НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування та спеціалізованих комп'ютерних систем

Лабораторна робота №3

з дисципліни
«Бази даних і засоби управління»

Тема: «Засоби оптимізації роботи СУБД PostgreSQL»

Виконав: студент III курсу
ФПМ групи КВ-84
Іванюк В. І.
Перевірив:

Київ – 2020

Метою роботи є здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

Завдання роботи полягає у наступному:

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.

Вимоги до пункту завдання №1

Для перетворення функцій, що реалізують запити до об’єктної бази даних, необхідно встановити бібліотеку sqlAlchemy, налаштувати програму на роботу з ORM, розробити класи-сутності для об’єктів-сутностей, представлених відповідними таблицями БД та пов’язаних зв’язками 1:M, M:M та 1:1 виконати опис схеми бази даних. Особливу увагу приділити контролю зовнішніх зв’язків між таблицями засобами ORM.

Замінити виклики запитів мовою SQL на відповідні запити засобами SQLAlchemy по роботі з об’єктами. Обов’язковим є реалізація вставки, вилучення та редагування екземплярів класів-сутностей. Розробка запитів на генерацію даних та пошук екземплярів класів-сутностей вітається, але не є обов’язковою.

Інтерфейси функцій (вхідні та вихідні аргументи функцій модуля “Модель”) мають залишитись без змін.

Вимоги до пункту завдання №2

Відповідно до варіанту індексування продемонструвати на прикладах запитів SQL SELECT підвищення швидкодії їх виконання з використанням індексів, а також пояснити чому для деяких випадків індексування використовувати недоцільно. При цьому для наочного представлення слід використати функцію генерування рандомізованих даних з лабораторної роботи №2, створивши необхідну кількість тестових даних. Навести 4-5 прикладів запитів SELECT (із виведенням результуючих даних), що містять фільтрацію, агрегатні функції, групування та сортування (у необхідних комбінаціях).

Вимоги до пункту завдання №3

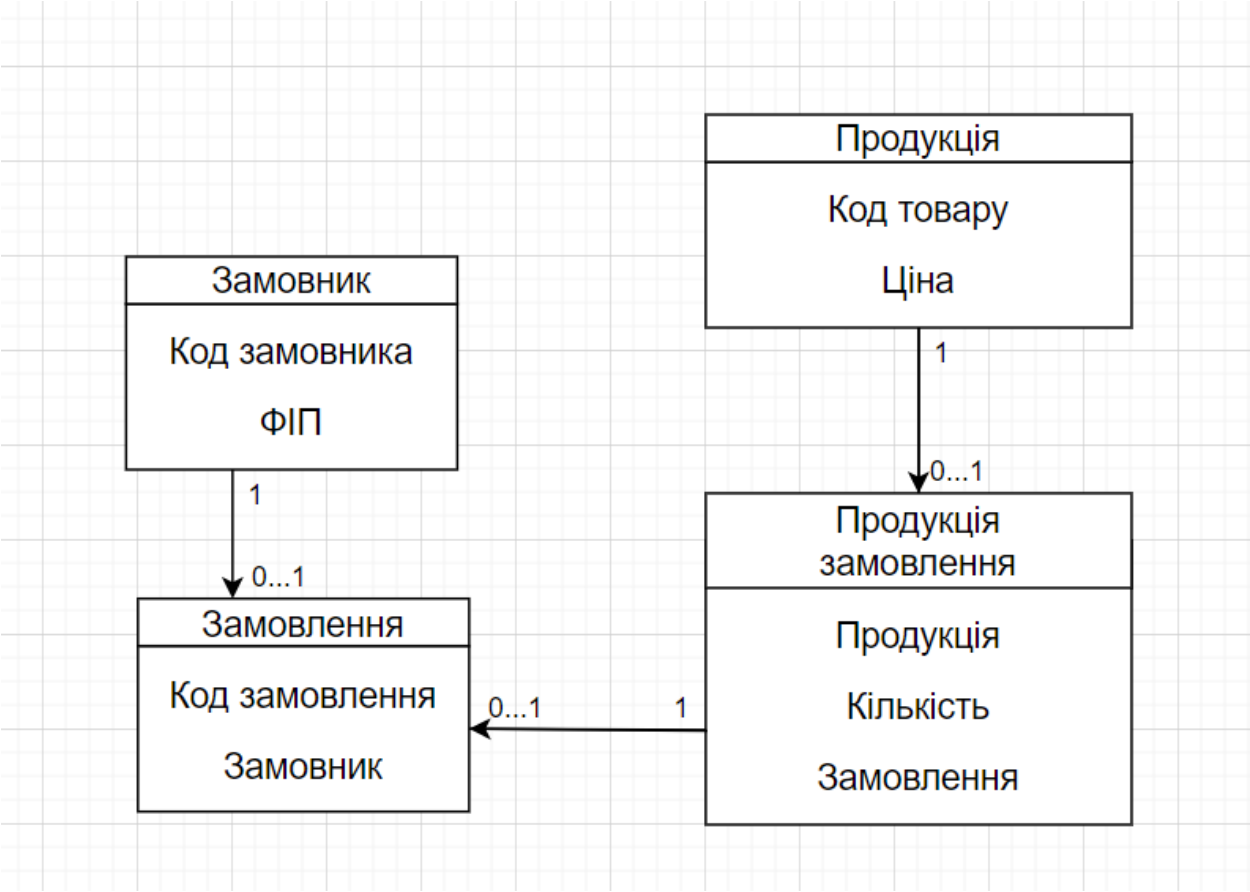
Створити тригер бази даних PostgreSQL відповідно до варіанта. Тригерна функція має включати обробку запису, що модифікується (вставляється або вилучається), умовні оператори, курсорні цикли та обробку виключних ситуацій. Виконати відлагодження тригера при різних вхідних даних, навівши 2-3 приклади його використання.

Варіант 13

13	*BTree, BRIN*	*before insert, delete*

Пункт №1

Схема бази даних у вигляді таблиць та зв'язків між ними



Класи в ORM :

```
Base = declarative_base()

class Customer(Base):
    __tablename__ = 'customer'
    Customer_ID = Column(Integer, primary_key = True)
    Name = Column(String)

class Order(Base):
    __tablename__ = 'order'
    Order_ID = Column(Integer, primary_key = True)
    _Customer = Column(String, ForeignKey('customer.name'))

class Products(Base):
    __tablename__ = 'products'
    Product_ID = Column(Integer, primary_key = True)
    Price = Column(Integer)


class Products_order(Base):
    __tablename__ = 'products_order'
    _order = Column(Integer, primary_key = True)
    _product = Column(Integer)
    quantity = Column(Integer)
```

Приклади запитів :

1. **Вставки**

```
def insert_customer(values: tuple):
    try:
        customer = Customer(Customer_ID = values[0], Name =values[1])
        s.add(customer)
        s.commit()
    except SQLAlchemyError as e:
        print(e)
```

2. **Видалення**

```
def delete(t_name, column, value):
    try:
        t_class = getattr(sys.modules[__name__], t_name.capitalize())
        t_class_col = getattr(t_class, column)
        s.query(t_class).filter(t_class_col == value).delete()
        s.commit()
    except SQLAlchemyError as e:
        print(e)
```

3. Редагування

```
def update(t_name, column, value, cond):
    try:
        t_class = getattr(sys.modules[__name__], t_name.capitalize())
        t_class_col = getattr(t_class, column)
        t_class_cond_col = getattr(t_class, cond[0])
        s.query(t_class).filter(t_class_cond_col == cond[1]).update({t_class_col: value})
        s.commit()
    except SQLAlchemyError as e:
        print(e)
```

Пункт №2

Створення індексу

Btree:

Query Editor | Query History

```
1  CREATE INDEX btree_price ON "Lab3" USING Btree(unit_price)
```

Explain | Data Output | Notifications | **Messages**

```
CREATE INDEX

Query returned successfully in 733 msec.
```

BRIN:

Query Editor | Query History

```
1  create index brin_id on lab3 using BRIN(product_id)
```

Explain | Data Output | **Messages** | Notifications

```
CREATE INDEX

Query returned successfully in 54 msec.
```

Приклад фільтрації даних

Без індексації

Query Editor | Query History

```
1  explain analyze select * from lab3  where unit_price = 33
```

Explain | **Data Output** | Messages | Notifications

	QUERY PLAN text
1	Seq Scan on lab3 (cost=0.00..1791.00 rows=1070 width=12) (actual time=0.015..8.860 rows=1020 loops=1)
2	Filter: (unit_price = 33)
3	Rows Removed by Filter: 98980
4	Planning Time: 1.010 ms
5	Execution Time: 8.894 ms

Btree

Query Editor | Query History

```
1  explain analyze select * from lab3  where unit_price = 33
```

Explain | **Data Output** | Messages | Notifications

	QUERY PLAN text
1	Bitmap Heap Scan on lab3 (cost=20.71..576.57 rows=1070 width=12) (actual time=0.337..0.705 rows=1020 loops=1)
2	Recheck Cond: (unit_price = 33)
3	Heap Blocks: exact=476
4	-> Bitmap Index Scan on btree_price (cost=0.00..20.44 rows=1070 width=0) (actual time=0.297..0.297 rows=1020 loops=1)
5	Index Cond: (unit_price = 33)
6	Planning Time: 9.441 ms
7	Execution Time: 0.755 ms

BRIN

Query Editor | Query History

```
1  explain analyze select * from lab3 where product_id = 33
```

Explain | **Data Output** | Messages | Notifications

	QUERY PLAN text
1	Index Scan using "Lab3_pkey" on lab3 (cost=0.29..8.31 rows=1 width=12) (actual time=0.011..0.011 rows=1 loops=1)
2	Index Cond: (product_id = 33)
3	Planning Time: 1.224 ms
4	Execution Time: 0.019 ms

Результат

Query Editor | Query History

```
1  select * from lab3  where unit_price = 33
```

Explain | **Data Output** | Messages | Notifications

	product_id [PK] integer	unit_price integer	date_of_update date
1	2	33	2021-01-29
2	47	33	2021-03-16
3	137	33	2021-03-11
4	475	33	2021-01-23
5	576	33	2021-03-02
6	662	33	2021-02-19

Приклад з агрегатними функціями

Без індексації

Query Editor | Query History

```
1  explain analyze select min(unit_price) from lab3
```

Explain | **Data Output** | Messages | Notifications

	QUERY PLAN text
1	Aggregate (cost=1791.00..1791.01 rows=1 width=4) (actual time=10.991..10.991 rows=1 loops=1)
2	-> Seq Scan on lab3 (cost=0.00..1541.00 rows=100000 width=4) (actual time=0.009..5.554 rows=100000 loops=1)
3	Planning Time: 0.089 ms
4	Execution Time: 11.009 ms

Btree

Query Editor | Query History

```
1  explain analyze select min(unit_price) from lab3
```

Explain | **Data Output** | Messages | Notifications

	QUERY PLAN text
1	Result (cost=0.47..0.48 rows=1 width=4) (actual time=0.163..0.163 rows=1 loops=1)
2	InitPlan 1 (returns $0)
3	-> Limit (cost=0.42..0.47 rows=1 width=4) (actual time=0.161..0.161 rows=1 loops=1)
4	-> Index Only Scan using btree_price on lab3 (cost=0.42..5026.13 rows=100000 width=4) (actual time=0.160..0.160 rows=1 loops=1)
5	Index Cond: (unit_price IS NOT NULL)
6	Heap Fetches: 1
7	Planning Time: 0.098 ms
8	Execution Time: 0.181 ms

BRIN

Query Editor | Query History

```
1  explain analyze select min(product_id) from lab3
```

Explain | **Data Output** | Messages | Notifications

	QUERY PLAN text
1	Result (cost=0.33..0.34 rows=1 width=4) (actual time=0.097..0.097 rows=1 loops=1)
2	InitPlan 1 (returns $0)
3	-> Limit (cost=0.29..0.33 rows=1 width=4) (actual time=0.095..0.095 rows=1 loops=1)
4	-> Index Only Scan using "Lab3_pkey" on lab3 (cost=0.29..3398.29 rows=100000 width=4) (actual time=0.094..0.094 ...
5	Index Cond: (product_id IS NOT NULL)
6	Heap Fetches: 1
7	Planning Time: 0.943 ms
8	Execution Time: 0.113 ms

Результат

Query Editor | Query History

```
1  select min(unit_price) from lab3
```

Explain | **Data Output** | Messages | Notifications

	min integer
1	1

Приклад з сортуванням

Без індексації

Query Editor | Query History

```
1  explain analyze select * from lab3 order by unit_price
```

Explain | **Data Output** | Messages | Notifications

	QUERY PLAN text
1	Sort (cost=9845.82..10095.82 rows=100000 width=12) (actual time=50.094..57.082 rows=100000 loops=1)
2	Sort Key: unit_price
3	Sort Method: external merge Disk: 2160kB
4	-> Seq Scan on lab3 (cost=0.00..1541.00 rows=100000 width=12) (actual time=0.009..5.647 rows=100000 loops=1)
5	Planning Time: 0.051 ms
6	Execution Time: 60.203 ms

Btree

Query Editor | Query History

```
1  explain analyze select * from lab3 order by unit_price
```

Explain | **Data Output** | Messages

	QUERY PLAN text
1	Index Scan using btree_price on lab3 (cost=0.42..4776.13 rows=100000 width=12) (actual time=0.019..22.163 rows=100000 loops=1)
2	Planning Time: 0.156 ms
3	Execution Time: 23.983 ms

BRIN

Query Editor | Query History

```
1  explain analyze select * from lab3 order by product_id
```

Explain | **Data Output** | Messages | Notifications

	QUERY PLAN text
1	Index Scan using "Lab3_pkey" on lab3 (cost=0.29..3148.29 rows=100000 width=12) (actual time=0.011..10.796 rows=100000 loops=1)
2	Planning Time: 0.057 ms
3	Execution Time: 12.724 ms

Результат

Query Editor | Query History

```
1  select * from lab3 order by unit_price
```

Explain | **Data Output** | Messages | Notifications

	product_id [PK] integer	unit_price integer	date_of_update date
1	35454	1	2021-02-12
2	33632	1	2021-03-13
3	14592	1	2021-02-18
4	57580	1	2021-04-17
5	12216	1	2021-04-09

Приклад з агрегатними функціями та сортуванням

Без індексації

Query Editor | Query History

```
1  explain analyze select min(unit_price) from lab3 group by unit_price
```

Explain | **Data Output** | Messages | Notifications

	QUERY PLAN text
1	HashAggregate (cost=2041.00..2042.00 rows=100 width=8) (actual time=19.117..19.123 rows=100 loops=1)
2	Group Key: unit_price
3	-> Seq Scan on lab3 (cost=0.00..1541.00 rows=100000 width=4) (actual time=0.009..5.506 rows=100000 loops=1)
4	Planning Time: 0.060 ms
5	Execution Time: 19.160 ms

Btree

Query Editor | Query History

```
1  explain analyze select min(product_id) from lab3 group by product_id
```

Explain | **Data Output** | Messages | Notifications

	QUERY PLAN text
1	GroupAggregate (cost=0.29..4648.29 rows=100000 width=8) (actual time=0.160..43.690 rows=100000 loops=1)
2	Group Key: product_id
3	-> Index Only Scan using btree_price on lab3 (cost=0.29..3148.29 rows=100000 width=4) (actual time=0.156..24.825 rows=...
4	Heap Fetches: 100000
5	Planning Time: 0.069 ms
6	Execution Time: 46.002 ms

BRIN

Query Editor | Query History

```
1  explain analyze select count(product_id) from lab3 group by product_id
```

Explain | **Data Output** | Messages | Notifications

	QUERY PLAN text
1	GroupAggregate (cost=0.29..4648.29 rows=100000 width=12) (actual time=0.117..39.043 rows=100000 loops=1)
2	Group Key: product_id
3	-> Index Only Scan using "Lab3_pkey" on lab3 (cost=0.29..3148.29 rows=100000 width=4) (actual time=0.113..16.551 rows=10...
4	Heap Fetches: 100000
5	Planning Time: 1.579 ms
6	Execution Time: 41.418 ms

Результат

```
1  select min(product_id) from lab3 group by product_id
```

Explain | **Data Output** | Messages | Notifications

	min integer
1	1
2	2
3	3
4	4
5	5

Висновки :

1. При фільтрації обидва індекси чудово виконують свою роботу в порівняні з послідовним пошуком при відсутності індексів, хоча BRIN спрацював швидше.

2. При використанні агрегатних функцій, використання індексів залежить від агрегатної функції. Наприклад, використання функції min дає змогу використовувати Btree індекс та збільшує швидкість пошуку оскільки необхідно лише дійти до крайнього лівого «листка». Використання функції count не дає змогу використати Btree індекс тому що в результаті маємо лише один запис. Але при цьому індекс BRIN може працювати при використанні цієї функції.

3. При сортуванні запити з індексами працюють ефективніше ніж без них. BRIN спрацював швидше, ніж Btree.

4. При сортуванні та використанні агрегатних функцій обидва індекси виявились неефективними та спрацювали повільніше аніж запит без індеків.

Пункт №3

Команди, що ініціюють виконання тригера, код тригера:

Query Editor Query History

```
CREATE OR REPLACE FUNCTION lab3_f() RETURNS TRIGGER AS $lab3_f$
    BEGIN
        IF(TG_OP = 'INSERT') THEN
            BEGIN
                IF (
                    (SELECT count(*) from lab3 WHERE id = ANY(NEW.prod_id::int[]))
                    !=
                    array_length(NEW.prod_id::int[], 1)
                ) THEN
                    RAISE EXCEPTION 'price must exist';
                END IF;
                RETURN NEW;
            END;
        ELSIF (TG_OP = 'DELETE') THEN
            DECLARE
            del_cur CURSOR FOR
            SELECT * FROM lab3 WHERE OLD.id = ANY(prod_id::int[]);
                BEGIN
                FOR record IN del_cur LOOP
                    BEGIN
                        UPDATE lab3 SET unit_price = array_remove(prod_id, OLD.id) WHERE id = record.id;
                    END;
                END LOOP;
            END;
            RETURN OLD;
        END IF;
    END;
$lab3_f$ LANGUAGE plpgsql;
```

Explain Data Output **Messages** Notifications

```
CREATE FUNCTION

Query returned successfully in 57 msec.
```

Query Editor Query History

```
CREATE TRIGGER t_product
BEFORE INSERT OR DELETE
ON lab3
FOR EACH ROW
EXECUTE PROCEDURE lab3_f();
```

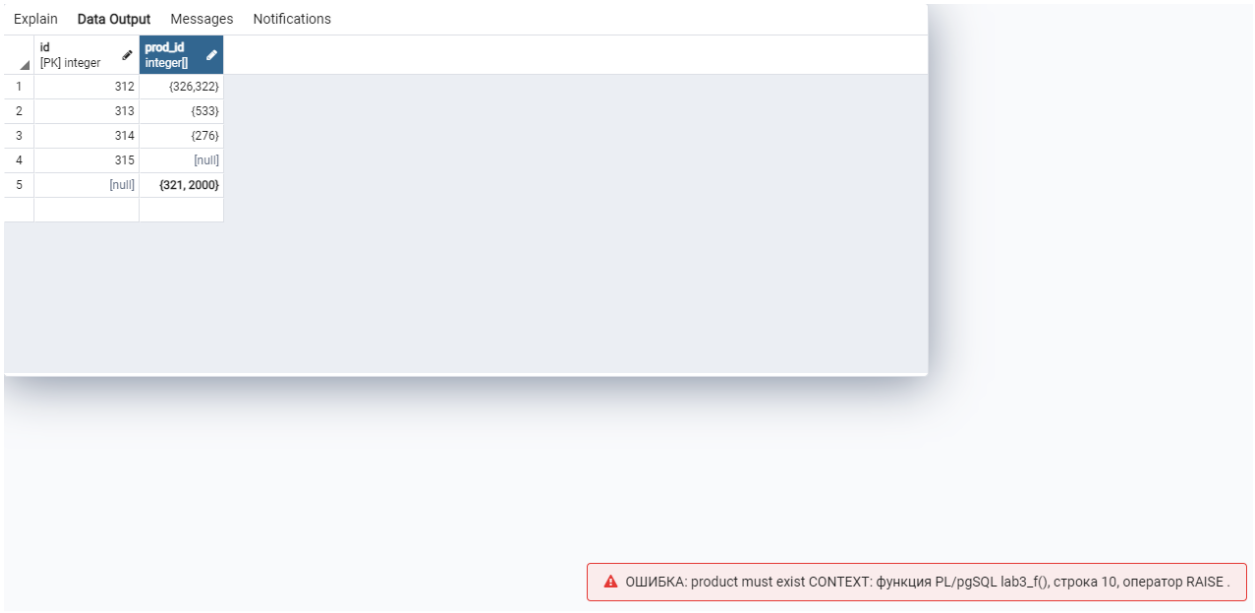
Explain Data Output **Messages** Notifications

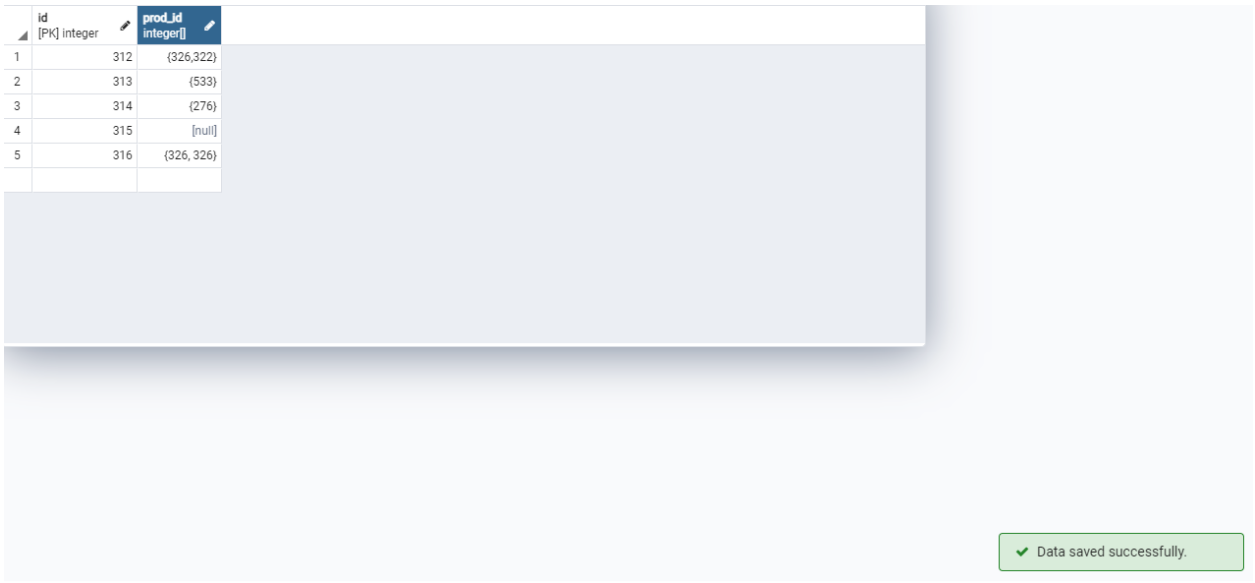
```
CREATE TRIGGER

Query returned successfully in 48 msec.
```

Таблиця :

	id [PK] integer	prod_id integer[]
1	312	{326,322}
2	313	{533}
3	314	{276}
4	315	[null]

1. Якщо додається новий запис до таблиці, то усі id що містяться у prod_id мають існувати.





2. При видаленні запису, записи що містили у prod_id id видаляємого запису, змінять вміст prod_id на такий самий масив, але вже без id видаляємого запису

	id [PK] integer	prod_id integer[]
1	312	{326,322}
2	326	{276}
3	315	[null]
4	316	{326,326}

```
1  DELETE from lab WHERE id=326
```

Explain Data Output **Messages** Notifications

```
DELETE 1
```

	id [PK] integer	prod_id integer[]
1	312	{322}
2	315	[null]
3	316	{}