МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

КУРСОВА РОБОТА

з дисципліни “Бази даних”

спеціальність 121 – Програмна інженерія

на тему: «База даних учнів навчального закладу»

Студентка групи КП-01 **Грабовська А.Д.**

Асистент кафедри СПіСКС

Радченко К.О.

Захищено з оцінкою _____

Київ – 2021

Анотація

Дана курсова робота складається з бази даних учнів навчального закладу та консольного додатку для адміністративної взаємодії з цією базою даних, написаного мовою програмування Python, який дозволяє виконувати певні операції над нею (читання, оновлення, запис, видалення, псевдовипадкова генерація записів, аналіз даних тощо).

У результаті розробки даної бази даних та даного консольного додатку було набуто практичні навички розробки сучасного програмного забезпечення, що взаємодіє з реляційними базами даних, а також здобуто навички оформлення відповідного текстового, програмного та ілюстративного матеріалу у формі проектної документації.

Зміст

Вступ

Навчальний заклад – це велика та складна система. З розвитком технологій, з’явилась можливість полегшити зберігання, додавання та пошук різноманітної інформації. Таким чином, було прийнято рішення зібрати детальну інформацію про навчальний заклад. У даній базі даних була зібрана детальна інформація про предмети, класи(групи) та вчителів, вона призначена для зручного та швидкого доступу до такої інформації.
База даних була спроектована відповідно до 3-ої нормальної форми, швидкість її роботи була оптимізована шляхом додавання індексів, також вона було убезпечена додаванням механізмів реплікації та резервного копіювання. Окрім того, розроблений консольний додаток може надавати результати певного аналізу даних у базі. Також наявна функція імпорту та експорту даних у форматі CSV.

Аналіз інструментарію для виконання курсової роботи

Для виконання даної роботи у якості системи керування базами даних було обрано PostgreSQl. Такий вибір був зроблений у зв'язку з такими факторами:

- Відкрите ПЗ відповідає стандарту SQL - PostgreSQL - безкоштовне ПЗ з відкритим вихідним кодом. Ця СУБД є дуже потужною системою.
- Підтримка великої кількості типів даних, включно з власними
- Цілісність даних з усіма необхідними обмеженнями
- Надійність, безпека
- PostgreSQL не просто реляційна, а об'єктно-реляційна СУБД, що надає певні переваги
- Працює з багатьма типами мереж
- Велика місткість
- Велика спільнота – просто знайти вирішення потенційних проблем при розробці
- Це повністю опен-сорсний проект
- Розширення - існує можливість розширення функціоналу за рахунок своїх процедур

Для взаємодії з базою даних було обрано бібліотеку psycopg2, оскільки:

- Добре підходить для зручного використання у мові програмування Python
- Розроблена спеціально для PostgreSQL
- Найпопулярніша для взаємодії з PostgreSQL у мові програмування Python
- Має чітку, зрозумілу та вичерпну документацію з хорошими прикладами

Для візуалізації результатів аналізу даних було обрано бібліотеку plotly, оскільки:

- Вона надає зручний інтерфейс для автоматичного будування графічних об’єктів
- Для графічних об’єктів наявна можливість дуже гнучкого налаштування з великою кількістю опцій для вигляду
- Наявна можливість будувати надзвичайно різноманітні графічні об’єкти
- Наявна чітка, зрозуміла та вичерпна документація з хорошими прикладами побудови різних графічних об’єктів з різними налаштуваннями

Структура бази даних

База даних має такі таблиці з полями:

1. teachers - вчителі
    - id - цілочисельний унікальний ідентифікатор
    - name - текстове поле, ім’я вчителя
    - surname - текстове поле, прізвище вчителя
2. groups – класи(групи)
    - id - цілочисельний унікальний ідентифікатор
    - name - текстове поле, назва профілю класу
    - year - цілочисельне значення року навчання
    - num_of_exc - цілочисельне значення кількості відмінників
    - teacher - цілочисельне значення id вчителя
3. classes - предмети
    - id - цілочисельний унікальний ідентифікатор
    - name - текстове поле, назва предмета
    - teacher - цілочисельне значення id вчителя

Опис програмного забезпечення

3.1. Загальна структура програмного забезпечення

Розроблене програмне забезпечення містить такі компоненти:

1. База даних, що зберігає інформацію про серії, набори та мініфігурки у чотирьох таблицях
2. Засоби CRUD-функціоналу
3. Засоби псевдовипадкової генерації даних
4. Засоби пошуку, фільтрації та валідації
5. Засоби реплікації з двома серверами-репліками

3.2. Опис модулів програмного забезпечення

Розроблене програмне забезпечення було розбите на такі модулі:

1. model

   Даний модуль напряму взаємодіє з базою даних. В цьому модулі знаходяться запити до бази даних, їх обробка та пов’язані з цим операції.

2. view

   Даний модуль відповідає за виведення даних у консоль та отримання даних від користувача з консолі. Усі використання функцій input та print знаходяться у цьому модулі.

3. controller

   Цей модуль пов’язує model та view і забезпечує обробку виключних ситуацій. Він отримує інформацію з model та передає її або іншу потрібну інформацію для відображення у view.

3.3. Опис основних алгоритмів роботи

При псевдовипадковій генерації даних для кожної з таблиць процес генерації був побудований так, аби генерувалися більш-менш адекватні дані для обраної предметної галузі та для обраної структури бази даних. Зокрема, у запитах були застосовані власноруч розроблені PostgreSQL-функції для забезпечення цілісності даних, унікальності тощо.

Для введенні даних відбувається обробка вхідних даних для забезпечення коректного їх занесення у таблиці.

Аналіз функціонування засобів реплікації

Тип реплікації, який було використано у даній курсовій роботі - logical replication. Для цього було виконано такі команди:

wal_level = logical

CREATE PUBLICATION mypub FOR TABLE classes, groups, teachers;

CREATE SUBSCRIPTION mysub CONNECTION 'dbname="school" host="localhost" user="postgres"' PUBLICATION mypub;

Зображення роботи функціонування засобів реплікації у додатках.

Аналіз функціонування засобів резервування/відновлення бази даних

Резервне копіювання необхідне для забезпечення безпечного та швидкого відновлення даних у разі втрати їх із бази даних. Тип резервного копіювання, який було реалізовано - повне. Це такий вид резервного копіювання, у якому щоразу копіюються повністю всі дані. Перевага такого різновиду резервного копіювання полягає у тому, що не потрібно об’єднувати різні файли для відновлення, натомість відновлюється все з одного файлу, за рахунок чого відновлення є помітно швидшим порівняно з іншими видами резервного копіювання. Було використано вбудоване резервне копіювання системи керування базами даних PostgreSQL.

Розроблений консольний додаток надає можливість користувачеві керувати резервним копіюванням та відновленням з консолі. Файли резервних копій зберігаються у корені проекту. У консольному додатку користувач може відновлювати базу даних.

Також додаток містить тригери, що дозволяє зберігати старі дані в окрему таблицю та при необхідності витягати їх. Тригерні табллиці містять ті ж стовпці, що й оригінальні. При видаленні чи редагуванні оригінальної таблиці дані автоматично записуються в додаткові таблиці. Приклад створення тригерів і таблиць можна знайти в додатках

Зображення роботи резервного копіювання у додатках.

Аналіз результатів підвищення швидкодії виконання запитів

З метою підвищення швидкодії запитів для отримання деяких даних було використано індексування деяких полів трьох таблиць. Тип індексування, який був використаний - BTREE. Індексування було застосовано до полів id таблиць teachers, groups, classes.

У випадку коли даних у таблиці багато (наприклад, 100 тисяч та більше) лінійний пошук стає заповільним, у зв’язку з чим для великих баз даних і потрібні індекси. Однак у разі малої бази даних індекси є неефективними. У зв’язку з цим індекси і застосовуються лише для великих баз даних.

Тестування індексів було проведено на базі даних у якій у кожній з трьох проіндексованих таблиць наявно 100 тисяч записів.

Запити створення індексів:

1. CREATE INDEX class_index ON classes USING BTREE(id);
2. CREATE INDEX ON groups USING BTREE(id);
3. CREATE INDEX ON teachers USING BTREE(id);

Запити, на яких тестувалися індекси:

1. SELECT * from groups WHERE id = 1753
2. SELECT * from teachers WHERE id = 1753
3. SELECT * from classes WHERE id = 1753

Результати наведені на діаграмі у додатках.

Опис результатів аналізу предметної галузі

У розробленому консольному додатку наявний такий аналіз даних, що містяться у базі:

- при псевдовипадковій генерації даних для кожної з таблиць процес генерації був побудований так, аби генерувалися більш-менш адекватні дані для обраної предметної галузі та для обраної структури бази даних. Зокрема, у запитах були застосовані власноруч розроблені PostgreSQL-функції для забезпечення цілісності даних, унікальності тощо.
- для введенні даних відбувається обробка вхідних даних для забезпечення коректного їх занесення у таблиці.

Приклади графіків та діаграми наведені у додатках.

Висновки

Під час виконання даної курсової роботи виконано таку роботу та отримано такі результати:

- Було розроблено базу даних, яка відповідає 3-ій нормальній формі та організована максимально зручно та просто
- Реплікацію було реалізовано виду physical streaming replication у конфігурації master-slave з можливістю перепідключення шляхом введення команди в консольному додатку
- Резервне копіювання було реалізовано повне, що дає можливість швидкого відновлення
- Була розроблена псевдовипадкова генерація для всіх таблиць, яка генерує реалістичні значення
- Була підвищена швидкодія запитів до бази даних шляхом індексування деяких полів деяких таблиць
- Були розроблені засоби для аналізу даних із бази, які також надають можливість виводити графічне представлення його результату для наочності висновків
- Були розроблені засоби генерації записів на основі готових датасетів (імпорт) а також засоби експорту даних у форматі CSV
- Був розроблений зручний консольний інтерфейс який також обробляє всі помилки, валідує дані та надає можливість виконання CRUD-операцій та фільтрації командами, а не SQL-запитами

У результаті виконання даної курсової роботи було набуто практичні навички розробки сучасного програмного забезпечення, що взаємодіє з реляційними базами даних, а також здобуто навички оформлення відповідного текстового, програмного та ілюстративного матеріалу у формі проектної документації.

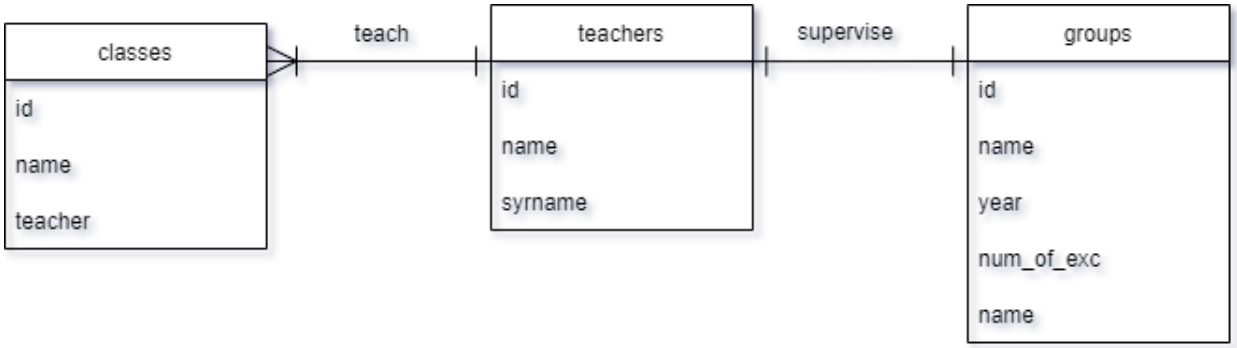
Завдяки виконанню даної роботи було здобуто вміння розробляти програмне забезпечення для реляційних баз даних, відбулося оволодіння основами використання СУБД, а також інструментальними засобами підтримки розробки додатків для подібних баз даних.

Література

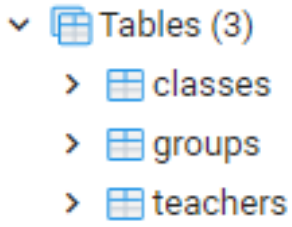
1. PostgreSQL 12.5 Documentation [Електронний ресурс] / The PostgreSQL Global Development Group // PostgreSQL: The World's Most Advanced Open Source Relational Database – Електрон. дані. – Режим доступу : https://www.postgresql.org/docs/12/index.html. – Назва з екрана.
2. Psycopg 2.8.7.dev0 documentation [Електронний ресурс] // Psycopg – Електрон. дані. – Режим доступу : https://www.psycopg.org/docs/. – Назва з екрана.
3. Plotly Python Open Source Graphing Library – Режим доступу : https://plotly.com/python/. – Назва з екрана.
4. time — Time access and conversions [Електронний ресурс] // Python. – Електрон. дані. – Режим доступу : https://docs.python.org/3/library/time.html. – Назва з екрана.
5. datetime — Basic date and time types [Електронний ресурс] // Python. – Електрон. дані. – Режим доступу : https://docs.python.org/3/library/datetime.html. – Назва з екрана.
6. pgAdmin 4 4.28 documentation [Електронний ресурс] // pgAdmin PostgreSQL Tools – Електрон. дані. – Режим доступу : https://www.pgadmin.org/docs/pgadmin4/4.28/index.html. – Назва з екрана.

Додатки

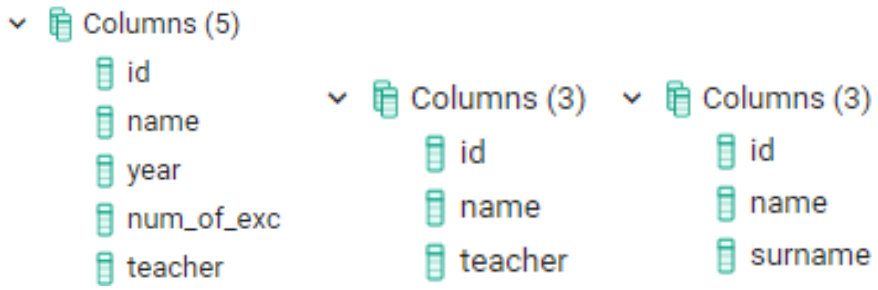
А. Графічні матеріали



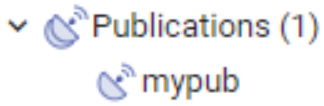
Структура бази даних



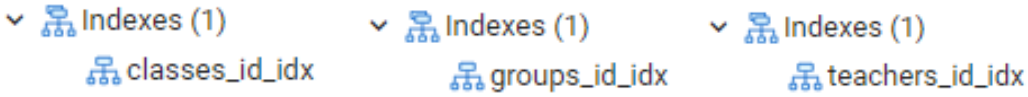
Таблиці у PgAdmin



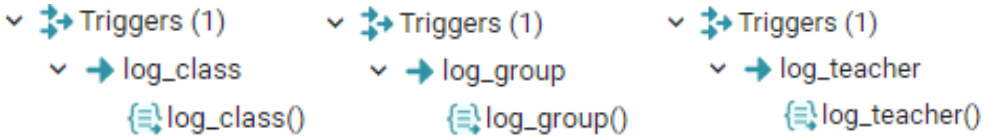
Колонки таблиці у PgAdmin



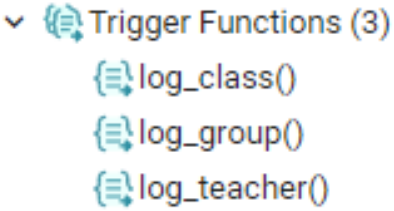
Публікації



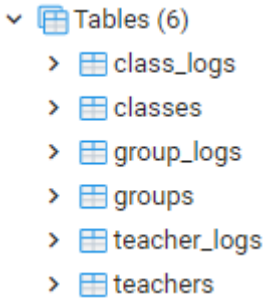
Індекси



Тригери



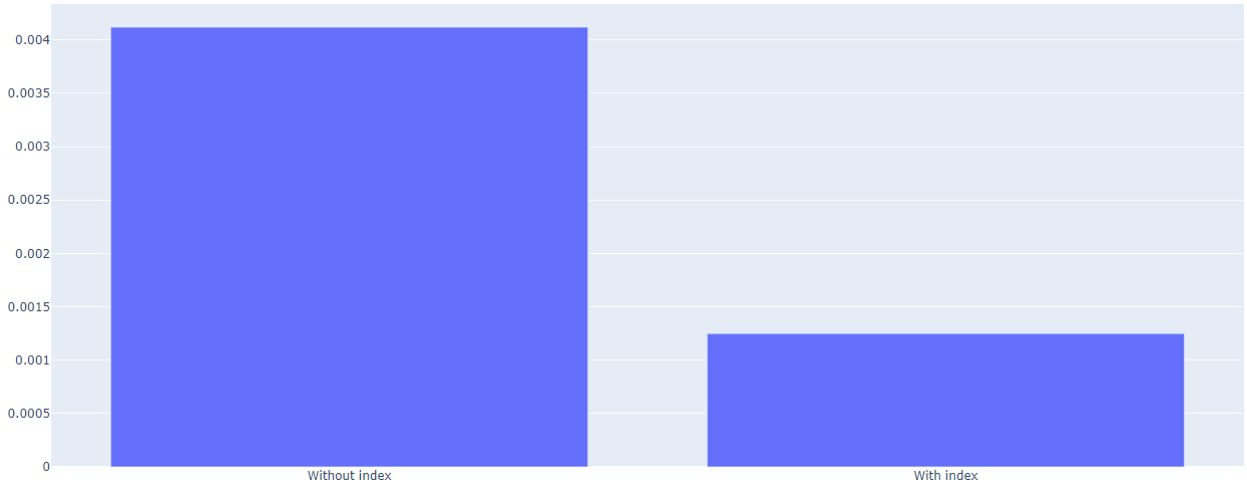
Тригерні функції



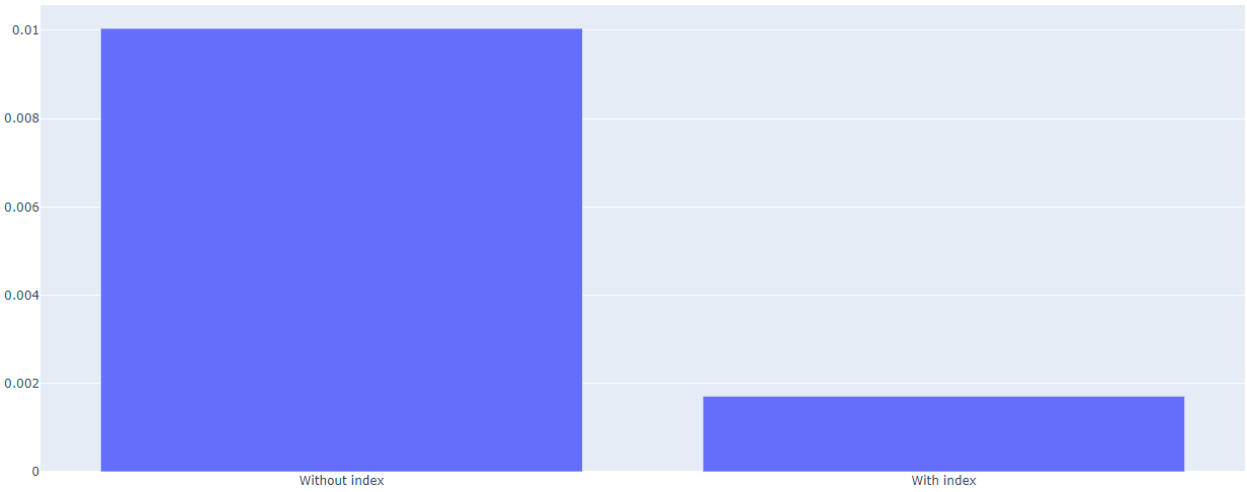
Таблиці …_logs для зберігання даних

```
10000 entities for teachers created. Time: 0.007131499999999402

10000 entities for groups created. Time: 0.004116200000000486

10000 entities for classes created. Time: 0.0013300000000000811
```

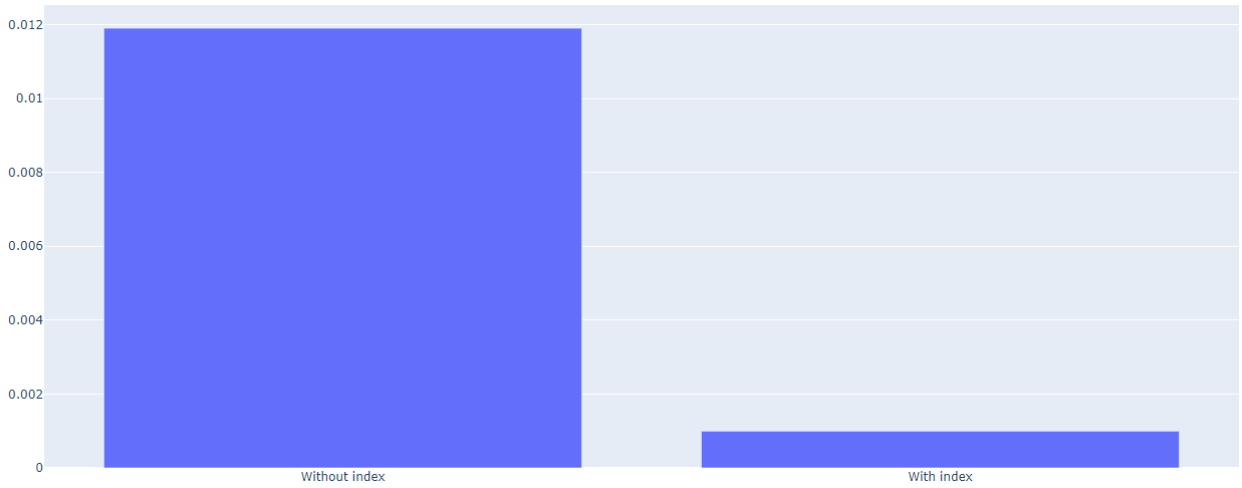
Час псевдорандомного створення сутностей



Графік порівняння пошуку в таблиці classes без і з індексом



Графік порівняння пошуку в таблиці groups без і з індексом



Графік порівняння пошуку в таблиці teachers без і з індексом

Б. Фрагменти програмного коду

Перевірка на коректність значень

```
class Validator:

    def __init__(self):
        print()

    def checkStr(self, val):
        if(not val):
            return False
        val = val.strip()
        if (len(val) < 2) :
            return False
        return True

    def checkNum(self, val):
        try:
            int(val)
        except ValueError:
            return False
        return True
```

Відновлення бази даних

```
class Restore:

    def Save(self):
        try:
            connection = psycopg2.connect(user="postgres",
                                          password="1",
                                          host="127.0.0.1",
                                          port="5432",
                                          database="school")
            cursor = connection.cursor()
            cursor.execute('SELECT groups.id, groups.name, groups.year,
groups.num_of_exc, groups.teacher, teachers.id, teachers.name,
teachers.surname, classes.id, classes.name, classes.teacher FROM groups,
teachers, classes')
            f = open('copy.backup', 'w')
            for row in cursor:
              f.write("insert into t values (" + str(row) + ");")
            connection.commit()
            print("Database saved")
```

```
        except (Exception, Error) as error:
            print("Error with PostgreSQL", error)
        finally:
            if connection:
                cursor.close()
                connection.close()
```

Створення тригерів бази даних

```
class Reserver:
    def ClassTrigger(self):
        try:
            connection = psycopg2.connect(user="postgres",
                                          password="1",
                                          host="127.0.0.1",
                                          port="5432",
                                          database="school")
            cursor = connection.cursor()
            query = """DROP TABLE IF EXISTS class_logs;
                        CREATE TABLE class_logs(id integer NOT NULL, name text,
teacher integer);
                        CREATE OR REPLACE FUNCTION log_class() RETURNS trigger
AS $BODY$
                        BEGIN
                            INSERT INTO class_logs VALUES(OLD.id, OLD.name,
OLD.teacher);
                            RETURN NEW;
                        END;
                    $BODY$ LANGUAGE plpgsql;
                    DROP TRIGGER IF EXISTS log_class ON classes;
                    CREATE TRIGGER log_class BEFORE UPDATE OR DELETE ON classes
                        FOR EACH ROW EXECUTE PROCEDURE log_class();"""
            cursor.execute(query)
            connection.commit()
            print('Trigger was created')
        except (Exception, Error) as error:
            print("Error with PostgreSQL", error)
        finally:
            if connection:
                cursor.close()

    def TeacherTrigger(self):
        try:
            connection = psycopg2.connect(user="postgres",
                                          password="1",
                                          host="127.0.0.1",
                                          port="5432",
                                          database="school")
            cursor = connection.cursor()
```

```
            query = """DROP TABLE IF EXISTS teacher_logs;
                        CREATE TABLE teacher_logs(id integer NOT NULL, name
text, surname text);
                        CREATE OR REPLACE FUNCTION log_teacher() RETURNS
trigger AS $BODY$
                        BEGIN
                            INSERT INTO teacher_logs VALUES(OLD.id, OLD.name,
OLD.surname);
                            RETURN NEW;
                        END;
                    $BODY$ LANGUAGE plpgsql;
                    DROP TRIGGER IF EXISTS log_teacher ON teachers;
                    CREATE TRIGGER log_teacher BEFORE UPDATE OR DELETE ON
teachers
                        FOR EACH ROW EXECUTE PROCEDURE log_teacher();"""
            cursor.execute(query)
            connection.commit()
            print('Trigger was created')
        except (Exception, Error) as error:
            print("Error with PostgreSQL", error)
        finally:
            if connection:
                cursor.close()

    def GroupTrigger(self):
        try:
            connection = psycopg2.connect(user="postgres",
                                          password="1",
                                          host="127.0.0.1",
                                          port="5432",
                                          database="school")
            cursor = connection.cursor()
            query = """DROP TABLE IF EXISTS group_logs;
                        CREATE TABLE group_logs(id integer NOT NULL, name text,
year integer, num_of_exc integer, teacher integer);
                        CREATE OR REPLACE FUNCTION log_group() RETURNS trigger
AS $BODY$
                        BEGIN
                            INSERT INTO group_logs VALUES(OLD.id, OLD.name,
OLD.year, OLD.num_of_exc, OLD.teacher);
                            RETURN NEW;
                        END;
                    $BODY$ LANGUAGE plpgsql;
                    DROP TRIGGER IF EXISTS log_group ON groups;
                    CREATE TRIGGER log_group BEFORE UPDATE OR DELETE ON groups
                        FOR EACH ROW EXECUTE PROCEDURE log_group();"""
            cursor.execute(query)
            connection.commit()
            print('Trigger was created')
        except (Exception, Error) as error:
```

```
            print("Error with PostgreSQL", error)
        finally:
            if connection:
                cursor.close()
```

Оптимізація з використанням індексів

```
class Index:

    def Create(self):
        try:
            connection = psycopg2.connect(user="postgres",
                                          password="1",
                                          host="127.0.0.1",
                                          port="5432",
                                          database="school")
            cursor = connection.cursor()
            selecr_query = """CREATE INDEX class_index ON classes USING
BTREE(id); CREATE INDEX  ON groups USING BTREE(id); CREATE INDEX ON teachers
USING BTREE(id);"""
            cursor.execute(selecr_query)
            connection.commit()
            print("Indexes were created.")
        except (Exception, Error) as error:
            print("Error with PostgreSQL", error)
        finally:
            if connection:
                cursor.close()
                connection.close()

    def Drop(self):
        try:
            connection = psycopg2.connect(user="postgres",
                                          password="1",
                                          host="127.0.0.1",
                                          port="5432",
                                          database="school")
            cursor = connection.cursor()
            selecr_query = """DROP INDEX class_index; DROP INDEX group_index;
DROP INDEX teacher_index; """
            cursor.execute(selecr_query)
            connection.commit()
            print("Indexes were created.")
        except (Exception, Error) as error:
            print("Error with PostgreSQL", error)
        finally:
            if connection:
                cursor.close()
                connection.close()
```

Генерація даних

```
class RandomClasses:
    def __init__(self):
        self.id = 0
        self.name = ""
        self.teacher = 0

    def random(self, n):
        res = n
        names = ['Math', 'English', 'Geography', 'History', 'PE', 'Ukrainian',
'Literature', 'Sciense', 'Law']
        try:
            connection = psycopg2.connect(user="postgres",
                                          password="1",
                                          host="127.0.0.1",
                                          port="5432",
                                          database="school")
            cursor = connection.cursor()
            cursor.execute("DROP TABLE IF EXISTS classes; CREATE TABLE
classes(id integer NOT NULL, name text, teacher integer);")
            for i in range(n):
                try:
                    name = names[random.randint(0, len(names) -
1)]
                    cursor.execute("INSERT INTO classes (id, name, teacher)
VALUES ((SELECT trunc(random() * %s + 1)::int), %s, (SELECT trunc(random() * %s
+ 1)::int))", [n, name, n])
                except(Exception, Error) as error:
                    print("Error with PostgreSQL", error)
                    res-=1
                connection.commit()
        except (Exception, Error) as error:
            print("Error with PostgreSQL", error)
        finally:
            if connection:
                cursor.close()
                connection.close()
        print(str(res) + " Entities added.")

class RandomTeachers:
    def __init__(self):
        self.id = 0
        self.name = ""
        self.surname = ""

    def random(self, n):
        res = n
        names = ['Inna', 'Kyrulo', 'Anna', 'Igor', 'Ivan', 'Evgeniya', 'Lesya',
'Sergey', 'Alla', 'Andriy']
```

```
        surnames = ['Kyrychenko', 'Lightman', 'Anfisov', 'Soldatenko',
'Kravchuk', 'Los', 'Tkach', 'Shevchenko', 'Ivanov', 'Petrov', 'Sydorov']
        try:
            connection = psycopg2.connect(user="postgres",
                                          password="1",
                                          host="127.0.0.1",
                                          port="5432",
                                          database="school")
            cursor = connection.cursor()
            cursor.execute("DROP TABLE IF EXISTS teachers; CREATE TABLE
teachers(id integer NOT NULL, name text, surname text)")
            for i in range(n):
                try:
                    name = names[random.randint(0, len(names) - 1)]
                    surname = surnames[random.randint(0, len(names) - 1)]
                    cursor.execute("INSERT INTO teachers (id, name, surname)
VALUES ((SELECT trunc(random() * %s + 1)::int), %s, %s)", [n, name, surname])
                except:
                    res-=1
                connection.commit()
        except (Exception, Error) as error:
            print("Error with PostgreSQL", error)
        finally:
            if connection:
                cursor.close()
                connection.close()
        print(str(res) + " Entities added.")

class RandomGroups:
    def __init__(self):
        self.id = 0
        self.name = ""
        self.year = 0
        self.num_of_exc = 0
        self.teacher = 0

    def random(self, n):
        res = n
        names = ['Math', 'Linguistic', 'History', 'Sciense']
        try:
            connection = psycopg2.connect(user="postgres",
                                          password="1",
                                          host="127.0.0.1",
                                          port="5432",
                                          database="school")
            cursor = connection.cursor()
            cursor.execute("DROP TABLE IF EXISTS groups; CREATE TABLE groups(id
integer NOT NULL, name text, year integer, num_of_exc integer, teacher
integer)")
            for i in range(n):
```

```
                try:
                    name = names[random.randint(0, len(names) - 1)]
                    cursor.execute("INSERT INTO groups (id, name, year,
num_of_exc, teacher) VALUES ((SELECT trunc(random() * %s + 1)::int), %s,
(SELECT trunc(random() * 11 + 1)::int), (SELECT trunc(random() * %s + 1)::int),
(SELECT trunc(random() * %s + 1)::int))", [n, name, n, n])
                except:
                    res-=1
                connection.commit()
        except (Exception, Error) as error:
            print("Error with PostgreSQL", error)
        finally:
            if connection:
                cursor.close()
                connection.close()
        print(str(res) + " Entities added.")
```

Зв’язок із базою даних (частина класу model)

```
class Classes:
    def __init__(self):
        self.id = 0
        self.name = ""
        self.teacher = 0

    def create(self, id, name, teacher):
        if (id < 1):
            print('Error with input!')
            return
        try:
            connection = psycopg2.connect(user="postgres",
                                          password="1",
                                          host="127.0.0.1",
                                          port="5432",
                                          database="school")
            cursor = connection.cursor()
            insert_query = """ INSERT INTO classes (id, name, teacher) VALUES
(%s, %s, %s)"""
            item_tuple = (id, name, teacher)
            cursor.execute(insert_query, item_tuple)
            connection.commit()
            print("Entity inserted")

        except (Exception, Error) as error:
            print("Error with PostgreSQL", error)
        finally:
            if connection:
                cursor.close()
                connection.close()
```

```
    def update(self, id, name, teacher):
        if (id < 1):
            print('Error with input!')
            return
        try:
            connection = psycopg2.connect(user="postgres",
                                          password="1",
                                          host="127.0.0.1",
                                          port="5432",
                                          database="school")
            cursor = connection.cursor()
            update_query = """Update classes SET name=%s, teacher=%s WHERE id =
%s"""
            item_tuple = (name, teacher, id)
            cursor.execute(update_query, item_tuple)
            connection.commit()
            count = cursor.rowcount
            print(count, "Entity updated")
        except (Exception, Error) as error:
            print("Error with PostgreSQL", error)
        finally:
            if connection:
                cursor.close()
                connection.close()


    def delete(self, id):
        if (id < 1):
            print('Error with input!')
            return
        try:
            connection = psycopg2.connect(user="postgres",
                                          password="1",
                                          host="127.0.0.1",
                                          port="5432",
                                          database="school")
            cursor = connection.cursor()
            cursor.execute("Delete from classes WHERE id = %s;", [id])
            connection.commit()
            count = cursor.rowcount
            print(count, "Entity deleted")
        except (Exception, Error) as error:
            print("Error with PostgreSQL", error)
        finally:
            if connection:
                cursor.close()
                connection.close()

    def read(self, id):
```

```
        if (id < 1):
            print('Error with input!')
            return
        try:
            connection = psycopg2.connect(user="postgres",
                                          password="1",
                                          host="127.0.0.1",
                                          port="5432",
                                          database="school")
            cursor = connection.cursor()
            selecr_query = """SELECT * from classes WHERE id = %s"""
            item_tuple = [id]
            cursor.execute(selecr_query, item_tuple)
            connection.commit()
            records = cursor.fetchall()
            print("Result:")
            for row in records:
                print("Id = ", row[0], "Name = ", row[1], "Teacher  = ",
row[2])
        except (Exception, Error) as error:
            print("Error with PostgreSQL", error)
        finally:
            if connection:
                cursor.close()
                connection.close()
```

Створення графіків

```
class Plot:

    def Create(self, a, b, file):
        types = ['Without index', 'With index']
        secs = [a, b]
        data = [go.Bar(
        x = types,
        y = secs
        )]
        fig = go.Figure(data=data)
        fig.write_image("img/" + file + ".pdf")
        fig.show()
```