НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
Кафедра системного програмування та спеціалізованих комп'ютерних систем

Лабораторна робота № 2

з дисципліни

«Бази даних та засоби управління»

Тема: «Засоби оптимізації роботи СУБД PostgreSQL»

Виконала: студентка групи КВ-11
Петрук Ольга

Telegram: @olyaaaaaaaaaaaaaaaaa

Київ – 2023

Проектування бази даних та ознайомлення з базовими операціями СУБД PostgreSQL

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

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.

Посилання на Github: https://github.com/kebabgirl/db.git

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.

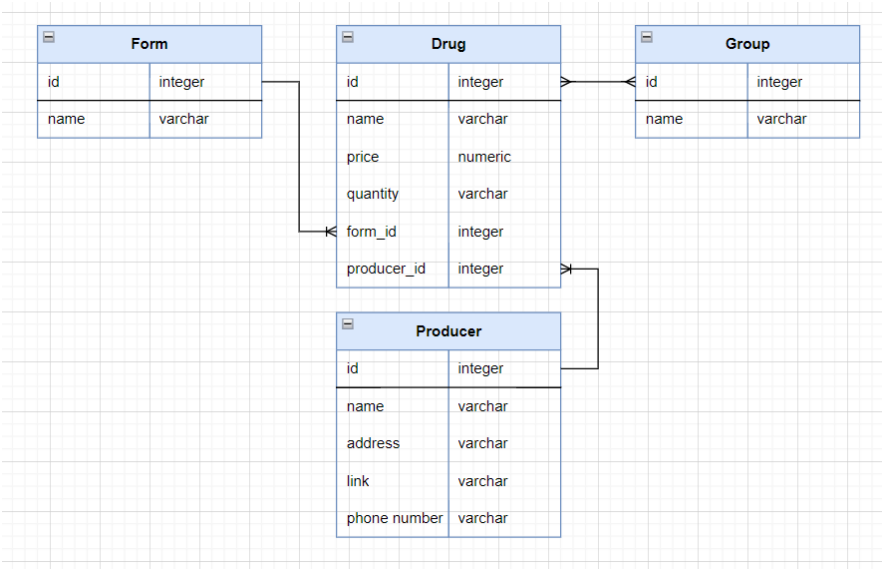
Вимоги до пункту завдання №4

Проаналізувати на прикладах використання рівнів ізоляції транзакцій READ COMMITTED, REPEATABLE READ та SERIALIZABLE, продемонструвавши феномени, які виникають, і спосіб їх уникнення завдяки встановленню відповідного рівня ізоляції транзакцій. Для виконання завдання необхідно відкрити дві транзакції у різних вікнах pgAdmin4 і виконати послідовність запитів INSERT, UPDATE або DELETE у обох транзакціях, що доводять наявність або відсутність певних феноменів.

Варіант №20

№ варіанта	*Види індексів*	*Умови для тригера*
20	*GIN, BRIN*	*after insert, update*

Скріншот розробленої моделі «сутність-зв’язок» предметної галузі «Довідник медичних препаратів»:



Сутності з описом призначення:

Предметна галузь «Електронний довідник медичних препаратів» включає в себе 4 сутності, кожна сутність містить декілька атрибутів:

1. Drug (drug_id, name, price, quantity, form_id, producer_id).
2. Group (group_id, name).
3. Form (form_id, name).
4. Producer (producer_id, name, address, link, phone_number).

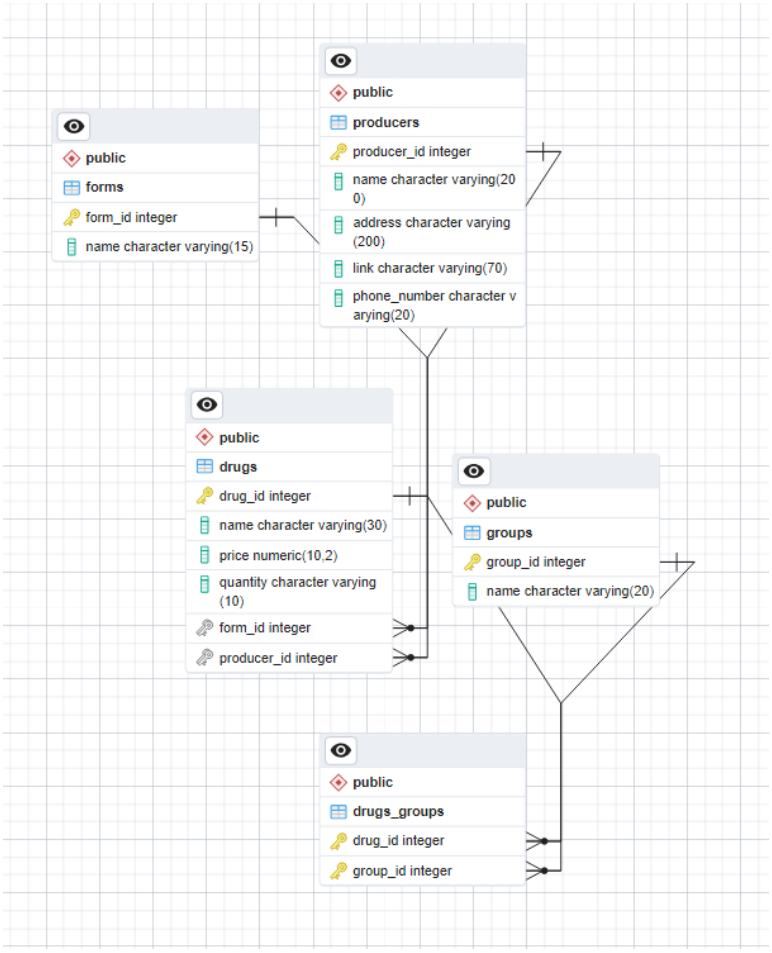
Сутність Drug описує медичні препарати, які є в довіднику. Кожен препарат містить свій унікальний id, назву, ціну, обсяг, id своєї форми та id свого виробника.

Сутність Group відповідає за групи медичних препаратів. У кожної групи є свій унікальний id та назва.

Сутність Form відповідає за форми медичних препаратів. У кожної форми є свій унікальний id та назва.

Сутність Producer описує виробників медичних препаратів. У кожного виробника є свій унікальний id, назва, адреса, посилання та номер телефону. У виробника може бути багато медичних препаратів, але для кожного препарату він є єдиним.

Схема бази даних у графічному вигляді:



Завдання №1

У даній лабораторній роботі було реалізовано 5 класів відповідно до 5 існуючих таблиць у розробленій базі даних, а саме:

1. Drug
2. Producer
3. Form
4. Group
5. Drug_Group

Drug

Таблиця Drugs має такі стовпці: drug_id (ідентифікатор препарату), name (назва препарату), price (ціна препарату), quantity (кількість препарату), form_id (зовнішній ключ, який пов'язує препарат із його формою), producer_id (зовнішній ключ, який пов'язує препарат із його виробником). Також наявний зв'язок із таблицею Drugs_Groups, тому в цьому класі встановлений зв'язок relationship ("Drug_Group").

Програмна реалізація класу Drug:

```
class Drug(Base):
    __tablename__ = 'drugs'
    drug_id = Column(Integer, primary_key=True)
    name = Column(String)
    price = Column(Numeric)
    quantity = Column(String)

    form_id = Column(Integer, ForeignKey('forms.form_id'))
    producer_id= Column(Integer, ForeignKey('producers.producer_id'))

    drug_group = relationship("Drug_Group")

    def __init__(self, name, price, quantity, form_id, producer_id):
        self.name = name
        self.price = price
        self.quantity = quantity
        self.form_id = form_id
        self.producer_id = producer_id

    def __repr__(self):
        return f"<Drugs(drug_id={self.drug_id}, name={self.name},
price={self.price}, quantity={self.quantity}, form_id={self.form_id},
producer_id={self.producer_id})>"
```

Producer

Таблиця Producers має такі стовпці: producer_id (ідентифікатор виробника), name (назва виробника), address (адреса виробника), link (сайт виробника), phone_number (номер телефону виробника). Також наявний зв'язок із таблицею Drugs, тому в цьому класі встановлений зв'язок relationship ("Drug").

Програмна реалізація класу Producer:

```
class Producer(Base):
    __tablename__ = 'producers'
    producer_id = Column(Integer, primary_key=True)
    name = Column(String)
    address = Column(String)
    link = Column(String)
    phone_number = Column(String)

    drug = relationship("Drug")

    def __init__(self, name, address, link, phone_number):
        self.name = name
        self.address = address
        self.link = link
        self.phone_number = phone_number

    def __repr__(self):
        return f"<Producers(producer_id={self.producer_id}, name={self.name},
address={self.address}, link={self.link}, phone_number={self.phone_number})>"
```

Form

Таблиця Forms має такі стовпці: form_id (ідентифікатор форми), name (назва форми). Також наявний зв’язок із таблицею Drugs, тому в цьому класі встановлений зв’язок relationship (“Drug”).

Програмна реалізація класу Form:

```
class Form(Base):
    __tablename__ = 'forms'
    form_id = Column(Integer, primary_key=True)
    name = Column(String)

    drug = relationship("Drug")

    def __init__(self, name):
        self.name = name

    def __repr__(self):
        return f"<Forms(form_id={self.form_id}, name={self.name})>"
```

Group

Таблиця Groups має такі стовпці: group_id (ідентифікатор групи), name (назва групи). Також наявний зв’язок із таблицею Drugs_Groups, тому в цьому класі встановлений зв’язок relationship (“Drug_Group”).

Програмна реалізація класу Group:

```
class Group(Base):
    __tablename__ = 'groups'
    group_id = Column(Integer, primary_key=True)
    name = Column(String)

    drug_group = relationship("Drug_Group")
```

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

    def __repr__(self):
        return f"<Groups(group_id={self.group_id}, name={self.name})>"
```

Drug_Group

Таблиця Drugs_Groups має такі стовпці: drug_id (зовнішній ключ, який посилається на препарат), group_id (зовнішній ключ, який посилається на групу).

Програмна реалізація класу Drug_Group:

```
class Drug_Group(Base):
    __tablename__ = 'drugs_groups'
    drug_id = Column(Integer, ForeignKey('drugs.drug_id'), primary_key=True)
    group_id = Column(Integer, ForeignKey('groups.group_id'),
primary_key=True)

    def __init__(self, drug_id, group_id):
        self.drug_id = drug_id
        self.group_id = group_id

    def __repr__(self):
        return f"<Drugs_Groups(drug_id={self.drug_id},
group_id={self.group_id})>"
```

Меню виглядає наступним чином і включає в себе 5 пунктів. Розглянемо їх далі.

```
Меню:
1. Додати рядок
2. Показати таблицю
3. Редагувати рядок
4. Видалити рядок
5. Вихід
Виберіть пункт:
```

Пункт №1 – «Додати рядок»

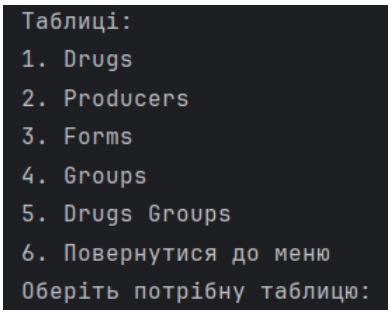
Після вибору цього пункту, відкривається список доступних таблиць, де потрібно вибрати таблицю, до якої бажаємо додати рядок:

```
Таблиці:
1. Drugs
2. Producers
3. Forms
4. Groups
5. Drugs Groups
6. Повернутися до меню
Оберіть потрібну таблицю:
```

Після вибору таблиці, користувачу потрібно ввести всі необхідні дані для нового рядка, після чого буде повідомлено про успішну дію або помилку.

Пункт №2 – «Показати таблицю»

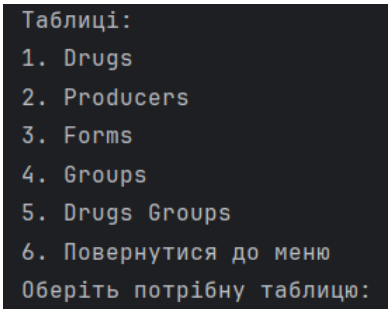
Після вибору цього пункту, відкривається список доступних таблиць, де потрібно вибрати таблицю, яку бажаємо побачити:



Після вибору таблиці, мають вивестися всі рядки і стовпці з обраної таблиці БД.

Пункт №3 – «Редагувати рядок»

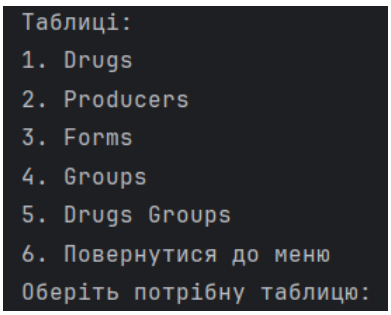
Після вибору цього пункту, відкривається список доступних таблиць, де потрібно вибрати таблицю, в якій бажаємо зробити зміну:



Після вибору таблиці, користувачу потрібно ввести ідентифікатор існуючого рядку в таблиці. Потім відповідно внести нові дані, після чого буде повідомлено про успішну дію або помилку.

Пункт №4 – «Видалити рядок»

Після вибору цього пункту, відкривається список доступних таблиць, де потрібно вибрати таблицю, в якій бажаємо видалити рядок:



Після вибору таблиці, користувачу потрібно ввести ідентифікатор існуючого рядку в таблиці, після чого рядок буде видалено в разі успіху або повідомлено про помилку.

Пункт №5 – «Вихід»

Пункт виходу з програми: закривається з’єднання і програма завершується.

Приклади запитів у вигляді ORM

Для демонстрації запитів виберемо по 1-2 таблиці до кожного.

Запити вставки реалізовані за допомогою функцій insert. Спочатку в меню користувач обирає опцію додавання, далі обирає таблицю, до якої хоче додати рядок і вводить необхідні дані.

Таблиця “drugs” до вставки:

	drug_id [PK] integer	name character varying (30)	price numeric (10,2)	quantity character varying (10)	form_id integer	producer_id integer
1	100017	drug_name2	333.00	333ml	3	3
2	100016	Drug2	1.00	1p	1	1
3	100015	IP	58.65	82	2	1
4	100014	YK	93.70	32	7	1
5	100013	KQ	1.66	96	5	4
6	100012	KS	99.73	81	3	3
7	100011	CL	36.26	13	6	2
8	100010	VH	0.15	29	2	3
9	100009	EH	29.75	93	7	2
10	100008	WP	18.75	9	6	4

```
Меню:
1. Додати рядок
2. Показати таблицю
3. Редагувати рядок
4. Видалити рядок
5. Вихід
Виберіть пункт: 1

Таблиці:
1. Drugs
2. Producers
3. Forms
4. Groups
5. Drugs Groups
6. Повернутися до меню
Оберіть потрібну таблицю: 1

Adding drug:
Enter drug name: тест1
Enter drug price: 88,88
It must be a number.
Enter drug price: 88.88
Enter drug quantity: тест1
Enter drug form_id: 1
Enter drug producer_id: 1
Drug added successfully!
```

Таблиця “drugs” після вставки:

	drug_id [PK] integer	name character varying (30)	price numeric (10,2)	quantity character varying (10)	form_id integer	producer_id integer
1	100019	тест1	88.88	тест1	1	1
2	100017	drug_name2	333.00	333ml	3	3
3	100016	Drug2	1.00	1p	1	1
4	100015	IP	58.65	82	2	1
5	100014	YK	93.70	32	7	1
6	100013	KQ	1.66	96	5	4
7	100012	KS	99.73	81	3	3
8	100011	CL	36.26	13	6	2
9	100010	VH	0.15	29	2	3
10	100009	EH	29.75	93	7	2

Лістинг функцій insert для кожної таблиці:

```
    @staticmethod
    def insert_drug(name: str, price: float, quantity: str, form_id: int,
producer_id: int) -> None:
        drug = Drug(name=name, price=price, quantity=quantity,
form_id=form_id, producer_id=producer_id)
        s.add(drug)
        s.commit()

    @staticmethod
    def insert_producer(name: str, address: str, link: str, phone_number:
str) -> None:
        producer = Producer(name=name, address=address, link=link,
phone_number=phone_number)
        s.add(producer)
        s.commit()

    @staticmethod
    def insert_form(name: str) -> None:
        form = Form(name=name)
        s.add(form)
```

```
        s.commit()

    @staticmethod
    def insert_group(name: str) -> None:
        group = Group(name=name)
        s.add(group)
        s.commit()

    @staticmethod
    def insert_drug_group(drug_id: int, group_id: int) -> None:
        drug_group = Drug_Group(drug_id=drug_id, group_id=group_id)
        s.add(drug_group)
        s.commit()
```

Запити показу реалізовані за допомогою функцій show. Спочатку в меню користувач обирає опцію показу, далі обирає таблицю, яку хоче побачити.

Таблиця “producers”:

	producer_id [PK] integer	name character varying (200)	address character varying (200)	link character varying (70)	phone_number character varying (20)
1	1	Реккітт Бенкізер Хелскер Інтернешнл Лімітед	Тейн Роуд, Ноттінгем, Ноттінгемшир, NG90 2DB, Велика Британія.	[null]	+44 20 1234 5678
2	2	АТ «КИЇВСЬКИЙ ВІТАМІННИЙ ЗАВОД»	04073, Україна, м. Київ, вул. Копилівська, 38	www.vitamin.com.ua	[null]
3	3	Товариство з обмеженою відповідальністю «ФАРМЕКС ГРУП»	Україна, 08301, Київська обл., місто Бориспіль, вулиця Шевченка, будино...	[null]	[null]
4	4	КРКА, д.д., Ново место, Словенія	Шмар'єшка цеста 6, 8501 Ново место, Словенія	[null]	[null]
5	5	ГСК Консьюмер Хелскер САРЛ / GSK Consumer Healthcare SA...	Рут де Летра, Ніон, 1260, Швейцарія/Route de l'Etraz, Nyon, 1260, Switzerla...	[null]	[null]

```
Меню:
1. Додати рядок
2. Показати таблицю
3. Редагувати рядок
4. Видалити рядок
5. Вихід
Виберіть пункт: 2

Таблиці:
1. Drugs
2. Producers
3. Forms
4. Groups
5. Drugs Groups
6. Повернутися до меню
Оберіть потрібну таблицю: 2

Producers:
ID: 1, Name: Реккітт Бенкізер Хелскер Інтернешнл Лімітед, Address: Тейн Роуд, Ноттінгем, Ноттінгемшир, NG90 2DB, Велика Британія., Link: None , Phone number: +44 20 1234 5678
ID: 2, Name: АТ «КИЇВСЬКИЙ ВІТАМІННИЙ ЗАВОД», Address: 04073, Україна, м. Київ, вул. Копилівська, 38, Link: www.vitamin.com.ua , Phone number: None
ID: 3, Name: Товариство з обмеженою відповідальністю «ФАРМЕКС ГРУП», Address: Україна, 08301, Київська обл., місто Бориспіль, вулиця Шевченка, будинок 100., Link: None , Phone
ID: 4, Name: КРКА, д.д., Ново место, Словенія, Address: Шмар'єшка цеста 6, 8501 Ново место, Словенія, Link: None , Phone number: None
ID: 5, Name: ГСК Консьюмер Хелскер САРЛ / GSK Consumer Healthcare SARL, Address: Рут де Летра, Ніон, 1260, Швейцарія/Route de l'Etraz, Nyon, 1260, Switzerland, Link: None , Ph
```

Лістинг функцій show для кожної таблиці:

```
    @staticmethod
    def show_drugs():
        return s.query(Drug.drug_id, Drug.name, Drug.price, Drug.quantity,
Drug.form_id, Drug.producer_id).all()

    @staticmethod
    def show_producers():
        return s.query(Producer.producer_id, Producer.name, Producer.address,
Producer.link, Producer.phone_number).all()

    @staticmethod
    def show_forms():
        return s.query(Form.form_id, Form.name).all()

    @staticmethod
    def show_groups():
        return s.query(Group.group_id, Group.name).all()
```

```python
@staticmethod
def show_drugs_groups():
    return s.query(Drug_Group.drug_id, Drug_Group.group_id).all()
```

Запит редагування реалізовано за допомогою функції update. Спочатку користувач обирає, у якій таблиці потрібно змінити запис і за яким ідентифікатором. Потім треба ввести всі необхідні дані для реадгування рядка.

Таблиця “groups” до редагування:

	group_id [PK] integer	name character varying (20)
2	2	антибіотики
3	3	противірусні
4	4	знеболюючі
5	5	очні
6	6	протизапальні
7	7	жарознижуючі
8	8	від кашлю
9	9	від нежиті
10	11	Group1
11	12	група2

```
Меню:
1. Додати рядок
2. Показати таблицю
3. Редагувати рядок
4. Видалити рядок
5. Вихід
Виберіть пункт: 3

Таблиці:
1. Drugs
2. Producers
3. Forms
4. Groups
5. Drugs Groups
6. Повернутися до меню
Оберіть потрібну таблицю: 4

Updating group:
Enter group ID: 12
Enter group name: тест1
Group updated successfully!
```

	group_id [PK] integer	name character varying (20)
1	1	вітаміни
2	2	антибіотики
3	3	противірусні
4	4	знеболюючі
5	5	очні
6	6	протизапальні
7	7	жарознижуючі
8	8	від кашлю
9	9	від нежиті
10	11	Group1
11	12	тест1

Лістинг функцій update для кожної таблиці:

```
@staticmethod
    def update_drug(drug_id: int, name: str, price: float, quantity: str,
form_id: int, producer_id: int) -> None:
        s.query(Drug).filter_by(drug_id=drug_id).update({Drug.name: name,
Drug.price: price, Drug.quantity: quantity, Drug.form_id: form_id,
Drug.producer_id: producer_id})
        s.commit()

    @staticmethod
    def update_producer(producer_id: int, name: str, address: str, link: str,
phone_number: str) -> None:

s.query(Producer).filter_by(producer_id=producer_id).update({Producer.name:
name, Producer.address: address, Producer.link: link, Producer.phone_number:
phone_number})
        s.commit()

    @staticmethod
    def update_form(form_id: int, name: str) -> None:
        s.query(Form).filter_by(form_id=form_id).update({Form.name: name})
        s.commit()

    @staticmethod
    def update_group(group_id: int, name: str) -> None:
        s.query(Group).filter_by(group_id=group_id).update({Group.name:
name})
        s.commit()

    @staticmethod
    def update_drug_group(drug_id: int, group_id: int) -> None:

s.query(Drug_Group).filter_by(drug_id=drug_id).update({Drug_Group.group_id:
group_id})
        s.commit()
```

Запити видалення реалізовані за допомогою функцій delete. Спочатку користувач обирає таблицю, з якої потрібно видалити дані. Потім потрібно ввести номер ідентифікатора рядка для видалення.

Таблиця “drugs” до видалення:

	drug_id [PK] integer	name character varying (30)	price numeric (10,2)	quantity character varying (10)	form_id integer	producer_id integer
1	100019	тест1	88.88	тест1	1	1
2	100017	drug_name2	333.00	333ml	3	3
3	100016	Drug2	1.00	1p	1	1
4	100015	IP	58.65	82	2	1
5	100014	YK	93.70	32	7	1
6	100013	KQ	1.66	96	5	4
7	100012	KS	99.73	81	3	3
8	100011	CL	36.26	13	6	2
9	100010	VH	0.15	29	2	3
10	100009	EH	29.75	93	7	2

```
Меню:
1. Додати рядок
2. Показати таблицю
3. Редагувати рядок
4. Видалити рядок
5. Вихід
Виберіть пункт: 4

Таблиці:
1. Drugs
2. Producers
3. Forms
4. Groups
5. Drugs Groups
6. Повернутися до меню
Оберіть потрібну таблицю: 1

Deleting drug:
Enter drug ID: 100019
Drug deleted successfully!
```

Таблиця “drugs” після видалення:

	drug_id [PK] integer	name character varying (30)	price numeric (10,2)	quantity character varying (10)	form_id integer	producer_id integer
1	100017	drug_name2	333.00	333ml	3	3
2	100016	Drug2	1.00	1p	1	1
3	100015	IP	58.65	82	2	1
4	100014	YK	93.70	32	7	1
5	100013	KQ	1.66	96	5	4
6	100012	KS	99.73	81	3	3
7	100011	CL	36.26	13	6	2
8	100010	VH	0.15	29	2	3
9	100009	EH	29.75	93	7	2
10	100008	WP	18.75	9	6	4

Лістинг функцій delete для кожної таблиці:

```
@staticmethod
def delete_drug(drug_id) -> None:
    drug = s.query(Drug).filter_by(drug_id=drug_id).one()
    s.delete(drug)
    s.commit()

@staticmethod
def delete_producer(producer_id) -> None:
    producer = s.query(Producer).filter_by(producer_id=producer_id).one()
    s.delete(producer)
    s.commit()

@staticmethod
def delete_form(form_id) -> None:
    form = s.query(Form).filter_by(form_id=form_id).one()
    s.delete(form)
    s.commit()

@staticmethod
def delete_group(group_id) -> None:
    group = s.query(Group).filter_by(group_id=group_id).one()
    s.delete(group)
    s.commit()

@staticmethod
def delete_drug_group(group_id) -> None:
    drugs_groups = s.query(Drug_Group).filter_by(group_id=group_id).all()
    for drug_group in drugs_groups:
        s.delete(drug_group)
    s.commit()
```

Завдання №2

Індекс – це спеціальна структура даних, яка зберігає групу ключових значень та покажчиків. Індекс використовується для управління даними. Для тестування індексів було створено окремі таблиці у базі даних test з 1000000 записів.

GIN

Індекс GIN (Generalized Inverted Index) — це тип індексу у системі управління базами даних, який дозволяє ефективно виконувати пошук та фільтрацію для деяких типів даних, які не обробляються традиційними B-деревами (Binary Tree Index).

GIN особливо корисний для оптимізації запитів, які використовують рядкові масиви, текстові дані, JSON-структури та інші складні типи даних. Онлайн-пошук, фільтрація, і взаємодія з елементами цих типів стає значно ефективнішою завдяки GIN індексу.

Для дослідження індексу була створена таблиця gin_test, яка має дві колонки: “id” та “string”:

```
CREATE TABLE "gin_test"("id" bigserial PRIMARY KEY, "string" varchar(100));
INSERT INTO "gin_test"("id", "string")
SELECT generate_series as "id", md5(random()::text)
FROM generate_series(1, 1000000)
```

	id [PK] bigint	string character varying (100)
1	1	28f79295b06ce66da89bb10927b8d222
2	2	3e1ec64b693c11b884ad1604e67094...
3	3	43c48cde8adec85a4d7d20984e1d92...
4	4	b434a365ac260e7554aac2930c1af9c9
5	5	8b71ea7265942d91b6e4c521212734...
6	6	1f725ab984619a2db2fc0670ad5b552a
7	7	ca2b8f07240c675de14109c2428cdf83
8	8	a8aeb9bf0b70a854ed5f7186be84ce12
9	9	de924b14e2c9bc2f9ff6bf95005a6542
10	10	dd988fb4ca375b170a8f72cdb1c0ad36

Total rows: 1000 of 1000000 Query complete 00:00:00.5(

Для тестування візьмемо 5 запитів:

```
SELECT * FROM gin_test WHERE "string" = 'de924b14e2c9bc2f9ff6bf95005a6542';
SELECT COUNT(*), "string" FROM gin_test GROUP BY "string";
SELECT * FROM gin_test ORDER BY "string" DESC;
SELECT COUNT(*), "string" FROM gin_test WHERE "id" BETWEEN 100 AND 1000 GROUP
BY "string";
SELECT COUNT(*), "string" FROM gin_test GROUP BY "string" ORDER BY COUNT(*)
DESC;
```

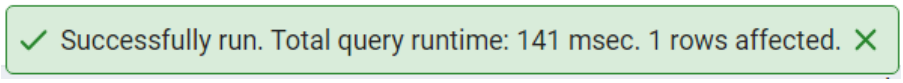
Створення індексу:

```
CREATE INDEX gin_index ON gin_test USING gin("string" gin_trgm_ops);
```

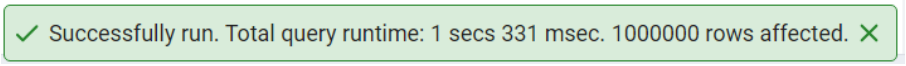
Результати виконання запитів

Без індекса gin

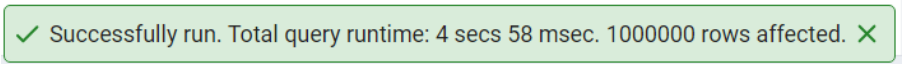
Запит №1



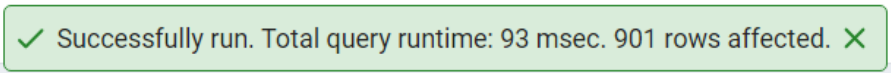
Запит №2



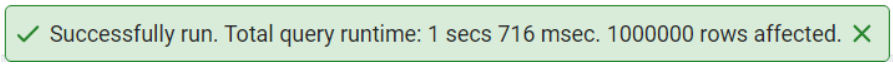
Запит №3



Запит №4

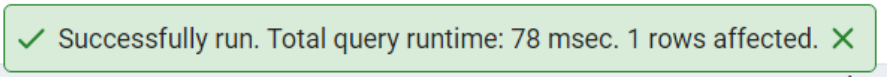


Запит №5

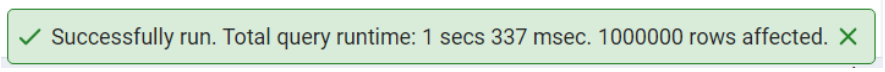


З індексом gin

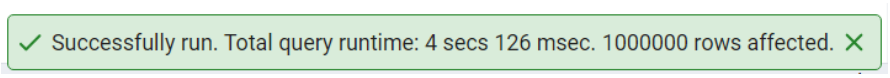
Запит №1



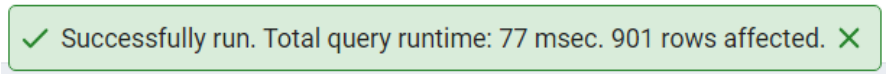
Запит №2



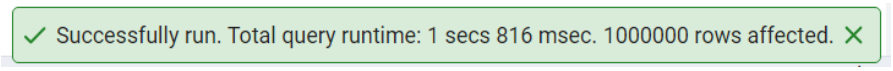
Запит №3



Запит №4



Запит №5



Отже, з отриманих результатів можна побачити, що запити №2, 3, 5 спрацювали швидше без індексу gin. Розглянемо кожен із них:

1. Запит №2 - GROUP BY

   Запит з агрегацією та GROUP BY може працювати ефективніше без індексу, оскільки база даних може проводити агрегацію на основі сортованого порядку стовпців без використання індексу. GIN-індекс може займати додатковий ресурс для таких запитів.

2. Запит №3 - ORDER BY

   Запити з сортуванням можуть працювати швидше без індексу, оскільки база даних може використовувати сортований порядок на основі фізичного розташування даних у таблиці. Використання індексу може вимагати додаткових операцій.

3. Запит №5 - GROUP BY та ORDER BY COUNT(*)

   Запит з використанням GROUP BY і ORDER BY COUNT(*) може бути швидше оброблений без індексу, оскільки GIN-індекс може вимагати додаткових операцій для обробки агрегаційних функцій та сортування.

BRIN

Індекс BRIN (Block Range INdex) - це інший тип індексу в PostgreSQL, який призначений для оптимізації пошуку та фільтрації даних великих таблиць, особливо тих, які мають упорядкований порядок значень за деяким стовпцем. BRIN індекси призначені для ефективної обробки діапазонних запитів.

Основна ідея BRIN індексу полягає в тому, щоб розділити дані на блоки та зберігати індексовані значення для кожного блоку. Це дозволяє базі даних

швидко визначати, які блоки можуть містити потрібні рядки, і використовувати індекс для обробки тільки цих блоків.

Для дослідження індексу була створена таблиця brin_test, яка має дві колонки: “id” та “string”:

```
CREATE TABLE "brin_test"("id" bigserial PRIMARY KEY, "string" varchar(100));
INSERT INTO "brin_test"("id", "string")
SELECT generate_series as "id", md5(random()::text)
FROM generate_series(1, 1000000)
```

	id [PK] bigint	string character varying (100)
1	1	97c766a6db55cf270b6e5608162534bc
2	2	27c1426a8f2434863c1128a059adf241
3	3	173b1c6b0a7e215d57a28d7dc9dc7de7
4	4	9f674b181324ae1c07544370bf48bb74
5	5	2433b4365cd0d32b4efcbfbdace002ba
6	6	6b8458b9794987936f2425dded87daa4
7	7	e23c036ee7916d6c5da675c8c01c8e4a
8	8	154d9c50d4d0033b6200c4903bb64f7a
9	9	5a63e29216859450af3cdaa817f612d9
10	10	1c991cef8dfcd1be7df920970204e10e

Total rows: 1000 of 1000000 Query complete 00:00:00.37

Для тестування візьмемо 5 запитів:

```
SELECT * FROM brin_test WHERE "string" = '2433b4365cd0d32b4efcbfbdace002ba';
SELECT COUNT(*), "string" FROM brin_test GROUP BY "string";
SELECT * FROM brin_test ORDER BY "string" DESC;
SELECT COUNT(*), "string" FROM brin_test WHERE "id" BETWEEN 100 AND 1000
GROUP BY "string";
SELECT COUNT(*), "string" FROM brin_test GROUP BY "string" ORDER BY COUNT(*)
DESC;
```

Створення індексу:

```
CREATE INDEX brin_index ON brin_test USING brin("string");
```

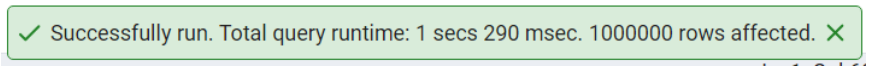
Результати виконання запитів

Без індекса brin

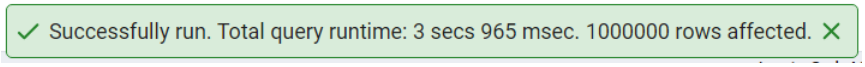
Запит №1

✓ Successfully run. Total query runtime: 229 msec. 1 rows affected. ✕

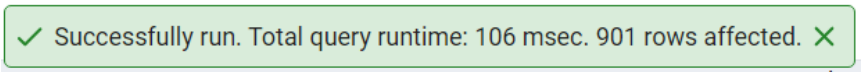
Запит №2



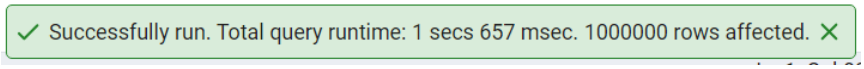
Запит №3



Запит №4

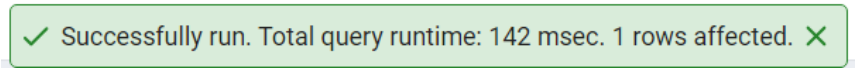


Запит №5

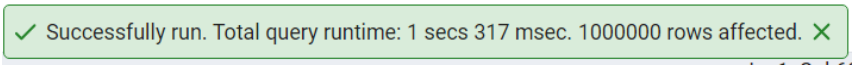


З індексом brin

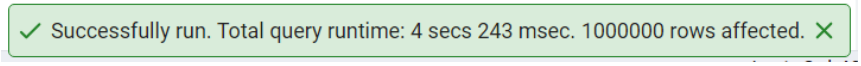
Запит №1



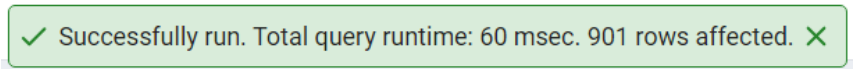
Запит №2



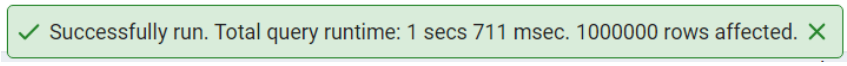
Запит №3



Запит №4



Запит №5



Отже, з отриманих результатів можна побачити, що запити №3, 5 спрацювали швидше без індексу brin. Розглянемо кожен із них:

1. Запит №3 - ORDER BY

   Запит з сортуванням (ORDER BY) може працювати ефективніше без індексу, особливо якщо дані вже відсортовані фізично в таблиці. BRIN-індекс, який орієнтований на діапазонні значення, може бути менш ефективним для сортування, особливо якщо діапазони не відображають точно впорядкованих значень.

2. Запит №5 - GROUP BY та ORDER BY COUNT(*)

Схоже на випадок з GIN-індексом, GROUP BY і ORDER BY COUNT(*) можуть працювати ефективніше без BRIN-індексу. BRIN, орієнтований на блоки, може мати обмежену точність для агрегаційних функцій та сортування в порівнянні з B-деревооб'єктними індексами.

Ефективність індексів може сильно залежати від конкретного сценарію використання, обсягу даних та конкретного плану виконання, який вибирає оптимізатор запитів бази даних.