

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих комп’ютерних систем**

**Лабораторна робота №3**

з дисципліни **Бази даних і засоби управління**

*на тему: “Засоби оптимізації роботи СУБД PostgreSQL”*

Виконав:

студент ІII курсу

групи КВ-02

Шпот І.Р.

Перевірив:

Київ – 2022

*Мета роботи:* здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

Завдання роботи:

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 у PostgreSQL.

4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

**Варіант 24**

Завдання роботи полягає у наступному:

1. Перетворити модуль «Model» з шаблону MVC лабораторної роботи №2 у вигляд об’єктно реляційної проекції «ORM».
2. Створити та проаналізувати різні типи індексів PostgreSQL. Види індексів: GIN, BRIN
3. Розробити тригер бази даних PostgreSQL. Умови тригера: after update, insert
4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

**Логічна модель (схема) БД «Школа»**

Логічна модель (схему бази даних) наведено на Рисунок 1.

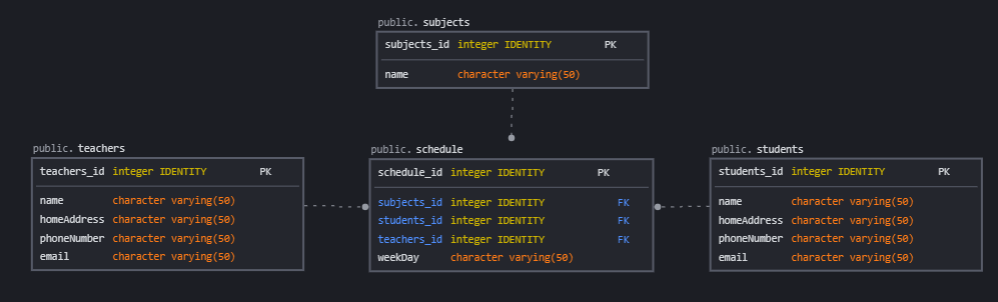


Рисунок 1 – Схема бази даних (інструмент sqldmb.com)

У базі даних були змінені назви таблиць та деяких атрибутів. Зміни полягають в тому, що всі назви були переведені в нижній регістр. Тобто Subjects → subjects, Teachers → teachers, Students → students, homeAddress → home\_address, phoneNumber → phone\_number та weekDay → week\_day.

**Середовище розробки та налаштування підключення до бази даних**

Для виконання лабораторної роботи використовувались мова програмування Python 3.10 та середовище розробки IDLE (Python 3.10 64-bit)

**Класи ORM**

**class** subjects(Base):

\_\_tablename\_\_ = 'subjects'

subjects\_id = Column(Integer, primary\_key=**True**)

name = Column(String)

sched = relationship("schedule", order\_by='schedule.schedule\_id',

back\_populates='subj')

**class** teachers(Base):

\_\_tablename\_\_ = 'teachers'

teachers\_id = Column(Integer, primary\_key=**True**)

name = Column(String)

home\_address = Column(String)

phone\_number = Column(String)

email = Column(String)

sched = relationship("schedule", order\_by='schedule.schedule\_id',

back\_populates='teach')

**class** students(Base):

\_\_tablename\_\_ = 'students'

students\_id = Column(Integer, primary\_key=**True**)

name = Column(String)

home\_address = Column(String)

phone\_number = Column(String)

email = Column(String)

sched = relationship("schedule", order\_by='schedule.schedule\_id',

back\_populates='stud')

**class** schedule(Base):

\_\_tablename\_\_ = 'schedule'

schedule\_id = Column(Integer, primary\_key=**True**)

subjects\_id = Column(Integer, ForeignKey('subjects.subjects\_id'))

teachers\_id = Column(Integer, ForeignKey('teachers.teachers\_id'))

students\_id = Column(Integer, ForeignKey('students.students\_id'))

week\_day = Column(String)

subj = relationship("subjects", back\_populates="sched")

teach = relationship("teachers", back\_populates="sched")

stud = relationship("students", back\_populates="sched"

**Результат роботи програми з використанням ORM**

Результат роботи «Insert» наведено на Рисунок 2. Стан таблиці «students» у застосунку pgAdmin4 до та після вставки наведено на Рисунок 3-4.

Зображення, що містить текст

Автоматично згенерований опис

Рисунок 2 – результат роботи «Insert»

Зображення, що містить стіл

Автоматично згенерований опис

Рисунок 3 – дані таблиці «students» у застосунку pgAdmin4 до вставки

Зображення, що містить стіл

Автоматично згенерований опис

Рисунок 4 – дані таблиці «students» у застосунку pgAdmin4 після вставки

Результат роботи «Delete» наведено на Рисунок 5. Стан таблиці «students» у застосунку pgAdmin4 після видалення наведено на Рисунок 6.

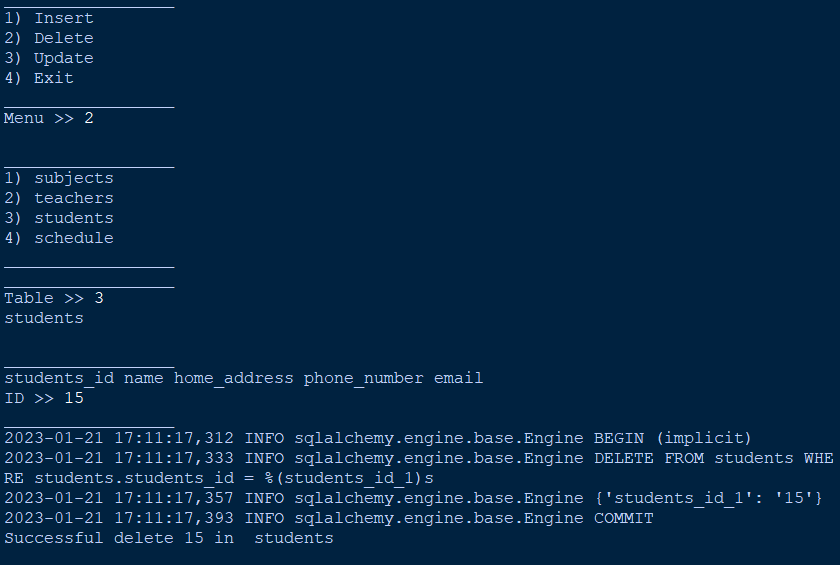


Рисунок 5– результат роботи «Delete»

Зображення, що містить стіл

Автоматично згенерований опис

Рисунок 6– дані таблиці «students» у застосунку pgAdmin4 після видалення

Результат роботи «Update» наведено на Рисунок 7. Стан таблиці «schedule» у застосунку pgAdmin4 до та після вставки наведено на Рисунок 8-9.

Зображення, що містить текст

Автоматично згенерований опис

Рисунок 7– результат роботи «Update»

Зображення, що містить стіл

Автоматично згенерований опис

Рисунок 8– дані таблиці «schedule» у застосунку pgAdmin4 до редагування

Зображення, що містить стіл

Автоматично згенерований опис

Рисунок 9 – дані таблиці «schedule» у застосунку pgAdmin4 після редагування

**Тригер after update, insert**

Робота тригера полягає у створенні нового елементу таблиці «schedule» після вставки чи редагування в таблиці «students». Результати роботи тригера зображення на Рисунок 10-12.

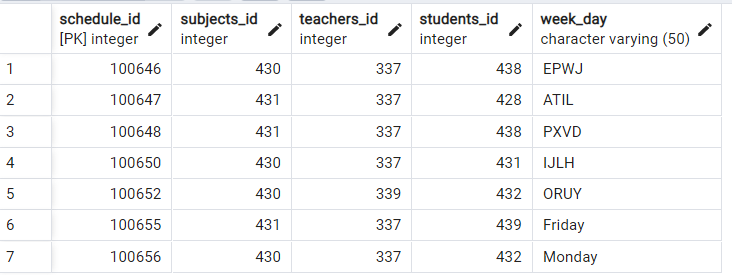


Рисунок 10 – стан таблиці «schedule» до роботи тригера

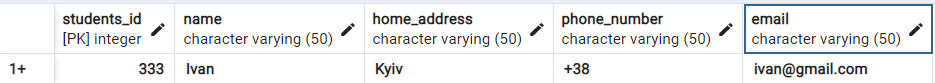


Рисунок 11 – додавання нового елементу у таблицю «students»

Зображення, що містить стіл

Автоматично згенерований опис

Рисунок 12 – стан таблиці «schedule» після роботи тригера

**Текст коду тригера**

Функція тригира:

**CREATE** **OR** **REPLACE** **FUNCTION** **public**.trigger\_fnc()

**RETURNS** **trigger**

**LANGUAGE** 'plpgsql'

COST 100

**VOLATILE** **NOT** LEAKPROOF

**AS** $BODY$

**BEGIN**

**INSERT** **INTO** "schedule" ("subjects\_id","teachers\_id","students\_id", "week\_day")

**VALUES**(1,1,**NEW**.students\_id,'Monday');

**RETURN** **NEW**;

**END**;

$BODY$;

**ALTER** **FUNCTION** **public**.trigger\_fnc()

**OWNER** **TO** postgres;

Тригер:

**CREATE** **TRIGGER** after\_insert\_update

**AFTER** **INSERT** **OR** **UPDATE**

**ON** **public**.students

**FOR** **EACH** **ROW**

**EXECUTE** **FUNCTION** **public**.trigger\_fnc();

**Текст програми**

orm.py

**from** **sqlalchemy** **import** create\_engine

**from** **sqlalchemy** **import** Column, ForeignKey, Integer, String

**from** **sqlalchemy.ext.declarative** **import** declarative\_base

**from** **sqlalchemy.orm** **import** relationship, sessionmaker

engine = create\_engine('postgresql+psycopg2://postgres:Pr0gr3s'

'@localhost:5432/School', echo=**True**)

Session = sessionmaker(bind=engine)

session = Session()

Base = declarative\_base()

**def** recreate():

Base.matadata.drop\_all(engine)

Base.matadata.create\_all(engine)

**class** subjects(Base):

\_\_tablename\_\_ = 'subjects'

subjects\_id = Column(Integer, primary\_key=**True**)

name = Column(String)

sched = relationship("schedule", order\_by='schedule.schedule\_id',

back\_populates='subj')

**class** teachers(Base):

\_\_tablename\_\_ = 'teachers'

teachers\_id = Column(Integer, primary\_key=**True**)

name = Column(String)

home\_address = Column(String)

phone\_number = Column(String)

email = Column(String)

sched = relationship("schedule", order\_by='schedule.schedule\_id',

back\_populates='teach')

**class** students(Base):

\_\_tablename\_\_ = 'students'

students\_id = Column(Integer, primary\_key=**True**)

name = Column(String)

home\_address = Column(String)

phone\_number = Column(String)

email = Column(String)

sched = relationship("schedule", order\_by='schedule.schedule\_id',

back\_populates='stud')

**class** schedule(Base):

\_\_tablename\_\_ = 'schedule'

schedule\_id = Column(Integer, primary\_key=**True**)

subjects\_id = Column(Integer, ForeignKey('subjects.subjects\_id'))

teachers\_id = Column(Integer, ForeignKey('teachers.teachers\_id'))

students\_id = Column(Integer, ForeignKey('students.students\_id'))

week\_day = Column(String)

subj = relationship("subjects", back\_populates="sched")

teach = relationship("teachers", back\_populates="sched")

stud = relationship("students", back\_populates="sched")

**class** Model:

**def** \_\_init\_\_(self):

self.session = Session()

self.connection = engine.connect()

@staticmethod

**def** insert\_data(table\_name, values):

**if** table\_name == 'subjects':

session.add(subjects(subjects\_id=values[0], name=values[1]))

session.commit()

**elif** table\_name == 'teachers':

session.add(teachers(teachers\_id=values[0], name=values[1],

home\_address=values[2], phone\_number=values[3],

email=values[4]))

session.commit()

**elif** table\_name == 'students':

session.add(students(students\_id=values[0], name=values[1],

home\_address=values[2], phone\_number=values[3],

email=values[4]))

session.commit()

**elif** table\_name == 'schedule':

session.add(schedule(schedule\_id=values[0], subjects\_id=values[1],

teachers\_id=values[2], students\_id=values[3],

week\_day=values[4]))

session.commit()

@staticmethod

**def** delete\_data(table\_name, del\_id):

**if** table\_name == 'subjects':

session.query(subjects).filter\_by(subjects\_id=del\_id).delete()

session.commit()

**elif** table\_name == 'teachers':

session.query(teachers).filter\_by(teachers\_id=del\_id).delete()

session.commit()

**elif** table\_name == 'students':

session.query(students).filter\_by(students\_id=del\_id).delete()

session.commit()

**elif** table\_name == 'schedule':

session.query(schedule).filter\_by(schedule\_id=del\_id).delete()

session.commit()

@staticmethod

**def** update\_data(table\_name, values):

**if** table\_name == 'subjects':

session.query(subjects).filter\_by(subjects\_id=values[0]).update({subjects.name: values[1]})

session.commit()

**elif** table\_name == 'teachers':

session.query(teachers).filter\_by(teachers\_id=values[0]).update({teachers.name: values[1],

teachers.home\_address: values[2],

teachers.phone\_number: values[3],

teachers.email: values[4]})

session.commit()

**elif** table\_name == 'students':

session.query(students).filter\_by(students\_id=values[0]).update({students.name: values[1],

students.home\_address: values[2],

students.phone\_number: values[3],

students.email: values[4]})

session.commit()

**elif** table\_name == 'schedule':

session.query(schedule).filter\_by(schedule\_id=values[0]).update({schedule.week\_day: values[1]})

session.commit()

view.py

**class** View:

@staticmethod

**def** show\_table(db):

i = 1

**for** element **in** db:

print(i, ')', element)

i += 1

@staticmethod

**def** show(db):

**for** element **in** db:

print(element)

@staticmethod

**def** display\_menu():

print('''

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1) Insert

2) Delete

3) Update

4) Exit

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_''')

@staticmethod

**def** display\_tables():

print('''

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1) subjects

2) teachers

3) students

4) schedule

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_''')

@staticmethod

**def** display\_subjects():

print('''

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

subjects\_id name''')

@staticmethod

**def** display\_teachers():

print('''

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

teachers\_id name home\_address phone\_number email''')

@staticmethod

**def** display\_students():

print('''

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

students\_id name home\_address phone\_number email''')

@staticmethod

**def** display\_schedule():

print('''

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

schedule\_id subjects\_id teachers\_id students\_id week\_day''')

@staticmethod

**def** display\_insert(table\_name, insert):

print('Successful insert', insert, 'in ', table\_name)

@staticmethod

**def** display\_delete(table\_name, delete):

print('Successful delete', delete, 'in ', table\_name)

@staticmethod

**def** display\_update(table\_name, update):

print('Successful update', update, 'in ', table\_name)

@staticmethod

**def** display\_generate(table\_name, count):

print('Successful generation of', count, 'data in ', table\_name)

controller.py

**import** **psycopg2**

**from** **view** **import** View

**from** **orm** **import** Model, session

**class** Controller:

**def** digit\_to\_table\_name():

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

number = input('Table >> ')

**if** str(number).isdigit():

**if** number == '1':

**return** 'subjects'

**elif** number == '2':

**return** 'teachers'

**elif** number == '3':

**return** 'students'

**elif** number == '4':

**return** 'schedule'

**else**:

print('Incorrect input')

self.digit\_to\_table\_name()

@staticmethod

**def** table\_name\_to\_pk(table\_name):

**if** table\_name == 'subjects':

**return** 'subjects\_id'

**elif** table\_name == 'teachers':

**return** 'teachers\_id'

**elif** table\_name == 'students':

**return** 'students\_id'

**elif** table\_name == 'schedule':

**return** 'schedule\_id'

**else**:

print('Incorrect input')

**return** ' '

@staticmethod

**def** get\_columns(table\_name):

**if** table\_name == 'subjects':

View.display\_subjects()

**elif** table\_name == 'teachers':

View.display\_teachers()

**elif** table\_name == 'students':

View.display\_students()

**elif** table\_name == 'schedule':

View.display\_schedule()

**else**:

print('Incorrect input')

**return** ' '

**def** insert(table\_name, values):

**try**:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

Model.insert\_data(table\_name, values)

View.display\_insert(table\_name, values)

**except** (**Exception**, psycopg2.Error) **as** error:

print("Failed to insert data to table", error)

**def** delete(table\_name, del\_id):

**try**:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

Model.delete\_data(table\_name, del\_id)

View.display\_delete(table\_name, del\_id)

**except** (**Exception**, psycopg2.Error) **as** error:

print("Failed to delete data in table", error)

**def** update(table\_name, values):

**try**:

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

Model.update\_data(table\_name, values)

View.display\_update(table\_name, values)

**except** (**Exception**, psycopg2.Error) **as** error:

print("Failed to update data in table", error)

**def** menu():

prog = **True**

**while** prog:

View.display\_menu()

choice = input('Menu >> ')

**if** choice == '1':

View.display\_tables()

table\_name = Controller.digit\_to\_table\_name()

print(table\_name)

Controller.get\_columns(table\_name)

values = input("Values >>").split(' ')

Controller.insert(table\_name, values)

**elif** choice == '2':

View.display\_tables()

table\_name = Controller.digit\_to\_table\_name()

print(table\_name)

Controller.get\_columns(table\_name)

del\_id = input("ID >> ")

**if** del\_id.isdigit():

Controller.delete(table\_name, del\_id)

**else**:

print("Input shoud be an integer!")

**elif** choice == '3':

View.display\_tables()

table\_name = Controller.digit\_to\_table\_name()

print(table\_name)

Controller.get\_columns(table\_name)

values = input("Values(not foreign keys) >>").split(' ')

Controller.update(table\_name,values)

**elif** choice == '4':

prog = **False**

**else**:

print('Incorrect input')

session.rollback()

main.py

**from** **orm** **import** Base, engine

**from** **controller** **import** Controller

**from** **sqlalchemy** **import** MetaData

metadata = MetaData()

Base.metadata.create\_all(engine)

Controller.menu()