Національний технічний університет України “Київський політехнічний інститут ім. Ігоря Сікорського”

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп’ютерних систем

Розрахункова графічна робота

“Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL”

Група: КВ-11

Виконав:Овчінніков Д.С.

Телеграм @dmytro_ovchinnikov

Оцінка:

Київ – 2023

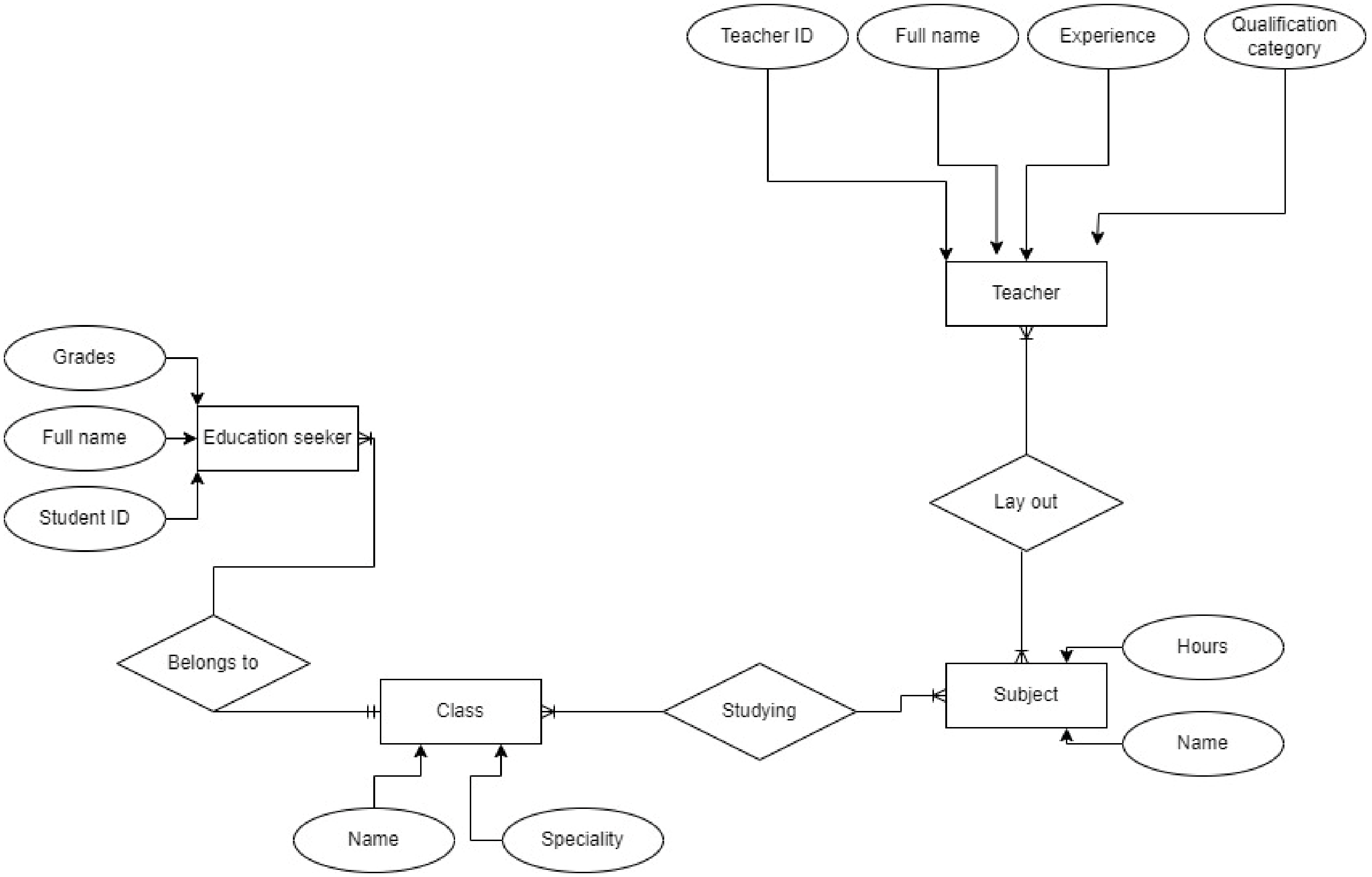
Посилання на репозиторій https://github.com/OvchinnikovDmytro/DataBase

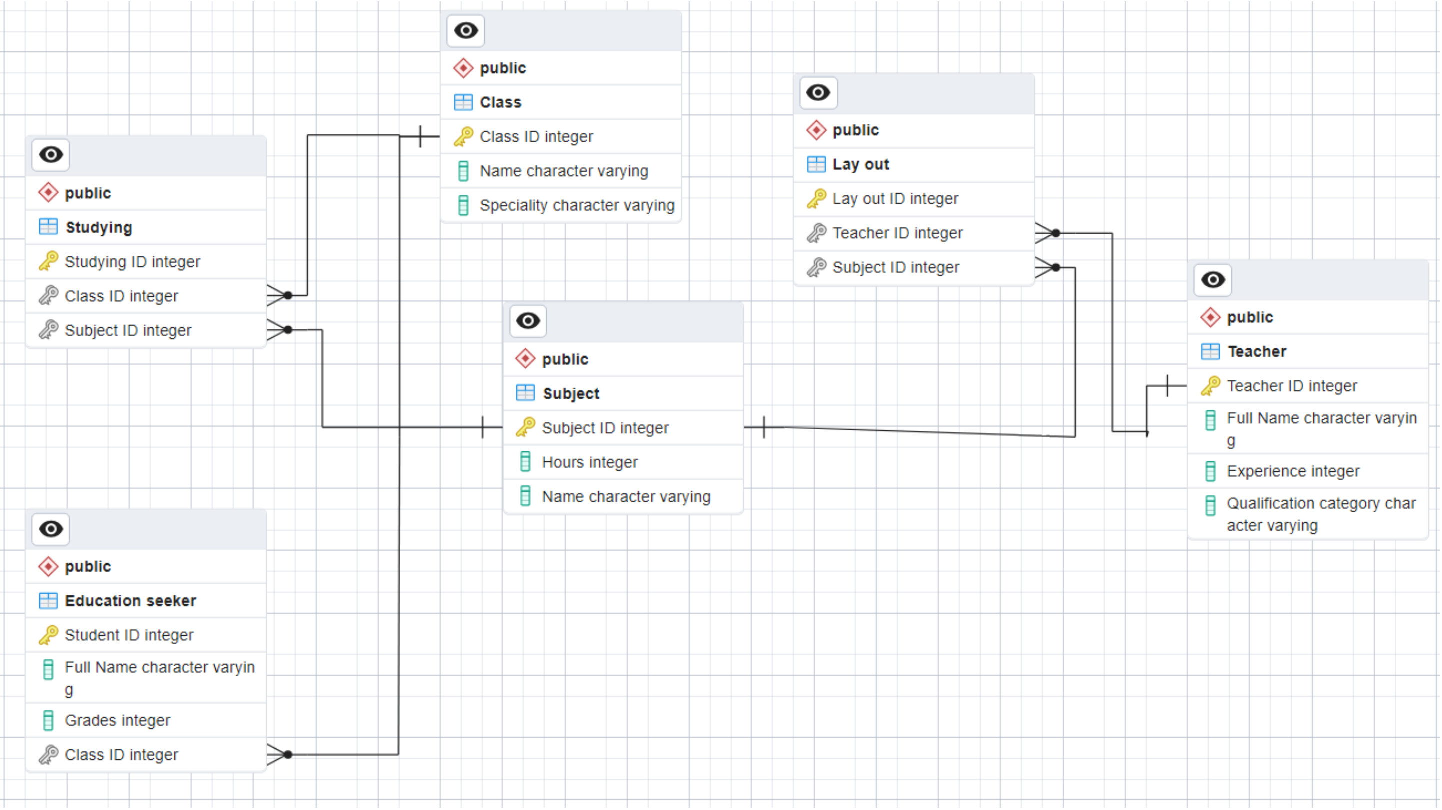
Метою роботи є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

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).

ER модель «Шкільна система управління навчанням.»
Перелік сутностей і опис їх призначення:
Здобувач освіти (Education seeker), сутність призначена для збереження даних про здобувача освіти – ПІБ, середня оцінка, та ID учнівського,ID класу.
Клас (Class), сутність призначена для збереження даних про клас здобувачів освіти для відслідковування року їх навчання та предмету їх поглибленого вивчення.
Предмет (Subject), сутність призначена для збереження даних про предмети що вивчаються в школі – назву предмету, годин вивчення предмета.
Вчитель (Teacher), сутність призначена для збереження даних про вчителя – ПІБ, Стаж, та Кваліфікаційна категорія.
Здобувач може навчатися тільки в одному класі. Для одного класу викладається декілька предметів і предмети викладаються для декількох класів. Вчитель може викладати декілька предметів і викладати ці ж предмети може і інший вчитель.





public
Class
Class ID integer
Name character varying
Speciality character varying

public
Lay out
Lay out ID integer
Teacher ID integer
Subject ID integer

public
Studying
Studying ID integer
Class ID integer
Subject ID integer

public
Subject
Subject ID integer
Hours integer
Name character varying

public
Teacher
Teacher ID integer
Full Name character varyin g
Experience integer
Qualification category char acter varying

public
Education seeker
Student ID integer
Full Name character varyin g
Grades integer
Class ID integer

Меню користувача

1. Class – взаємодія з таблицею Class
    1.1. Add Class – додати значення до таблиці
    1.2. View Class – вивести всі значення таблиці
    1.3. Update Class – оновити значення таблиці
    1.4. Delete Class – видалити значення таблиці
    1.5.Generate Class – згенерувати випадкові дані в таблицю
    1.6.Find Class – знайти деякий клас
    1.7.Quit - вийти

2. Education seeker – взаємодія з таблицею Education seeker
    2.1. Add Education seeker – додати значення до таблиці
    2.2. View Education seeker – вивести всі значення таблиці
    2.3. Update Education seeker – оновити значення таблиці
    2.4. Delete Education seeker – видалити значення таблиці
    2.5.Generate Education seeker – згенерувати випадкові дані в таблицю
    2.6.Find Education seeker – знайти деякого учня
    2.7.Quit – вийти

3. Subject – взаємодія з таблицею Subject
    3.1. Add Subject – додати значення до таблиці
    3.2. View Subject – вивести всі значення таблиці
    3.3. Update Subject – оновити значення таблиці
    3.4. Delete Subject – видалити значення таблиці
    3.5.Generate Subject – згенерувати випадкові дані в таблицю
    3.6.Find Subject – знайти деякий предмет
    3.7.Quit – вийти

4. Teacher – взаємодія з таблицею Teacher
    4.1. Add Teacher – додати значення до таблиці
    4.2. View Teacher – вивести всі значення таблиці
    4.3. Update Teacher – оновити значення таблиці
    4.4. Delete Teacher – видалити значення таблиці
    4.5.Generate Teacher – згенерувати випадкові дані в таблицю
    4.6.Find Teacher – знайти деякого вчителя
    4.7.Quit – вийти

5. Lay out – взаємодія з таблицею Lay out
    5.1. Add Lay out – додати значення до таблиці
    5.2. View Lay out – вивести всі значення таблиці
    5.3. Update Lay out – оновити значення таблиці
    5.4. Delete Lay out – видалити значення таблиці
    5.5.Generate Lay out – згенерувати випадкові дані в таблицю

    5.6.Quit – вийти

6. Studying – взаємодія з таблицею Studying
    6.1. Add Studying – додати значення до таблиці
    6.2. View Teacher – вивести всі значення таблиці
    6.3. Update Studying – оновити значення таблиці
    6.4. Delete Studying – видалити значення таблиці
    6.5.Generate Studying – згенерувати випадкові дані в таблицю
    6.6.Quit – вийти

7. Find – знайти записи за певними критеріями
8. Exit – вихід з програми

Пункт №1

Початкові таблиці

```
Classes:
ID: 1, Name: 11, Speciality: Math
Education seekers:
ID: 28, Full Name: Oleksa Oleksandr Olga, Grade: 9, Class ID 1
```

Операція вилучення запису батьківської таблиці

```
Menu:
1. Class
2. Education Seeker
3. Subject
4. Teacher
5. Lay out
6. Studying
7. Find
8. Exit
Enter your choice: 1

Class Menu:
```

```
1. Add Class
2. View Class
3. Update Class
4. Delete Class
5. Generate Class
6. Find Class
7. Quit
Enter your choice: 4
Enter class ID: 1
Class is not deleted.Delete all student from it first
```

Клас не видалився, бо до класу був прив'язаний здобувач освіти,після видалення його можна буде видалити і клас.

Спроба додати запис в дочірню таблицю, без відповідного запису в батьківській:

```
Menu:
1. Class
2. Education Seeker
3. Subject
4. Teacher
5. Lay out
6. Studying
7. Find
8. Exit
Enter your choice: 2

Education seeker Menu:
1. Add Education seeker
2. View Education seeker
3. Update Education seeker
4. Delete Education seeker
5. Generate Education seeker
6. Find Education seeker
7. Quit
```

```
Enter your choice: 1
Enter Student Full Name: Vasyl Vovchenko
Enter Student grades: 10
Enter Student class: 3
Education seeker isnt added.There is no such a class id
```

Здобувач освіти не був доданий так як не існує класу з заданим айді

Пункт №2

Генерація 100 записів в таблиці Class

```
Classes:
ID: 1, Name: 11, Speciality: Math
ID: 2, Name: 9, Speciality: Art
ID: 3, Name: 7, Speciality: Ukrainian
ID: 4, Name: 12, Speciality: Literature
ID: 5, Name: 12, Speciality: History
ID: 6, Name: 9, Speciality: Art
ID: 7, Name: 12, Speciality: Ukrainian
ID: 8, Name: 7, Speciality: Ukrainian
ID: 9, Name: 8, Speciality: Ukrainian
ID: 10, Name: 10, Speciality: Ukrainian
ID: 11, Name: 7, Speciality: Science
ID: 12, Name: 11, Speciality: Science
ID: 13, Name: 7, Speciality: Math
ID: 14, Name: 7, Speciality: Science
ID: 15, Name: 8, Speciality: History
ID: 16, Name: 8, Speciality: Science
ID: 17, Name: 11, Speciality: Art
ID: 18, Name: 9, Speciality: Literature
```

ID: 19, Name: 7, Speciality: Literature

ID: 20, Name: 9, Speciality: Science

ID: 21, Name: 9, Speciality: Art

ID: 22, Name: 12, Speciality: Science

……………………………………………

ID: 90, Name: 12, Speciality: Literature

ID: 91, Name: 12, Speciality: History

ID: 92, Name: 9, Speciality: Science

ID: 93, Name: 9, Speciality: History

ID: 94, Name: 10, Speciality: Science

ID: 95, Name: 11, Speciality: History

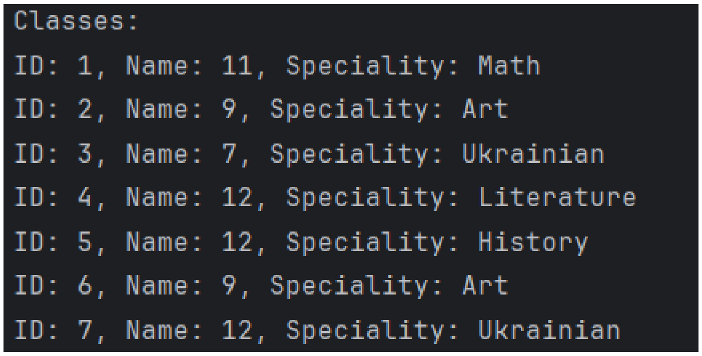
ID: 96, Name: 10, Speciality: Literature

ID: 97, Name: 9, Speciality: Art

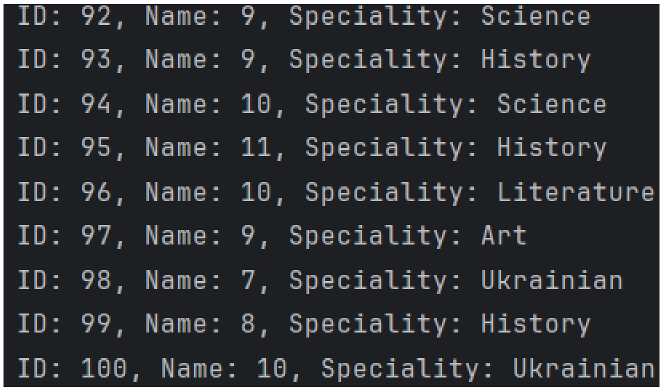
ID: 98, Name: 7, Speciality: Ukrainian

ID: 99, Name: 8, Speciality: History

ID: 100, Name: 10, Speciality: Ukrainian

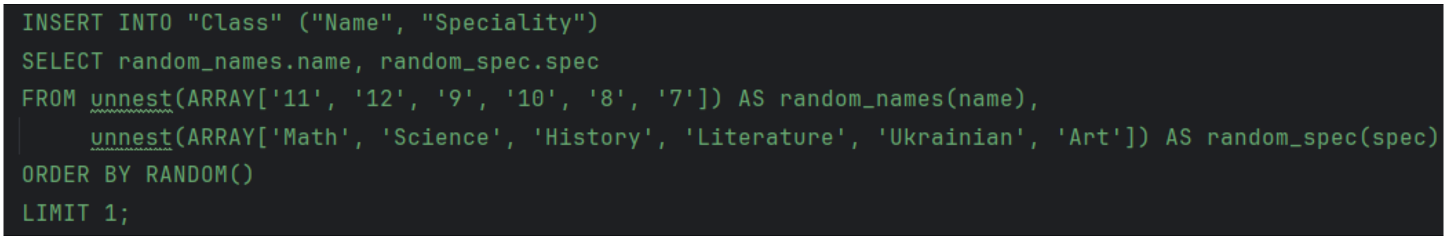


```
Classes:
ID: 1, Name: 11, Speciality: Math
ID: 2, Name: 9, Speciality: Art
ID: 3, Name: 7, Speciality: Ukrainian
ID: 4, Name: 12, Speciality: Literature
ID: 5, Name: 12, Speciality: History
ID: 6, Name: 9, Speciality: Art
ID: 7, Name: 12, Speciality: Ukrainian
```

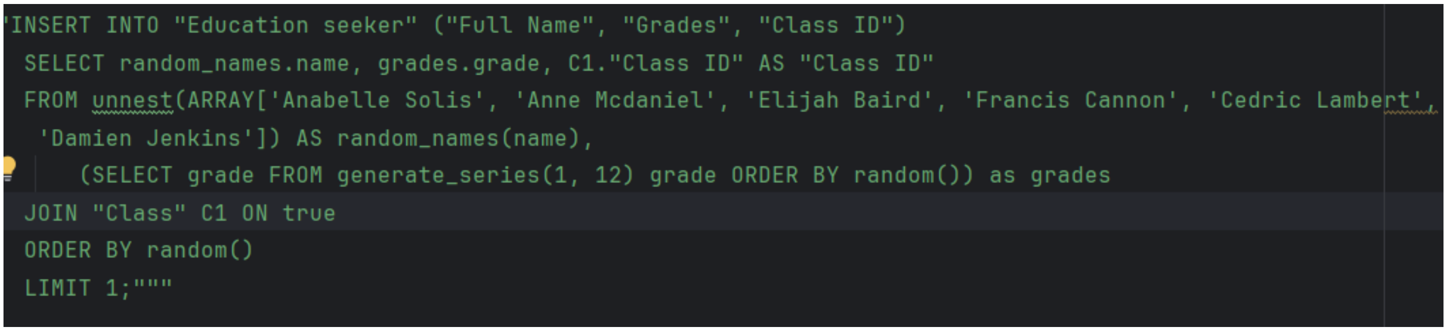


```
ID: 92, Name: 9, Speciality: Science
ID: 93, Name: 9, Speciality: History
ID: 94, Name: 10, Speciality: Science
ID: 95, Name: 11, Speciality: History
ID: 96, Name: 10, Speciality: Literature
ID: 97, Name: 9, Speciality: Art
ID: 98, Name: 7, Speciality: Ukrainian
ID: 99, Name: 8, Speciality: History
ID: 100, Name: 10, Speciality: Ukrainian
```

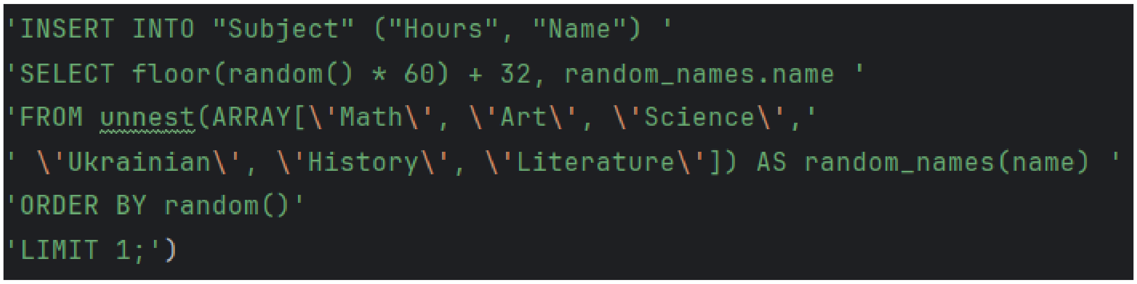
Копії SQL-запитів для генерації записів таблиць:



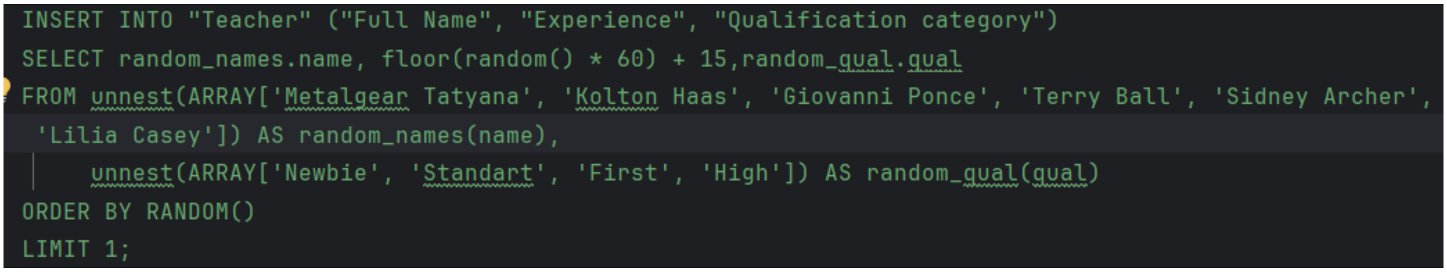
```sql
INSERT INTO "Class" ("Name", "Speciality")
SELECT random_names.name, random_spec.spec
FROM unnest(ARRAY['11', '12', '9', '10', '8', '7']) AS random_names(name),
     unnest(ARRAY['Math', 'Science', 'History', 'Literature', 'Ukrainian', 'Art']) AS random_spec(spec)
ORDER BY RANDOM()
LIMIT 1;
```



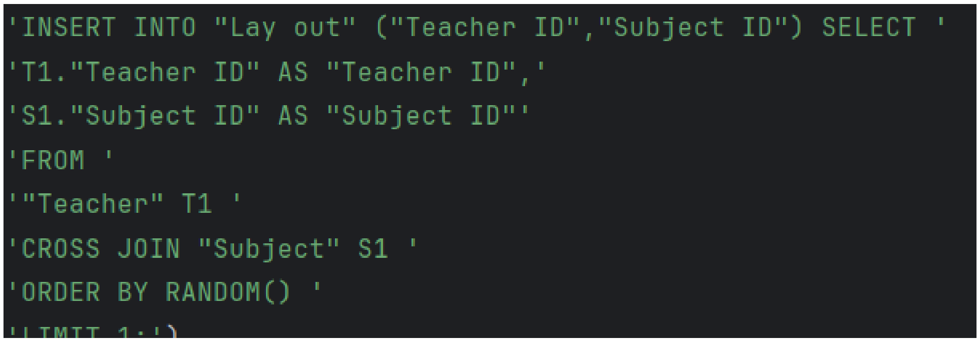
```sql
'INSERT INTO "Education seeker" ("Full Name", "Grades", "Class ID")
 SELECT random_names.name, grades.grade, C1."Class ID" AS "Class ID"
 FROM unnest(ARRAY['Anabelle Solis', 'Anne Mcdaniel', 'Elijah Baird', 'Francis Cannon', 'Cedric Lambert',
  'Damien Jenkins']) AS random_names(name),
     (SELECT grade FROM generate_series(1, 12) grade ORDER BY random()) as grades
 JOIN "Class" C1 ON true
 ORDER BY random()
 LIMIT 1;"""
```



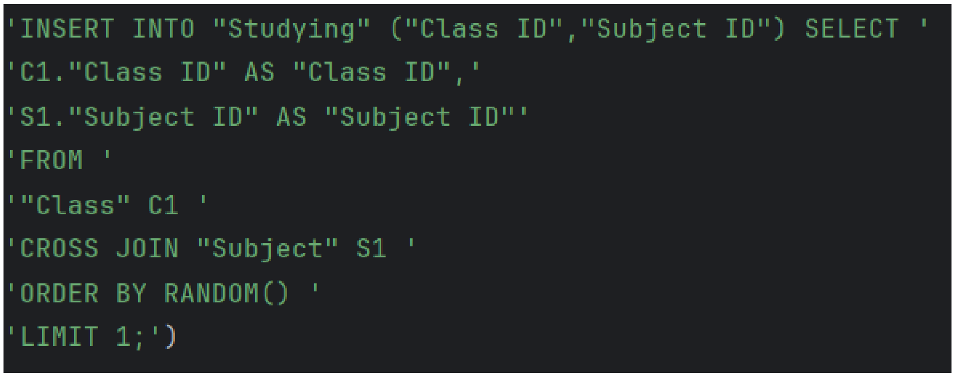
```sql
'INSERT INTO "Subject" ("Hours", "Name") '
'SELECT floor(random() * 60) + 32, random_names.name '
'FROM unnest(ARRAY[\'Math\', \'Art\', \'Science\',' 
' \'Ukrainian\', \'History\', \'Literature\']) AS random_names(name) '
'ORDER BY random()'
'LIMIT 1;')
```



```sql
INSERT INTO "Teacher" ("Full Name", "Experience", "Qualification category")
SELECT random_names.name, floor(random() * 60) + 15,random_qual.qual
FROM unnest(ARRAY['Metalgear Tatyana', 'Kolton Haas', 'Giovanni Ponce', 'Terry Ball', 'Sidney Archer',
 'Lilia Casey']) AS random_names(name),
     unnest(ARRAY['Newbie', 'Standart', 'First', 'High']) AS random_qual(qual)
ORDER BY RANDOM()
LIMIT 1;
```

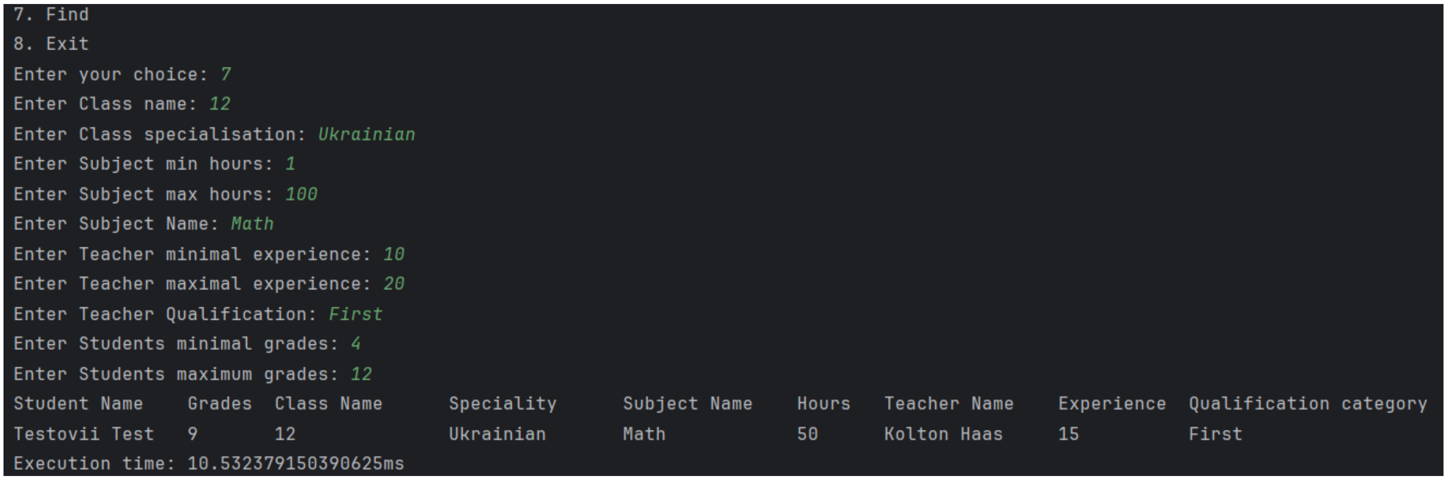


```sql
'INSERT INTO "Lay out" ("Teacher ID","Subject ID") SELECT '
'T1."Teacher ID" AS "Teacher ID",'
'S1."Subject ID" AS "Subject ID"'
'FROM '
'"Teacher" T1 '
'CROSS JOIN "Subject" S1 '
'ORDER BY RANDOM() '
'LIMIT 1;')
```



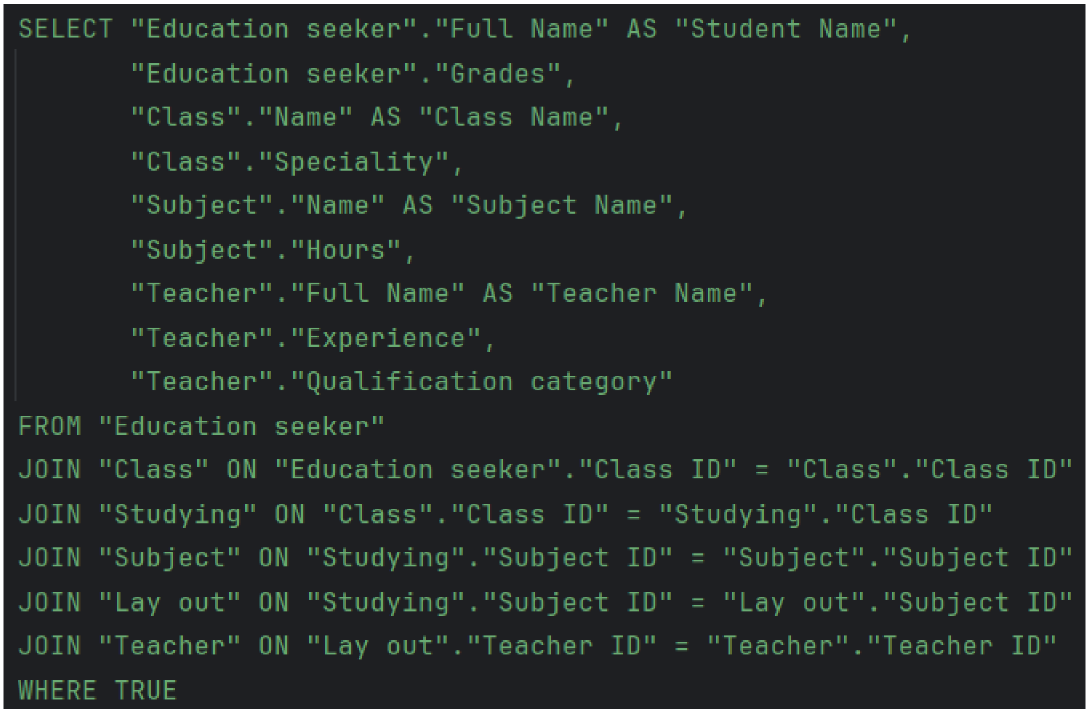
```
'INSERT INTO "Studying" ("Class ID","Subject ID") SELECT '
'C1."Class ID" AS "Class ID",'
'S1."Subject ID" AS "Subject ID"'
'FROM '
'"Class" C1 '
'CROSS JOIN "Subject" S1 '
'ORDER BY RANDOM() '
'LIMIT 1;')
```

Пункт №3

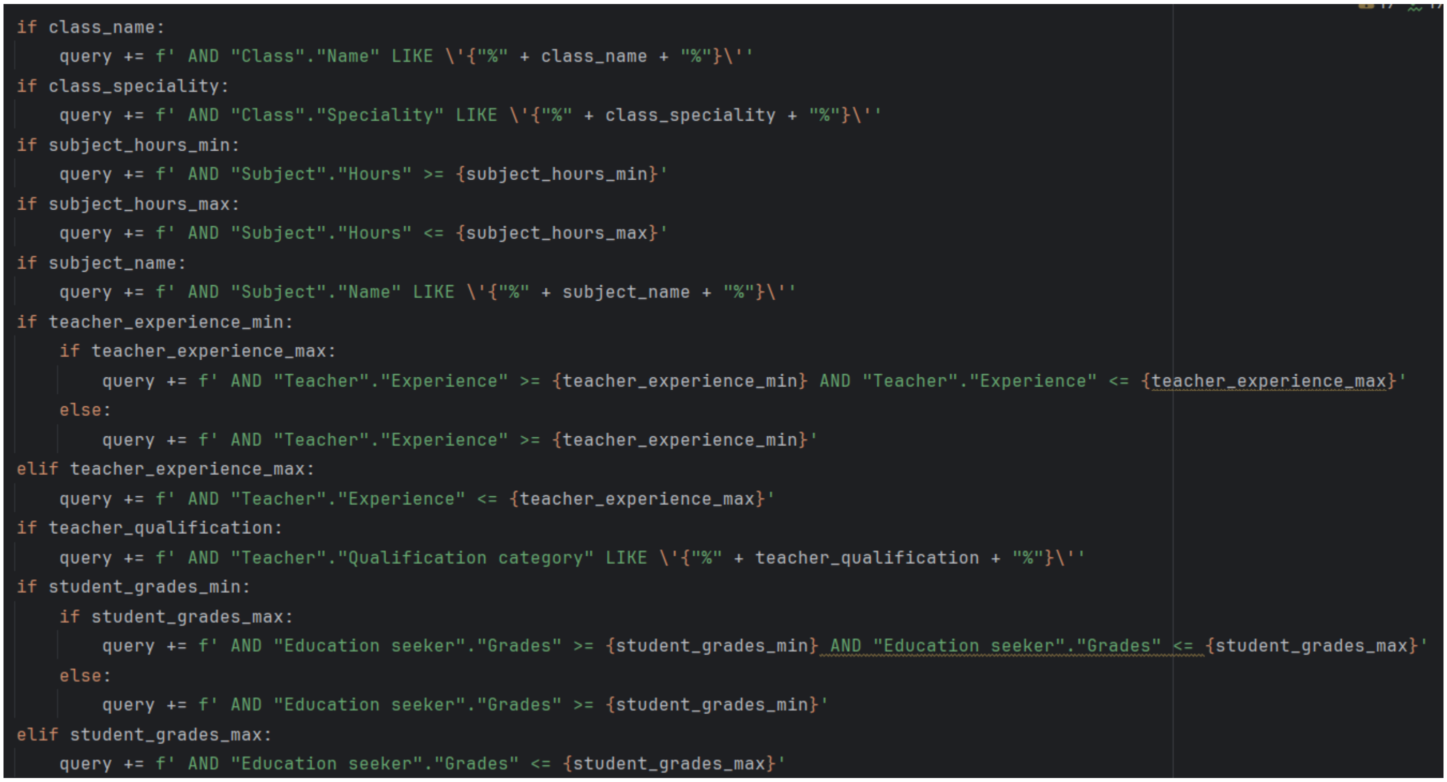


```
7. Find
8. Exit
Enter your choice: 7
Enter Class name: 12
Enter Class specialisation: Ukrainian
Enter Subject min hours: 1
Enter Subject max hours: 100
Enter Subject Name: Math
Enter Teacher minimal experience: 10
Enter Teacher maximal experience: 20
Enter Teacher Qualification: First
Enter Students minimal grades: 4
Enter Students maximum grades: 12
Student Name    Grades  Class Name      Speciality      Subject Name    Hours   Teacher Name    Experience  Qualification category
Testovii Test   9       12              Ukrainian       Math            50      Kolton Haas     15          First
Execution time: 10.532379150390625ms
```

Копія SQL-запиту для пошуку:



```
SELECT "Education seeker"."Full Name" AS "Student Name",
       "Education seeker"."Grades",
       "Class"."Name" AS "Class Name",
       "Class"."Speciality",
       "Subject"."Name" AS "Subject Name",
       "Subject"."Hours",
       "Teacher"."Full Name" AS "Teacher Name",
       "Teacher"."Experience",
       "Teacher"."Qualification category"
FROM "Education seeker"
JOIN "Class" ON "Education seeker"."Class ID" = "Class"."Class ID"
JOIN "Studying" ON "Class"."Class ID" = "Studying"."Class ID"
JOIN "Subject" ON "Studying"."Subject ID" = "Subject"."Subject ID"
JOIN "Lay out" ON "Studying"."Subject ID" = "Lay out"."Subject ID"
JOIN "Teacher" ON "Lay out"."Teacher ID" = "Teacher"."Teacher ID"
WHERE TRUE
```



```
if class_name:
    query += f' AND "Class"."Name" LIKE \'{"%" + class_name + "%"}\''
if class_speciality:
    query += f' AND "Class"."Speciality" LIKE \'{"%" + class_speciality + "%"}\''
if subject_hours_min:
    query += f' AND "Subject"."Hours" >= {subject_hours_min}'
if subject_hours_max:
    query += f' AND "Subject"."Hours" <= {subject_hours_max}'
if subject_name:
    query += f' AND "Subject"."Name" LIKE \'{"%" + subject_name + "%"}\''
if teacher_experience_min:
    if teacher_experience_max:
        query += f' AND "Teacher"."Experience" >= {teacher_experience_min} AND "Teacher"."Experience" <= {teacher_experience_max}'
    else:
        query += f' AND "Teacher"."Experience" >= {teacher_experience_min}'
elif teacher_experience_max:
    query += f' AND "Teacher"."Experience" <= {teacher_experience_max}'
if teacher_qualification:
    query += f' AND "Teacher"."Qualification category" LIKE \'{"%" + teacher_qualification + "%"}\''
if student_grades_min:
    if student_grades_max:
        query += f' AND "Education seeker"."Grades" >= {student_grades_min} AND "Education seeker"."Grades" <= {student_grades_max}'
    else:
        query += f' AND "Education seeker"."Grades" >= {student_grades_min}'
elif student_grades_max:
    query += f' AND "Education seeker"."Grades" <= {student_grades_max}'
```

Пункт №4

Програмний код модуля model:

```
import psycopg2
import psycopg2.extensions
import time


class Model:
    def __init__(self):
        self.conn = psycopg2.connect(
            dbname='postgres',
            user='postgres',
            password='admin',
            host='localhost',
            port=5434
        )
        self.create_table_class()
        self.create_table_education_seeker()
        self.create_table_subject()
        self.create_table_teacher()
        self.create_table_laying()
        self.create_table_studying()

    def create_table_class(self):
        c = self.conn.cursor()
        c.execute('''
            CREATE TABLE IF NOT EXISTS "Class" (
                "Class ID" SERIAL PRIMARY KEY,
                "Name" CHARACTER VARYING NOT NULL,
                "Speciality" CHARACTER VARYING NOT NULL
            )
        ''')

        # Check if the table exists
        c.execute("SELECT EXISTS (SELECT 1 FROM information_schema.tables WHERE
table_name = 'Class')")
        table_exists = c.fetchone()[0]

        if not table_exists:
            # Table does not exist, so create it
            c.execute('''
                CREATE TABLE IF NOT EXISTS "Class" (
                    "Class ID" SERIAL PRIMARY KEY,
                    "Name" CHARACTER VARYING NOT NULL,
                    "Speciality" CHARACTER VARYING NOT NULL
                )
            ''')

        self.conn.commit()

    def add_class(self, name, spec):
        c = self.conn.cursor()
        c.execute('INSERT INTO "Class" ("Name", "Speciality") VALUES (%s, %s)', (name,
spec))
        self.conn.commit()

    def get_all_class(self):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Class"')
        return c.fetchall()

    def update_class(self, class_id, name, spec):
        c = self.conn.cursor()
        c.execute(f'SELECT EXISTS(SELECT "Class ID" FROM "Class" WHERE "Class ID" =
{class_id})')
        id_exist = c.fetchone()[0]
        if id_exist:
            c.execute('UPDATE "Class" SET "Name"=%s, "Speciality"=%s WHERE "Class
ID"=%s', (name, spec, class_id))
            self.conn.commit()
            return 1
        else:
            return 0

    def delete_class(self, class_id):
        c = self.conn.cursor()
        c.execute(f'SELECT EXISTS(SELECT "Class ID" FROM "Education seeker" WHERE "Class
ID" = {class_id})')
        student_exist = c.fetchone()[0]
        c.execute(f'SELECT EXISTS(SELECT "Class ID" FROM "Class" WHERE "Class ID" =
{class_id})')
        id_exist = c.fetchone()[0]
        c.execute(f'SELECT EXISTS(SELECT "Class ID" FROM "Studying" WHERE "Class ID" =
{class_id})')
        studying_id = c.fetchone()[0]
        if id_exist:
            if student_exist:
                return "Class is not deleted.Delete all student from it first"
            if studying_id:
                return "Clas is not deleted. Delete it from Studying table first"
            c.execute('DELETE FROM "Class" WHERE "Class ID"=%s', (class_id,))
            self.conn.commit()

            c.execute('SELECT MAX("Class ID") FROM "Class"')
            max_id = c.fetchone()[0]

            if max_id is None:
                c.execute('ALTER SEQUENCE "Class_Class ID_seq" RESTART WITH %s', (1,))
                self.conn.commit()
            return "Class deleted successfuly"
        else:
            return "There is no such class ID"

    def generate_class(self, count):
        c = self.conn.cursor()

        for i in range(count):
            c.execute('''
                INSERT INTO "Class" ("Name", "Speciality")
                SELECT random_names.name, random_spec.spec
                FROM unnest(ARRAY['11', '12', '9', '10', '8', '7']) AS
random_names(name),
                     unnest(ARRAY['Math', 'Science', 'History', 'Literature',
'Ukrainian', 'Art']) AS random_spec(spec)
                ORDER BY RANDOM()
                LIMIT 1;
            ''')

        self.conn.commit()

    def find_class(self, name, spec):
        c = self.conn.cursor()
        c.execute("""SELECT * FROM "Class" WHERE "Name" LIKE %s AND "Speciality" LIKE %s
""", (name, spec))
        return c.fetchall()

    def create_table_education_seeker(self):
        c = self.conn.cursor()
        c.execute('''
            CREATE TABLE IF NOT EXISTS "Education seeker" (
                "Student ID" SERIAL PRIMARY KEY,
                "Full Name" CHARACTER VARYING NOT NULL,
                "Grades" INTEGER NOT NULL,
                "Class ID" INTEGER NOT NULL
            )
        ''')

        c.execute("SELECT EXISTS (SELECT 1 FROM information_schema.tables WHERE
table_name = 'Education seeker')")
        table_exists = c.fetchone()[0]

        if not table_exists:
            # Table does not exist, so create it
            c.execute('''
                CREATE TABLE IF NOT EXISTS "Education seeker" (
                    "Student ID" SERIAL PRIMARY KEY,
                    "Full Name" CHARACTER VARYING NOT NULL,
                    "Grades" INTEGER NOT NULL,
                    "Class ID" INTEGER NOT NULL
                )
            ''')

        self.conn.commit()

    def add_education_seeker(self, name, grades, class_id):
        c = self.conn.cursor()
        c.execute(f'SELECT EXISTS(SELECT "Class ID" FROM "Class" WHERE "Class ID" =
{class_id})')
        id_exist = c.fetchone()[0]
        if id_exist:
            c.execute('INSERT INTO "Education seeker" ("Full Name", "Grades", "Class ID")
VALUES (%s, %s, %s)', (name, grades, class_id))
            self.conn.commit()
            return 1
        return 0

    def get_all_education_seekers(self):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Education seeker"')
        return c.fetchall()

    def update_education_seeker(self, student_id, name, grades, class_id):
        c = self.conn.cursor()
        c.execute(f'SELECT EXISTS(SELECT "Class ID" FROM "Class" WHERE "Class ID" =
{class_id})')
        class_exist = c.fetchone()[0]
        c.execute(f'SELECT EXISTS(SELECT "Student ID" FROM "Education seeker" WHERE
"Student ID" = {student_id})')
        id_exist = c.fetchone()[0]
        if id_exist and class_exist:
            c.execute('UPDATE "Education seeker" SET "Full Name"=%s, "Grades"=%s, "Class
ID"=%s WHERE "Student ID"=%s', (name, grades, class_id, student_id))
            self.conn.commit()
            return 1
        else:
            return 0

    def delete_education_seeker(self, student_id):
        c = self.conn.cursor()
        c.execute(f'SELECT EXISTS(SELECT "Student ID" FROM "Education seeker" WHERE
"Student ID" = {student_id})')
        id_exist = c.fetchone()[0]
        if id_exist:
            c.execute('DELETE FROM "Education seeker" WHERE "Student ID"=%s',
(student_id,))
            self.conn.commit()

            c.execute('SELECT MAX("Student ID") FROM "Education seeker"')
            max_id = c.fetchone()[0]

            if max_id is None:
                c.execute('ALTER SEQUENCE "Student_Education seeker ID_seq" RESTART WITH
%s', (1,))
                self.conn.commit()

            return 1
        else:
            return 0

    def generate_education_seeker(self, count):
        c = self.conn.cursor()

        for i in range(count):
            c.execute(
                """INSERT INTO "Education seeker" ("Full Name", "Grades", "Class ID")
                    SELECT random_names.name, grades.grade, C1."Class ID" AS "Class ID"
                    FROM unnest(ARRAY['Anabelle Solis', 'Anne Mcdaniel', 'Elijah Baird',
'Francis Cannon', 'Cedric Lambert', 'Damien Jenkins']) AS random_names(name),
                        (SELECT grade FROM generate_series(1, 12) grade ORDER BY
random()) as grades
                    JOIN "Class" C1 ON true
                    ORDER BY random()
                    LIMIT 1;"""
            )
        self.conn.commit()

    def find_education_seeker(self, grades_min, grades_max, class_id):
        c = self.conn.cursor()
        c.execute("""SELECT * FROM "Education seeker" WHERE "Grades" BETWEEN %s AND %s
AND "Class ID" = %s""", (grades_min, grades_max, class_id))
        return c.fetchall()

    def create_table_subject(self):
        c = self.conn.cursor()
        c.execute('''
            CREATE TABLE IF NOT EXISTS "Subject" (
                "Subject ID" SERIAL PRIMARY KEY,
                "Hours" INTEGER NOT NULL,
                "Name" CHARACTER VARYING NOT NULL
            )
        ''')

        c.execute("SELECT EXISTS (SELECT 1 FROM information_schema.tables WHERE
table_name = 'Subject')")
        table_exists = c.fetchone()[0]

        if not table_exists:
            c.execute('''
                CREATE TABLE IF NOT EXISTS "Subject" (
                    "Subject ID" SERIAL PRIMARY KEY,
                    "Hours" INTEGER NOT NULL,
                    "Name" CHARACTER VARYING NOT NULL
                )
            ''')

        self.conn.commit()

    def add_subject(self, hours, name):
        c = self.conn.cursor()
        c.execute('INSERT INTO "Subject" ("Hours", "Name") VALUES (%s, %s)', (hours,
name))
        self.conn.commit()

    def get_all_subject(self):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Subject"')
        return c.fetchall()

    def update_subject(self, subject_id, hours, name):
        c = self.conn.cursor()
        c.execute(f'SELECT EXISTS(SELECT "Subject ID" FROM "Subject" WHERE "Subject ID" =
{subject_id})')
        id_exist = c.fetchone()[0]
        if id_exist:
            c.execute('UPDATE "Subject" SET "Hours"=%s, "Name"=%s WHERE "Subject ID"=%s',
(hours, name, subject_id))
            self.conn.commit()
            return 1
        else:
            return 0

    def delete_subject(self, subject_id):
        c = self.conn.cursor()
        c.execute(f'SELECT EXISTS(SELECT "Subject ID" FROM "Subject" WHERE "Subject ID" =
{subject_id})')
        id_exist = c.fetchone()[0]
        c.execute(f'SELECT EXISTS(SELECT "Subject ID" FROM "Lay out" WHERE "Subject ID" =
{subject_id})')
        layout_id = c.fetchone()[0]
        c.execute(f'SELECT EXISTS(SELECT "Subject ID" FROM "Studying" WHERE "Subject ID"
= {subject_id})')
        studying_id = c.fetchone()[0]
        if id_exist:
            if layout_id:
                return "Subject is not deleted. Delete this subject from Lay out table
first"
            if studying_id:
                return "Subject is not deleted. Delete this subject from Studying table
first"
            c.execute('DELETE FROM "Subject" WHERE "Subject ID"=%s', (subject_id,))
            self.conn.commit()

            c.execute('SELECT MAX("Subject ID") FROM "Subject"')
            max_id = c.fetchone()[0]

            if max_id is None:
                c.execute('ALTER SEQUENCE "Subject_Subject ID_seq" RESTART WITH %s',
(1,))
                self.conn.commit()

            return "Subject deleted"
        else:
            return "Subject is not deleted there is no such ID"

    def find_subject(self, hours_min, hours_max, name):
        c = self.conn.cursor()
        c.execute("""SELECT * FROM "Subject" WHERE "Hours" BETWEEN %s AND %s AND "Name"
LIKE %s""", (hours_min, hours_max, name))
        return c.fetchall()

    def generate_subject(self, count):
        c = self.conn.cursor()

        for i in range(count):
            c.execute('INSERT INTO "Subject" ("Hours", "Name") '
                      'SELECT floor(random() * 60) + 32, random_names.name '
                      'FROM unnest(ARRAY[\'Math\', \'Art\', \'Science\',' 
                      ' \'Ukrainian\', \'History\', \'Literature\']) AS
random_names(name) '
                      'ORDER BY random()'
                      'LIMIT 1;')

        self.conn.commit()

    def create_table_teacher(self):
        c = self.conn.cursor()
        c.execute('''
            CREATE TABLE IF NOT EXISTS "Teacher" (
                "Teacher ID" SERIAL PRIMARY KEY,
                "Full Name" CHARACTER VARYING NOT NULL,
                "Experience" INTEGER NOT NULL,
                "Qualification category" CHARACTER VARYING NOT NULL
            )
        ''')

        c.execute("SELECT EXISTS (SELECT 1 FROM information_schema.tables WHERE
table_name = 'Teacher')")
        table_exists = c.fetchone()[0]
        if not table_exists:
            c.execute('''
                CREATE TABLE IF NOT EXISTS "Teacher" (
                    "Teacher ID" SERIAL PRIMARY KEY,
                    "Full Name" CHARACTER VARYING NOT NULL,
                    "Experience" INTEGER NOT NULL,
                    "Qualification category" CHARACTER VARYING NOT NULL
                )
            ''')

        self.conn.commit()

    def add_teacher(self, name, exp, qual):
        c = self.conn.cursor()
        c.execute('INSERT INTO "Teacher" ("Full Name", "Experience", "Qualification
category") VALUES (%s, %s, %s)', (name, exp, qual))
        self.conn.commit()

    def get_all_teacher(self):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Teacher"')
        return c.fetchall()

    def update_teacher(self, teacher_id, name, exp, qual):
        c = self.conn.cursor()
        c.execute(f'SELECT EXISTS(SELECT "Teacher ID" FROM "Teacher" WHERE "Teacher ID" =
{teacher_id})')
        id_exist = c.fetchone()[0]
        if id_exist:
            c.execute('UPDATE "Teacher" SET "Full Name"=%s, "Experience"=%s,
"Qualification category"=%s  WHERE "Teacher ID"=%s', (name, exp, qual, teacher_id))
            self.conn.commit()
            return 1
        else:
            return 0

    def delete_teacher(self, teacher_id):
        c = self.conn.cursor()
        c.execute(f'SELECT EXISTS(SELECT "Teacher ID" FROM "Teacher" WHERE "Teacher ID" =
{teacher_id})')
        id_exist = c.fetchone()[0]
        c.execute(f'SELECT EXISTS(SELECT "Teacher ID" FROM "Lay out" WHERE "Teacher ID" =
{teacher_id})')
        layout_id = c.fetchone()[0]
        if id_exist:
            if layout_id:
                return "Teacher is not deleted. delete it from Lay out table first"
            c.execute('DELETE FROM "Teacher" WHERE "Teacher ID"=%s', (teacher_id,))
            self.conn.commit()

            c.execute('SELECT MAX("Teacher ID") FROM "Teacher"')
            max_id = c.fetchone()[0]

            if max_id is None:
                c.execute('ALTER SEQUENCE "Teacher_Teacher ID_seq" RESTART WITH %s',
(1,))
                self.conn.commit()

            return "Teacher deleted"
        else:
            return "Teacher is not deleted.There is no such id"

    def find_teacher(self, exp_min, exp_max, qual, hours_min, hours_max, subject_name):
        c = self.conn.cursor()
        c.execute("""SELECT * FROM "Teacher" WHERE "Experience" BETWEEN %s AND %s AND
"Qualification category" LIKE %s""", (exp_min, exp_max, qual))
        return c.fetchall()

    def generate_teacher(self, count):
        c = self.conn.cursor()

        for i in range(count):
            c.execute('''
                INSERT INTO "Teacher" ("Full Name", "Experience", "Qualification
category")
                SELECT random_names.name, floor(random() * 60) + 15,random_qual.qual
                FROM unnest(ARRAY['Metalgear Tatyana', 'Kolton Haas', 'Giovanni Ponce',
'Terry Ball', 'Sidney Archer', 'Lilia Casey']) AS random_names(name),
                     unnest(ARRAY['Newbie', 'Standart', 'First', 'High']) AS
random_qual(qual)
                ORDER BY RANDOM()
                LIMIT 1;
            ''')
        self.conn.commit()

    def create_table_laying(self):
        c = self.conn.cursor()
        c.execute('''
            CREATE TABLE IF NOT EXISTS "Lay out" (
                "Lay out ID" SERIAL PRIMARY KEY,
                "Teacher ID" INTEGER NOT NULL,
                "Subject ID" INTEGER NOT NULL
            )
        ''')

        c.execute("SELECT EXISTS (SELECT 1 FROM information_schema.tables WHERE
table_name = 'Lay out')")
        table_exists = c.fetchone()[0]

        if not table_exists:
            c.execute('''
                CREATE TABLE IF NOT EXISTS "Lay out" (
                    "Lay out ID" SERIAL PRIMARY KEY,
                    "Teacher ID" INTEGER NOT NULL,
                    "Subject ID" INTEGER NOT NULL
                )
            ''')

        self.conn.commit()

    def add_laying(self, teach_id, subject_id):
        c = self.conn.cursor()
        c.execute(f'SELECT EXISTS(SELECT "Teacher ID" FROM "Teacher" WHERE "Teacher ID" =
{teach_id})')
        teach_exist = c.fetchone()[0]
        c.execute(f'SELECT EXISTS(SELECT "Subject ID" FROM "Subject" WHERE "Subject ID" =
{subject_id})')
        subject_exist = c.fetchone()[0]
        if teach_exist and subject_exist:
            c.execute('INSERT INTO "Lay out" ("Teacher ID","Subject ID") VALUES (%s,%s)',
                      (teach_id, subject_id))
            self.conn.commit()
            return 1
        else:
            return 0
    def get_all_laying(self):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Lay out"')
        return c.fetchall()

    def update_laying(self, laying_id, teach_id, subject_id):
        c = self.conn.cursor()
        c.execute(f'SELECT EXISTS(SELECT "Teacher ID" FROM "Teacher" WHERE "Teacher ID" =
{teach_id})')
        teach_exist = c.fetchone()[0]
        c.execute(f'SELECT EXISTS(SELECT "Subject ID" FROM "Subject" WHERE "Subject ID" =
{subject_id})')
        subject_exist = c.fetchone()[0]
        c.execute(f'SELECT EXISTS(SELECT "Lay out ID" FROM "Lay out" WHERE "Lay out ID" =
{laying_id})')
        id_exist = c.fetchone()[0]
        if id_exist and teach_exist and subject_exist:
            c.execute(
                'UPDATE "Lay out" SET "Teacher ID"=%s, "Subject ID"=%s WHERE "Lay out
ID"=%s',
                (teach_id, subject_id, laying_id))
            self.conn.commit()
            return 1
        else:
            return 0

    def delete_laying(self, laying_id):
        c = self.conn.cursor()
        c.execute(f'SELECT EXISTS(SELECT "Lay out ID" FROM "Lay out" WHERE "Lay out ID" =
{laying_id})')
        id_exist = c.fetchone()[0]
        if id_exist:
            c.execute('DELETE FROM "Lay out" WHERE "Lay out ID"=%s', (laying_id,))
            self.conn.commit()

            c.execute('SELECT MAX("Lay out ID") FROM "Lay out"')
            max_id = c.fetchone()[0]

            if max_id is None:
                c.execute('ALTER SEQUENCE "Laying_Lay out ID_seq" RESTART WITH %s', (1,))
                self.conn.commit()

            return 1
        else:
            return 0

    def generate_laying(self, count):
        c = self.conn.cursor()

        for i in range(count):
            c.execute('INSERT INTO "Lay out" ("Teacher ID","Subject ID") SELECT '
                      'T1."Teacher ID" AS "Teacher ID",'
                      'S1."Subject ID" AS "Subject ID"'
                      'FROM '
                      '"Teacher" T1 '
                      'CROSS JOIN "Subject" S1 '
                      'ORDER BY RANDOM() '
                      'LIMIT 1;')

        self.conn.commit()

    def create_table_studying(self):
        c = self.conn.cursor()
        c.execute('''
            CREATE TABLE IF NOT EXISTS "Studying" (
                "Studying ID" SERIAL PRIMARY KEY,
                "Class ID" INTEGER NOT NULL,
                "Subject ID" INTEGER NOT NULL
            )
        ''')

        c.execute("SELECT EXISTS (SELECT 1 FROM information_schema.tables WHERE
table_name = 'Studying')")
        table_exists = c.fetchone()[0]

        if not table_exists:
            c.execute('''
                CREATE TABLE IF NOT EXISTS "Studying" (
                    "Studying ID" SERIAL PRIMARY KEY,
                    "Class ID" INTEGER NOT NULL,
                    "Subject ID" INTEGER NOT NULL
                )
            ''')

        self.conn.commit()

    def add_studying(self, class_id, subject_id):
        c = self.conn.cursor()
        c.execute(f'SELECT EXISTS(SELECT "Class ID" FROM "Class" WHERE "Class ID" =
{class_id})')
        class_exist = c.fetchone()[0]
        c.execute(f'SELECT EXISTS(SELECT "Subject ID" FROM "Subject" WHERE "Subject ID" =
{subject_id})')
        subject_exist = c.fetchone()[0]
        if class_exist and subject_exist:
            c.execute('INSERT INTO "Studying" ("Class ID","Subject ID") VALUES (%s,%s)',
                      (class_id, subject_id))
            self.conn.commit()
            return 1
        else:
            return 0

    def get_all_studying(self):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Studying"')
        return c.fetchall()

    def update_studying(self, studying_id, class_id, subject_id):
        c = self.conn.cursor()
        c.execute(f'SELECT EXISTS(SELECT "Class ID" FROM "Class" WHERE "Class ID" =
{class_id})')
        class_exist = c.fetchone()[0]
        c.execute(f'SELECT EXISTS(SELECT "Subject ID" FROM "Subject" WHERE "Subject ID" =
{subject_id})')
        subject_exist = c.fetchone()[0]
        c.execute(f'SELECT EXISTS(SELECT "Studying ID" FROM "Studying" WHERE "Studying
ID" = {studying_id})')
        id_exist = c.fetchone()[0]
        if id_exist and class_exist and subject_exist:
            c.execute(
                'UPDATE "Studying" SET "Class ID"=%s, "Subject ID"=%s WHERE "Studying
ID"=%s',
                (class_id, subject_id, studying_id))
            self.conn.commit()
            return 1
        else:
            return 0

    def delete_studying(self, studying_id):
        c = self.conn.cursor()
        c.execute(f'SELECT EXISTS(SELECT "Studying ID" FROM "Studying" WHERE "Studying
ID" = {studying_id})')
        id_exist = c.fetchone()[0]
        if id_exist:
            c.execute('DELETE FROM "Studying" WHERE "Studying ID"=%s', (studying_id,))
            self.conn.commit()

            c.execute('SELECT MAX("Studying ID") FROM "Studying"')
            max_id = c.fetchone()[0]

            if max_id is None:
                c.execute('ALTER SEQUENCE "Studying_Studying ID_seq" RESTART WITH %s',
(1,))
                self.conn.commit()

            return 1
        else:
            return 0

    def generate_studying(self, count):
        c = self.conn.cursor()

        for i in range(count):
            c.execute('INSERT INTO "Studying" ("Class ID","Subject ID") SELECT '
                      'C1."Class ID" AS "Class ID",'
                      'S1."Subject ID" AS "Subject ID"'
                      'FROM '
                      '"Class" C1 '
                      'CROSS JOIN "Subject" S1 '
                      'ORDER BY RANDOM() '
                      'LIMIT 1;')

        self.conn.commit()

    def find(self, class_name, class_speciality, subject_hours_min, subject_hours_max,
subject_name,
             teacher_experience_min, teacher_experience_max, teacher_qualification,
student_grades_min,
             student_grades_max):
            query = ('''
                     SELECT "Education seeker"."Full Name" AS "Student Name",
                            "Education seeker"."Grades",
                            "Class"."Name" AS "Class Name",
                            "Class"."Speciality",
                            "Subject"."Name" AS "Subject Name",
                            "Subject"."Hours",
                            "Teacher"."Full Name" AS "Teacher Name",
                            "Teacher"."Experience",
                            "Teacher"."Qualification category"
                     FROM "Education seeker"
                     JOIN "Class" ON "Education seeker"."Class ID" = "Class"."Class ID"
                     JOIN "Studying" ON "Class"."Class ID" = "Studying"."Class ID"
                     JOIN "Subject" ON "Studying"."Subject ID" = "Subject"."Subject ID"
                     JOIN "Lay out" ON "Studying"."Subject ID" = "Lay out"."Subject ID"
                     JOIN "Teacher" ON "Lay out"."Teacher ID" = "Teacher"."Teacher ID"
                     WHERE TRUE
                     ''')

            if class_name:
                query += f' AND "Class"."Name" LIKE \'{"%" + class_name + "%"}\''
            if class_speciality:
                query += f' AND "Class"."Speciality" LIKE \'{"%" + class_speciality +
"%"}\''
            if subject_hours_min:
                query += f' AND "Subject"."Hours" >= {subject_hours_min}'
            if subject_hours_max:
                query += f' AND "Subject"."Hours" <= {subject_hours_max}'
            if subject_name:
                query += f' AND "Subject"."Name" LIKE \'{"%" + subject_name + "%"}\''
            if teacher_experience_min:
                if teacher_experience_max:
                    query += f' AND "Teacher"."Experience" >= {teacher_experience_min}
AND "Teacher"."Experience" <= {teacher_experience_max}'
                else:
                    query += f' AND "Teacher"."Experience" >= {teacher_experience_min}'
            elif teacher_experience_max:
                query += f' AND "Teacher"."Experience" <= {teacher_experience_max}'
            if teacher_qualification:
                query += f' AND "Teacher"."Qualification category" LIKE \'{"%" +
teacher_qualification + "%"}\''
            if student_grades_min:
                if student_grades_max:
                    query += f' AND "Education seeker"."Grades" >= {student_grades_min}
AND "Education seeker"."Grades" <= {student_grades_max}'
                else:
                    query += f' AND "Education seeker"."Grades" >= {student_grades_min}'
            elif student_grades_max:
                query += f' AND "Education seeker"."Grades" <= {student_grades_max}'

            c = self.conn.cursor()
            start_time = time.time()
            c.execute(query)
            end_time = time.time()
            return (c.fetchall(), end_time - start_time)

    def find_teaching_classes(self, class_name,  subject_name, teacher_name):
        c = self.conn.cursor()
        c.execute('''
            SELECT l."Subject Name", c."Class Name", t."Teacher Name"
            FROM "Lay out" l
            JOIN "Class" c ON l."Class ID" = c."Class ID"
            JOIN "Teacher" t ON l."Teacher ID" = t."Teacher ID"
            WHERE l."Subject Name" = %s AND c."Class Name" = %s AND t."Teacher Name" = %s
            GROUP BY l."Subject Name", c."Class Name", t."Teacher Name";
        ''', (subject_name, class_name, teacher_name))

        rows = c.fetchall()
        for row in rows:
            print(row)
```

Опис функцій

__init__ – функція ініціалізації об’єкту, в ній відбувається підключення до бази даних та створюються таблиці

create_tables – функція створення таблиць бази даних, якщо вони не існують

add_<table_name> – додати запис до відповідної таблиці get_all_<table_name> – отримати всі записи з таблиці

update_<table_name> – змінити запис таблиці

delete_<table_name> – видалити запис з таблиці

generate_<table_name> – згенерувати задану кількість записів таблиці

find – пошук записів за параметрами