops=1)
        Sort Key: birthday_date
        Sort Method: quicksort  Memory: 37kB
        ->  Seq Scan on "Client"  (cost=0.00..2334.09 rows=163 width=29) (actual time=0.310..17.597
              Filter: (((name)::text ~~ 'a%'::text) AND ((email)::text ~~ 'a%'::text))
              Rows Removed by Filter: 99849
Planning Time: 0.267 ms
Execution Time: 17.946 ms
```

Результат з індексацією (1 ms)

```
  Sort Method: quicksort  Memory: 47kB
  ->  HashAggregate  (cost=440.40..442.82 rows=161 width=68) (actual time=0.648..0.699 rows=155 loops=1)
        Group Key: birthday_date
        ->  Bitmap Heap Scan on "Client"  (cost=29.67..439.18 rows=163 width=29) (actual time=0.338..0.551 rows=
              Recheck Cond: (((name)::text ~~ 'a%'::text) AND ((email)::text ~~ 'a%'::text))
              Heap Blocks: exact=147
              ->  Bitmap Index Scan on gin_all_inx  (cost=0.00..29.63 rows=163 width=0) (actual time=0.319..0.31
                    Index Cond: (((name)::text ~~ 'a%'::text) AND ((email)::text ~~ 'a%'::text))
Planning Time: 0.108 ms
Execution Time: 0.788 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 gin_all_idx
ON "Category" using gin(name gin_trgm_ops, type gin_trgm_ops)
```

Запит

```
explain analyse Select name, array_agg(type) from "Category"
Where name like 'a%'
group by name
```

Результат без індексації (30 ms)

```
  ->  Sort  (cost=2161.27..2168.85 rows=3030 wid…
        Sort Key: name
        Sort Method: quicksort  Memory: 407kB
        ->  Seq Scan on "Category"  (cost=0.00..…
              Filter: ((name)::text ~~ 'a%'::tex…
              Rows Removed by Filter: 96037
Planning Time: 0.136 ms
Execution Time: 29.516 ms
```

Результат з індексацією (12 ms)

```
  ->  Sort  (cost=992.57..1000.15 rows=3030 width=20) (actual time=8.998..9.398 row
        Sort Key: name
        Sort Method: quicksort  Memory: 407kB
        ->  Bitmap Heap Scan on "Category"  (cost=43.49..817.36 rows=3030 width=20)
              Recheck Cond: ((name)::text ~~ 'a%'::text)
              Heap Blocks: exact=735
              ->  Bitmap Index Scan on gin_all_idx  (cost=0.00..42.73 rows=3030 wid
                    Index Cond: ((name)::text ~~ 'a%'::text)
Planning Time: 0.123 ms
Execution Time: 11.998 ms
```

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

3

Створення індексації

```
CREATE INDEX hash_name_idx
ON "Client" using hash(name)

Create Index hash_taxes_sum_idx
On "Order" using hash(taxes_sum)
```

Запит

```
explain analyse select "Client".id, name, taxes_sum from "Client"
INNER JOIN "Link_Client-Order" links on "Client".id = links.client_id
INNER JOIN "Order" Ord on Ord.id = links.order_id
where name = 'plvpoknmrw' and taxes_sum = 951
```

Результат без індексації (34ms)

```
Nested Loop  (cost=829.67..4239.94 rows=1 width=20) (actual tim
  ->  Merge Join  (cost=829.25..4193.70 rows=10 width=9) (actua
        Merge Cond: (ord.id = links.order_id)
        ->  Index Scan using "Order_pkey" on "Order" ord  (cost
              Filter: (taxes_sum = '951'::numeric)
              Rows Removed by Filter: 102151
        ->  Sort  (cost=828.80..853.82 rows=10005 width=8) (act
              Sort Key: links.order_id
              Sort Method: quicksort  Memory: 900kB
              ->  Seq Scan on "Link_Client-Order" links  (cost=
  ->  Index Scan using "Client_pkey" on "Client"  (cost=0.43..4
        Index Cond: (id = links.client_id)
        Filter: ((name)::text = 'plvpoknmrw'::text)
        Rows Removed by Filter: 1
Planning Time: 0.688 ms
Execution Time: 33.649 ms
```

Результат з індексацією (3ms)

```
Nested Loop  (cost=8.46..202.18 rows=1 width=20) (actual time=(
  ->  Hash Join  (cost=8.03..198.34 rows=1 width=19) (actual t
        Hash Cond: (links.client_id = "Client".id)
        ->  Seq Scan on "Link_Client-Order" links  (cost=0.00.
        ->  Hash  (cost=8.02..8.02 rows=1 width=15) (actual ti
              Buckets: 1024  Batches: 1  Memory Usage: 9kB
              ->  Index Scan using hash_name_idx on "Client"
                    Index Cond: ((name)::text = 'plvpoknmrw'::
  ->  Index Scan using "Order_pkey" on "Order" ord  (cost=0.43
        Index Cond: (id = links.order_id)
        Filter: (taxes_sum = '951'::numeric)
Planning Time: 0.362 ms
Execution Time: 2.438 ms
```

Висновок: оскільки ми використовуємо селектор, де перевіряємо умову знаком рівності, хешування значно прискорює виконання запиту

4

Створення індексації

```
Create Index hash_taxes_sum_idx
On "Order" using hash(taxes_sum)
```

Запит

```
explain analyse select taxes_sum, array_agg(transaction_date) from "Order
where taxes_sum = 100
group by (taxes_sum)
```

Результат без індексації (263ms)

```
GroupAggregate  (cost=1000.00..12798.06 rows=665 width=37) (actual time=258.
  Group Key: taxes_sum
  ->  Gather  (cost=1000.00..12784.29 rows=1091 width=9) (actual time=1.355.
        Workers Planned: 2
        Workers Launched: 2
        ->  Parallel Seq Scan on "Order"  (cost=0.00..11675.19 rows=455 widt
              Filter: (taxes_sum = '100'::numeric)
              Rows Removed by Filter: 366313
Planning Time: 3.711 ms
Execution Time: 263.405 ms
```

Результат з індексацією (3ms)

```
1 GroupAggregate  (cost=32.46..2829.57 rows=665 width=37) (actual time=2.607..2
2   Group Key: taxes_sum
3   ->  Bitmap Heap Scan on "Order"  (cost=32.46..2815.80 rows=1091 width=9) (a
4         Recheck Cond: (taxes_sum = '100'::numeric)
5         Heap Blocks: exact=967
6         ->  Bitmap Index Scan on hash_taxes_sum_idx  (cost=0.00..32.18 rows=1
7               Index Cond: (taxes_sum = '100'::numeric)
8 Planning Time: 0.202 ms
9 Execution Time: 2.651 ms
```

Висновок: оскільки ми використовуємо селектор, де перевіряємо умову знаком рівності, хешування значно прискорює виконання запиту

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 Category, тригер ставить всім записам з таблиці Product, що містили як FK змінений, або видалений запис з Category, загальний Category (category_id = 0).

Текст тригера

```
create function trigger() returns trigger
      language plpgsql
as
$$
DECLARE
      ref_cursor_update NO SCROLL CURSOR FOR Select "Product".id,
```

```
category_id from "Product"
                                        INNER JOIN "Category" C
on C.id = "Product".category_id
                                        where category_id =
New.id
                                        order by "Product".id;

      ref_cursor_delete NO SCROLL CURSOR FOR Select "Product".id,
category_id from "Product"
                                        INNER JOIN "Category" C
on C.id = "Product".category_id
                                        where category_id =
Old.id
                                        order by "Product".id;
      record record;
      BEGIN
      if (TG_OP = 'DELETE') then
            if old.id = 0 then raise EXCEPTION 'You cannot delete
general category';
            end if;

            OPEN ref_cursor_delete;
            LOOP
                  FETCH ref_cursor_delete INTO record;
                  if not found then
                  exit;
                  end if;
                  UPDATE "Product" SET category_id = 0 WHERE
record.id = "Product".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 "Product" SET category_id = 0 WHERE
record.id = "Product".id;
            END LOOP;
            RETURN new;
      end if;
      END;
$$;
```

Команди, що ініціюють виконання тригера

```
Update "Category" SET name = 'trigger_test' Where id = 75857

Delete from "Category" WHERE id = 75857

Delete from "Category" WHERE id = 0
```

Записи до змін:

	id	name	category_id
1	5432	xkvvvwhybbrkmxq	75857
2	6352	kkyefomgnuxtxgd	75857
3	6732	cmacsewcsmascps	75857

	count	category_id
1	20	0
2	8	1
3	3	75857

Записи після змін:

	id	name	category_id

	count	category_id
1	23	0
2	8	1
3	3	1922
4	3	74048