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

Пильова Діана

Telegram: @smesharik29

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’єктнореляційної 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/pylova/DataBase

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.

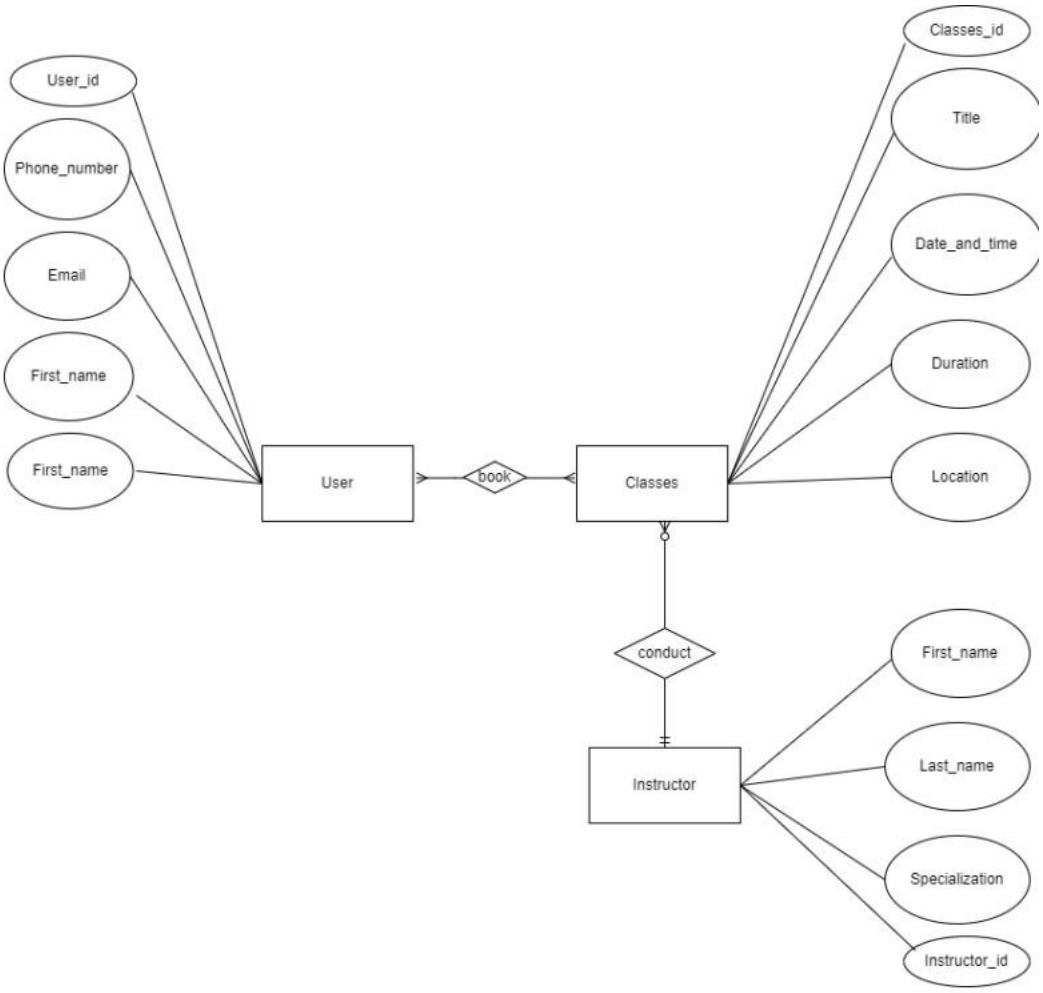
Вимоги до пункту завдання №4

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 №21

№ варіанта	*Види індексів*	*Умови для тригера*
21	*BTree, Hash*	*before delete, update*

ER-діаграма, побудована за нотацією Чена



Сутності з описом призначення

Для побудови бази даних предметної області було виділено декілька сутностей:

1. Заняття (Classes) з атрибутами: назва заняття, дата і час проведення, тривалість, місце проведення, індифікаційний номер.

2. Користувач (User) з атрибутами: ім'я, прізвище, електронна адреса, номер телефону, індифікаційний номер.

3. Тренер (Instructor) з атрибутами: ім'я, прізвище, спеціалізація, індифікаційний номер.

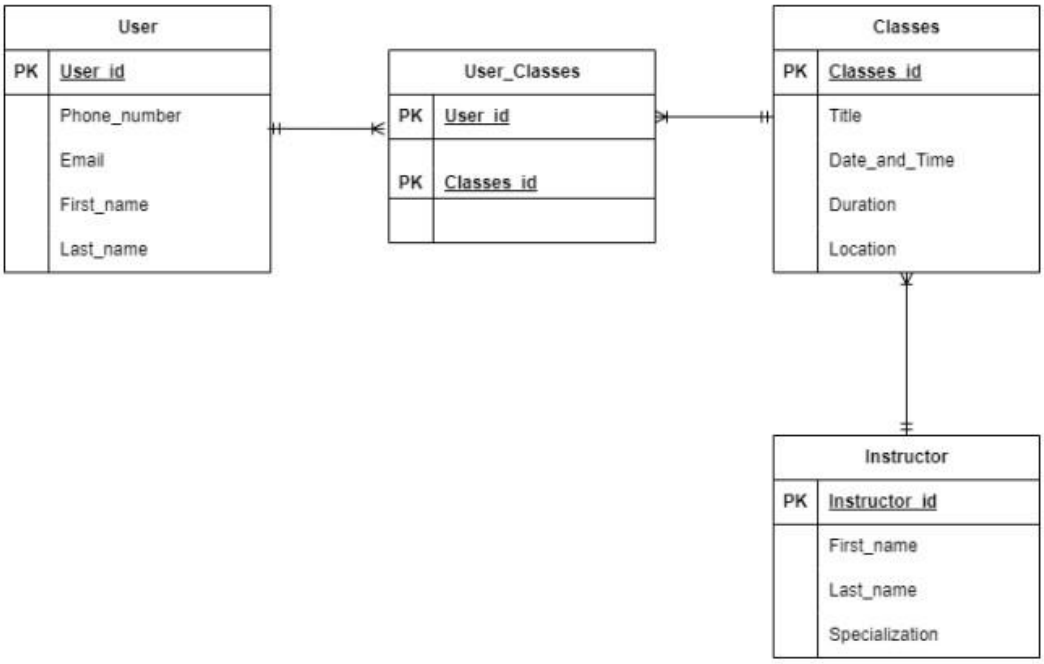
Опис зв’язків між сутностями предметної області

Сутність "Заняття" встановлює зв'язок багато-до-багатьох (M:N) з сутністю "Користувач", оскільки кожне заняття може мати багатьох

користувачів, які на нього записані, і кожен користувач може бути записаний на багато занять.

Сутність "Заняття" має зв'язок один-до-багатьох (1:N) з сутністю "Тренер", оскільки кожне заняття має одного тренера, який його проводить, і один тренер може проводити багато занять.

Схема бази даних у графічному вигляді



Завдання №1

У даній лабораторній роботі було реалізовано 4 класи відповідно до 4 існуючих таблиць у розробленій базі даних, а саме:

1. User
2. Class
3. Instructor
4. UserClasses

User

Таблиця User має такі стовпці: User_id (ідентифікатор користувача), Phone_number (номер телефону користувача), Email (електронна пошта користувача), First_name (ім’я користувача), Last_name (прізвище користувача). Також наявний зв’язок із таблицею User_Classes, тому в цьому класі встановлений зв’язок relationship (“UserClasses”).

Програмна реалізація класу User:

```
class User(Base):
    __tablename__ = 'User'
    User_id = Column(Integer, primary_key=True)
    Phone_number = Column(String)
    Email = Column(String)
    First_name = Column(String)
    Last_name = Column(String)

    user_class = relationship("UserClasses")
     def __init__(self, user_id, phone_number, email, first_name,
last_name):
        self.User_id = user_id
self.Phone_number = phone_number
self.Email = email          self.First_name
= first_name          self.Last_name =
last_name
     def
__repr__(self):
        return f"<Users(User_id={self.User_id},
Phone_number={self.Phone_number}, Email={self.Email},
First_name={self.First_name}, Last_name={self.Last_name})>"
```

Class

Таблиця Classes має такі стовпці: Classes_id (ідентифікатор заняття), Title (назва заняття), Date_and_Time (дата та час проведення заняття), Duration (тривалість заняття), Location (місце проведення заняття). Також наявні зв’язки із таблицями Instructor та User_Classes, тому в цьому класі встановлені зв’язки relationship (“Instructor”) та relationship (“UserClasses”).

Програмна реалізація класу Class:

```
class Class(Base):
    __tablename__ = 'Classes'
    Classes_id = Column(Integer, primary_key=True)
    Title = Column(String)
    Date_and_Time = Column(DateTime)
    Duration = Column(Numeric)
    Location = Column(String)
     instructor = relationship("Instructor")
user_class = relationship("UserClasses")
     def __init__(self, class_id, title, date_and_time, duration,
location):
        self.Classes_id = class_id
self.Title = title
        self.Date_and_Time = date_and_time
self.Duration = duration          self.Location
= location
     def
__repr__(self):
        return f"<Classes(Classes_id={self.Classes_id}, Title={self.Title},
Date_and_Time={self.Date_and_Time}, Duration={self.Duration},
Location={self.Location})>"
```

Instructor

Таблиця Instructor має такі стовпці: Instructor_id (ідентифікатор інструктора), First_name (ім’я інструктора), Last_name (прізвище інструктора), Classes_id (зовнішній ключ, який посилається на заняття).

Програмна реалізація класу Instructor:

```
class Instructor(Base):
    __tablename__ = 'Instructor'
    Instructor_id = Column(Integer, primary_key=True)
    First_name = Column(String)
    Last_name = Column(String)

    Classes_id = Column(Integer, ForeignKey('Classes.Classes_id'))
     def __init__(self, instructor_id, first_name, last_name,
class_id):
        self.Instructor_id = instructor_id
self.First_name = first_name
self.Last_name = last_name         self.Classes_id
= class_id
     def
__repr__(self):
        return f"<Instructor(Instructor_id={self.Instructor_id},
First_name={self.First_name}, Last_name={self.Last_name},
Classes_id={self.Classes_id})>"
```

UserClasses

Таблиця User_Classes має такі стовпці: User_id (зовнішній ключ, який посилається на користувача), Classes_id (зовнішній ключ, який посилається на заняття).

Програмна реалізація класу UserClasses:

```
class UserClasses(Base):
    __tablename__ = 'User_Classes'

    User_id = Column(Integer, ForeignKey('User.User_id'), primary_key=True)
    Classes_id = Column(Integer, ForeignKey('Classes.Classes_id'),
primary_key=True)
     def __init__(self, user_id,
classes_id):
        self.User_id = user_id
self.Classes_id = classes_id
     def __repr__(self):         return
f"<User_Classes(User_id={self.User_id},
Classes_id={self.Classes_id})>"
```

Меню складається із 5 пунктів, кожен з яких буде розглянуто далі.

```
Menu:
1. Add row
2. Show table
3. Update row
4. Delete row
5. Exit
Select your choice:
```

1) **Add row (Додати рядок)**
   Цей пункт створений для додавання рядка у таблицю. Після його вибору, відкривається список всіх таблиць БД, де потрібно обрати таблицю, до якої хочемо додати рядок:

```
Menu:
1. Add row
2. Show table
3. Update row
4. Delete row
5. Exit
Select your choice: 1

Tables:
1. Users
2. Classes
3. Instructors
4. User Classes
5. Back to menu
Select table:
```

   Після вибору таблиці, користувачу потрібно ввести всі необхідні дані для нового рядка.

2) **Show table (Показ таблиці)**
   Цей пункт створений для показу таблиць. Після його вибору, відкривається список доступних таблиць БД, де потрібно вибрати таблицю, яку бажаємо побачити.

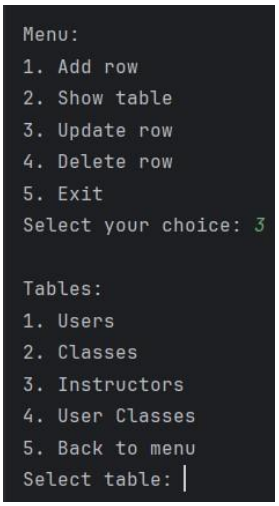
```
Menu:
1. Add row
2. Show table
3. Update row
4. Delete row
5. Exit
Select your choice: 2

Tables:
1. Users
2. Classes
3. Instructors
4. User Classes
5. Back to menu
Select table:
```

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

3) **Update row (Редагувати рядок)**

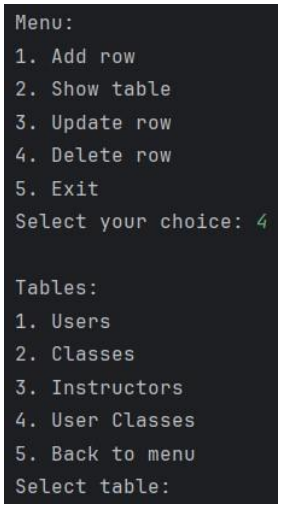
Цей пункт створений для редагування рядків у таблицях. Після вибору цього пункту, відкривається список доступних таблиць, де потрібно вибрати таблицю, в якій бажаємо зробити зміну.



Після вибору таблиці, користувачу потрібно ввести ідентифікатор існуючого рядка в таблиці. Потім записати нові дані для обраного рядка.

4) **Delete row (Видалити рядок)**

Цей пункт створений для видалення рядків у таблицях. Після вибору цього пункту, відкривається список доступних таблиць БД, де потрібно вибрати таблицю, в якій бажаємо видалити рядок.



Після вибору таблиці, користувачу потрібно ввести ідентифікатор існуючого рядка в таблиці для видалення.

5) **Exit (Вихід)**

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

```
Menu:
1. Add row
2. Show table
3. Update row
4. Delete row
5. Exit
Select your choice: 5

Process finished with exit code 0
```

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 “User” до вставки:

	User_id [PK] integer	Phone_number character varying (15)	Email character varying (255)	First_name character varying (255)	Last_name character varying (255)
1	1	0997328921	qkwk@gmail.com	Steven	Doyle
2	2	0962683804	hdjf@gmail.com	Niall	James
3	3	555	test	test	test
4	4	0116745491	0	VS	MO
5	5	0339144684	2f8@example.com	A	TL
6	7	0997328921	one@gmail.com	Petro	Peter
7	8	0962683804	nastya@gmail.com	Nastya	Turenko
8	9	0912345678	katya@gmail.com	Katya	Maistrenko
9	10	0987654321	mykola@gmail.com	Mykola	Koval
10	11	0985555555	lev@gmail.com	Lev	Dan
11	12	0976543210	dima@gmail.com	Dima	Bogdan
12	13	0981111111	masha@gmail.com	Masha	Teteruk
13	14	0637441228	newyear2024@gmail.com	Name1	Surname1

```
Menu:
1. Add row
2. Show table
3. Update row
4. Delete row
5. Exit
Select your choice: 1

Tables:
1. Users
2. Classes
3. Instructors
4. User Classes
5. Back to menu
Select table: 1

Adding user:
Enter user_id: 15
Enter phone number: 0123456789
Enter email: test1@test.com
Enter first name: Test1
Enter last name: Test1
User added successfully!
```

Таблиця “User” після вставки:

	User_id [PK] integer	Phone_number character varying (15)	Email character varying (255)	First_name character varying (255)	Last_name character varying (255)
2	2	0962683804	hdjf@gmail.com	Niall	James
3	3	555	test	test	test
4	4	0116745491	0	VS	MO
5	5	0339144684	2f8@example.com	A	TL
6	7	0997328921	one@gmail.com	Petro	Peter
7	8	0962683804	nastya@gmail.com	Nastya	Turenko
8	9	0912345678	katya@gmail.com	Katya	Maistrenko
9	10	0987654321	mykola@gmail.com	Mykola	Koval
10	11	0985555555	lev@gmail.com	Lev	Dan
11	12	0976543210	dima@gmail.com	Dima	Bogdan
12	13	0981111111	masha@gmail.com	Masha	Teteruk
13	14	0637441228	newyear2024@gmail.com	Name1	Surname1
14	15	0123456789	test1@test.com	Test1	Test1

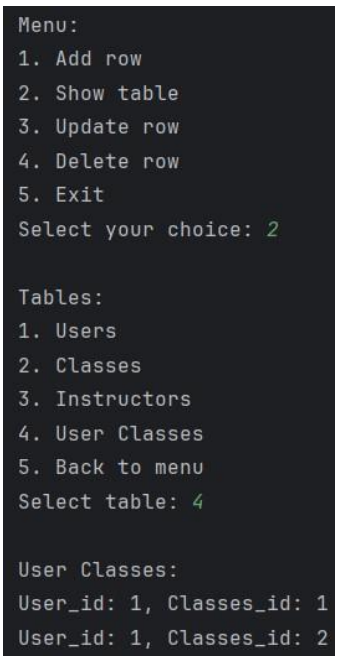
Лістинг функцій insert для кожної таблиці:

```
    def insert_user(self, user_id: int, phone_number: str, email: str,
first_name: str, last_name: str) -> None:          user =
User(user_id=user_id, phone_number=phone_number, email=email,
first_name=first_name, last_name=last_name)
        s.add(user)
        s.commit()      def insert_class(self, class_id: int, title:
str, date_and_time: DateTime, duration: int, location: str) -> None:
        clas = Class(class_id=class_id, title=title,
date_and_time=date_and_time, duration=duration, location=location)
        s.add(clas)
        s.commit()      def insert_instructor(self, instructor_id:
int, first_name: str, last_name: str, class_id: int) -> None:
        instructor = Instructor(instructor_id=instructor_id,
first_name=first_name, last_name=last_name, class_id=class_id)
        s.add(instructor)
        s.commit()      def insert_user_class(self, user_id: int,
classes_id: int) -> None:          user_class =
UserClasses(user_id=user_id, classes_id=classes_id)
        s.add(user_class)
        s.commit()
```

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

Таблиця “User_Classes”:

	User_id [PK] integer	Classes_id [PK] integer
1	1	1
2	1	2



Лістинг функцій show для кожної таблиці:

```
@staticmethod     def
show_users():
        return s.query(User.User_id, User.Phone_number, User.Email,
User.First_name, User.Last_name).all()

    @staticmethod
    def show_classes():
        return s.query(Class.Classes_id, Class.Title, Class.Date_and_Time,
Class.Duration, Class.Location).all()

    @staticmethod     def
show_instructors():
        return s.query(Instructor.Instructor_id, Instructor.First_name,
Instructor.Last_name, Instructor.Classes_id).all()

    @staticmethod     def
show_user_classes():
        return s.query(UserClasses.User_id, UserClasses.Classes_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 “User” до редагування:

	User_id [PK] integer	Phone_number character varying (15)	Email character varying (255)	First_name character varying (255)	Last_name character varying (255)
2	2	0962683804	hdjf@gmail.com	Niall	James
3	3	555	test	test	test
4	4	0116745491	0	VS	MO
5	5	0339144684	2f8@example.com	A	TL
6	7	0997328921	one@gmail.com	Petro	Peter
7	8	0962683804	nastya@gmail.com	Nastya	Turenko
8	9	0912345678	katya@gmail.com	Katya	Maistrenko
9	10	0987654321	mykola@gmail.com	Mykola	Koval
10	11	0985555555	lev@gmail.com	Lev	Dan
11	12	0976543210	dima@gmail.com	Dima	Bogdan
12	13	0981111111	masha@gmail.com	Masha	Teteruk
13	14	0637441228	newyear2024@gmail.com	Name1	Surname1
14	15	0123456789	test1@test.com	Test1	Test1

```
Menu:
1. Add row
2. Show table
3. Update row
4. Delete row
5. Exit
Select your choice: 3

Tables:
1. Users
2. Classes
3. Instructors
4. User Classes
5. Back to menu
Select table: 1

Updating user:
Enter user ID: 15
Enter user_id: 15
Enter phone number: 1111111111
Enter email: test2@test.com
Enter first name: test2
Enter last name: test2
User updated successfully!
```

Таблиця “User” після редагування:

	User_id [PK] integer	Phone_number character varying (15)	Email character varying (255)	First_name character varying (255)	Last_name character varying (255)
2	2	0962683804	hdjf@gmail.com	Niall	James
3	3	555	test	test	test
4	4	0116745491	0	VS	MO
5	5	0339144684	2f8@example.com	A	TL
6	7	0997328921	one@gmail.com	Petro	Peter
7	8	0962683804	nastya@gmail.com	Nastya	Turenko
8	9	0912345678	katya@gmail.com	Katya	Maistrenko
9	10	0987654321	mykola@gmail.com	Mykola	Koval
10	11	0985555555	lev@gmail.com	Lev	Dan
11	12	0976543210	dima@gmail.com	Dima	Bogdan
12	13	0981111111	masha@gmail.com	Masha	Teteruk
13	14	0637441228	newyear2024@gmail.com	Name1	Surname1
14	15	1111111111	test2@test.com	test2	test2

Лістинг функцій update для кожної таблиці:

```
    @staticmethod     def update_user(user_id: int, phone_number: str,
email: str, first_name: str, last_name: str, id: int) -> None:
        s.query(User).filter_by(User_id=id).update({User.User_id: user_id,
```

```
User.Phone_number: phone_number, User.Email: email, User.First_name:
first_name, User.Last_name: last_name})
        s.commit()

    @staticmethod
    def update_class(class_id: int, title: str, date_and_time: DateTime,
duration: int, location: str, id: int) -> None:
        s.query(Class).filter_by(Classes_id=id).update({Class.Classes_id:
class_id, Class.Title: title, Class.Date_and_Time: date_and_time,
Class.Duration: duration, Class.Location: location})
        s.commit()

    @staticmethod     def update_instructor(instructor_id: int,
first_name: str, last_name:
str, class_id: int, id: int) -> None:

s.query(Instructor).filter_by(Instructor_id=id).update({Instructor.Instructor
_id: instructor_id, Instructor.First_name: first_name, Instructor.Last_name:
last_name, Instructor.Classes_id: class_id})
        s.commit()

    @staticmethod     def update_user_class(user_id: int, class_id: int,
id: int) -> None:

s.query(UserClasses).filter_by(User_id=id).update({UserClasses.User_id:
user_id, UserClasses.Classes_id: class_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 “Instructor” до видалення:

	Instructor_id [PK] integer	First_name character varying (255)	Last_name character varying (255)	Classes_id integer
3	4	a	a	1
4	5	Conor	Smith	5
5	6	Sophie	Lee	10
6	7	Michael	Johnson	5
7	8	Emma	White	5
8	9	Alex	Taylor	1
9	10	Grace	Brown	1
10	11	Daniel	Miller	2
11	12	Olivia	Wilson	5
12	13	William	Clark	2
13	14	Sophia	Moore	2
14	15	Test2	Test2	999
15	16	test2	test2	10000

```
Menu:
1. Add row
2. Show table
3. Update row
4. Delete row
5. Exit
Select your choice: 4

Tables:
1. Users
2. Classes
3. Instructors
4. User Classes
5. Back to menu
Select table: 3

Deleting instructor:
Enter instructor ID: 16
Instructor deleted successfully!
```

Таблиця “Instructor” після видалення:

	Instructor_id [PK] integer	First_name character varying (255)	Last_name character varying (255)	Classes_id integer
2	3	name2	lastname2	1
3	4	a	a	1
4	5	Conor	Smith	5
5	6	Sophie	Lee	10
6	7	Michael	Johnson	5
7	8	Emma	White	5
8	9	Alex	Taylor	1
9	10	Grace	Brown	1
10	11	Daniel	Miller	2
11	12	Olivia	Wilson	5
12	13	William	Clark	2
13	14	Sophia	Moore	2
14	15	Test2	Test2	999

Лістинг функцій delete для кожної таблиці:

```
    @staticmethod     def
delete_user(user_id) -> None:
        user = s.query(User).filter_by(User_id=user_id).one()
        s.delete(user)
        s.commit()
```

```
    @staticmethod     def
delete_class(class_id) -> None:
        clas = s.query(Class).filter_by(Classes_id=class_id).one()
        s.delete(clas)
        s.commit()

    @staticmethod     def
delete_instructor(instructor_id) -> None:
        instructor =
s.query(Instructor).filter_by(Instructor_id=instructor_id).one()
        s.delete(instructor)
        s.commit()

    @staticmethod     def
delete_user_class(user_id) -> None:
        user_classes = s.query(UserClasses).filter_by(User_id=user_id).all()
for user_class in user_classes:
            s.delete(user_class)
        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 записами.

BTree

Індекс B-дерева (B-tree index) використовується для прискорення операцій пошуку в базі даних. Це особлива структура даних, яка забезпечує швидкий доступ до записів в таблиці за значенням індексованого стовпця.

Основна ідея застосування індексів B-дерева в SQL полягає в тому, щоб створити структуру, яка дозволяє ефективно виконувати операції пошуку (наприклад, SELECT), а також деякі операції сортування та об'єднання даних.

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

```
CREATE TABLE "btree_test"("id" bigserial PRIMARY KEY, "string" varchar(100));
INSERT INTO "btree_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	a3339c3bb03924797eead56e58caea...
2	2	7f3292a98ca49dba9306a41e5e8afe13
3	3	ca61e6fc98db6430a7317a2aac9c1ff6
4	4	bd2b5c157c17114553b631307bf2b9...
5	5	87ea52aa8a73d043f07db75eccc001af
6	6	495d9fae15603090ba1363c9279540...
7	7	5f67a57dc6fb4e0c580997f94efe2bda
8	8	5b9cc668d29ad517c99e1174c870e8...
9	9	e85ac229444f01022b0648f7ea6f5f24
10	10	f37cc16dc96bde3ec4b71ab0257d19b9
11	11	8a3ec52131cc973cea4e4d00b4d75b...
12	12	f64d9171c83b6e03c91ad1594a7d5b...
13	13	0e8812608bfbdb6516b59e7a34b782...

Для тестування візьмемо 5 запитів:

```
SELECT * FROM btree_test WHERE "string" = 'ca61e6fc98db6430a7317a2aac9c1ff6';
SELECT COUNT(*), "string" FROM btree_test GROUP BY "string";
SELECT * FROM btree_test ORDER BY "string" DESC;
```

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

Створення індексу:

```
CREATE INDEX btree_index ON btree_test (string);
```

Результати виконання запитів

Без індекса BTree

Запит №1

Запит №2

Запит №3

Запит №4

Запит №5

З індексом BTree

Запит №1

Запит №2

Запит №3

Запит №4

Запит №5

Отже, з отриманих результатів можна побачити, що виконання обраних запитів з використанням індексу BTree є швидшим за виконання без індексу.

Hash

Індекс Hash (або хеш-індекс) — це структура даних в базах даних, призначена для швидкого пошуку даних за хеш-значенням ключа. У порівнянні з індексами B-дерева, які зберігають ключі в впорядкованому вигляді, індекси Hash використовують хеш-функції для отримання адреси, де зберігаються відповідні дані. Основні особливості індексів Hash:

1. Хеш-функція: Кожен ключ індексується за допомогою хеш-функції, яка перетворює його в унікальний ідентифікатор (хеш-код). Цей хеш-код потім використовується для швидкого визначення місця зберігання відповідних даних.
2. Швидкий доступ: Операції пошуку за ключем в індексі Hash можуть бути дуже ефективними, оскільки можна швидко обчислити адресу, де повинні знаходитися дані.
3. Невпорядковані: Відмінність від індексів B-дерева, індекси Hash не зберігають ключі впорядковано. Це робить їх менш ефективними для діапазонних пошуків та сортування даних.
4. Колізії: Оскільки хеш-функції зображають ключі в обмежений простір хеш-кодів, можливі колізії, коли два різних ключа мають однаковий хеш-код. Це вирішується за допомогою методів управління колізіями, таких як використання списків або відкритого хешування.

Індекси Hash часто використовуються в базах даних для прискорення операцій швидкого пошуку за унікальними ключами, але їх ефективність може залежати від конкретних умов використання та характеристик даних.

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

```
CREATE TABLE "hash_test"("id" bigserial PRIMARY KEY, "string" varchar(100));
INSERT INTO "hash_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	1a06f67494d42c933a01421679218…
2	2	ed61dc751924d03cdb2add6d6a794…
3	3	0fb22466c6038f9d1ef5c812891f22c1
4	4	b49ca8cbc867d1ad14e8520adf1ac3…
5	5	23d5c7844f5829270b9fe0595ebde5…
6	6	def39299140cb7e48057fbe23903de…
7	7	5c4dac77b7c6d2a89dd0ca748ac53…
8	8	7850292eccf69f5a7aa038d75f76047b
9	9	5ad3f5a0fc3b696754e7412bef417e8d
10	10	b303d8a49b8e111aff6ace7f2b13d81b
11	11	3122f06946365f05a9e4f7c3a187884c
12	12	1d7aaec2d52fba83400a1ef40d61d2…
13	13	35a2c9e100dcf3f2b5cf6a4b46aec02f

Для тестування візьмемо 5 запитів:

```
SELECT * FROM hash_test WHERE "string" = '23d5c7844f5829270b9fe0595ebde545';
SELECT COUNT(*), "string" FROM hash_test GROUP BY "string";
SELECT * FROM hash_test ORDER BY "string" DESC;
SELECT COUNT(*), "string" FROM hash_test WHERE "id" BETWEEN 100 AND 1000 GROUP
BY "string";
SELECT COUNT(*), "string" FROM hash_test GROUP BY "string" ORDER BY COUNT(*)
DESC;
```

Створення індексу:

```
CREATE INDEX "hash_index" ON "hash_test" USING hash("string");
```

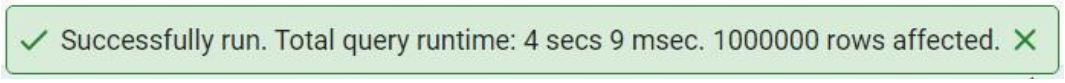
Результати виконання запитів

Без індекса Hash

Запит №1

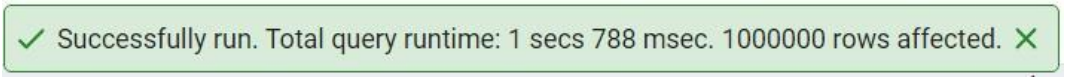
Запит №2

Запит №3



Запит №4

Запит №5



З індексом Hash

Запит №1

Запит №2

Запит №3

Запит №4

Запит №5

Індекси Hash можуть бути ефективними для точних (пошук за конкретним значенням) та швидких пошуків, але вони можуть виявитися менш ефективними в деяких сценаріях, зокрема, коли виникають колізії або коли потрібні операції діапазонного пошуку чи сортування.

Із отриманих результатів можна побачити, що запити №3, 4 та 5 виконалися швидше без індексу Hash ніж з. Розглянемо їх:

Запит №3. Цей запит може виконуватися швидше без індексу Hash, оскільки індекс Hash не допомагає оптимізувати операції сортування.

Запит №4. Цей запит може виконуватися швидше без індексу Hash, особливо якщо індекс використовується неефективно через діапазонне використання "id".

Запит №5. Цей запит може виконуватися швидше без індексу Hash, оскільки оптимізатор може використовувати інші методи групування та сортування без індексу hash.