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

Факультет прикладної математики
Кафедра системного програмування і спеціалізованих комп’ютерних систем

Лабораторна робота № 3

з дисципліни «Бази даних і засоби управління»

«Засоби оптимізації роботи СУБД PostgreSQL»

Виконала: Денисенко Олександра
Студентка групи КВ-91
Перевірив: Павловський В.І.

Київ 2021

Лабораторна робота №3

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 №2 у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.
4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

Варіант 8

№ варіанта	*Види індексів*	*Умови для тригера*
8	*Btree, Gin*	*After insert, update*

Посилання на Github: https://github.com/OleksandraDenysenko/Data-Base.git

Завдання 1

Логічна модель предметної області «Школа»

Обрана предметна галузь передбачає обробку даних про учнів, їхні номери телефонів, вчителів, предмети та розклад школи.

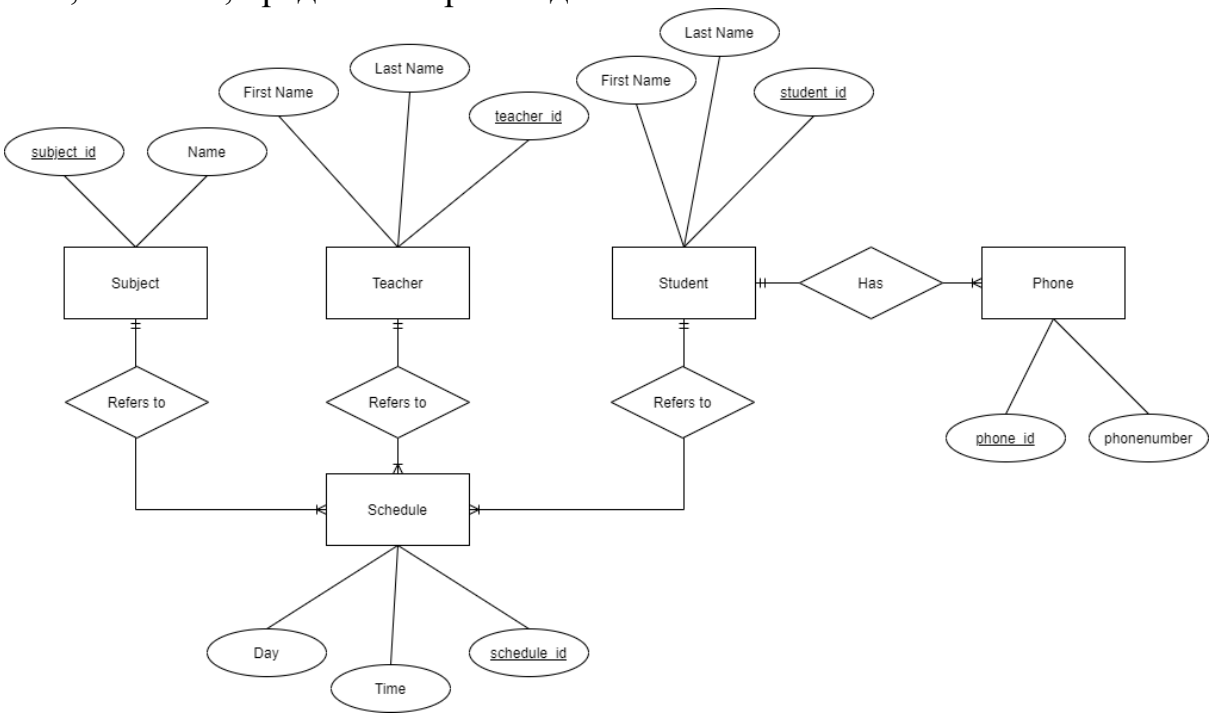


Рисунок 1. ER-діаграма, побудована за нотацією Чена

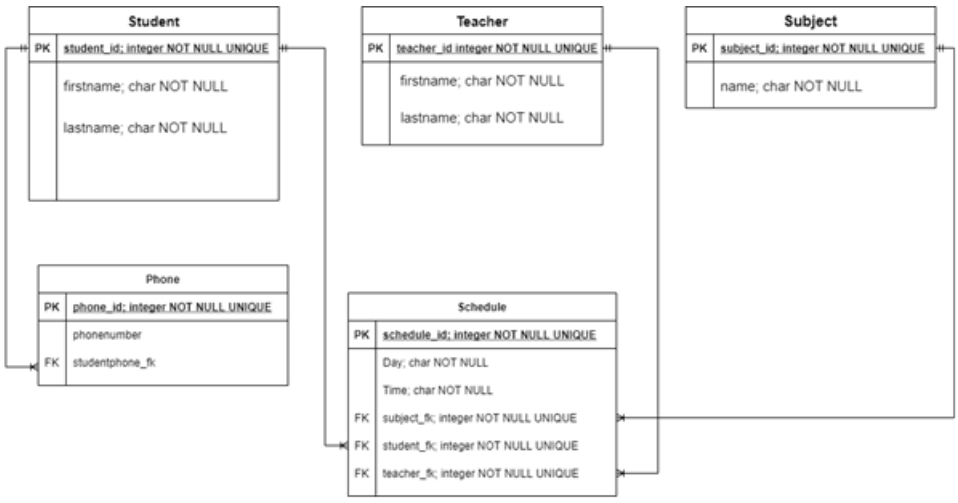


Рисунок 2 - ER-діаграма, переведена у таблиці БД

Зміни у порівнянні з першою лабораторною роботою відсутні.

Середовище розробки та налаштування підключення до бази даних

Для виконання лабораторної роботи використовувалась мова програмування Python та середовище розробки PyCharm 2021.2.1.

Для підключення до серверу бази даних PostgreSQL використано сторонню бібліотеку psycorg2, для реалізації моделі ORM використовувалася

стороння бібліотека SQLAlchemy, середовище для відлагодження SQL-запитів до бази даних – pgAdmin 4.

Класи ORM

У даній лабораторній роботі було реалізовано 5 класів відповідно до 5 існуючих таблиць: Subject, Teacher, Student, Phone, Schedule.

Таблиця Subject має стовпчики id (ідентифікатор) та name (ім'я), а також зв'язок 1:N із таблицею Schedule, тому в класі Subject встановлений зв'язок relationship('Schedule').

Таблиця Teacher має стовпчики id (ідентифікатор), firstname (ім'я), lastname (прізвище), а також зв'язок 1:N із таблицею Schedule, тому в класі Teacher встановлений зв'язок relationship('Schedule').

Таблиця Student має стовпчики id (ідентифікатор), firstname (ім'я), lastname (прізвище), а також зв'язок 1:N із таблицею Schedule, тому в класі Student встановлений зв'язок relationship('Schedule'). Ця таблиця має зв'язок 1:N з таблицею Phone, тому в класі Student встановлений зв'язок relationship('Phone').

Таблиця Phone має стовпчики id (ідентифікатор), phonenumber (номер телефону), studentphone_fk (зовнішній ключ, що пов'язує учня з його номером телефону), а також зв'язок 1:N із таблицею Student, тому в класі Phone встановлений зв'язок relationship('Student').

Таблиця Schedule має стовпчики id (ідентифікатор), day (день тижня), time (час), subject_fk (зовнішній ключ, що пов'язує предмет з розкладом), teacher_fk (зовнішній ключ, що пов'язує вчителя з розкладом), student_fk (зовнішній учня, що пов'язує предмет з розкладом). Ця таблиця має зв'язки 1:N з таблицями Student, Teacher, Subject, тому в класі Schedule встановлені зв'язки relationship('Subject'), relationship('Teacher'), relationship('Student').

Нижче наведена програмна реалізація класів ORM мовою Python (лістинги усіх модулей надані нижче):

```
class Subject(Base):
    __tablename__ = 'subject'
    subject_id = Column(Integer, nullable=False, primary_key=True)
    name = Column(String, nullable=False)
    schedule = relationship('Schedule')

    def __init__(self, subject_id, name):
        self.subject_id = subject_id
        self.name = name

    def __repr__(self):
        return "{:>10}{:>35}".format(self.subject_id, self.name)


class Student(Base):
    __tablename__ = 'student'
    student_id = Column(Integer, primary_key=True)
    firstname = Column(String, nullable=False)
    lastname = Column(String, nullable=False)
    phones = relationship('Phone')
    schedule = relationship('Schedule')

    def __init__(self, student_id, firstname, lastname):
        self.student_id = student_id
```

```
        self.firstname = firstname
        self.lastname = lastname

    def __repr__(self):
        return "{:>10}{:>35}{:>35}".format(self.student_id, self.firstname,
self.lastname)


class Teacher(Base):
    __tablename__ = 'teacher'
    teacher_id = Column(Integer, primary_key=True)
    firstname = Column(String, nullable=False)
    lastname = Column(String, nullable=False)
    schedule = relationship('Schedule')

    def __init__(self, teacher_id, firstname, lastname):
        self.teacher_id = teacher_id
        self.firstname = firstname
        self.lastname = lastname

    def __repr__(self):
        return "{:>10}{:>35}{:>35}".format(self.teacher_id, self.firstname,
self.lastname)


class Phone(Base):
    __tablename__ = 'phone'
    phone_id = Column(Integer, primary_key=True)
    phonenumber = Column(String, nullable=False)
    studentphone_fk = Column(Integer, ForeignKey('student.student_id'))

    def __init__(self, phone_id, phonenumber, studentphone_fk):
        self.phone_id = phone_id
        self.phonenumber = phonenumber
        self.studentphone_fk = studentphone_fk

    def __repr__(self):
        return "{:>10}{:>35}{:>10}".format(self.phone_id, self.phonenumber,
self.studentphone_fk)


class Schedule(Base):
    __tablename__ = 'schedule'
    schedule_id = Column(Integer, primary_key=True)
    day = Column(String, nullable=False)
    time = Column(String, nullable=False)
    subject_fk = Column(Integer, ForeignKey('subject.subject_id'))
    student_fk = Column(Integer, ForeignKey('student.student_id'))
    teacher_fk = Column(Integer, ForeignKey('teacher.teacher_id'))

    def __init__(self, schedule_id, day, time, subject_fk, student_fk,
teacher_fk):
        self.schedule_id = schedule_id
        self.day = day
        self.time = time
        self.subject_fk = subject_fk
        self.student_fk = student_fk
        self.teacher_fk = teacher_fk

    def __repr__(self):
        return "{:>10}{:>35}{:>35}{:>10}{:>10}{:>10}".format(self.schedule_id,
self.day, self.time, self.subject_fk,
```

```
                                                    self.student_fk,
self.teacher_fk)
```

Приклади запитів у вигляді ORM

Запит вставки

Цей запит реалізовано за допомогою функції insert. Спочатку у меню користувач обирає опцію вставки, далі обирає таблицю, до якої хоче додати запис і вводить необхідні дані. Є перевірка введених даних. У разі успішного додавання запису користувач бачить відповідне повідомлення. Реалізацію запиту вставки продемонструємо на прикладі таблиці subject.

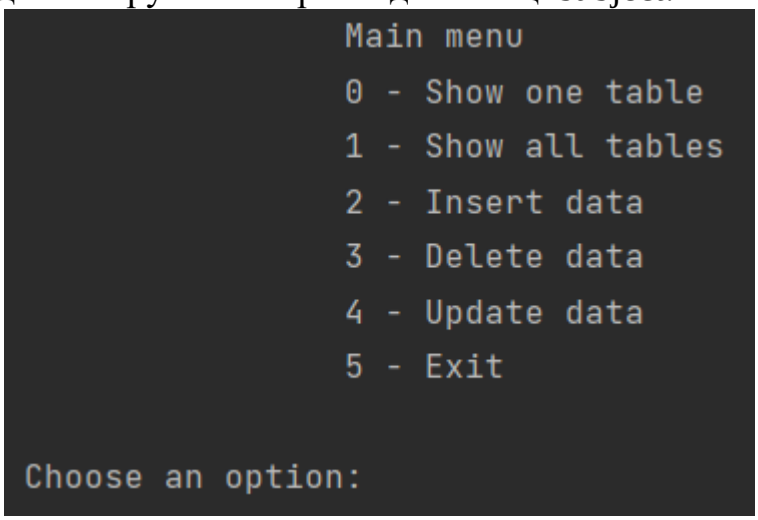


Рис.3 – Початкове меню для вибору опцій

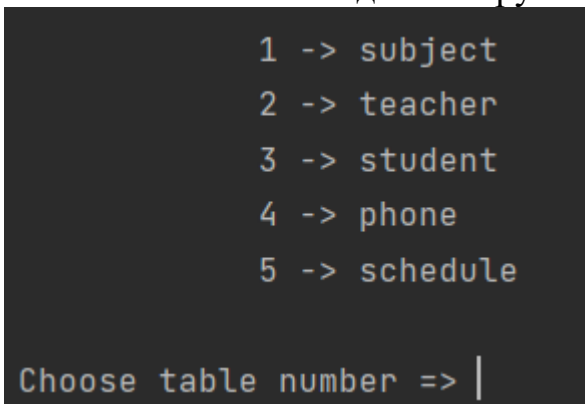


Рис.4 – Меню для вибору таблиці

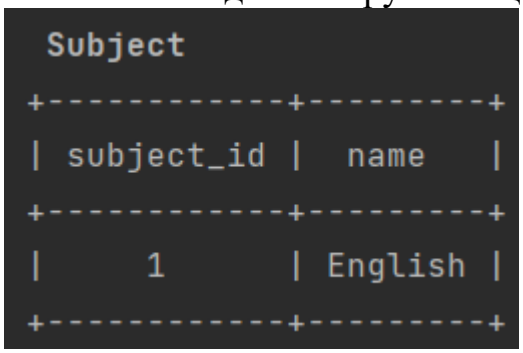


Рис.5 – Початковий стан таблиці Subject

```
Choose table number => 1
Enter subject_id: 2
Enter name: Maths
The row with 'subject_id' = '2' in table 'subject' was inserted successfully.
```

Рис.6 – Повідомлення про успішну вставку запису

Subject

subject_id	name
1	English
2	Maths

Рис.7 – Таблиця Subject після успішної вставки даних

Лістинг функції insert

```
def insert() -> None:
    Session = sessionmaker(bind=engine)
    session = Session()
    go_on = True
    while go_on:
        View.list()
        table = controller.validtable()
        if table == 1:
            id = controller.validate_input_items("subject_id")
            name = controller.validate_input_items("name")
            s = Subject(subject_id=id, name=name)
            session.add(s)
            View.complete_message("subject_id", id, "subject", "inserted")
            go_on = False
        elif table == 2:
            id = controller.validate_input_items("teacher_id")
            firstname = controller.validate_input_items("firstname")
            lastname = controller.validate_input_items("lastname")
            s = Teacher(teacher_id=id, firstname=firstname, lastname=lastname)
            session.add(s)
            View.complete_message("teacher_id", id, "teacher", "inserted")
            go_on = False
        elif table == 3:
            id = controller.validate_input_items("student_id")
            firstname = controller.validate_input_items("firstname")
            lastname = controller.validate_input_items("lastname")
            s = Student(student_id=id, firstname=firstname, lastname=lastname)
            session.add(s)
            View.complete_message("student_id", id, "student", "inserted")
            go_on = False
        elif table == 4:
            id = controller.validate_input_items("phone_id")
            phonenumber = controller.validate_input_items("phonenumber")
            studentphone_fk = controller.validate_input_items("student_id")
            s = Phone(phone_id=id, phonenumber=phonenumber,
studentphone_fk=studentphone_fk)
            session.add(s)
            View.complete_message("phone_id", id, "phone", "inserted")
```

```
            go_on = False
        elif table == 5:
            id = controller.validate_input_items("schedule_id")
            day = controller.validate_input_items("day")
            time = controller.validate_input_items("time")
            subject_fk = controller.validate_input_items("subject_id")
            student_fk = controller.validate_input_items("student_id")
            teacher_fk = controller.validate_input_items("teacher_id")
            s = Schedule(schedule_id=id, day=day, time=time,
subject_fk=subject_fk, student_fk=student_fk,
                         teacher_fk=teacher_fk)
            session.add(s)
            View.complete_message("schedule_id", id, "schedule", "inserted")
            go_on = False
        session.commit()
```

Запит видалення

Цей запит реалізовано за допомогою функції delete. Спочатку користувач обирає таблицю, з якої потрібно видалити дані. Потім потрібно ввести номер ідентифікатора запису для видалення. Якщо такого ідентифікатора не існує, то користувач побачить повідомлення про помилку. У разі успішного видалення запису користувач побачить відповідне повідомлення.

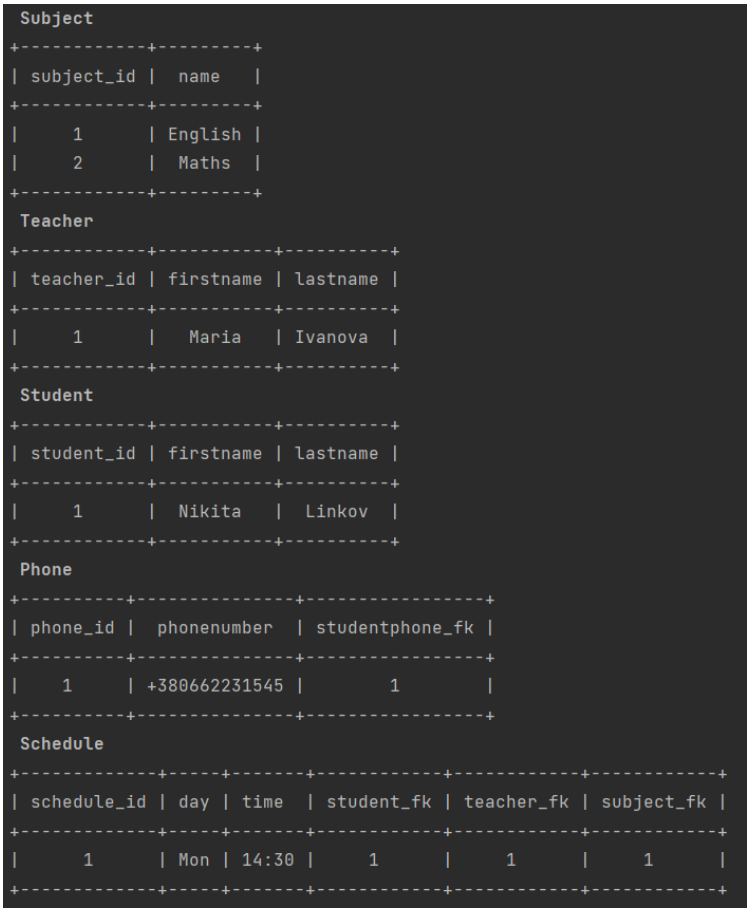
Особливу увагу потрібно приділити зв’язкам між таблицями. Наприклад, таблиці subject та schedule пов’язані зв’язком 1:N через зовнішній ключ subject_fk у таблиці schedule. Тому якщо користувач обирає запис про предмет, з яким є зв’язок у таблиці schedule, то відповідні записи видаляться в обох таблицях.

Реалізацію запиту видалення продемонструємо на таблиці student.

```
1 -> subject
2 -> teacher
3 -> student
4 -> phone
5 -> schedule

Choose table number =>
```

Рис.8 – Меню для вибору таблиці, з якої видалятиметься рядок



Subject

subject_id	name
1	English
2	Maths

Teacher

teacher_id	firstname	lastname
1	Maria	Ivanova

Student

student_id	firstname	lastname
1	Nikita	Linkov

Phone

phone_id	phonenumber	studentphone_fk
1	+380662231545	1

Schedule

schedule_id	day	time	student_fk	teacher_fk	subject_fk
1	Mon	14:30	1	1	1

Рис.9 – Початковий стан усіх таблиць

```
Choose table number => 1
Enter subject_id: 1
The row with 'subject_fk' = '1' in table 'schedule' was deleted successfully.
The row with 'subject_id' = '1' in table 'subject' was deleted successfully.
```

Рис.10 – Повідомлення про успішне видалення запису

```
 Subject
+------------+-------+
| subject_id | name  |
+------------+-------+
|     2      | Maths |
+------------+-------+
 Teacher
+------------+-----------+----------+
| teacher_id | firstname | lastname |
+------------+-----------+----------+
|     1      |   Maria   | Ivanova  |
+------------+-----------+----------+
 Student
+------------+-----------+----------+
| student_id | firstname | lastname |
+------------+-----------+----------+
|     1      |  Nikita   |  Linkov  |
+------------+-----------+----------+
 Phone
+----------+--------------+-----------------+
| phone_id |  phonenumber  | studentphone_fk |
+----------+--------------+-----------------+
|    1     | +380662231545 |        1        |
+----------+--------------+-----------------+
 Schedule
+-------------+-----+------+------------+------------+------------+
| schedule_id | day | time | student_fk | teacher_fk | subject_fk |
+-------------+-----+------+------------+------------+------------+
+-------------+-----+------+------------+------------+------------+
```

Рис.11 – Стан таблиць після видалення запису з subject та schedule

Лістинг функції delete

```
def delete():
    Session = sessionmaker(bind=engine)
    session = Session()
    go_on = True
    while go_on:
        View.list()
        table = controller.validtable()
        if table == 1:
            id = controller.validate_input_items("subject_id")
            records = session.query(Subject).get(id)
            if records is not None:
                records = session.query(Schedule).get(id)
                if records is not None:
                    delete = session.query(Schedule).filter(Schedule.subject_fk
== id)
                    for i in delete:
                        session.delete(i)
                    View.complete_message("subject_fk", id, "schedule",
"deleted")
                session.delete(session.query(Subject).filter(Subject.subject_id
== id).one())
                View.complete_message("subject_id", id, "subject", "deleted")
            else:
                controller.message("No ID found")
            go_on = False
```

```
        elif table == 2:
            id = controller.validate_input_items("teacher_id")
            records = session.query(Teacher).get(id)
            if records is not None:
                records = session.query(Schedule).get(id)
                if records is not None:
                    delete = session.query(Schedule).filter(Schedule.teacher_fk
== id)
                    for i in delete:
                        session.delete(i)
                    View.complete_message("teacher_fk", id, "schedule",
"deleted")
                session.delete(session.query(Teacher).filter(Teacher.teacher_id
== id).one())
                View.complete_message("teacher_id", id, "teacher", "deleted")
            else:
                controller.message("No ID found")
            go_on = False
        elif table == 3:
            id = controller.validate_input_items("student_id")
            records = session.query(Student).get(id)
            if records is not None:
                records = session.query(Phone).get(id)
                if records is not None:
                    delete =
session.query(Schedule).filter(Schedule.studentphone_fk == id)
                    for i in delete:
                        session.delete(i)
                    View.complete message("studentphone fk", id, "phone",
"deleted")
                records = session.query(Schedule).get(id)
                if records is not None:
                    delete = session.query(Schedule).filter(Schedule.student_fk
== id)
                    for i in delete:
                        session.delete(i)
                    View.complete_message("student_fk", id, "schedule",
"deleted")
                session.delete(session.query(Student).filter(Student.student_id
== id).one())
                View.complete_message("student_id", id, "student", "deleted")
            else:
                controller.message("No ID found")
            go_on = False
        elif table == 4:
            id = controller.validate_input_items("phone_id")
            records = session.query(Phone).get(id)
            if records is not None:
                session.delete(session.query(Phone).filter(Phone.phone_id ==
id).one())
                View.complete_message("phone_id", id, "phone", "deleted")
            else:
                controller.message("No ID found")
            go_on = False
        elif table == 5:
            id = controller.validate_input_items("schedule_id")
            records = session.query(Schedule).get(id)
            if records is not None:

session.delete(session.query(Schedule).filter(Schedule.schedule_id == id).one())
                View.complete_message("schedule_id", id, "schedule", "deleted")
            else:
                controller.message("No ID found")
```

```
            go_on = False
        else:
            "Input correct number"
        session.commit()
        pass
```

Запит редагування

Цей запит реалізовано за допомогою функції update. Спочатку користувач обирає, у якій таблиці потрібно змінити запис і за яким ідентифікатором. Також потрібно обрати атрибут, що редагується. У разі успішного редагування користувач побачить відповідне повідомлення. Редагування запиту продемонструємо на прикладі таблиці student.

```
 Student
+------------+-----------+----------+
| student_id | firstname | lastname |
+------------+-----------+----------+
|     1      |  Nikita   |  Linkov  |
+------------+-----------+----------+
```

Рис.12 – Початковий стан таблиці student

```
            1 -> subject
            2 -> teacher
            3 -> student
            4 -> phone
            5 -> schedule

Choose table number => 3
Enter student_id: 1

                1 -> firstname
                2 -> lastname

Choose a number of column to update: 1
```

Рис.13 – Меню для вибору таблиці та атрибуту запису для редагування

```
Choose a number of column to update: 1
Enter firstname: Maksym
The row with 'student_id' = '1' in table 'student' was updated successfully.
Continue working with update? Enter Yes or No
```

Рис.14 – Повідомлення про успішне редагування рядка

Student

student_id	firstname	lastname
1	Maksym	Linkov

Рис.15 – Стан таблиці після виконання запиту редагування

Лістинг функції update

```
def update():
    Session = sessionmaker(bind=engine)
    session = Session()
    go_on = True
    while go_on:
        View.list()
        table = controller.validtable()
        if table == 1:
            id = controller.validate_input_items("subject_id")
            records = session.query(Subject).get(id)
            if records is not None:
                value = controller.validate_input_items("name")
                upd = session.query(Subject).get(id)
                upd.name = value
                session.add(upd)
            else:
                print("No subject with this ID")
            go_on = False
        elif table == 2:
            id = controller.validate_input_items("teacher_id")
            records = session.query(Teacher).get(id)
            if records is not None:
                View.columns(2)
                continue_update = True
                while continue_update:
                    attr = input("Choose a number of column to update ")
                    if attr == '1':
                        value = controller.validate_input_items("firstname")
                        upd = session.query(Teacher).get(id)
                        upd.firstname = value
                        session.add(upd)
                        continue_update = False
                    elif attr == '2':
                        value = controller.validate_input_items("lastname")
                        upd = session.query(Teacher).get(id)
                        upd.lastname = value
                        session.add(upd)
                        continue_update = False
                    else:
                        print("Enter correct number ")
                View.complete_message("teacher_id", id, "teacher", "updated")
                go_on = False
                pass
            else:
                print("No teacher with this ID")
        elif table == 3:
            id = controller.validate_input_items("student_id")
```

```
            records = session.query(Student).get(id)
            if records is not None:
                View.columns(3)
                continue_update = True
                while continue_update:
                    attr = input("Choose a number of column to update: ")
                    if attr == '1':
                        value = controller.validate_input_items("firstname")
                        upd = session.query(Student).get(id)
                        upd.firstname = value
                        session.add(upd)
                        continue_update = False
                    elif attr == '2':
                        value = controller.validate_input_items("lastname")
                        upd = session.query(Student).get(id)
                        upd.lastname = value
                        session.add(upd)
                        continue_update = False
                    else:
                        print("Enter correct number ")
                    View.complete_message("student_id", id, "student",
"updated")
                    go_on = False
                    pass
            else:
                print("No student with this ID")
        elif table == 4:
            id = controller.validate_input_items("phone_id")
            records = session.query(Phone).get(id)
            if records is not None:
                View.columns(4)
                continue_update = True
                while continue_update:
                    attr = input("Choose a number of column to update: ")
                    if attr == '1':
                        value = controller.validate_input_items("phonenumber")
                        upd = session.query(Phone).get(id)
                        upd.phonenumber = value
                        session.add(upd)
                        continue_update = False
                    elif attr == '2':
                        value = controller.validate_input_items("student_id")
                        upd = session.query(Phone).get(id)
                        upd.studentphone_fk = value
                        session.add(upd)
                        continue_update = False
                    go_on = False
                    pass
            else:
                print("No phone with this ID")
        elif table == 5:
            id = controller.validate_input_items("schedule_id")
            records = session.query(Schedule).get(id)
            if records is not None:
                View.columns(5)
                continue_update = True
                while continue_update:
                    attr = input("Choose a number of column to update: ")
                    if attr == '1':
                        value = controller.validate_input_items("day")
                        upd = session.query(Schedule).get(id)
                        upd.day = value
                        session.add(upd)
```

```
                    continue_update = False
                elif attr == '2':
                    value = controller.validate_input_items("time")
                    upd = session.query(Schedule).get(id)
                    upd.time = value
                    session.add(upd)
                    continue_update = False
                elif table == '3':
                    value = controller.validate_input_items("subject_id")
                    upd = session.query(Schedule).get(id)
                    upd.subject_fk = value
                    session.add(upd)
                    continue_update = False
                elif table == '4':
                    value = controller.validate_input_items("student_id")
                    upd = session.query(Schedule).get(id)
                    upd.student_fk = value
                    session.add(upd)
                    continue_update = False
                elif table == '5':
                    value = controller.validate_input_items("teacher_id")
                    upd = session.query(Schedule).get(id)
                    upd.teacher_fk = value
                    session.add(upd)
                    continue_update = False
            go_on = False
            pass
        else:
            print("No schedule with this ID")
    else:
        print("Please enter correct number ")
session.commit()
pass
```

Завдання 2

Для тестування індексів було створено окремі таблиці у базі даних test з 1000000 записів.

GIN

GIN призначений для обробки випадків, коли елементи, що підлягають індексації, є складеними значеннями (наприклад - реченнями), а запити, які обробляються індексом, мають шукати значення елементів, які з'являються в складених елементах (повторювані частини слів або речень). Індекс GIN зберігає набір пар (ключ, список появи ключа), де список появи — це набір ідентифікаторів рядків, у яких міститься ключ. Один і той самий ідентифікатор рядка може знаходитись у кількох списках, оскільки елемент може містити більше одного ключа. Кожне значення ключа зберігається лише один раз, тому індекс GIN дуже швидкий для випадків, коли один і той же ключ з’являється багато разів. Цей індекс може взаємодіяти тільки з полем типу tsvector.

SQL запити

Створення таблиці БД:

```
DROP TABLE IF EXISTS "gin_test";

CREATE TABLE "gin_test"("id" bigserial PRIMARY KEY, "string" text, "gin_vector" tsvector);
```

```
INSERT INTO "gin_test"("string") SELECT substr(characters, (random()*length(characters)+1)::integer, 10) FROM (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters), generate_series(1, 1000000) as q;
```

```
UPDATE "gin_test" set "gin_vector" = to_tsvector("string");
```

Запити для тестування:

Було протестовано 4 запити: 1 - виведення записів, у яких ідентифікатор кратний числу 2; 2 - виведення записів, у яких наявне сполучення букв bnm; 3 - виведення суми ідентифікаторів записів, у яких наявні сполучення букв QWERTYUIO або bnm; 4 - виведення мінімального ідентифікатора та максимального ідентифікатора записів, де є сполучення букв bnm, сортування за кратними 2 ідентифікаторами.

```
SELECT COUNT(*) FROM "gin_test" WHERE "id" % 2 = 0;
```

```
SELECT COUNT(*) FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('bnm'));
```

```
SELECT SUM("id") FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('QWERTYUIOP')) OR ("gin_vector" @@ to_tsquery('bnm'));
```

```
SELECT MIN("id"), MAX("id") FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('bnm')) GROUP BY "id" % 2;
```

Створення індексу:

```
DROP INDEX IF EXISTS "gin_index";
CREATE INDEX "gin_index" ON "gin_test" USING gin("gin_vector");
```

Результати виконання запитів

Запити вводились у psql.exe.

Запити без індексування:

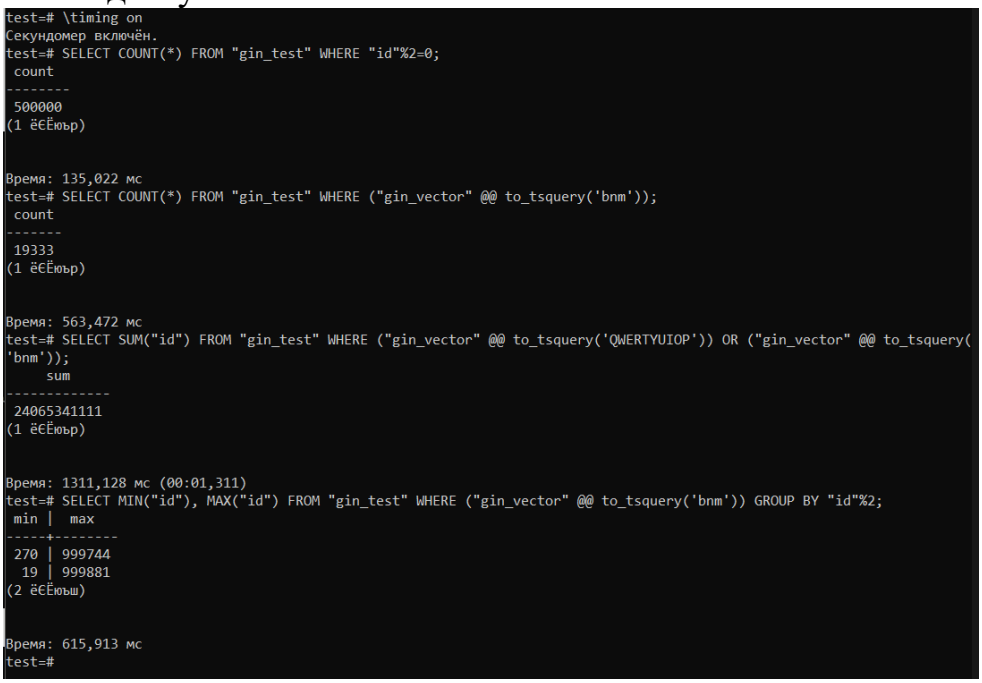


Рис.16 – Запити без індексування

Час виконання запитів

Операція 1	Операція 2	Операція 3	Операція 4
135,022 мс	563,472 мс	1311,128 мс	615,913 мс

Табл.1 – Час виконання запитів без індексу GIN

```
Время: 615,913 мс
test=# CREATE INDEX "gin_index" ON "gin_test" USING gin("gin_vector");
CREATE INDEX
Время: 445,428 мс
test=# SELECT COUNT(*) FROM "gin_test" WHERE "id"%2=0;
 count
--------
 500000
(1 ёЄЁюър)


Время: 117,865 мс
test=# SELECT COUNT(*) FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('bnm'));
 count
-------
 19333
(1 ёЄЁюър)


Время: 108,245 мс
test=# SELECT SUM("id") FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('QWERTYUIOP')) OR ("gin_vector" @@ to_tsquery(
'bnm'));
     sum
-------------
 24065341111
(1 ёЄЁюър)


Время: 97,675 мс
test=# SELECT MIN("id"), MAX("id") FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('bnm')) GROUP BY "id"%2;
 min |  max
-----+--------
 270 | 999744
  19 | 999881
(2 ёЄЁюъш)


Время: 80,801 мс
```

Рис.17 – Запити з індексуванням

Час виконання запитів

Операція 1	Операція 2	Операція 3	Операція 4
117,865 мс	108,245 мс	97,675 мс	80,801 мс

Табл.2 – Час виконання операції з індексом GIN

З отриманих результатів бачимо, що в усіх заданих випадках пошук з індексацією відбувається значно швидше, ніж пошук без індексації (окрім першого, оскільки на перший запит дана індексація не впливає). Це відбувається завдяки головній особливості індексування GIN: кожне значення шуканого ключа зберігається один раз і запит іде не по всій таблиці, а лише по тим даним, що містяться у списку появи цього ключа. Для даних типу numeric даний тип індексування використовувати недоцільно і неможливо.

BTree

Індекс BTree призначений для даних, які можна відсортувати. Іншими словами, для типу даних мають бути визначені оператори «більше», «більше або дорівнює», «менше», «менше або дорівнює» та «дорівнює». Пошук починається з кореня вузла, і потрібно визначити, по якому з дочірніх вузлів спускатися. Знаючи ключи в корені, можна зрозуміти діапазони значень в дочірніх вузлах. Процедура повторюється до тих пір, поки не буде знайдено вузол, з якого можна отримати необхідні дані.

SQL запити

Створення таблиці БД і внесення 1000000 записів:

```
DROP TABLE IF EXISTS "btree_test";
CREATE TABLE "btree_test"("id" bigserial PRIMARY KEY, "time" timestamp);
INSERT INTO "btree_test"("time") SELECT (timestamp '2021-01-01' + random()*(timestamp '2020-01-01'-timestamp '2022-01-01')) FROM (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters), generate_series(1, 1000000) as q;
```

Запити для тестування:

Було протестовано 4 запити: 1 – виведення записів, ідентифікатор яких кратний 2 (рис.18 та рис.21); 2 - виведення записів, у яких час більше або дорівнює 2019-10-01 (рис.19 та рис.21); 3 - виведення середнього значення ідентифікаторів записів, у яких час знаходиться в проміжку між 2019-10-01 та 2021-12-7 (рис.19 і рис.21); 4 - виведення суми ідентифікаторів, а також максимального ідентифікатора записів, у яких час знаходиться в промяжку між 2020-05-05 та 2021-05-05, сортування за кратними 2 ідентифікаторами (рис.20 і рис.21).

```
SELECT COUNT(*) FROM "btree_test" WHERE "id" % 2 = 0;

SELECT COUNT(*) FROM "btree_test" WHERE “time” >= ‘20191001’;

SELECT AVG(“id”) FROM “btree_test” WHERE “time” >= ‘20191001’ AND “time” <= ‘20211207’;

SELECT SUM(“id”), MAX(“id”) FROM “btree_test” WHERE “time” >= ‘20200505’ AND “time” <= ‘20210505’ GROUP BY “id”%2;
```

Створення індексу:

```
DROP INDEX IF EXISTS "btree_index";
CREATE INDEX "btree_time_index" ON "btree_test" ("id");
```

Результати виконання запитів

Запити вводились у psql.exe.

Запити без індексування:

```
test=# \timing on
Секундомер включён.
test=# SELECT COUNT(*) FROM "btree_test" WHERE "id"%2=0;
 count
--------
 500000
(1 ёЄЁюър)
```

Рис.18 – Запит 1

```
test=# SELECT COUNT(*) FROM "btree_test" WHERE "time" >= '20191001';
 count
--------
 626306
(1 ёЄЁюър)


Время: 92,365 мс
test=# SELECT AVG("id") FROM "btree_test" WHERE "time" >= '20191001' AND "time" <= '20211207';
         avg
---------------------
 499895.262130651790
(1 ёЄЁюър)
```

Рис.19 – Запит 2 і 3

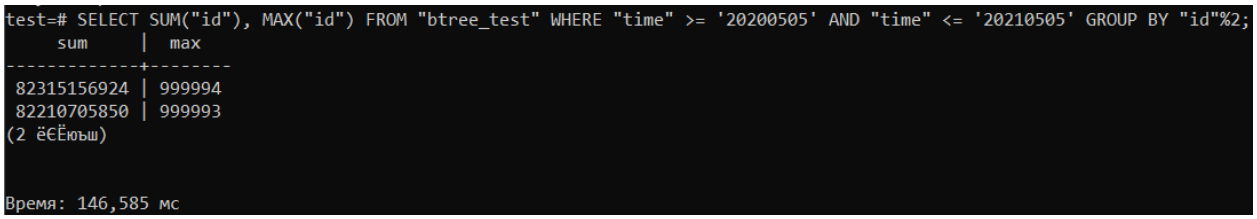


Рис.20 – Запит 4

Час виконання запитів

Операція 1	Операція 2	Операція 3	Операція 4
111,491 мс	92,365 мс	154,643	146,584 мс

Табл.3 – Час виконання запитів без індексу BTree

Запити з індексуванням:

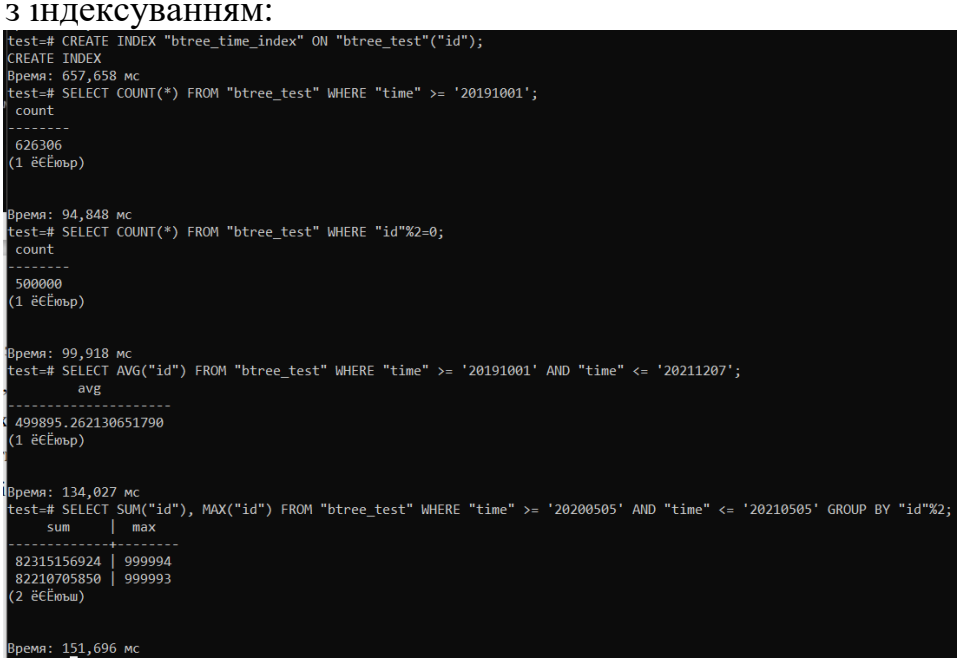


Рис.21 – Запити з індексуванням

Час виконання запитів

Операція 1	Операція 2	Операція 3	Операція 4
94,848 мс	99,918 мс	134,027	151,696 мс

Табл.4 – Час виконання запитів з індексом BTree

Як бачимо з результатів, за допомогою використання індексу BTree виконання операцій дещо пришвидшилося. Це пов’язано з тим, що дерево утворює багато гілок, і через це B-дерево виходить неглибоким навіть для дуже великих таблиць. Цей індекс також рекомендовано використовувати саме для операцій пошуку с порівнянням (нерівностями), що і було продемонстровано в запитах.

Завдання 3

Для тестування тригера було створено дві таблиці в базі даних test: таблиця trigger_test з атрибутами trigger_testID (ідентифікатор) trigger_textName (ім’я), trigger_test_log з атрибутами id (ідентифікатор), trigger_test_log_ID (зовнішній ключ для зв’язку з таблицею trigger_test), trigger_test_log_name (ім’я).

Тригер спрацьовує після операції вставки (after insert) та під час операції редагування (update). Серед усіх записів таблиці trigger_test у курсорному циклі

обираються ті, що мають ідентифікатори кратні 2. Якщо цей ідентифікатор також кратний 3, то висвічується повідомлення, що число ділиться на 2 і 3. Також якщо ідентифікатор кратний 2 і 3, то в таблицю trigger_test_log вставляються рядки з цими ідентифікаторами та відповідними іменами. В іншому випадку (якщо число не ділиться на 3, але ділиться на 2), викликається повідомлення - «Число парне» і в таблицю trigger_test_log вставляються рядки з цими ідентифікаторами та відповідними іменами. Далі з атрибуту trigger_test_log_name видаляються набори символів ‘log’. Якщо число не ділиться на 2, то висвічується повідомлення «Число непарне» і виконується редагування в курсорному циклі: для всіх записів таблиці trigger_test_log, що мають в назві сполучення букв ‘_id’ потрібно замінити ім’я на ‘_’ та trigger_test_log_name та ‘_log’.

Тригер спрацьовує, якщо викликати операцію вставки (insert) або редагування (update). Нижче на знімках екрану продемонстровано коректну роботу тригера.

Запити для створення таблиць:

```
DROP TABLE IF EXISTS "trigger_test";
CREATE TABLE "trigger_test"(
"trigger_testID" bigserial PRIMARY KEY,
      "trigger_testName" text
);
DROP TABLE IF EXISTS "trigger_test_log";
CREATE TABLE "trigger_test_log"(
"id" bigserial PRIMARY KEY,
      "trigger_test_log_ID" bigint,
      "trigger_test_log_name" text
);
```

Команди, що ініціюють виконання тригера:

```
CREATE TRIGGER "after_insert_update_trigger"

AFTER INSERT OR UPDATE ON "trigger_test"

FOR EACH ROW

EXECUTE procedure after_insert_func();
```

Початковий вміст таблиці trigger_test було задано запитом:

```
INSERT INTO trigger_test("trigger_testName") VALUES ('test1'), ('test2'), ('test3'), ('test4'), ('test5'), ('test6');
```

Текст тригера:

```
CREATE OR REPLACE FUNCTION after_insert_func() RETURNS TRIGGER AS $trigger$
DECLARE
CURSOR_LOG CURSOR FOR SELECT * FROM "trigger_test_log";
row_ "trigger_test_log"%ROWTYPE;
BEGIN
IF NEW."trigger_testID"%2=0 THEN
IF NEW."trigger_testID"%3=0 THEN
RAISE NOTICE 'trigger_testID is multiple of 2 and 3';
FOR row_ IN CURSOR_LOG LOOP
-- UPDATE "trigger_test_log" SET "trigger_test_log_name"='_' || row_."trigger_test_log_name" || '_log' WHERE "id"=row_."id";
INSERT INTO "trigger_test_log"("trigger_test_log_ID", "trigger_test_log_name") VALUES (NEW."trigger_testID", NEW."trigger_testName");
END LOOP;
RETURN NEW;
```

```
	ELSE
	RAISE NOTICE 'trigger_testID is even';
	INSERT INTO "trigger_test_log"("trigger_test_log_ID", "trigger_test_log_name")
VALUES (NEW."trigger_testID", NEW."trigger_testName");
	UPDATE "trigger_test_log" SET "trigger_test_log_name" = trim(BOTH '_log' FROM
"trigger_test_log_name");
	RETURN NEW;
	END IF;
	ELSE
	RAISE NOTICE 'trigger_testID is odd';
	FOR row_ IN CURSOR_LOG LOOP
	UPDATE "trigger_test_log" SET "trigger_test_log_name" = '_' ||
row_."trigger_test_log_name" || '_log' WHERE "id" = row_."id";
	END LOOP;
	RETURN NEW;
	END IF;
	END;
	$trigger$ LANGUAGE plpgsql;

	CREATE TRIGGER after_insert_test
	AFTER INSERT OR UPDATE ON "trigger_test"
	FOR EACH ROW EXECUTE PROCEDURE after_insert_func();
```

Результат виконання

Приклад виконання наведено для таблиць trigger_test та trigger_test_log.

	trigger_testID [PK] bigint	trigger_testName text

Рис.22 – Початковий стан таблиці trigger_test

	id [PK] bigint	trigger_test_log_ID bigint	trigger_test_log_name text

Рис.23 – Початковий стан таблиці trigger_test_log

Далі було виконано запит на вставку.

```
INSERT INTO trigger_test("trigger_testName") VALUES ('test7'), ('test8');
```

	trigger_testID [PK] bigint	trigger_testName text
1	1	test1
2	2	test2
3	3	test3
4	4	test4
5	5	test5
6	6	test6

Рис.24 – Таблиця trigger_test після виконання операції вставки

	id [PK] bigint	trigger_test_log_ID bigint	trigger_test_log_name text
1	1	2	_test2_log
2	2	4	_test4_log
3	3	6	test6
4	4	6	test6

Рис.25 – Таблиця trigger_test_log після виконання операції вставки

Запит на редагування (додає у всі записи із парними ідентифікаторами в імені сполучення символів ‘_2’):

```
UPDATE "trigger_test" SET "trigger_testName" = "trigger_testName" || '_2' WHERE "trigger_testID"%2=0
```

	trigger_testID [PK] bigint	trigger_testName text
1	1	test1
2	2	test2
3	3	test3
4	4	test4
5	5	test5
6	6	test6

Рис.26 – Початковий стан таблиці trigger_test

	id [PK] bigint	trigger_test_log_ID bigint	trigger_test_log_name text
1	1	2	_test2_log
2	2	4	_test4_log
3	3	6	test6
4	4	6	test6

Рис.27 – Початковий стан таблиці trigger_test_log

	trigger_testID [PK] bigint	trigger_testName text
1	1	test1
2	2	test2_2
3	3	test3
4	4	test4_2
5	5	test5
6	6	test6_2

Рис.28 – Стан таблиці trigger_test після виконання операції редагування

	id [PK] bigint	trigger_test_log_ID bigint	trigger_test_log_name text
1	1	2	test2_2
2	2	4	test4_2
3	3	6	test6_2
4	4	6	test6_2

Рис.29 – Стан таблиці trigger_test_log після виконання операції редагування та спрацювання тригера

Як видно на знімках, записи із парними ідентифікаторами записалися в таблицю trigger_test_log, аналогічно до першого випадку із запитом вставки. Алгоритмічно нічого не змінилося, оскільки всі дії виконуються в залежності від значення ідентифікатора, який не змінювався.

Якщо виконати запити вставки та редагування по черзі, то ситуація дещо зміниться. Спочатку у таблицю trigger_test вставляються записи з ідентифікаторами від 1 до 8. Далі для парних ідентифікаторів записи копіюються в trigger_test_log, а запис з ідентифікатором 6 копіюється в таблицю тричі, оскільки він ділиться і на 2, і на 3.

	trigger_testID [PK] bigint	trigger_testName text

Рис.30 – Стан таблиці trigger_test до почергової вставки та редагування

	id [PK] bigint	trigger_test_log_ID bigint	trigger_test_log_name text

Рис.31 – Стан таблиці trigger_test_log до почергової вставки та редагування

	trigger_testID [PK] bigint	trigger_testName text
1	1	test1
2	2	test2_2
3	3	test3
4	4	test4_2
5	5	test5
6	6	test6_2
7	7	test7
8	8	test8_2

Рис.32 – Стан таблиці trigger_test після вставки та редагування

	id [PK] bigint	trigger_test_log_ID bigint	trigger_test_log_name text
1	1	8	test8
2	2	2	test2_2
3	3	4	test4_2
4	4	6	test6_2
5	5	6	test6_2
6	6	6	test6_2
7	7	8	test8_2

Рис.33 – Стан таблиці trigger_test_log після вставки та редагування

Завдання 4

Для цього завдання знадобилася окрема таблиця “transactions” з атрибутами id (ідентифікатор), numeric (число), text (текст). Також у таблицю було додано три записи за допомогою запиту вставки insert.

Запит на створення таблиці та вставку:

```
DROP TABLE IF EXISTS "transactions";
CREATE TABLE "transactions"(
"id" bigserial PRIMARY KEY,
      "numeric" bigint,
      "text" text
);
INSERT INTO "transactions"("numeric", "text") VALUES (222, 'string1'), (223, 'string2'), (224, 'string3');
```

READ COMMINTTED

На цьому рівні ізоляції одна транзакція не бачить змін у базі даних, викликаних іншою доки та не завершить своє виконання (командою COMMIT або ROLLBACK).

Спочатку у транзакціях 1 і 2 таблиця має однаковий стан. Якщо у транзакції 1 виконати редагування одного рядка, то в транзакції 2 цих змін не буде помітно, поки в першій транзакції не буде команди commit. Таким чином, феномен «брудного читання» на цьому рівні ізоляції неможливий.

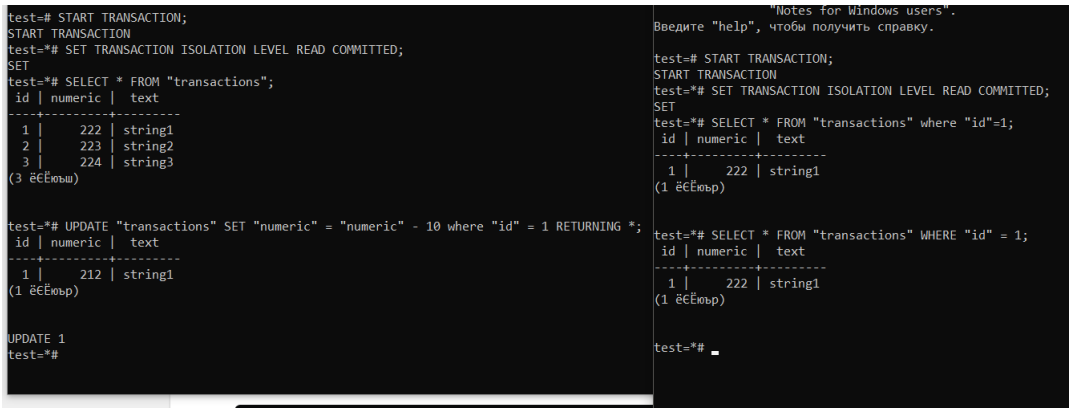


Рис.34 – Виконання редагування в одній з одночасних транзакцій на рівні ізоляції Rad committed

Тепер дослідимо феномен «фантомного читання». Після команди commit у першій транзакції у другій ми побачимо, що умові numeric>=220 відповідають лише 2 рядки, а не 3, як раніше, оскільки зміни були внесені і збережені в обох транзакціях.

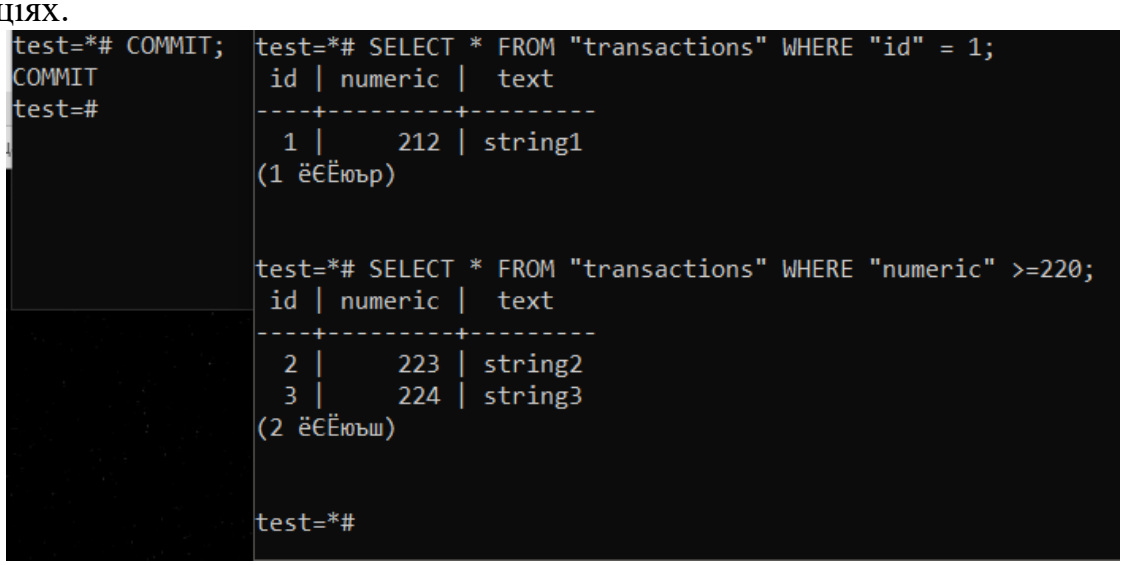


Рис.35 – Феномен «фантомного читання»

REPEATABLE READ

Почнемо дві транзакції на рівні ізоляції repeatable read. У другій транзакції обираємо запис з id = 1 і виводимо записи з numeric >=212. Тепер віднімемо від numeric у рядку з ідентифікатором 1 першої транзакції 10. Викликаємо команду commit. У другій транзакції ніяких змін із цим рядком немає, хоча commit був виконаний. Це сталося через використання рівня ізоляції repeatable read, тобто один і той самий запит має повертати той самий результат. Це призводить до того, що феномен «неповторного читання» неможливий на цьому рівні ізоляції.

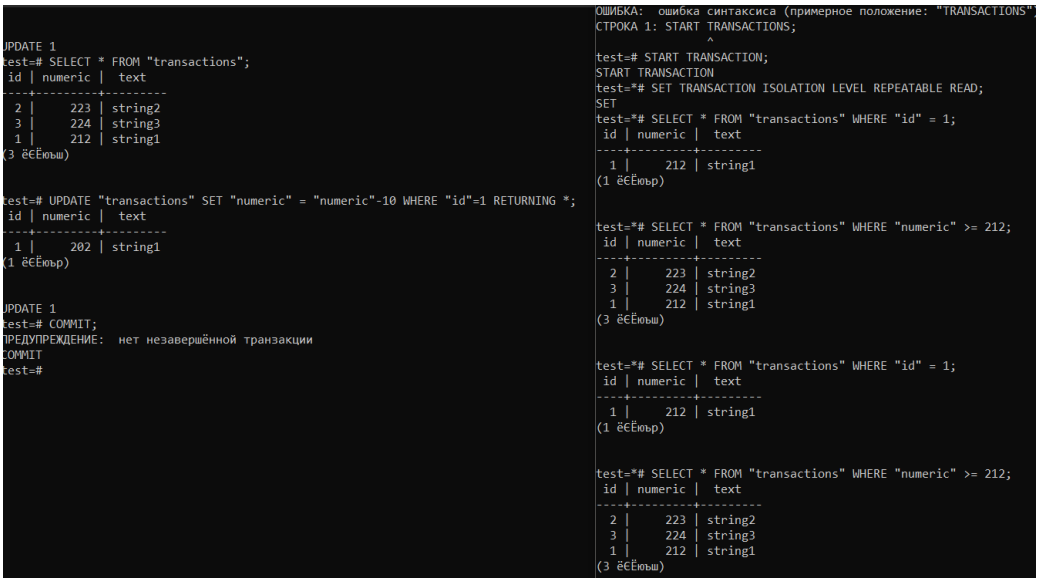


Рис.36 – Спроба одночасного внесення змін у таблиці на рівні ізоляції Repeatable read

Якщо спробувати в другій транзакції виконати запит редагування того самого рядка і відняти від numeric 10, то нам висвітиться помилка через паралельні зміни в транзакціях. Це є перевагою repeatable read, оскільки тоді ми можемо уникнути ситуацій, коли при відніманні 10 від 212 ми отримаємо 192 замість 202. Щоб відмінити транзакцію, було викликано команду rollback.

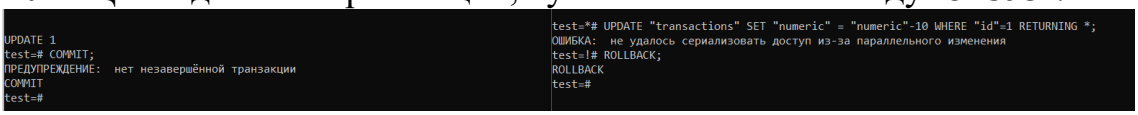


Рис.37 – Помилка через паралельні зміни на рівні ізоляції Repeatable read

Дослідимо аномалію серіалізації. На рівні ізоляції repeatable read запустимо дві транзакції. У першій виведемо всі рядки і порахуємо суму стовпчика numeric у всіх записах. Додаємо запис із цим значенням в таблицю. Якщо у другій транзакції повторити ті ж самі операції, то стан таблиці на початку ще не змінений, сума буде такою ж, як у першій транзакції. Таким чином, ми додамо до таблиці такий самий рядок, як і першій транзакції. Виконуючи commit в обох транзакціях, ми побачимо два однакових записи в таблиці. Це і є феномен «серіалізації», що пояснюється серійним виконанням двох транзакцій однієї за одною, причому порядок виконання транзакції неважливий.

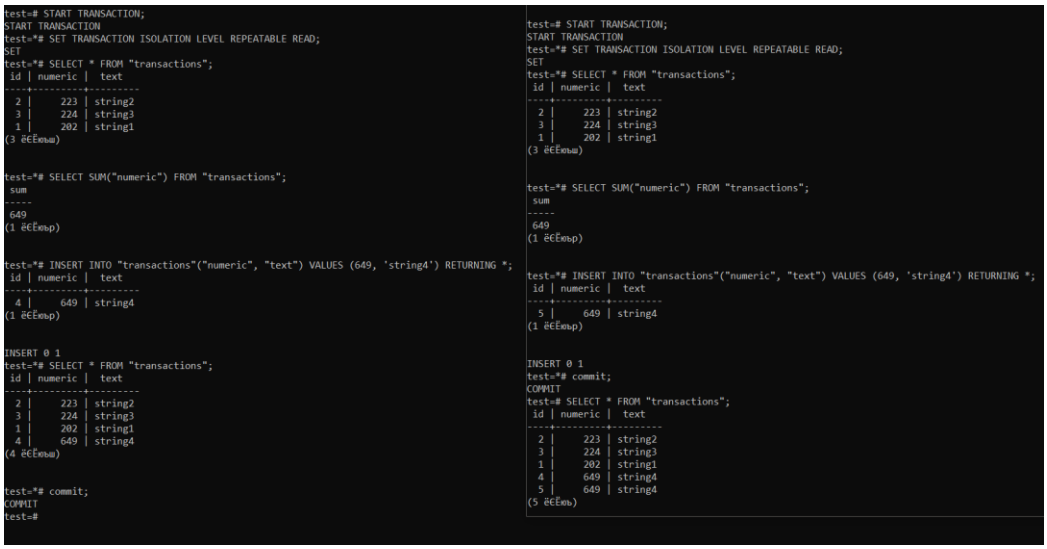


Рис.38 – Аномалія серіалізації

SERIALIZABLE

Запустимо дві транзакції на рівні Serializable. Спочатку стан таблиці однаковий. У першій транзакції видалимо рядок з id = 5 та виконаємо редагування рядку з id = 4. Якщо у другій транзакції спробувати зробити ті ж операції, то ми повинні будемо очікувати, доки перша транзакція не завершиться. Коли команда commit у першій транзакції виконана, у другій виникає помилка через паралельне видалення. Це неможливо, оскільки якщо запис уже видалений в першій транзакції, то видалити рядок з неіснуючим ідентифікатором неможливо. Ситуацію рятує команда rollback, і після цього бачимо, що зміни внесені і в другу транзакцію.

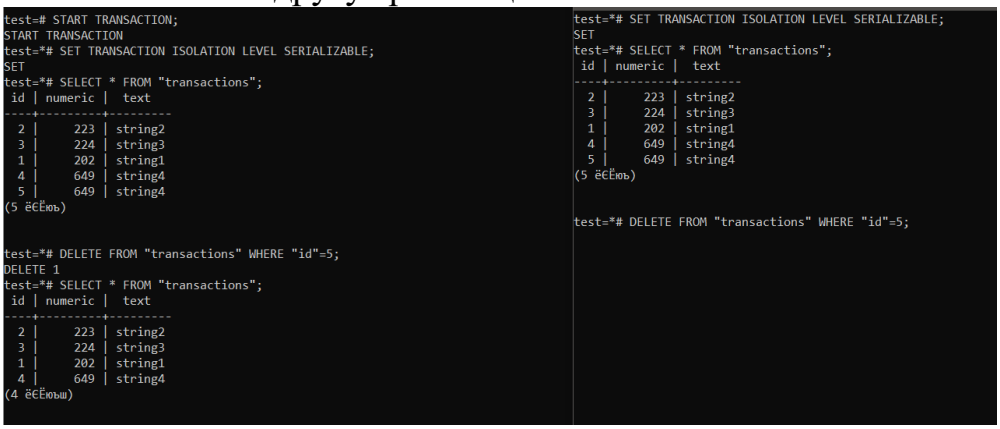


Рис.39 – Спроба одночасного виконання запитів на рівні ізоляції Serializable

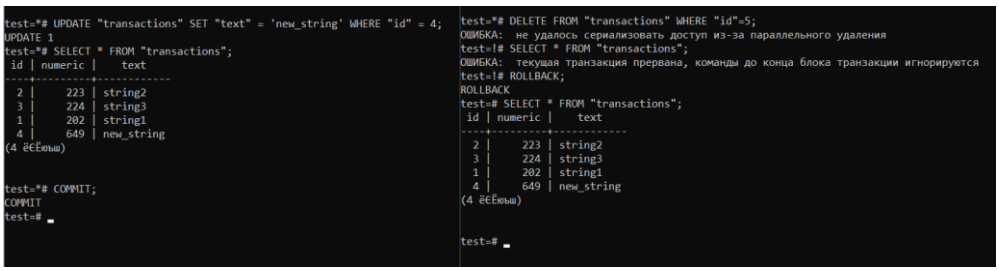


Рис.40 – Помилка через паралельне вилучення запису на рівні ізоляції Serializable

Посилання для навігації по тексту програми

Текст програми

main.py

```
from menu import Menu

Menu.menu()
```

controller.py

```
from view import View


def message(text):
    return print(text)


def validtable():
    incorrect = True
    while incorrect:
        table = input('Choose table number => ')
        if table.isdigit():
            table = int(table)
            if 1 <= table <= 5:
                incorrect = False
            else:
                print('Incorrect input, try again.')
        else:
            print('Incorrect input, try again.')
    return table


def validate_input_items(name):
    if name == "id":
        value = View.enter_item("id")
        if value.isdecimal():
            return value
    elif name == "subject_id":
        value = View.enter_item("subject_id")
        if value.isdecimal():
            return value
    elif name == "teacher_id":
        value = View.enter_item("teacher_id")
        if value.isdecimal():
```

```
            return value
        else:
            message("enter only number")
            validate_input_items("teacher_id")
    elif name == "phone_id":
        value = View.enter_item("phone_id")
        if value.isdecimal():
            return value
    elif name == "schedule_id":
        value = View.enter_item("schedule_id")
        if value.isdecimal():
            return value
    elif name == "student_id":
        value = View.enter_item("student_id")
        if value.isdecimal():
            return value
    elif name == "name":
        value = View.enter_item("name")
        if value.isalpha() is False:
            return validate_input_items(name)
        return value
    elif name == "firstname":
        value = View.enter_item("firstname")
        if value.isalpha() is False:
            return validate_input_items(name)
        return value
    elif name == "lastname":
        value = View.enter_item("lastname")
        if value.isalpha() is False:
            return validate_input_items(name)
        return value
    elif name == "phonenumber":
        value = View.enter_item("phone_number")
        if value.isdigit() is False:
            print("Enter like in example: 0998889999")
            return validate_input_items(name)
        else:
            if len(value) != 10:
                print("Phone should have 10 numbers")
                return validate_input_items(name)
            return str("+38" + value)
    elif name == "day":
        value = View.enter_item("day")
        if value.isalpha() is False:
            return validate_input_items(name)
        return value
    elif name == "time":
        value = View.enter_item("time")
        li = list(value.split(":"))
        li_hour = li[0]
        li_min = li[1]
        if li_hour.isdecimal() and li_min.isdecimal():
            if 20 > int(li_hour) > 7 and -1 < int(li_min) < 60:
                li = [li_hour, li_min]
                return ':'.join(li)
            else:
                message("Please, enter hours from 7 to 20 and minutes from 0 to
59")
                return validate_input_items(name)
        else:
            message("Please, enter only digits with : between")
```

Лістинг view.py

```
class View:
    def __init__(self, table, records):
        self.table = table
        self.records = records

    @staticmethod
    def complete_message(attribute, value, table, action):
        print(f"The row with '{attribute}' = '{value}' in table '{table}' was
{action} successfully.")

    @staticmethod
    def enter_item(item):
        data = input("Enter {}: ".format(item))
        return data

    @staticmethod
    def list():
        print('''
            1 -> subject
            2 -> teacher
            3 -> student
            4 -> phone
            5 -> schedule
            ''')

    @staticmethod
    def columns(table):
        if table == 1:
            print('''
                1 -> name
            ''')
        elif table == 2:
            print('''
                1 -> firstname
                2 -> lastname
            ''')
        elif table == 3:
            print('''
                1 -> firstname
                2 -> lastname
            ''')
        elif table == 4:
            print('''
                1 -> phone number
                2 -> student ID
            ''')
        elif table == 5:
            print('''
                1 -> day
                2 -> time
                3 -> subject ID
                4 -> student ID
                5 -> teacher Id
            ''')
```

model.py

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Table, Index, Column, Integer, String, Text,
create_engine, Boolean, PrimaryKeyConstraint, \
    UniqueConstraint, ForeignKeyConstraint, ForeignKey
from sqlalchemy.orm import relationship
from sqlalchemy.orm import Session, sessionmaker
import controller
from view import View
from numpy import array
from tabulate import tabulate
from pandas import DataFrame

tables = {
    1: 'subject',
    2: 'teacher',
    3: 'student',
    4: 'phone',
    5: 'schedule',
}

Base = declarative_base()
engine =
create_engine('postgresql+psycopg2://postgres:1111@localhost:5432/school')


class Subject(Base):
    __tablename__ = 'subject'
    subject_id = Column(Integer, nullable=False, primary_key=True)
    name = Column(String, nullable=False)
    schedule = relationship('Schedule')

    def __init__(self, subject_id, name):
        self.subject_id = subject_id
        self.name = name

    def __repr__(self):
        return "{:>10}{:>35}".format(self.subject_id, self.name)


class Student(Base):
    __tablename__ = 'student'
    student_id = Column(Integer, primary_key=True)
    firstname = Column(String, nullable=False)
    lastname = Column(String, nullable=False)
    phones = relationship('Phone')
    schedule = relationship('Schedule')

    def __init__(self, student_id, firstname, lastname):
        self.student_id = student_id
        self.firstname = firstname
        self.lastname = lastname

    def __repr__(self):
        return "{:>10}{:>35}{:>35}".format(self.student_id, self.firstname,
self.lastname)
```

```
class Teacher(Base):
    __tablename__ = 'teacher'
    teacher_id = Column(Integer, primary_key=True)
    firstname = Column(String, nullable=False)
    lastname = Column(String, nullable=False)
    schedule = relationship('Schedule')

    def __init__(self, teacher_id, firstname, lastname):
        self.teacher_id = teacher_id
        self.firstname = firstname
        self.lastname = lastname

    def __repr__(self):
        return "{:>10}{:>35}{:>35}".format(self.teacher_id, self.firstname,
self.lastname)


class Phone(Base):
    __tablename__ = 'phone'
    phone_id = Column(Integer, primary_key=True)
    phonenumber = Column(String, nullable=False)
    studentphone_fk = Column(Integer, ForeignKey('student.student_id'))

    def __init__(self, phone_id, phonenumber, studentphone_fk):
        self.phone_id = phone_id
        self.phonenumber = phonenumber
        self.studentphone_fk = studentphone_fk

    def __repr__(self):
        return "{:>10}{:>35}{:>10}".format(self.phone_id, self.phonenumber,
self.studentphone_fk)


class Schedule(Base):
    __tablename__ = 'schedule'
    schedule_id = Column(Integer, primary_key=True)
    day = Column(String, nullable=False)
    time = Column(String, nullable=False)
    subject_fk = Column(Integer, ForeignKey('subject.subject_id'))
    student_fk = Column(Integer, ForeignKey('student.student_id'))
    teacher_fk = Column(Integer, ForeignKey('teacher.teacher_id'))

    def __init__(self, schedule_id, day, time, subject_fk, student_fk,
teacher_fk):
        self.schedule_id = schedule_id
        self.day = day
        self.time = time
        self.subject_fk = subject_fk
        self.student_fk = student_fk
        self.teacher_fk = teacher_fk

    def __repr__(self):
        return "{:>10}{:>35}{:>35}{:>10}{:>10}{:>10}".format(self.schedule_id,
self.day, self.time, self.subject_fk,
                                                             self.student_fk,
self.teacher_fk)


def display_query(rows, headers):
    df = DataFrame([array(el) for el in rows], columns=array(headers))
    print(tabulate(df, headers="keys", tablefmt="pretty", showindex=False))
```

```
def show_subject():
    print("\033[1m Subject \033[0m")
    Session = sessionmaker(bind=engine)
    session = Session()
    records = session.query(Subject.subject_id, Subject.name).all()
    return records


def show_teacher():
    print("\033[1m Teacher \033[0m")
    Session = sessionmaker(bind=engine)
    session = Session()
    records = session.query(Teacher.teacher_id, Teacher.firstname,
Teacher.lastname).all()
    return records


def show_student():
    print("\033[1m Student \033[0m")
    Session = sessionmaker(bind=engine)
    session = Session()
    records = session.query(Student.student_id, Student.firstname,
Student.lastname).all()
    return records


def show_phone():
    print("\033[1m Phone \033[0m")
    Session = sessionmaker(bind=engine)
    session = Session()
    records = session.query(Phone.phone_id, Phone.phonenumber,
Phone.studentphone_fk).all()
    return records


def show_schedule():
    print("\033[1m Schedule \033[0m")
    Session = sessionmaker(bind=engine)
    session = Session()
    records = session.query(Schedule.schedule_id, Schedule.day, Schedule.time,
Schedule.student_fk,
                            Schedule.teacher_fk, Schedule.subject_fk).all()
    return records


def insert() -> None:
    Session = sessionmaker(bind=engine)
    session = Session()
    go_on = True
    while go_on:
        View.list()
        table = controller.validtable()
        if table == 1:
            id = controller.validate_input_items("subject_id")
            name = controller.validate_input_items("name")
            s = Subject(subject_id=id, name=name)
            session.add(s)
            View.complete_message("subject_id", id, "subject", "inserted")
            go_on = False
        elif table == 2:
            id = controller.validate_input_items("teacher_id")
            firstname = controller.validate_input_items("firstname")
            lastname = controller.validate_input_items("lastname")
```

```
            s = Teacher(teacher_id=id, firstname=firstname, lastname=lastname)
            session.add(s)
            View.complete_message("teacher_id", id, "teacher", "inserted")
            go_on = False
        elif table == 3:
            id = controller.validate_input_items("student_id")
            firstname = controller.validate_input_items("firstname")
            lastname = controller.validate_input_items("lastname")
            s = Student(student_id=id, firstname=firstname, lastname=lastname)
            session.add(s)
            View.complete_message("student_id", id, "student", "inserted")
            go_on = False
        elif table == 4:
            id = controller.validate_input_items("phone_id")
            phonenumber = controller.validate_input_items("phonenumber")
            studentphone_fk = controller.validate_input_items("student_id")
            s = Phone(phone_id=id, phonenumber=phonenumber,
studentphone_fk=studentphone_fk)
            session.add(s)
            View.complete_message("phone_id", id, "phone", "inserted")
            go_on = False
        elif table == 5:
            id = controller.validate_input_items("schedule_id")
            day = controller.validate_input_items("day")
            time = controller.validate_input_items("time")
            subject_fk = controller.validate_input_items("subject_id")
            student_fk = controller.validate_input_items("student_id")
            teacher_fk = controller.validate_input_items("teacher_id")
            s = Schedule(schedule_id=id, day=day, time=time,
subject_fk=subject_fk, student_fk=student_fk,
                         teacher_fk=teacher_fk)
            session.add(s)
            View.complete_message("schedule_id", id, "schedule", "inserted")
            go_on = False
        session.commit()


def delete():
    Session = sessionmaker(bind=engine)
    session = Session()
    go_on = True
    while go_on:
        View.list()
        table = controller.validtable()
        if table == 1:
            id = controller.validate_input_items("subject_id")
            records = session.query(Subject).get(id)
            if records is not None:
                records = session.query(Schedule).get(id)
                if records is not None:
                    delete = session.query(Schedule).filter(Schedule.subject_fk
== id)
                    for i in delete:
                        session.delete(i)
                    View.complete_message("subject_fk", id, "schedule",
"deleted")
                session.delete(session.query(Subject).filter(Subject.subject_id
== id).one())
                View.complete_message("subject_id", id, "subject", "deleted")
            else:
                controller.message("No ID found")
            go_on = False
        elif table == 2:
```

```
            id = controller.validate_input_items("teacher_id")
            records = session.query(Teacher).get(id)
            if records is not None:
                records = session.query(Schedule).get(id)
                if records is not None:
                    delete = session.query(Schedule).filter(Schedule.teacher_fk
== id)
                    for i in delete:
                        session.delete(i)
                    View.complete_message("teacher_fk", id, "schedule",
"deleted")
                session.delete(session.query(Teacher).filter(Teacher.teacher_id
== id).one())
                View.complete_message("teacher_id", id, "teacher", "deleted")
            else:
                controller.message("No ID found")
            go_on = False
        elif table == 3:
            id = controller.validate_input_items("student_id")
            records = session.query(Student).get(id)
            if records is not None:
                records = session.query(Phone).get(id)
                if records is not None:
                    delete =
session.query(Schedule).filter(Schedule.studentphone_fk == id)
                    for i in delete:
                        session.delete(i)
                    View.complete_message("studentphone_fk", id, "phone",
"deleted")
                records = session.query(Schedule).get(id)
                if records is not None:
                    delete = session.query(Schedule).filter(Schedule.student_fk
== id)
                    for i in delete:
                        session.delete(i)
                    View.complete_message("student_fk", id, "schedule",
"deleted")
                session.delete(session.query(Student).filter(Student.student_id
== id).one())
                View.complete_message("student_id", id, "student", "deleted")
            else:
                controller.message("No ID found")
            go_on = False
        elif table == 4:
            id = controller.validate_input_items("phone_id")
            records = session.query(Phone).get(id)
            if records is not None:
                session.delete(session.query(Phone).filter(Phone.phone_id ==
id).one())
                View.complete_message("phone_id", id, "phone", "deleted")
            else:
                controller.message("No ID found")
            go_on = False
        elif table == 5:
            id = controller.validate_input_items("schedule_id")
            records = session.query(Schedule).get(id)
            if records is not None:

session.delete(session.query(Schedule).filter(Schedule.schedule_id == id).one())
                View.complete_message("schedule_id", id, "schedule", "deleted")
            else:
                controller.message("No ID found")
            go_on = False
```

```
        else:
            "Input correct number"
        session.commit()
        pass


def update():
    Session = sessionmaker(bind=engine)
    session = Session()
    go_on = True
    while go_on:
        View.list()
        table = controller.validtable()
        if table == 1:
            id = controller.validate_input_items("subject_id")
            records = session.query(Subject).get(id)
            if records is not None:
                value = controller.validate_input_items("name")
                upd = session.query(Subject).get(id)
                upd.name = value
                session.add(upd)
            else:
                print("No subject with this ID")
            go_on = False
        elif table == 2:
            id = controller.validate_input_items("teacher_id")
            records = session.query(Teacher).get(id)
            if records is not None:
                View.columns(2)
                continue_update = True
                while continue_update:
                    attr = input("Choose a number of column to update ")
                    if attr == '1':
                        value = controller.validate_input_items("firstname")
                        upd = session.query(Teacher).get(id)
                        upd.firstname = value
                        session.add(upd)
                        continue_update = False
                    elif attr == '2':
                        value = controller.validate_input_items("lastname")
                        upd = session.query(Teacher).get(id)
                        upd.lastname = value
                        session.add(upd)
                        continue_update = False
                    else:
                        print("Enter correct number ")
                View.complete_message("teacher_id", id, "teacher", "updated")
                go_on = False
                pass
            else:
                print("No teacher with this ID")
        elif table == 3:
            id = controller.validate_input_items("student_id")
            records = session.query(Student).get(id)
            if records is not None:
                View.columns(3)
                continue_update = True
                while continue_update:
                    attr = input("Choose a number of column to update: ")
                    if attr == '1':
                        value = controller.validate_input_items("firstname")
                        upd = session.query(Student).get(id)
                        upd.firstname = value
```

```
                        session.add(upd)
                        continue_update = False
                    elif attr == '2':
                        value = controller.validate_input_items("lastname")
                        upd = session.query(Student).get(id)
                        upd.lastname = value
                        session.add(upd)
                        continue_update = False
                    else:
                        print("Enter correct number ")
                    View.complete_message("student_id", id, "student",
"updated")
                    go_on = False
                    pass
            else:
                print("No student with this ID")
        elif table == 4:
            id = controller.validate_input_items("phone_id")
            records = session.query(Phone).get(id)
            if records is not None:
                View.columns(4)
                continue_update = True
                while continue_update:
                    attr = input("Choose a number of column to update: ")
                    if attr == '1':
                        value = controller.validate_input_items("phonenumber")
                        upd = session.query(Phone).get(id)
                        upd.phonenumber = value
                        session.add(upd)
                        continue_update = False
                    elif attr == '2':
                        value = controller.validate_input_items("student_id")
                        upd = session.query(Phone).get(id)
                        upd.studentphone_fk = value
                        session.add(upd)
                        continue_update = False
                    go_on = False
                    pass
            else:
                print("No phone with this ID")
        elif table == 5:
            id = controller.validate_input_items("schedule_id")
            records = session.query(Schedule).get(id)
            if records is not None:
                View.columns(5)
                continue_update = True
                while continue_update:
                    attr = input("Choose a number of column to update: ")
                    if attr == '1':
                        value = controller.validate_input_items("day")
                        upd = session.query(Schedule).get(id)
                        upd.day = value
                        session.add(upd)
                        continue_update = False
                    elif attr == '2':
                        value = controller.validate_input_items("time")
                        upd = session.query(Schedule).get(id)
                        upd.time = value
                        session.add(upd)
                        continue_update = False
                    elif table == '3':
                        value = controller.validate_input_items("subject_id")
                        upd = session.query(Schedule).get(id)
```

```
                        upd.subject_fk = value
                        session.add(upd)
                        continue_update = False
                    elif table == '4':
                        value = controller.validate_input_items("student_id")
                        upd = session.query(Schedule).get(id)
                        upd.student_fk = value
                        session.add(upd)
                        continue_update = False
                    elif table == '5':
                        value = controller.validate_input_items("teacher_id")
                        upd = session.query(Schedule).get(id)
                        upd.teacher_fk = value
                        session.add(upd)
                        continue_update = False
                go_on = False
                pass
            else:
                print("No schedule with this ID")
        else:
            print("Please enter correct number ")
    session.commit()
    pass
```

menu.py

```
import model
from view import View
import controller


class Menu:
    @staticmethod
    def menu():
        while True:
            print('''
                Main menu
                0 - Show one table
                1 - Show all tables
                2 - Insert data
                3 - Delete data
                4 - Update data
                5 - Exit
                ''')
            choice = input('Choose an option: ')
            if choice == '0':
                View.list()
                table = controller.validtable()
                if table == 1:
                    model.display_query(model.show_subject(), ["subject_id",
"name"])
                elif table == 2:
                    model.display_query(model.show_teacher(), ["teacher_id",
"firstname", "lastname"])
                elif table == 3:
                    model.display_query(model.show_student(), ["student_id",
"firstname", "lastname"])
                elif table == 4:
                    model.display_query(model.show_phone(), ["phone_id",
```

```
"phonenumber", "studentphone_fk"])
                elif table == 5:
                    model.display_query(model.show_schedule(),
                                        ["schedule_id", "day", "time",
"student_fk", "teacher_fk", "subject_fk"])
            elif choice == '1':
                model.display_query(model.show_subject(), ["subject_id",
"name"])
                model.display_query(model.show_teacher(), ["teacher_id",
"firstname", "lastname"])
                model.display_query(model.show_student(), ["student_id",
"firstname", "lastname"])
                model.display_query(model.show_phone(), ["phone_id",
"phonenumber", "studentphone_fk"])
                model.display_query(model.show_schedule(),
                                    ["schedule_id", "day", "time", "student_fk",
"teacher_fk", "subject_fk"])
            elif choice == '2':
                end_insert = False
                while not end_insert:
                    model.insert()
                    incorrect = True
                    while incorrect:
                        answer = input('Continue working with insert? Enter Yes
or No ')
                        if answer == 'No':
                            end_insert = True
                            incorrect = False
                        elif answer == 'Yes':
                            incorrect = False
                            pass
                        else:
                            print('Please, enter Yes or No')
            elif choice == '3':
                end_delete = False
                while not end_delete:
                    model.delete()
                    incorrect = True
                    while incorrect:
                        answer = input('Continue working with delete? Enter Yes
or No ')
                        if answer == 'No':
                            end_delete = True
                            incorrect = False
                        elif answer == 'Yes':
                            incorrect = False
                            pass
                        else:
                            print('Please, enter Yes or No ')
            elif choice == '4':
                end_update = False
                while not end_update:
                    model.update()
                    incorrect = True
                    while incorrect:
                        answer = input('Continue working with update? Enter Yes
or No ')
                        if answer == 'No':
                            end_update = True
                            incorrect = False
                        elif answer == 'Yes':
                            incorrect = False
                            pass
```

```
                    else:
                        print('Please, enter Yes or No ')
            elif choice == '5':
                break
            else:
                print('Please, enter valid number')
```