МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

Лабораторна робота № 3

з дисципліни “Реляційні БД”

тема “Засоби оптимізації роботи СУБД PostgreSQL”

Варіант 17

Виконав

студент III курсу

групи КП-81

Подлеснюк Богдан Анатолійович

Зарахована

“____” “_____” 20___ р.

викладачем

Радченко К.О.

Київ 2020

Завдання

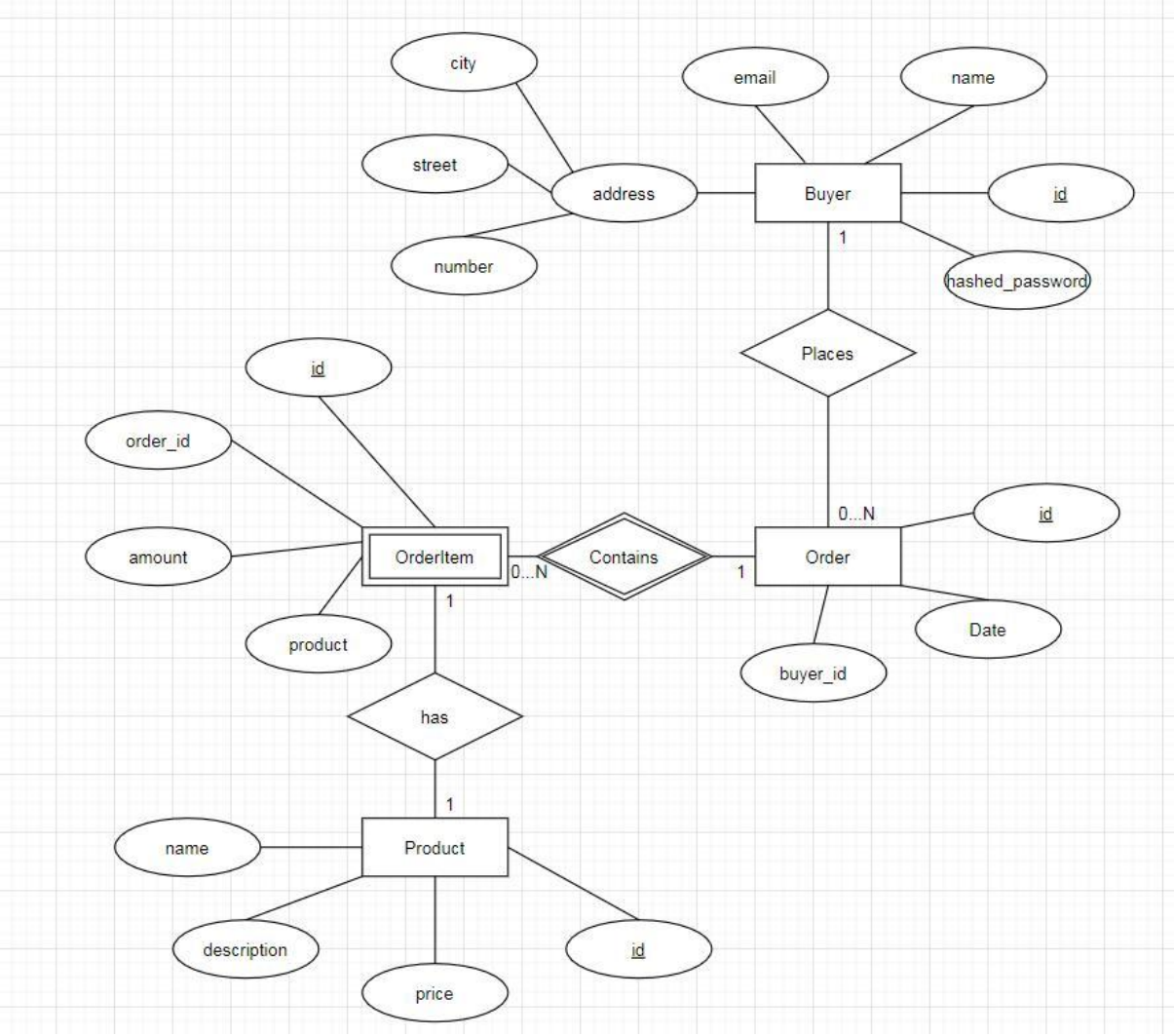
Завдання роботи полягає у наступному:

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.

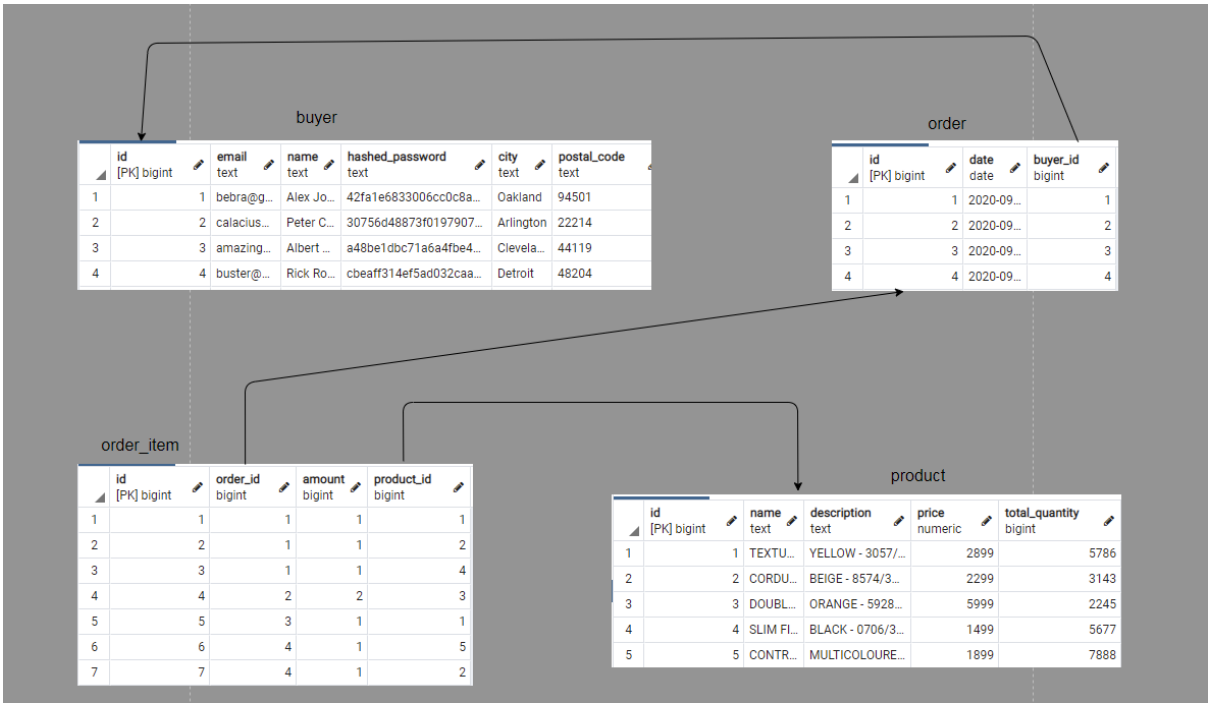
17	*GIN, BRIN*	*before update, delete*

Завдання 1

Графічна ER модель(нотація Чена)



Зв’язки між таблицями:



Сутності БД:

Relation	Attribute	Data type
buyer	`id` - unique attribute of every user, primary key `name` - name of the user `email` - email of the buyer `hashed_password` - password of the buyer saved in hash form `city` - city of the buyer `postal_code` - postal code of the buyer	bigint text text text text text
product	`id` - unique attribute of the product `name` - name of clothes `description` - description of clothes, its color, sizes available and charasteristics `price` - asking price of the product `total_quantity` - total quantity of the product	bigint text text numeric bigint
order	`id` - unique attribute of every order `date` - cut-off date when an order is closed `buyer_id` - id of the buyer, who placed the order	bigint date bigint
order_item	`id` - unique attribute of the order item `order_id` - id of the order, to which this item belongs `product_id` - id of the product, which lays in this order item `amount` - the total number or quantity of the product	bigint bigint bigint bigint

Меню додатку:

```
Select the table to work with | command:

    1 - buyer
    2 - order_item
    3 - order
    4 - product
    5 - Fill table "product" by random data (1000 items)
    6 - commit
    7 - Exit
```

```
GET buyer
Enter condition (SQL) or leave empty:
id | email | name | hashed_password | city | postal_code


Table :  `buyer`

<Buyer>{'postal_code': '94501', 'hashed_password': '42fa1e6833086cc8c8a242080d08be47c36ecaa4411855b89f7b2ca8fc4a9055', 'email': 'bebra@gmail.com', 'city': 'Oakland', 'name': 'Alex Johnson', 'id': 1})
<Buyer>{'postal_code': '22214', 'hashed_password': '30756d48873f81979076cd9e18c77da90384e2a5c5833979e325a46194cfcd84', 'email': 'calaciuspeter@gmail.com', 'city': 'Arlington', 'name': 'Peter Calaska', 'id': 2})
<Buyer>{'postal_code': '44119', 'hashed_password': 'a48be1dbc71a6a4fbe4d60fd8b5e1e2804a07d4acbd4092d860e85263febbbd9', 'email': 'amazingsauce@gmail.com', 'city': 'Cleveland', 'name': 'Albert Raymond', 'id': 3})
<Buyer>{'postal_code': '48204', 'hashed_password': 'cbeaff314ef5ad032caa60ee2e8d8144ae52a8572c7d6f75631f3bd4088a7b16', 'email': 'buster@gmail.com', 'city': 'Detroit', 'name': 'Rick Roller', 'id': 4})
<Buyer>{'postal_code': '13456', 'hashed_password': 'adadad8ady7ada76a', 'email': 'test@gmail.com', 'city': 'Tsss', 'name': 'Tes', 'id': 5})
```

Класи в моделі:

```python
class Buyer(Base, Repr):
    __tablename__ = 'buyer'

    id = Column(Integer, primary_key=True)
    email = Column(String)
    name = Column(String)
    hashed_password = Column(String)
    city = Column(String)
    postal_code = Column(String)

    orders = relationship('Order')

    def __init__(self, email=None, name=None, hashed_password=None, city=None, postal_code=None):
        self.email = email
        self.name = name
        self.hashed_password = hashed_password
        self.city = city
        self.postal_code = postal_code
```

```python
class Order(Base, Repr):
    __tablename__ = 'order'

    id = Column(Integer, primary_key=True)
    date = Column(Date)
    buyer_id = Column(Integer, ForeignKey('buyer.id'))

    order_items = relationship('OrderItem')

    def __init__(self, date=None, buyer_id=None):
        self.date = date
        self.buyer_id = buyer_id
```

```python
class OrderItem(Base, Repr):
    __tablename__ = 'order_item'

    id = Column(Integer, primary_key=True)
    order_id = Column(Integer, ForeignKey('order.id'))
    amount = Column(Integer)
    product_id = Column(Integer, ForeignKey('product.id'))

    def __init__(self, order_id=None, amount=None, product_id=None):
        self.order_id = order_id
        self.amount = amount
        self.product_id = product_id
```

```python
class Product(Base, Repr):
    __tablename__ = 'product'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    description = Column(String)
    price = Column(Integer)
    total_quantity = Column(Integer)

    order_items = relationship('OrderItem')

    def __init__(self, name=None, description=None, price=None, total_quantity=None):
        self.name = name
        self.description = description
        self.price = price
        self.total_quantity = total_quantity
```

Запити ORM:

```python
def get(self, tableName):
    try:
        condition = getInput(
            f'GET {tableName}\nEnter condition (SQL) or leave empty:', tableName)
        data = self.model.get(tableName, condition)
        self.view.print_entities(tableName, data)
        pressEnter()
        self.show_entity_menu(tableName)
    except Exception as err:
        self.show_entity_menu(tableName, str(err))

def insert(self, tableName):
    try:
        columns, values = getInsertInput(
            f"INSERT {tableName}\nEnter columns divided with commas, then do the same for values in format: ['value1', 'value2', ...]", tableName)
        self.model.insert(tableName, columns, values)
        self.show_entity_menu(tableName, 'Insert is successful!')
    except Exception as err:
        self.show_entity_menu(tableName, str(err))

def delete(self, tableName):
    try:
        condition = getInput(
            f'DELETE {tableName}\n Enter condition (SQL):', tableName)
        self.model.delete(tableName, condition)
        self.show_entity_menu(tableName, 'Delete is successful')
    except Exception as err:
        self.show_entity_menu(tableName, str(err))

def update(self, tableName):
    try:
        condition = getInput(
            f'UPDATE {tableName}\nEnter condition (SQL):', tableName)
        statement = getInput(
            "Enter SQL statement in format [<key>='<value>']", tableName)

        self.model.update(tableName, condition, statement)
        self.show_entity_menu(tableName, 'Update is successful')
    except Exception as err:
        self.show_entity_menu(tableName, str(err))
```

Завдання 2

Індекси

Порядок звертання до таблиці без використання фільтру по колонці, на яку додано індекс (створений індекс не використовується):

```
1  EXPLAIN ANALYZE SELECT * FROM product
2
```

	QUERY PLAN text
1	Seq Scan on product (cost=0.00..20.19 rows=1019 width=48) (actual time=0.058..0.268 rows=1019 loops=1)
2	Planning Time: 0.877 ms
3	Execution Time: 0.381 ms

Пошук з фільтрацією(без індексів):

```
1  EXPLAIN ANALYZE SELECT * FROM product WHERE id < 1000
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Seq Scan on product (cost=0.00..35.74 rows=987 width=71) (actual time=0.014..0.177 rows=987 loops=1)
2	Filter: (id < 1000)
3	Rows Removed by Filter: 32
4	Planning Time: 0.095 ms
5	Execution Time: 0.213 ms

Пошук з фільтрацією та сортуванням(без індексів):

```
1  EXPLAIN ANALYZE SELECT * FROM product WHERE id < 1000 ORDER BY id
```

	QUERY PLAN text
1	Index Scan using product_pkey on product (cost=0.28..87.55 rows=987 width=71) (actual time=0.013..0.198 rows=...
2	Index Cond: (id < 1000)
3	Planning Time: 0.080 ms
4	Execution Time: 0.235 ms

Створення GIN індексу:

```
1  ALTER TABLE product
2      ADD COLUMN ts_vector tsvector;
3
4  UPDATE product
5  SET ts_vector = to_tsvector(name)
6  WHERE true;
7
8  CREATE INDEX ginIndex ON product USING gin (ts_vector);
9
10
```

Порядок звертання до таблиці з використанням фільтру по колонці, на яку додано індекс (пошук відбувається за допомогою створеного індексу):

```
1  EXPLAIN ANALYZE SELECT * FROM product WHERE to_tsquery('a') @@ ts_vector;
```

	QUERY PLAN text
1	Bitmap Heap Scan on product (cost=0.00..4.26 rows=1 width=71) (actual time=0.048..0.049 rows=0 loops=1)
2	Recheck Cond: (to_tsquery('a'::text) @@ ts_vector)
3	-> Bitmap Index Scan on ginindex (cost=0.00..0.00 rows=1 width=0) (actual time=0.047..0.047 rows=0 loops=1)
4	Index Cond: (to_tsquery('a'::text) @@ ts_vector)
5	Planning Time: 0.333 ms
6	Execution Time: 0.070 ms

Створення BRIN індексу

```
1  CREATE INDEX brin ON product using brin(id)
```

Запит з фільтрацією(brin):

```
1  EXPLAIN ANALYZE SELECT * FROM product WHERE id < 1000
```

	QUERY PLAN text
1	Seq Scan on product (cost=0.00..35.74 rows=987 width=71) (actual time=0.011..0.140 rows=987 loops=1)
2	Filter: (id < 1000)
3	Rows Removed by Filter: 32
4	Planning Time: 0.242 ms
5	Execution Time: 0.175 ms

Запит з фільтрацією та фільтруванням(brin):

```
1  EXPLAIN ANALYZE SELECT * FROM product WHERE id < 1000 ORDER BY id
```

	QUERY PLAN text
1	Index Scan using product_pkey on product (cost=0.28..87.55 rows=987 width=71) (actual time=0.068..0.186 rows=...
2	Index Cond: (id < 1000)
3	Planning Time: 0.086 ms
4	Execution Time: 0.208 ms

Висновки: BRIN та GIN покращують швидкодію на достатньо великому наборі даних, в порівнянні з пошуком без індексів

Випадки коли має сенс використовувати індекси кожного типа:

B-Tree - For most datatypes and queries

GIN - For JSONB/hstore/arrays

GiST - For full text search and geospatial datatypes

SP-GiST - For larger datasets with natural but uneven clustering

BRIN - For really large datasets that line up sequentially

Hash - For equality operations, and generally B-Tree still what you want here

Завдання 3
Тригери

Створимо тригер, який буде перевіряти валідність даних перед оновленням

```
CREATE FUNCTION product_func() RETURNS trigger AS $product_func$
    BEGIN
        IF NEW.name IS NULL THEN
            RAISE EXCEPTION 'name cannot be null';
        END IF;
        IF NEW.price IS NULL THEN
            RAISE EXCEPTION '% cannot have null price', NEW.name;
        END IF;

        IF NEW.price < 0 THEN
            RAISE EXCEPTION '% cannot have a negative price', NEW.name;
        END IF;

        RETURN NEW;
    END;
$product_func$ LANGUAGE plpgsql;

CREATE TRIGGER product_trigger BEFORE UPDATE ON product
    FOR EACH ROW EXECUTE PROCEDURE product_func();
```

Data Output | Explain | Messages | Notifications

```
CREATE TRIGGER

Query returned successfully in 126 msec.
```

Приклади невалідних запитів:

Query Editor | Query History

```
UPDATE public.product
SET name = NULL, description = 'desc', price = -1
WHERE id = 6;
```

```
ERROR:  name cannot be null
CONTEXT:  PL/pgSQL function product_func() line 4 at RAISE
SQL state: P0001
```

```sql
UPDATE public.product
SET name = 'name', description = 'desc', price = NULL
WHERE id = 6;
```

```
ERROR:  name cannot have null price
CONTEXT:  PL/pgSQL function product_func() line 7 at RAISE
SQL state: P0001
```

```sql
UPDATE public.product
SET name = 'name', description = 'desc', price = -1
WHERE id = 6;
```

```
ERROR:  name cannot have a negative price
CONTEXT:  PL/pgSQL function product_func() line 11 at RAISE
SQL state: P0001
```

Тригер для валідаціїї перед видаленням:

```sql
CREATE OR REPLACE FUNCTION before_delete()
  RETURNS trigger AS $before_delete$
BEGIN
    IF OLD.total_quantity > 0 THEN
        RAISE EXCEPTION 'cannot delete % product with quantity(%) left', OLD.name, OLD.total_quantity;
    END IF;
    RETURN NULL;
END;
$before_delete$ LANGUAGE 'plpgsql';

CREATE TRIGGER before_delete BEFORE DELETE ON product
    FOR EACH ROW EXECUTE PROCEDURE before_delete();
```

Приклад невалідного запиту:

```sql
DELETE FROM product
WHERE id = 23;
```

```
ERROR:  cannot delete 2ffeb50040 product with quantity(6744) left
CONTEXT:  PL/pgSQL function before_delete() line 4 at RAISE
SQL state: P0001
```

Висновки

В результаті виконання лабораторної роботи було опановано способи оптимізації додатку для роботи з базою даних за допомогою ORM, досліджено вплив використання індексів, створено і протестовано тригери.