Лабораторна робота №3

По дисципліні “Бази даних”

Засоби оптимізації роботи СУБД PostgreSQL

Студента

Групи КП-02

Литвиненка Артема Сергійовича

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

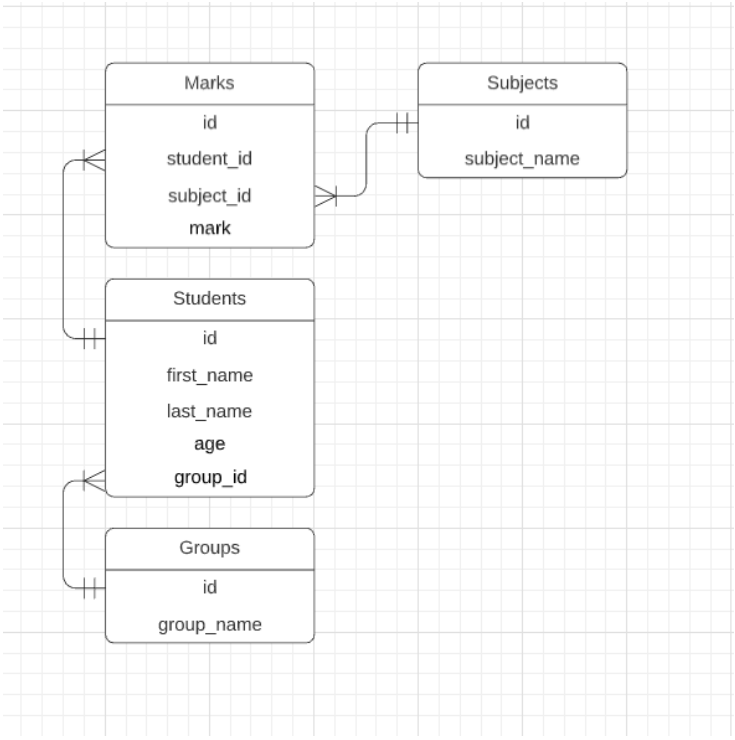
Завдання роботи полягає у наступному:

1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи No2 у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.

№ варіанта	Види індексів	Умови для тригера
8	BTree, Hash	after insert, update

Завдання

1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи No2 у вигляд об’єктно-реляційної проекції (ORM).



models.py

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String, ForeignKey

Base = declarative_base()


class Student(Base):
    __tablename__ = "students"
    student_id = Column(Integer, primary_key=True)
    first_name = Column(String)
    last_name = Column(String)
    age = Column(Integer)
    group_id = Column(String, ForeignKey("groups.group_id"))

    def __repr__(self):
        return "<Student(first_name='{}', last_name='{}', age={},
group_id={})>"\
            .format(self.first_name, self.last_name, self.age,
self.group_id)


class Group(Base):
    __tablename__ = "groups"
    group_id = Column(Integer, primary_key=True)
```

```
    title = Column(String)

    def __repr__(self):
        return "<Group(title='{}')>"\
            .format(self.title)


class Subject(Base):
    __tablename__ = "subjects"
    subject_id = Column(Integer, primary_key=True)
    title = Column(String)

    def __repr__(self):
        return "<Subject(title='{}')>"\
            .format(self.title)


class Mark(Base):
    __tablename__ = "marks"
    mark_id = Column(Integer, primary_key=True)
    student_id = Column(Integer, ForeignKey("students.student_id"))
    subject_id = Column(Integer, ForeignKey("subjects.subject_id"))
    mark = Column(Integer)

    def __repr__(self):
        return "<Mark(student_id={}, subject_id={}, mark={})>"\
            .format(self.student_id, self.subject_id, self.mark)
```

controller.py

```
from sqlalchemy.orm import sessionmaker
from sqlalchemy import create_engine
from ast import literal_eval

from config import DATABASE_URI

engine = create_engine(DATABASE_URI)
Session = sessionmaker(bind=engine)


class Controller():

    def __init__(self, model, view, autocommit=True):
        self.model = model
        self.view = view
        self.session = Session(autocommit=autocommit)

    def show_items(self, bullet_points=False):
        items = self.session.query(self.model).all()
        item_name = self.model.__tablename__
        if bullet_points:
```

```
            self.view.show_bullet_point_list(item_name, items)
        else:
            self.view.show_number_point_list(item_name, items)

    def show_item(self, item_id: int):
        item_name = self.model.__tablename__
        try:
            item = self.session.query(self.model).get(item_id)
            self.view.show_item(item, item_name)
        except Exception as _ex:
            self.view.display_missing_item_error(item_name, item_id,
_ex)

    def show_filtered_items(self, attrs, bullet_points=False):
        items =
self.session.query(self.model).filter_by(**literal_eval(attrs))
        item_name = self.model.__tablename__
        if bullet_points:
            self.view.show_bullet_point_list(item_name, items)
        else:
            self.view.show_number_point_list(item_name, items)

    def insert_item(self, item_data):
        item_data = literal_eval(item_data)
        item = self.model(**item_data)
        item_name = self.model.__tablename__
        try:
            self.session.add(item)
            self.view.display_item_insertion(item_name)
        except Exception as _ex:
            self.view.display_insert_item_error(item, _ex)

    def update_item(self, item_data):
        item_data = literal_eval(item_data)
        item_id_dict = {}
        id_column_name = list(item_data.keys())[0]
        item_id_dict[id_column_name] = item_data[id_column_name]
        item =
self.session.query(self.model).filter_by(**item_id_dict).first()
        item_type = self.model.__tablename__
        is_item_found = bool(item)
        try:
            if not is_item_found:
                raise Exception('Item not found exception')

            for key, value in item_data.items():
                setattr(item, key, value)

            self.view.display_item_updated(
                item_type, item_id_dict[id_column_name])
        except Exception as _ex:
```

```
            self.view.display_missing_item_error(
                item_type, item_id_dict[id_column_name], _ex)

    def delete_item(self, item_id: int):
        item_type = self.model.__tablename__
        id_column_name = self.model.__table__.columns.keys()[0]
        item = self.session.query(self.model).filter_by(
            **{id_column_name: item_id}
        )
        try:
            if not item:
                raise Exception('Item not found exception')

            item.delete()
            self.view.display_item_deletion(item_type, item_id)
        except Exception as _ex:
            self.view.display_missing_item_error(item_type, item_id,
_ex)

    def __del__(self):
        self.session.close()
```

```
Enter a command ('help' for all commands): show_item 2
Item from students found
[INFO] <Student(first_name='Abra', last_name='Cadabra', age=18, group_id=2)>
Enter a command ('help' for all commands):
```

2. Створити та проаналізувати різні типи індексів у PostgreSQL.

```
CREATE INDEX students_group_id_idx
    ON public.students USING btree
    (group_id ASC NULLS LAST)
    TABLESPACE pg_default;
```

```
CREATE INDEX students_age_idx
    ON public.students USING hash
    (age)
    TABLESPACE pg_default;
```

При порівнянні результатів було виявлено, що фільтрація без індексування виконувалася 0,19мс., у той час коли при використанні індексів ця операція зайняла 0,9мс., що приблизно у два рази швидше.

Індекси прискорюють швидкість виконання запитів оскільки для різних типів індексів використовуються різні структури даних. Ту, яку структуру даних треба вибирати залежить вид виду даних. Hash – знаходження даних по ключу, без ітерації; btree – рекурсивний пошук даних, тощо.

3. Розробити тригер бази даних PostgreSQL.

Процедура яку викликає тригер при умові

```
CREATE FUNCTION public.update_on_last_insert()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
BEGIN
    INSERT INTO last_inserts(last_insert_student) VALUES (NOW());
    RETURN NEW;
END;
$BODY$;

ALTER FUNCTION public.update_on_last_insert()
    OWNER TO postgres;
```

Сам тригер

```
CREATE TRIGGER on_last_insert_trigger
    AFTER INSERT
    ON public.students
    FOR EACH ROW
    EXECUTE FUNCTION public.update_on_last_insert();
```

Приклад

```
INSERT INTO students (first_name, last_name, age, group_id)
VALUES ('some', 'guy', 21, 2);
```

	last_insert_id [PK] bigint	last_insert_student date
1	1	2021-12-08
2	2	2021-12-08

Контрольні запитання

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

ORM - технологія, яка зв'язує БД з концепціями мов ООП, створюючи «віртуальну об'єктну базу даних»

Суть проблеми полягає в перетворенні таких об'єктів у форму, в якій вони можуть бути збережені у файлах або базах даних, і які легко можуть бути витягнуті в подальшому, зі збереженням властивостей об'єктів і відношень між ними.

ORM позбавляє програміста від написання великої кількості коду, часто одноманітного і схильного до помилок, тим самим значно підвищуючи швидкість розробки. Крім того, більшість сучасних реалізацій ORM дозволяє програмістові при необхідності жорстко задати код SQL-запитів, який використовуватиметься при тих чи інших діях (збереження в базу даних, завантаження, пошук тощо) з постійним об'єктом.

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

Індекс btree, він же B-дерево, придатний для даних, які можна відсортувати. Іншими словами, для типу даних повинні бути визначені оператори «більше», «більше або одно», «менше», «менше або одно» та «рівно».
B-дерева мають кілька важливих властивостей: Вони збалансовані, тобто будь-яку листову сторінку відокремлює від кореня те саме число внутрішніх сторінок.

Ідея BRIN не в тому, щоб швидко знайти потрібні рядки, а в тому, щоб уникнути перегляду непотрібних. Це завжди неточний індекс: він взагалі не містить TID-ів табличних рядків. Спрощено кажучи, BRIN добре працює для тих стовпців, значення яких корелюють з їх фізичним розташуванням у таблиці.

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 цьому індексуються не самі значення, а окремі елементи; кожен елемент посилається ті значення, у яких зустрічається.

Ідея хешування у тому, щоб значенню будь-якого типу даних зіставити деяке невелике число (від 0 до N−1, всього N значень). Таке зіставлення називають хеш-функцією.

Всі перераховані види індексів слугують для оптимізації алгоритмів пошуку, сортування, фільтрації, тощо. Переваги та недоліки кожного описуються в залежності від типу даних. Жоден з них не є гарним або поганим – якщо треба забити гвоздика – молоток; закрутити гайку – викрутка.

3. Пояснити призначення тригерів та функцій у базах даних.

Тригер – це процедура особливого типу, яку користувач не викликає явно, а використання якої обумовлено настанням визначеної події (дії) у реляційній БД:

- додаванням INSERT,
- вилученням рядка в заданій таблиці DELETE,
- або зміною даних у певному стовпці заданої таблиці UPDATE.

Тригери застосовуються для забезпечення цілісності даних і реалізації складної бізнес-логіки. Тригер запускається сервером автоматично при спробі зміни даних у таблиці, з якою він пов'язаний. Всі здійснені ним модифікації даних розглядаються як виконані в транзакції, в якій виконано дію, що викликало спрацьовування тригера. Відповідно, у разі виявлення помилки або порушення цілісності даних може відбутися відкат цієї транзакції.