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: ***«Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»***

Виконав: студент III курсу

ФПМ групи КВ-02
Телеганенко Б.В.

Перевірив:

Київ
2022

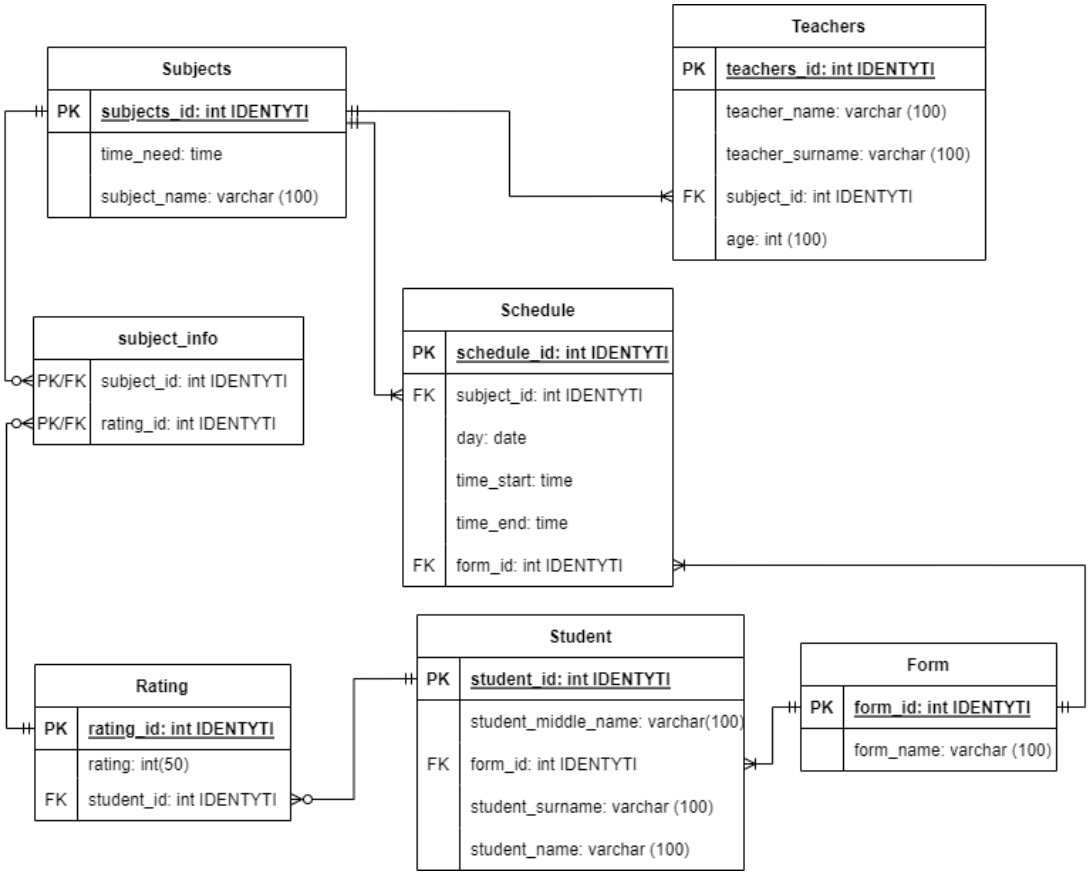
Метою роботи є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

Загальне завдання роботи полягає у наступному:

1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

Репозиторій GitHub з вихідним кодом:
https://github.com/nothing549/database/tree/lab2

Логічна модель для предметної галузі: ***Школа.***



Опис бази даних:

1)*Предмети(Subjects)*, з атрибутами: назва предмету, кількість потрібних для засвоєння матеріалу годин. Призначена для збереження інформації про предмети, які викладаються в школі.

2)*Вчителі(Teachers)*, з атрибутами: ім’я вчителя, прізвище вчителя, предмет, що викладає, вік вчителя. Призначена для збереження інформації про вчителів, які працюють у школі.

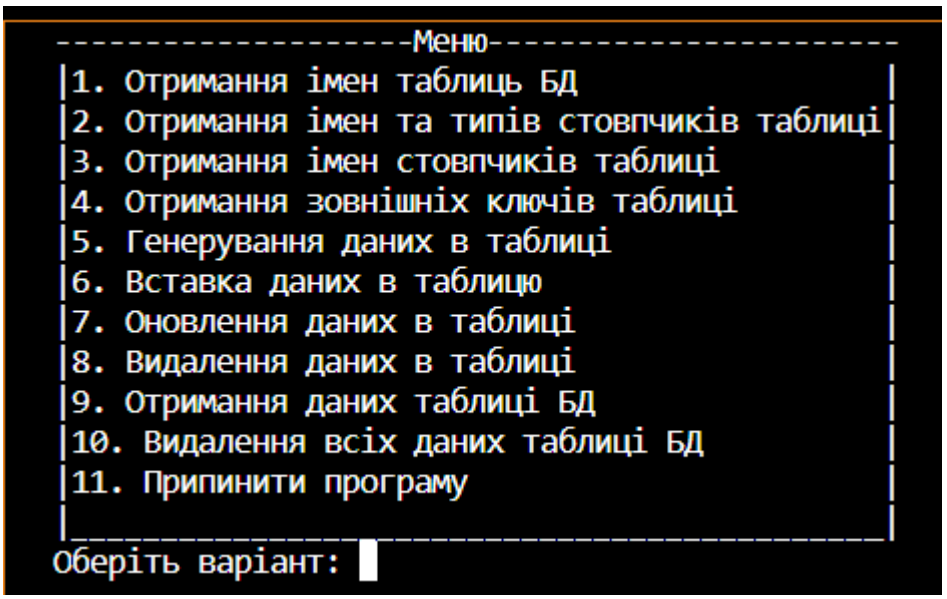
3)*Учні(Student),* з атрибутами: ім’я учня, прізвище учня, по батькові учня, назва класу. Призначена для збереження інформації про учнів, які вчаться в школі.

4)*Розклад(Schedule)*, з атрибутами: назва предмету, день тижня, час початку, час кінця, назва класу. Призначена для збереження інформації про розклад предметів.

5)*Оцінки(Rating)*, з атрибутами: назва предмету, інд. номер учня, оцінка. Призначена для збереження інформації про досягнення учнів, що навчаються у школі.

6)*Клас (Form),* з атрибутами: назва класу призначена для збереження інформації про класи у школі.

Меню програми:



Меню для користувача складається з одинадцяти пунктів:

1. Отримання імен таблиць бази даних.
2. Отримання імен та типів стовпчиків таблиці.
3. Отримання імен стовпчиків таблиці.
4. Отримання зовнішніх ключів таблиці.
5. Генерування даних в таблиці.
6. Вставка даних в таблицю.
7. Оновлення даних в таблиці.
8. Видалення даних з таблиці.
9. Отримання даних таблиці.
10. Видалення всіх даних таблиці БД.
11. Припинення роботи програми.

Меню інтуїтивно зрозуміле, бо назва кожного пункту повністю відповідає тому, що саме він робить.

Інструментарій

Середовище для відлагодження SQL-запитів до бази даних – PgAdmin4.

Мова програмування – Python 3.10

Середовище розробки програмного забезпечення – PyCharm.

Стороння бібліотека для роботи з PostgreSQL: *Psycopg2.*

Шаблон проектування

Програма виконана по шаблону MVC.

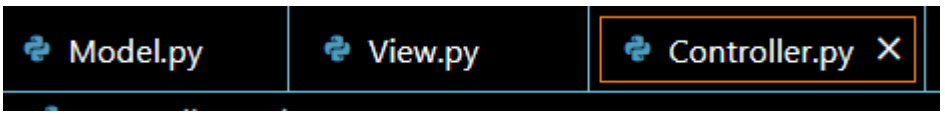
Model – представляє клас, що описує логіку використовуваних даних.

View – консольний інтерфейс з яким буде взаємодіяти користувач.

Controller – представляє клас, що забезпечує зв'язок між користувачем і системою, поданням і сховищем даних.

Структура програми

Програма поділена на 3 модулі: Model, View та Controller.



Кожен модуль містить відповідно до шаблону MVC інформацію.

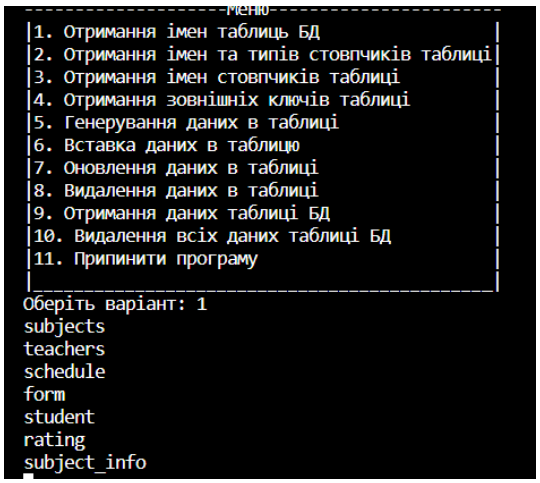
Модуль Model містить визначення усіх методів, які використовуються в програмі. Також цей модуль відповідає за підключення до БД.

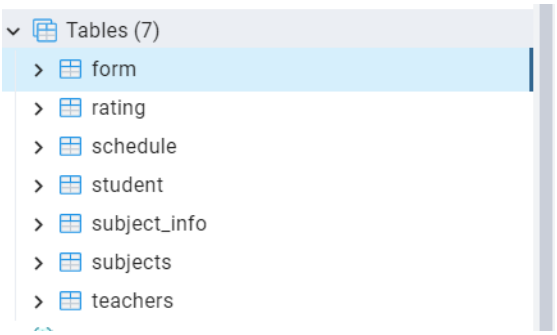
Модуль View описує зовнішній вигляд інтерфейсу, з яким користувач буде взаємодіяти. В конкретному випадку він описує консольний інтерфейс.

Модуль Controller є своєрідним зв’язком між модулем View та Model, він контролює увесь процес роботи програми.

Функції перегляду, внесення, редагування та вилучення даних

Перегляд даних:





```
--------------------Меню-----------------------
|1. Отримання імен таблиць БД                 |
|2. Отримання імен та типів стовпчиків таблиці|
|3. Отримання імен стовпчиків таблиці         |
|4. Отримання зовнішніх ключів таблиці        |
|5. Генерування даних в таблиці               |
|6. Вставка даних в таблицю                   |
|7. Оновлення даних в таблиці                 |
|8. Видалення даних в таблиці                 |
|9. Отримання даних таблиці БД                |
|10. Видалення всіх даних таблиці БД          |
|11. Припинити програму                       |
|_____________________________________________|
Оберіть варіант: 2
Введіть назву таблиці: form
('form_id', 'integer')
('form_name', 'character varying')
```

form

General | Columns | Advanced | Constraints | Parameters | Security | SQL

Inherited from table(s) — Select to inherit from...

Columns

		Name	Data type	Length/Precision	Scale	N
✎	🗑	form_id	integer			
✎	🗑	form_name	character varying	100		

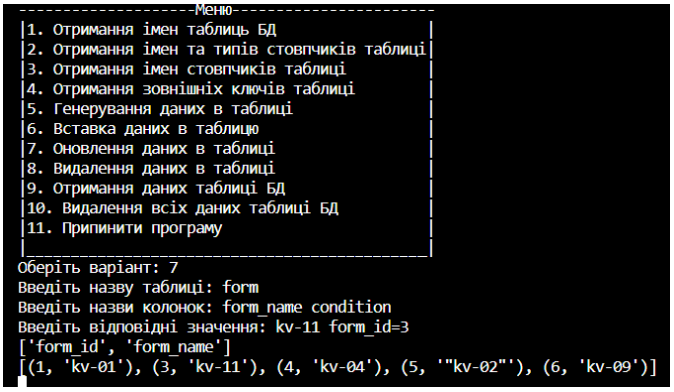
Внесення даних:

```
--------------------Меню-----------------------
|1. Отримання імен таблиць БД                 |
|2. Отримання імен та типів стовпчиків таблиці|
|3. Отримання імен стовпчиків таблиці         |
|4. Отримання зовнішніх ключів таблиці        |
|5. Генерування даних в таблиці               |
|6. Вставка даних в таблицю                   |
|7. Оновлення даних в таблиці                 |
|8. Видалення даних в таблиці                 |
|9. Отримання даних таблиці БД                |
|10. Видалення всіх даних таблиці БД          |
|11. Припинити програму                       |
|_____________________________________________|
Оберіть варіант: 6
Введіть назву таблиці: form
Введіть назви колонок: form_name
Введіть відповідні значення: kv-09
result:

['form_id', 'form_name']
[(1, 'kv-01'), (3, 'kv-03'), (4, 'kv-04'), (5, '"kv-02"'), (6, 'kv-09')]
```

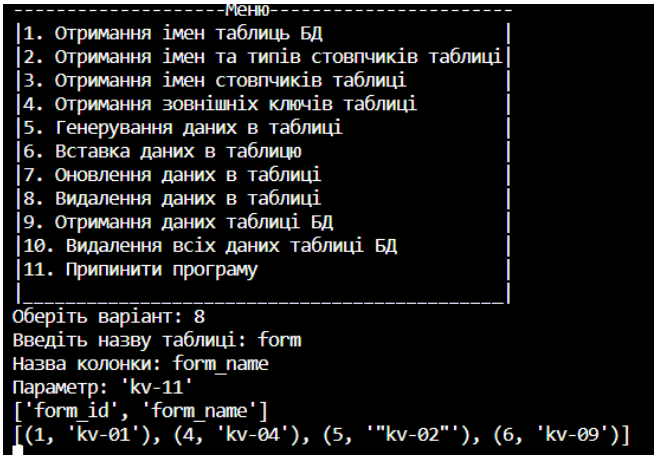
	form_id [PK] integer	form_name character varying (100)
1	1	kv-01
2	3	kv-03
3	4	kv-04
4	5	"kv-02"
5	6	kv-09

Редагування даних:



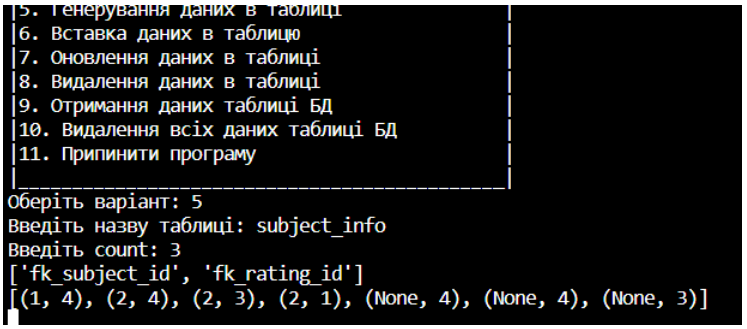
	form_id [PK] integer	form_name character varying (100)
1	1	kv-01
2	4	kv-04
3	5	"kv-02"
4	6	kv-09
5	3	kv-11

Вилучення даних:



	form_id [PK] integer	form_name character varying (100)
1	1	kv-01
2	4	kv-04
3	5	"kv-02"
4	6	kv-09

Генерування даних



	fk_subject_id integer	fk_rating_id integer
1	2	3
2	1	4
3	2	4
4	2	1
5	[null]	4
6	[null]	4
7	[null]	3

Реалізація перегляду, внесення, редагування та вилучення даних програмним кодом

Отримання імен таблиць

```
def get_table_names(self):
    if self.__table_names is None:
        self.__cursor.execute("""SELECT table_name
                    FROM information_schema.tables
                    WHERE table_schema = 'public'""")
        self.__table_names = [table[0] for table in self.__cursor]
    return self.__table_names
```

Вставка даних в таблицю

```
def insert_data(self, table_name, values):
    #Формування рядка значень VALUES запиту INSERT
    line = ''
    columns = '('
    for key in values:
        if values[key]:
            line += '%(' + key + ')s,'
            columns += key + ','
    columns = columns[:-1] + ')'
    #Виконання SQL-запиту на вставку даних в таблицю
    self.__cursor.execute(
        sql.SQL('INSERT INTO {} {} VALUES (' + line[:-1] + ')')
            .format(sql.Identifier(table_name), sql.SQL(columns)),
        values)
```

```
        self.__context.commit()
```

Генерація випадкових даних і заповнення таблиці

```
    def generate_data(self, table_name, count):
        types = self.get_column_types(table_name)
        fk_array = self.get_foreign_key_info(table_name)
        select_subquery = ""
        insert_query = "INSERT INTO " + table_name + " ("
        for i in range(1, len(types)):
            t = types[i]
            name = t[0]
            type = t[1]
            fk = [x for x in fk_array if x[0] == name]
            if fk:
                select_subquery += ('(SELECT {} FROM {} ORDER BY RANDOM(), ser LIMIT
1)'.format(fk[0][2], fk[0][1]))
            elif type == 'integer':
                select_subquery += 'trunc(random()*100)::INT'
            elif type == 'character varying':
                select_subquery += 'chr(trunc(65 + random()*25)::INT) || chr(trunc(65 +
ran-dom()*25)::INT)'
            elif type == 'date':
                select_subquery += """ date(timestamp '2014-01-10' + random() * (timestamp
'2020-01-20' - timestamp '2014-01-10'))"""
            elif type == 'time without time zone':
                select_subquery += "time '00:00:00' + DATE_TRUNC('second',RANDOM() * time
'24:00:00')"
            else:
                continue
```

```
            insert_query += name
            if i != len(types) - 1:
                select_subquery += ','
                insert_query += ','
            else:
                insert_query += ') '

        self.__cursor.execute(
            insert_query + "SELECT " + select_subquery +
                "FROM generate_series(1," + str(count) + ") as ser")
        self.__context.commit()
```

Оновлення даних в таблиці

```
    def change_data(self, table_name, values):
        #Формування рядка значень SET запиту UPDATE
        line = ''
        condition = values.pop('condition')
        for key in values:
            if values[key]:
                line += key + '=%(' + key + ')s,'
        #Виконання SQL-запиту на оновлення даних в таблиці
        self.__cursor.execute(
            sql.SQL('UPDATE {} SET ' + line[:-1] + ' WHERE {} ')
                .format(sql.Identifier(table_name), sql.SQL(condition)),
            values)
        self.__context.commit()
```

Видалення даних в таблиці

```
def delete_data(self, table_name, value, cond):
    self.__cursor.execute(
        sql.SQL('DELETE FROM {} WHERE {} = {}')
            .format(sql.Identifier(table_name), sql.Identifier(value), sql.SQL(cond)))
    self.__context.commit()


def delete_all_data(self, table_name):
    self.__cursor.execute(
        sql.SQL('DELETE * FROM {}')
            .format(sql.Identifier(table_name)))
    self.__context.commit()


def join_general(self, main_query, condition=""):
    new_cond = condition
    if condition:
        new_cond = "WHERE " + condition
    t1 = time.time()
    self.__cursor.execute(main_query.format(new_cond))
    t2 = time.time()
    return ((t2 - t1) * 1000, self.__cursor.fetchall())
```