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

Факультет прикладної математики
Кафедра програмного забезпечення комп’ютерних систем

Лабораторна робота № 3

з дисципліни “Бази даних”

тема “Засоби оптимізації роботи СУБД PostgreSQL”

Виконала

студентка II курсу

групи КП-01

Левицька Юлія Володимирівна

Варіант №12

Перевірив

“____” “_____” 20___р.

Радченко Костянтин
Олександрович

Київ 2021

Мета роботи

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

Постановка задачі

Завдання роботи полягає у наступному:

1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM).

2. Створити та проаналізувати різні типи індексів у PostgreSQL.

3. Розробити тригер бази даних PostgreSQL.

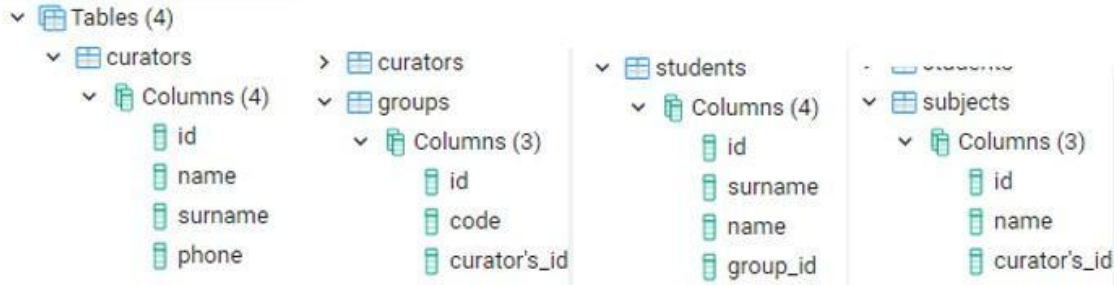
Варіант №12

Види індексів: BTree, GIN

Умови для тригера: before update, delete

Копії екрану, що підтверджують виконання вимог:

1. **Таблиці бази даних:**



Класи ORM:

```
class Curator(Base):
    __tablename__ = 'curators'
    id = Column(Integer, primary_key=True, unique=True, nullable=False)
    name = Column(String(50))
    surname = Column(String(50))
    phone = Column(String(50))
    group = relationship('Group', uselist=False)
    subjects = relationship('Subject')
    __table_args__ = {'extend_existing': True}

class Subject(Base):
    __tablename__ = 'subjects'
    id = Column(Integer, primary_key=True, unique=True, nullable=False)
    name = Column(String(50))
    curator_id = Column(Integer, ForeignKey('curators.id'))
    curator = relationship("Curator", foreign_keys=[curator_id, id, name])
    __table_args__ = {'extend_existing': True}

class Group(Base):
    __tablename__ = 'groups'
    id = Column(Integer, primary_key=True, unique=True, nullable=False)
    code = Column(String(50))
    curator_id = Column(Integer, ForeignKey('curators.id'))
    curator = relationship("Curator", foreign_keys=[curator_id, id, code])
    students = relationship("Students")
    __table_args__ = {'extend_existing': True}

class Student(Base):
    __tablename__ = 'curators'
    id = Column(Integer, primary_key=True, unique=True, nullable=False)
    name = Column(String(50))
    surname = Column(String(50))
    group_id = Column(Integer, ForeignKey('groups.id'))
    group = relationship("Group", foreign_keys=[group_id])
    __table_args__ = {'extend_existing': True}
```

Приклади команд:

```
def create(self, id, name, surname, phone):
    if (id < 1):
        print('Error with input!')
        return
    try:
        session = Session()
        session.add(Curator(
            id = id,
            name = name,
            surname = surname,
            phone = phone
        ))
        session.commit()
        print("Entity inserted")
```

```
def update(self, id, name, surname, phone):
    if (id < 1):
        print('Error with input!')
        return
    try:
        i = session.query(Curator).get(id)
        i.name = name
        i.surname = surname
        i.phone = phone
        session.add(i)
        session.commit()
        print("Entity updated")
```

Запит програми до користувача:

```
----------Welcome----------
Enter command (create, delete, update, random, readCuratorGroup, readCuratorSubject, readStudentCurator or exit): create
Enter table name: subjects
Enter a number: 55
Enter a value: Group dynamics
Enter a number: 7
```

Приклад результату:

34	55	Group dynamics	7

2. Приклади індексування:

Команди створення індексів:

- BTree:

```
connection = psycopg2.connect(user="postgres",
                              password="1",
                              host="127.0.0.1",
                              port="5432",
                              database="university")
cursor = connection.cursor()
selecr_query = """CREATE INDEX ON curators USING BTREE(id);"""
cursor.execute(selecr_query)
connection.commit()
print("Index created")
```

B-tree – це збалансоване дерево пошуку з кількома гілками, тому воно оптимальне, якщо використовується для пошуку.

- GIN:

```
connection = psycopg2.connect(user="postgres",
                              password="1",
                              host="127.0.0.1",
                              port="5432",
                              database="university")
cursor = connection.cursor()
selecr_query = """CREATE INDEX group_code ON groups USING gin (to_tsvector('english', code));"""
cursor.execute(selecr_query)
connection.commit()
print("Index created")
```

GIN розшифровується як Generalized Inverted Index – це так званий зворотний індекс. Він працює з типами даних, значення яких не є атомарними, а складаються з елементів. У ньому індексуються не самі значення, а окремі елементи; кожен елемент посилається ті значення, у яких зустрічається. Також оптимальне, якщо використовується для пошуку, а в інших випадках є неефективним.

Приклади запитів і результатів:

1.

```
selecr_query = """SELECT curators.name, curators.surname, groups.code from curators, groups
WHERE curators.id = 7 AND groups.curator_id = 7;"""
Result [('Ruslan', 'Hadynyak', 'KP-02')]
Time for operation 0.03505269999999994
selecr_query = """SELECT curators.name, curators.surname, groups.code from curators, groups
WHERE curators.id = groups.curator_id AND groups.code LIKE 'KM%';"""
Result [('Ivan', 'Melnyk', 'KM-03'), ('Anna', 'Sulema', 'KM-13'), ('Pylyp', 'Pylypenko', 'KM-14'), ('Tkh', 'Vss', 'KM-92')]
Time for operation -0.004729900000000065
```

3.

```
selecr_query = """SELECT curators.id, curators.name, curators.surname, groups.code from curators, groups
WHERE curators.id = groups.curator_id ORDER BY curators.id;"""
```

```
, 'Zabolotnya', 'AI=02'), (4, 'Tatiana', 'Zabolotnya', 'SR-34'), (4, 'Tatiana', 'Zabolotnya', 'PL-44'), (5, 'Vadym', 'Ivanenko', 'JI-2
2'), (5, 'Vadym', 'Ivanenko', 'KB-92'), (6, 'Pylyp', 'Pylypenko', 'YV-36'), (6, 'Pylyp', 'Pylypenko', 'BD-39'), (6, 'Pylyp', 'Pylypenk
o', 'KM-14'), (7, 'Ruslan', 'Hadynyak', 'KP-02'), (8, 'Ivan', 'Melnyk', 'KM-03'), (10, 'Hrq', 'Kqe', 'PP-50'), (12, 'Yan', 'Kulyk', 'V
D-51'), (13, 'Eyn', 'Dwn', 'TT-37'), (13, 'Eyn', 'Dwn', 'XY-84'), (14, 'Nazar', 'Salyha', 'HU-64'), (17, 'Ban', 'Brb', 'HR-50'), (17,
'Ban', 'Brb', 'KY-69'), (21, 'Qot', 'Lkg', 'KI-69'), (22, 'Pnk', 'Jdr', 'NX-34'), (26, 'Akh', 'Uen', 'FJ-15'), (27, 'Wyf', 'Iub', 'PD-
42'), (32, 'Tkh', 'Vss', 'KM-92'), (51, 'Kbw', 'Joo', 'IA-13')]
Time for operation -0.007487099999999858
```

4.

```
selecr_query = """SELECT groups.code from curators, groups
WHERE curators.id = groups.curator_id AND curators.id > 20 GROUP BY groups.code;"""
```

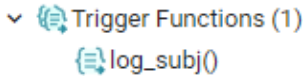
```
Result [('FJ-15',), ('IA-13',), ('KI-69',), ('KM-92',), ('NX-34',), ('PD-42',)]
Time for operation 0.004166500000000073
```

3. Тригери:

Команди створення тригера:

```
query = """DROP TABLE IF EXISTS subj_logs;
            CREATE TABLE subj_logs(id integer NOT NULL, old_name text, new_name text, curator_id integer);
            CREATE OR REPLACE FUNCTION log_subj() RETURNS trigger AS $BODY$
            BEGIN
                IF NEW.name IS NULL THEN
                    RAISE EXCEPTION 'Name cannot be null';
                END IF;
                IF NEW.curator_id IS NULL THEN
                    RAISE EXCEPTION 'Subject cannot have null curator id';
                END IF;
                INSERT INTO subj_logs VALUES(OLD.id, OLD.name, NEW.name, NEW.curator_id);
                RETURN NEW;
            END;
        $BODY$ LANGUAGE plpgsql;
        DROP TRIGGER IF EXISTS log_subj ON subjects;
        CREATE TRIGGER log_subj BEFORE UPDATE OR DELETE ON subjects
            FOR EACH ROW EXECUTE PROCEDURE log_subj();"""
cursor.execute(query)
```

Тригер додано (скріншот з PgAdmin4):



Приклади роботи:

1. Редагування:

```
----------Welcome----------
Enter command (create, delete, update or exit): update
Enter table name: subjects
Enter a number: 2
Enter a value: Programming
Enter a number: 9
1 Entity updated
```

```
def update(self, id, name, c_id):
    if (id < 1):
        print('Error with input!')
        return
    try:
        i = session.query(Subject).get(id)
        i.name = name,
        i.curator_id = c_id
        session.add(i)
        session.commit()
        print("Entity updated")
    except (Exception, Error) as error:
        print("Error with PostgreSQL", error)
    finally:
        print()
```

	id integer	old_name text	new_name text	curator_id integer
1	2	Programming	Proga	9

2. Вилучення:

```
----------Welcome----------
Enter command (create, delete, update or exit): delete
Enter table name: subjects
Enter a number: 2
1 Entity deleted
```

```
def delete(self, id):
    if (id < 1):
        print('Error with input!')
        return
    try:
        i = session.query(Subject).get(id)
        session.delete(i)
        session.commit()
        print("Entity deleted")
    except (Exception, Error) as error:
        print("Error with PostgreSQL", error)
    finally:
        print()
```

	id integer	old_name text	new_name text	curator_id integer
1	2	Programming	Proga	9

Контрольні запитання:

1. Сформулювати призначення та задачі об’єктно-реляційної проекції (ORM).

Призначенням є пов'язати бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних». Необхідно забезпечити роботу з даними в термінах класів, а не таблиць даних, і, навпаки, перетворити терміни та дані класів на дані, придатні для зберігання в СУБД. Необхідно також забезпечити інтерфейс для CRUD- операцій над даними. Загалом, необхідно позбутися необхідності писати SQL-код для взаємодії в СУБД.

2. Проаналізувати основні види індексів у PostgreSQL (BTree, BRIN, GIN, Hash): призначення, сфера застосування, переваги та недоліки.

Незважаючи на всі відмінності між типами індексів (названими також методами доступу), зрештою будь-який з них встановлює відповідність між ключем (наприклад, значенням проіндексованого стовпця) та рядками таблиці, в яких цей ключ зустрічається. У PostgreSQL використовуються такі основні види індексів: BTree, BRIN, GIN, Hash. Рядки ідентифікуються за допомогою TID (tuple id), який складається з номера блоку файлу та позиції рядка всередині блоку. Тоді, знаючи ключ або деяку інформацію про нього, можна швидко прочитати ті рядки, в яких може знаходитися інформація, що цікавить нас, не переглядаючи всю таблицю повністю. Важливо розуміти, що індекс, прискорюючи доступ до даних, натомість потребує певних витрат на свою підтримку.

3. Пояснити призначення тригерів та функцій у базах даних.

Тригер запускається сервером автоматично при спробі зміни даних в таблиці, з якою він зв'язаний. Функції такого типу дозволяють реалізувати складнішу логіку.

Висновки:

Під час виконання лабораторної роботи були здобуті практичні навички використання засобів оптимізації СУБД PostgreSQL.