**Національний технічний університет України**

**“Київський політехнічний інститут** імені Ігоря Сікорського**”**

**Факультет прикладної математики**

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

**Лабораторна робота №2**

***з дисципліни***

***“Бази даних та засоби управління”***

**ТЕМА: “ Засоби оптимізації роботи СУБД PostgreSQL”**

**Група: КВ-12**

**Виконала: Гнатюк С.В.**

**Оцінка:**

**Київ – 2023**

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

*Завдання* роботи полягає у наступному:

1. Перетворити модуль “Модель” з шаблону MVC РГР 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.

**Завдання за варіантом 5**

Види індексів: Btree, GIN

Умови для тригера: before update, delete

**Предметна галузь: Онлайн-сервіс для бронювання квитків на транспорт**

**Опис сутностей та зв’язків між ними**

Сутність User - користувач, який бронює квитки. Має char атрибути name (ім’я), surname (прізвище), phone (номер телефону) та key attribute user\_id (атрибут, значення якого не може повторюватися, оскільки він використовується для ідентифікації).

Сутність Vehicle - транспорт, на якому здійснюється поїздка. Має char атрибут vehicle\_type (тип транспорту, наприклад, автобус, поїзд тощо) та key attribute vehicle\_id.

Сутність Booking - бронювання. Має integer атрибут price (ціна поїздки), атрибут time типу time without timezone (час бронювання), key attribute booking\_id. Дана сутність пов’язана з двома іншими сутностями зв’язками 1:N, тобто один до багатьох. Один користувач може здійснювати багато бронювань, а один транспортний засіб можна бронювати багато разів. З іншого боку, одне бронювання може бути здійснене лише одним користувачем, і може стосуватися лише одного транспортного засобу.

Сутність staff – це працівник. Має ключовий атрибут staff\_id, character varying атрибути name (ім’я) та position (посада). Має 1:N зв’язок з сутністю Vehicle (транспорт), оскільки на одному транспортному засобі можуть працювати кілька людей (наприклад, водій, провідник тощо), але один працівник працює лише на 1 транспортному засобі.

Изображение выглядит как диаграмма, линия, оригами, шаблон

Автоматически созданное описание

Рисунок 1 - ER-діаграма за нотацією Crow’s foot

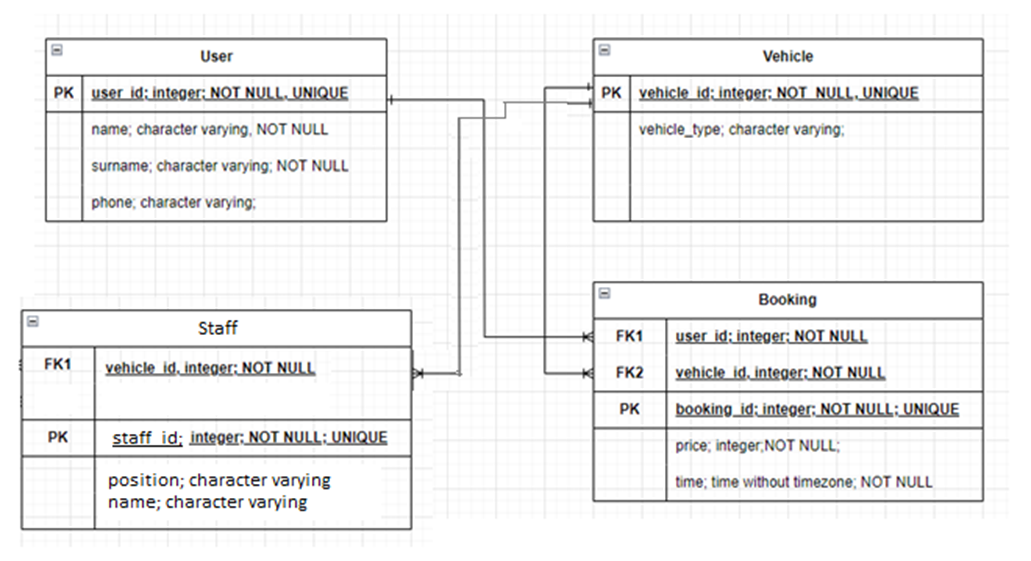


Рисунок 2 - Схема бази даних PostgreSQL на основі ER-моделі предметної галузі “Онлайн-сервіс бронювання квитків на транспорт”

**Мова програмування, середовище та компоненти розробки**

В даній роботі використано мову програмування Python, середовище розробки програмного забезпечення PyCharm Community Edition, середовище для відлагодження SQL-запитів до бази даних pgAdmin4, стороння бібліотека psycopg2 для роботи з базами даних PostgreSQl в Python, бібліотека для реалізації ORM SQLAlchemy.

**Завдання 1**

**Класи ORM**

class user(Base):

\_\_tablename\_\_ = 'user'

user\_id = Column(Integer, primary\_key=True)

phone = Column(String)

name = Column(String)

surname = Column(String)

Рисунок 4 - клас user (користувач)

class vehicle(Base):

\_\_tablename\_\_ = 'vehicle'

vehicle\_id = Column(Integer, primary\_key=True)

vehicle\_type = Column(String)

Рисунок 5 - клас vehicle (транспорт)

class staff(Base):

\_\_tablename\_\_ = 'staff'

staff\_id = Column(Integer, primary\_key=True)

position = Column(String)

name = Column(String)

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

Рисунок 6 - клас staff (працівник)

class booking(Base):

\_\_tablename\_\_ = 'booking'

booking\_id = Column(Integer, primary\_key=True)

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

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

price = Column(Integer)

booking\_time = Column(Time(timezone=False))

Рисунок 7 - клас booking (бронювання)

На рисунках 4-7 описані класи ORM (код з файлу Model.py), зокрема користувач, транспорт, працівник, бронювання. Клас працівник пов’язаний із класом транспорт зв’язком один до одного. Клас бронювання пов’язаний із класами користувач та транспорт зв’язками один до багатьох.

**Приклади запитів ORM**

def add\_vehicle(self, vehicle\_id, vehicle\_type):

s = Session()

check = s.query(exists().where(vehicle.vehicle\_id == vehicle\_id)).scalar()

if (check):

print("Error! Identifier already exists")

else:

v = vehicle(

vehicle\_id=vehicle\_id,

vehicle\_type=vehicle\_type

)

s.add(v)

s.commit()

print("Added successfully")

s.close()

Рисунок 8 - функція додавання даних до таблиці Транспорт

def update\_vehicle(self, vehicle\_id, vehicle\_t):

s = Session()

v = s.query(vehicle).filter\_by(vehicle\_id=vehicle\_id).first()

if v:

v.vehicle\_type = vehicle\_t

s.commit()

print("Catalog updated successfully!")

else:

print("Error! Catalog does not exist")

s.close()

Рисунок 9 - функція оновлення даних таблиці Транспорт

def delete\_vehicle(self, vehicle\_id):

with session\_scope() as s:

b = s.query(booking).filter\_by(vehicle\_id=vehicle\_id).all()

if b:

for i in b:

s.delete(i)

s.commit()

print("Deleted successfully!")

v = s.query(vehicle).get(vehicle\_id)

if v:

s.delete(v)

s.commit()

print("Deleted successfully!")

else:

print("Error! Vehicle doesn't exist")

Рисунок 10 - функція видалення даних із таблиці Транспорт

**Завдання 2**

Для використання індексу GIN встановимо тип даних в стопці “name” як tsvector в таблиці «User».

Використаємо дану функцію, щоб згенерувати випадкові дані:

def gener\_add\_user(self, num1, num2):

c = self.conn.cursor()

num = 0

for i in range(num1, num2 + 1):

c.execute('SELECT \* FROM "user" WHERE "user\_id" = %s', (i,))

check = c.fetchall()

if check:

print("user\_id %s already exists", i)

num = 1

if num == 0:

c.execute('INSERT INTO "user" ("user\_id", "phone" "name", "surname") SELECT generate\_series as "user\_id",to\_tsvector(chr(trunc(65+random()\*50)::int) || chr(trunc(65+random()\*25)::int)|| chr(trunc(65+random()\*25)::int)) as name, chr(trunc(65+random()\*25)::int) || chr(trunc(65+random()\*25)::int) as phone, chr(trunc(65+random()\*25)::int) || chr(trunc(65+random()\*25)::int) as surname FROM generate\_series(%s, %s)',(num1, num2))

self.conn.commit()

else:

print("Error! ID already in use")

Рисунок 11 - функція генерування випадкових даних для таблиці Користувач

За варіантом 5 необхідно використати такі види індексів: Btree, GIN.

**Запити без використання індекса Btree**

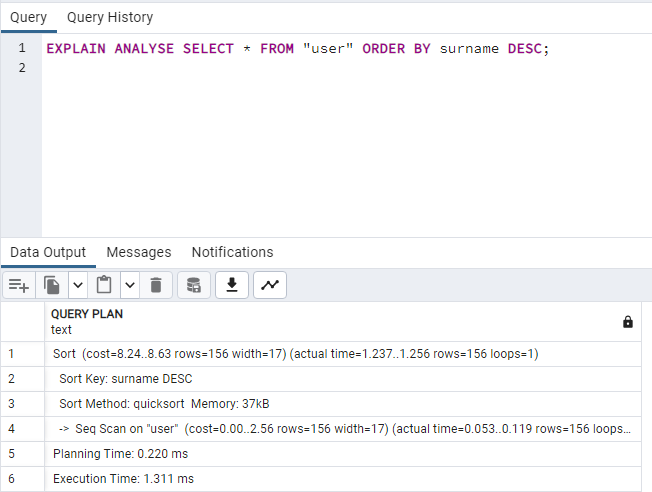


Рисунок 12 - запит на сортування без використання індексів

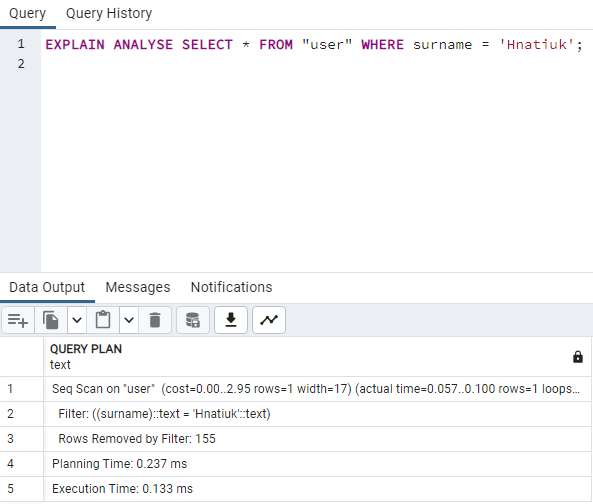


Рисунок 13 - запит на фільтрацію даних без використання індексу

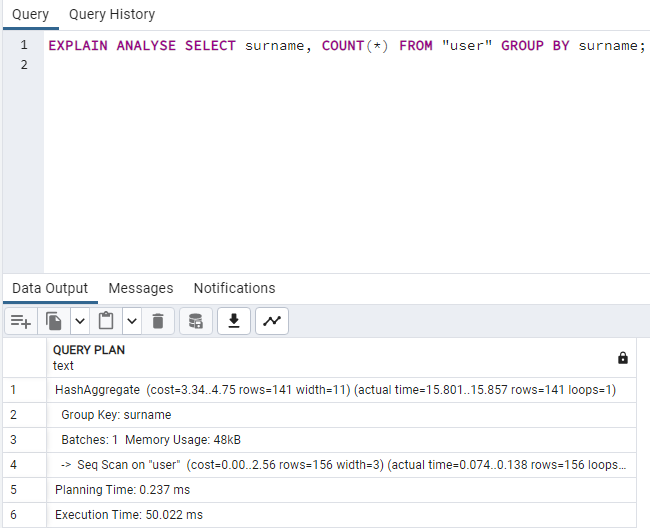


Рисунок 14 - запит на групування даних без використання індекса



Рисунок 15 - запит з агрегатними функціями без використання індекса

**Запити з використанням індекса Btree**

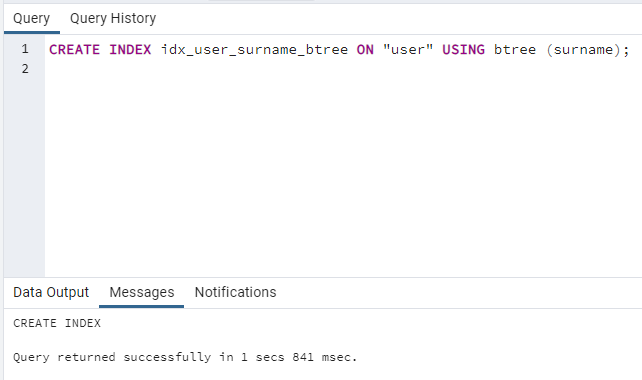


Рисунок 16 - створення індекса Btree

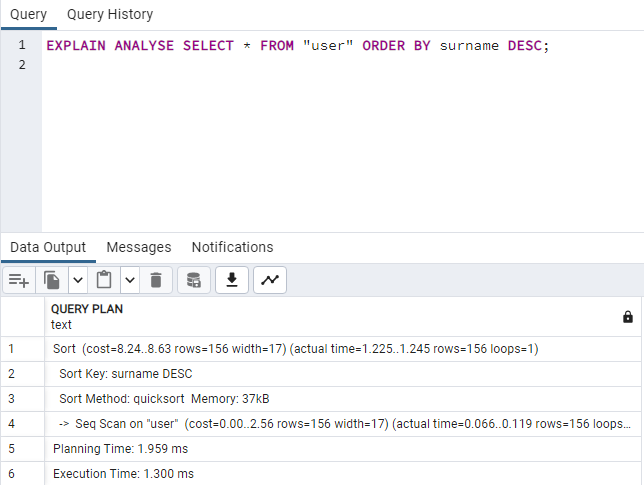


Рисунок 17 - запит на сортування з використанням індексу Btree

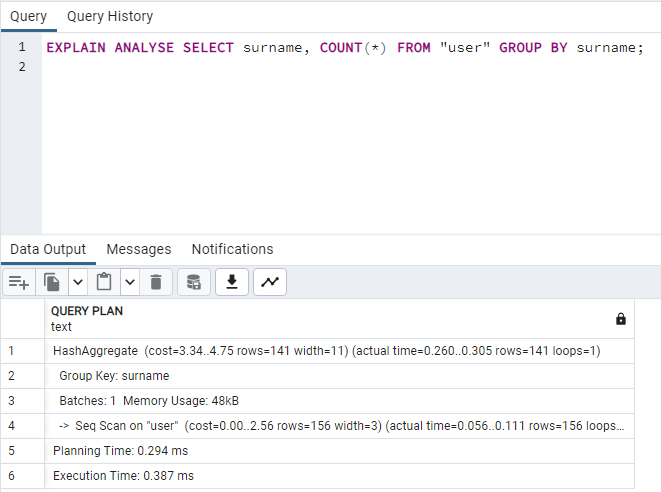


Рисунок 18 - запит на групування даних з використанням індекса Btree

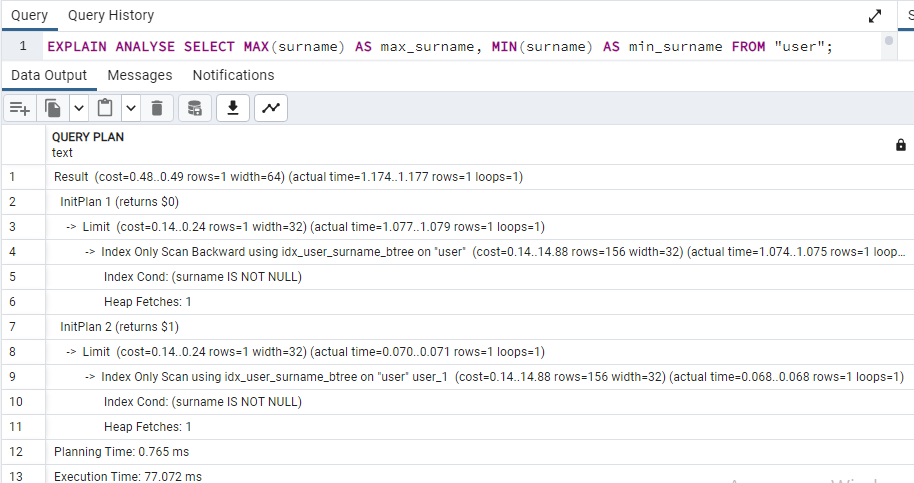


Рисунок 19 - запит з використанням агрегатних функцій та індекса Btree

**Index GIN**

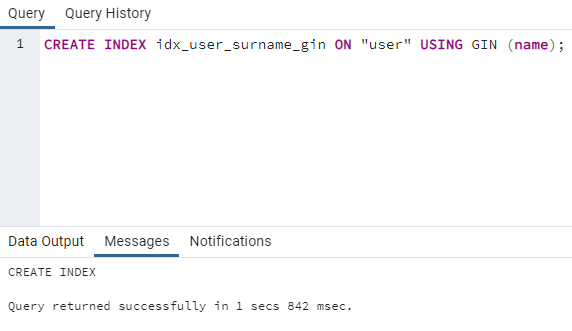


Рисунок 20 - створення індекса GIN

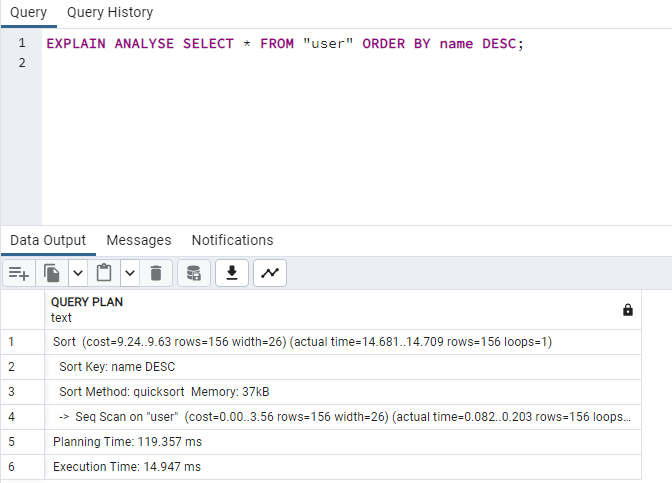


Рисунок 21 - запит на сортування

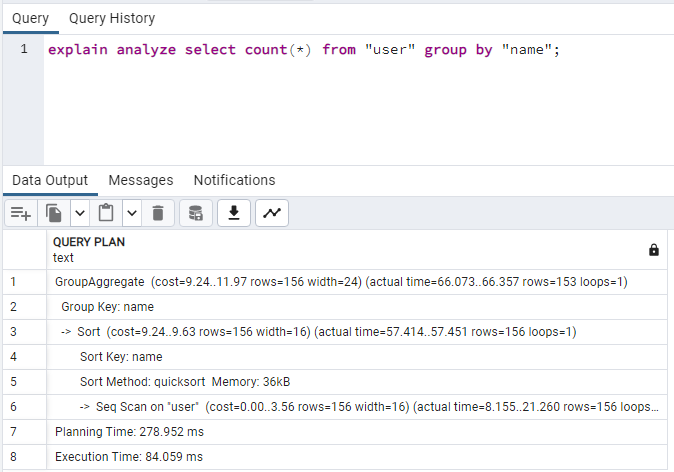


Рисунок 22 - запит з агрегатною функцією

**Завдання 3**

За умовою необхідно створити тригер баз даних з умовою before update, delete. Для цього створимо додаткові дві таблиці users\_1, users\_2. Заповнимо 150 рядків у першій таблиці, а потім скопіюємо всі ці дані у другу таблицю.

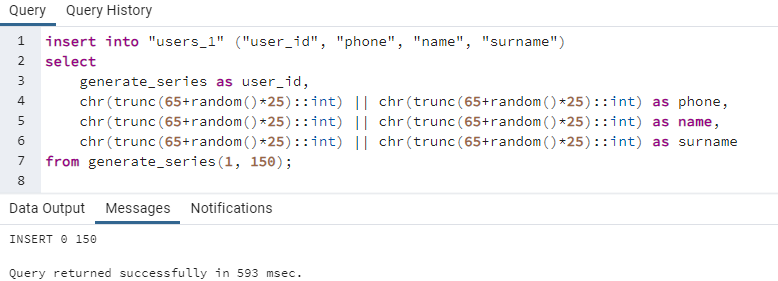


Рисунок 23 - заповнення таблиці users\_1

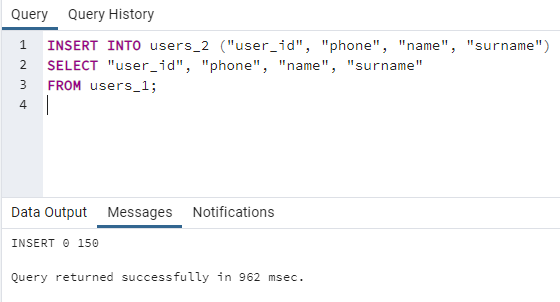


Рисунок 24 - заповнення таблиці users\_2

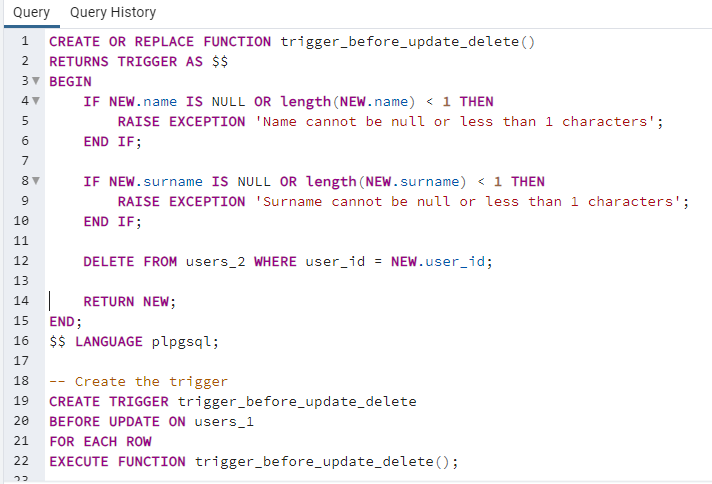


Рисунок 23 - створення тригера

Даний тригер видаляє рядок з таблиці users\_2 перед тим, як змінити відповідний рядок у таблиці users\_1. У випадку, якщо нові ім’я або прізвище користувача не відповідають вимогам, отримуємо повідомлення про помилку. Перевіримо, як працює цей тригер. Спробуємо змінити дані у першій таблиці:

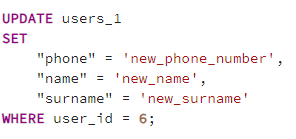


Рисунок 24 - зміна даних у users\_1

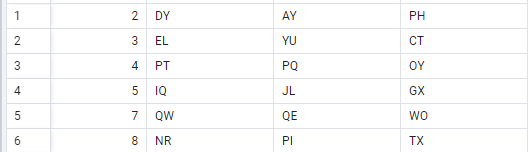


Рисунок 25 - фрагмент таблиці users\_2

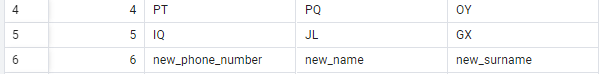


Рисунок 26 - фрагмент таблиці users\_1

Отже, перед оновленням даних першої таблиці дійсно відбулось видаленя відповідного рядка з другої таблиці.

Такий тригер може знадобитися у випадку, якщо потрібно зберігати таблицю з користувачами, дані яких ще ні разу не оновлювали.

**Завдання 4**

**Рівень ізоляції транзакцій READ COMMITTED**

При такому рівні ізоляції, транзакція повторно зчитує дані і виявляє зміни, здійснені іншою транзакцією з моменту зчитування.

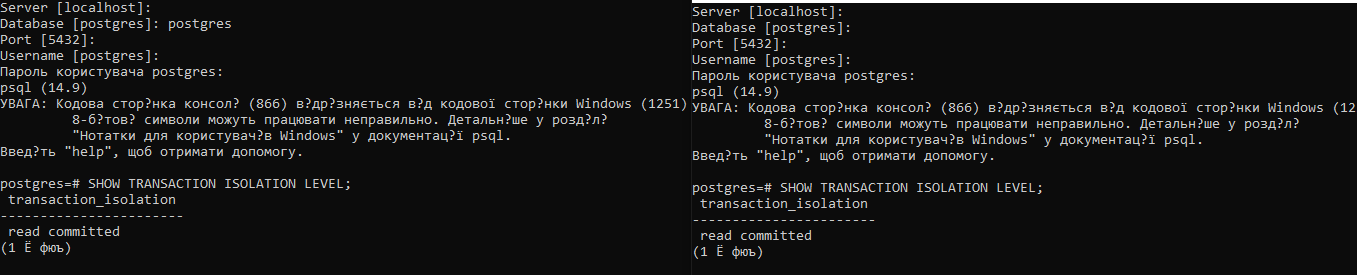


Рисунок 27 - рівень ізоляції READ COMMITTED

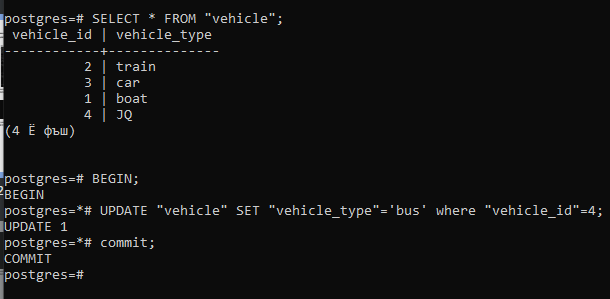


Рисунок 28 - зміна даних в таблиці “vehicle”

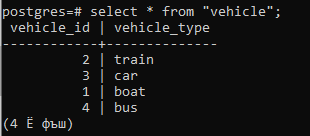


Рисунок 29 - результат зміни даних

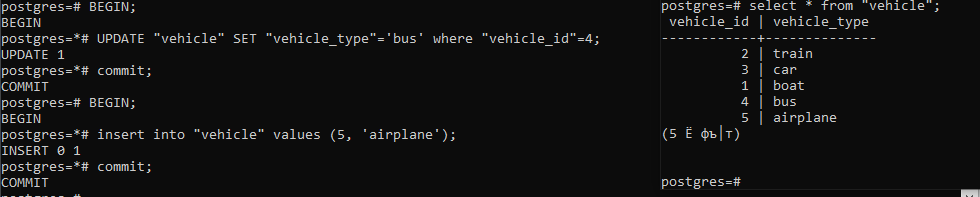


Рисунок 30 - вставка даних при рівні ізоляції READ COMMITTED

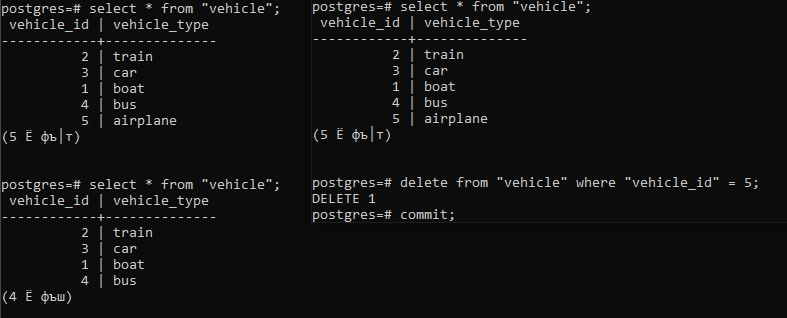


Рисунок 31 - видалення даних при ізоляції READ COMMITTED

**Рівень ізоляції REPEATABLE READ**

При рівні ізоляції REPEATABLE READ при повторному зчитуванні дані не змінюються після виконання запиту іншою транзакцією.

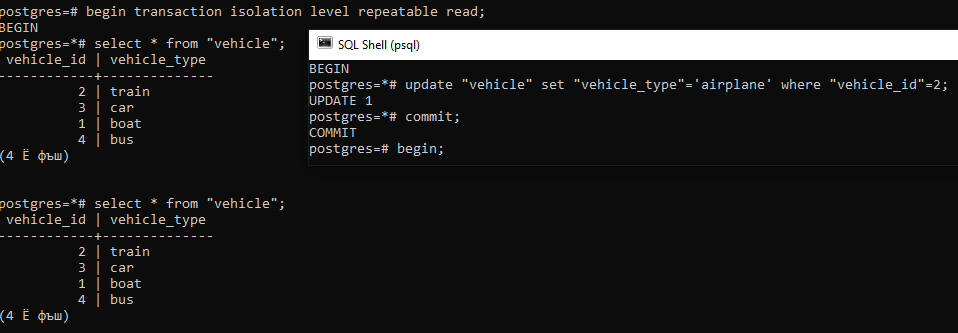


Рисунок 32 - запит на редагування даних при рівні ізоляції REPEATABLE READ

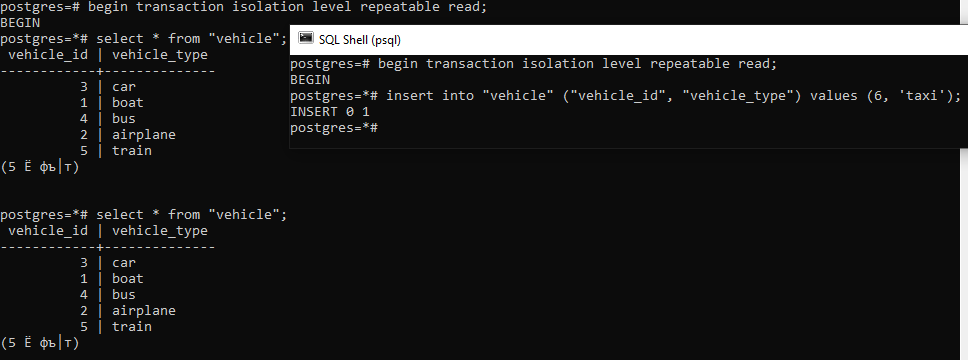


Рисунок 33 - запит на вставку даних при REPEATABLE READ

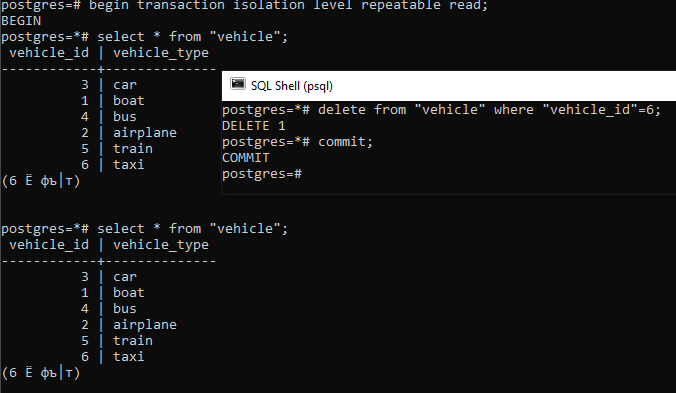


Рисунок 34 - запит на видалення даних при REPEATABLE READ

**Рівень ізоляції транзакції SERIALIZABLE**

При рівні ізоляції транзакцій SERIALIZABLE відбувається блокування запису і читання даних, якими ми працюємо на даний момент.

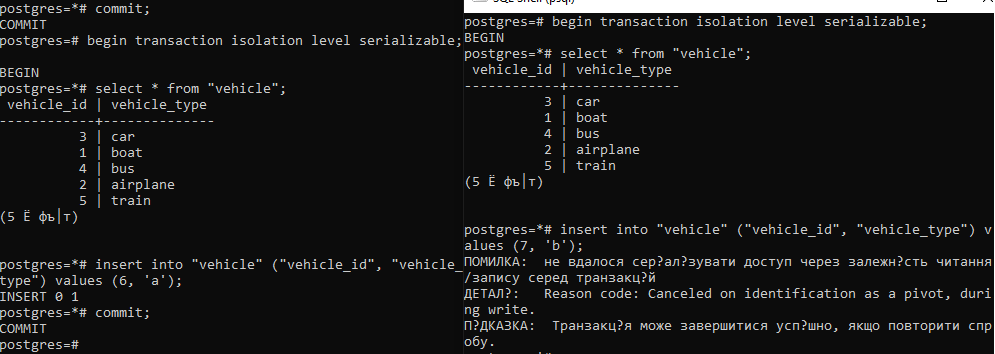


Рисунок 35 - запит на вставку даних при рівні ізоляції SERIALIZABLE

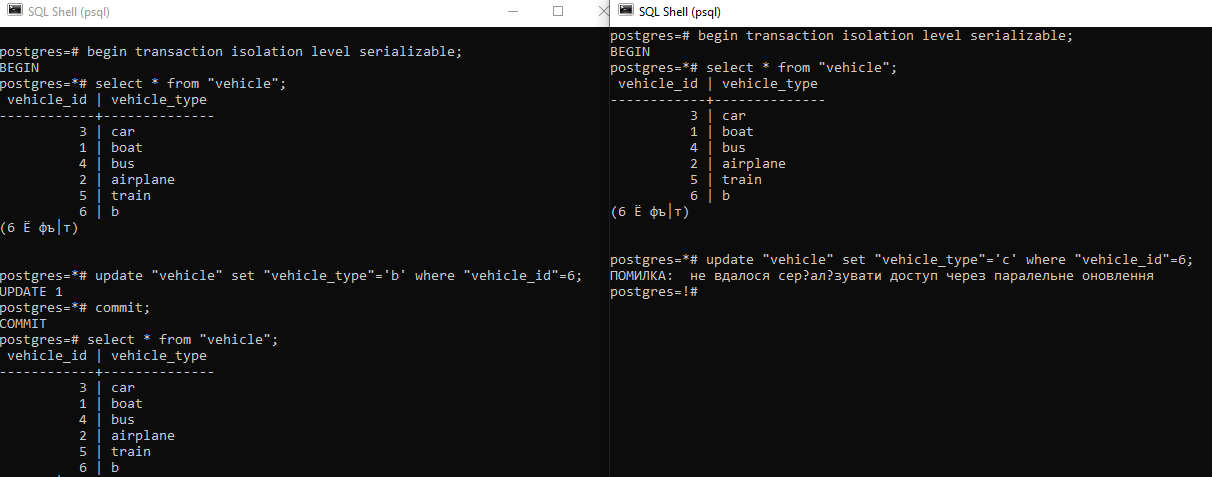


Рисунок 36 - запит на редагування даних при SERIALIZABLE

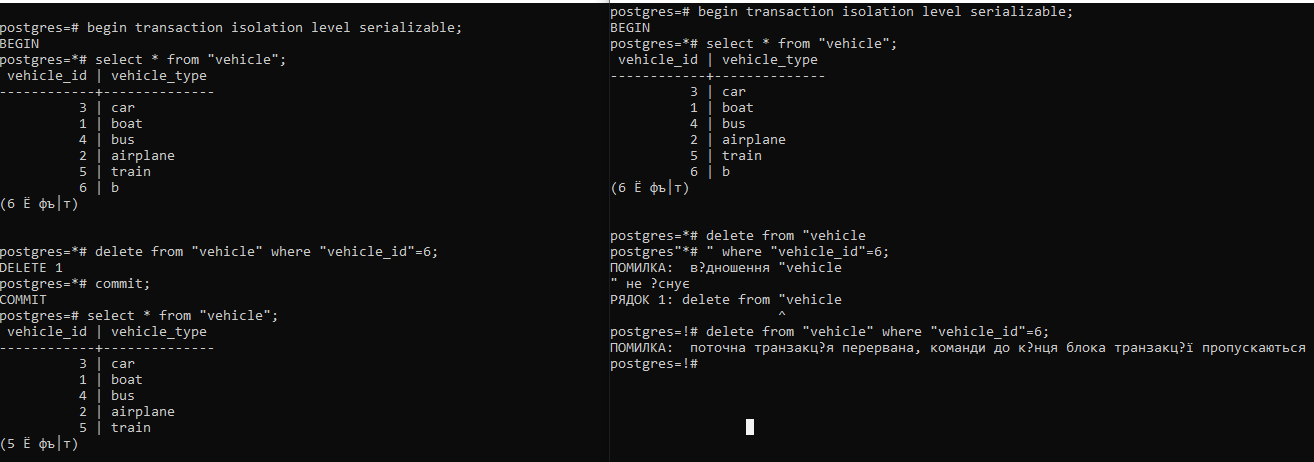


Рисунок 37 - запит на видалення даних при SERIALIZABLE

**Повний код програми**

**Main.py**

from Controller import Controller  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 controller = Controller()  
 controller.run()

**Model.py**

import sqlalchemy  
from sqlalchemy import create\_engine  
from sqlalchemy import MetaData, Table, Column, Integer, String, ForeignKey, Time  
import time  
from sqlalchemy.orm import declarative\_base  
from sqlalchemy.orm import relationship, backref, sessionmaker  
from contextlib import contextmanager  
from sqlalchemy import exists  
import psycopg2  
  
  
engine = create\_engine('postgresql://postgres:2004@localhost:5432/postgres')  
  
Session = sessionmaker(bind=engine)  
  
@contextmanager  
def session\_scope():  
 session = Session()  
 try:  
 yield session  
 session.commit()  
 except Exception:  
 session.rollback()  
 raise  
 finally:  
 session.close()  
Base = declarative\_base()  
class user(Base):  
 \_\_tablename\_\_ = 'user'  
 user\_id = Column(Integer, primary\_key=True)  
 phone = Column(String)  
 name = Column(String)  
 surname = Column(String)  
  
class vehicle(Base):  
 \_\_tablename\_\_ = 'vehicle'  
 vehicle\_id = Column(Integer, primary\_key=True)  
 vehicle\_type = Column(String)  
  
  
class staff(Base):  
 \_\_tablename\_\_ = 'staff'  
 staff\_id = Column(Integer, primary\_key=True)  
 position = Column(String)  
 name = Column(String)  
 vehicle\_id = Column(Integer, ForeignKey('vehicle.vehicle\_id'))  
  
  
class booking(Base):  
 \_\_tablename\_\_ = 'booking'  
 booking\_id = Column(Integer, primary\_key=True)  
 user\_id = Column(Integer, ForeignKey('user.user\_id'))  
 vehicle\_id = Column(Integer, ForeignKey('vehicle.vehicle\_id'))  
 price = Column(Integer)  
 booking\_time = Column(Time(timezone=False))  
  
class Model():  
  
 def \_\_init\_\_(self):  
 self.conn = psycopg2.connect(  
 dbname='postgres',  
 user='postgres',  
 password='2004',  
 host='localhost',  
 port=5432  
 )  
  
 def gener\_add\_user(self, num1, num2):  
 c = self.conn.cursor()  
 num = 0  
 for i in range(num1, num2 + 1):  
 c.execute('SELECT \* FROM "user" WHERE "user\_id" = %s', (i,))  
 check = c.fetchall()  
 if check:  
 print("user\_id %s already exists", i)  
 num = 1  
  
 if num == 0:  
 c.execute('INSERT INTO "user" ("user\_id", "phone", "name", "surname") SELECT generate\_series as "user\_id", chr(trunc(65+random()\*25)::int) || chr(trunc(65+random()\*25)::int) as phone, to\_tsvector(chr(trunc(65+random()\*50)::int) || chr(trunc(65+random()\*25)::int)|| chr(trunc(65+random()\*25)::int)) as name, chr(trunc(65+random()\*25)::int) || chr(trunc(65+random()\*25)::int) as surname FROM generate\_series(%s, %s)',(num1, num2))  
  
 self.conn.commit()  
 else:  
 print("Error! ID already in use")  
  
 def get\_all\_vehicle(self):  
 s = Session()  
 return s.query(vehicle).all()  
  
 def add\_vehicle(self, vehicle\_id, vehicle\_type):  
 s = Session()  
 check = s.query(exists().where(vehicle.vehicle\_id == vehicle\_id)).scalar()  
 if (check):  
 print("Error! Identifier already exists")  
 else:  
 v = vehicle(  
 vehicle\_id=vehicle\_id,  
 vehicle\_type=vehicle\_type  
 )  
 s.add(v)  
 s.commit()  
 print("Added successfully")  
 s.close()  
  
 def update\_vehicle(self, vehicle\_id, vehicle\_t):  
 s = Session()  
 v = s.query(vehicle).filter\_by(vehicle\_id=vehicle\_id).first()  
 if v:  
 v.vehicle\_type = vehicle\_t  
 s.commit()  
 print("Catalog updated successfully!")  
 else:  
 print("Error! Catalog does not exist")  
 s.close()  
  
 def delete\_vehicle(self, vehicle\_id):  
 with session\_scope() as s:  
 b = s.query(booking).filter\_by(vehicle\_id=vehicle\_id).all()  
 if b:  
 for i in b:  
 s.delete(i)  
 s.commit()  
 print("Deleted successfully!")  
 v = s.query(vehicle).get(vehicle\_id)  
 if v:  
 s.delete(v)  
 s.commit()  
 print("Deleted successfully!")  
 else:  
 print("Error! Vehicle doesn't exist")  
  
 def add\_staff(self, staff\_id, vehicle\_id, name, position):  
 s = Session()  
 check = s.query(exists().where(staff.staff\_id == staff\_id)).scalar()  
 if (check):  
 print("Error! Identifier already exists")  
 else:  
 check1 = s.query(exists().where(vehicle.vehicle\_id == vehicle\_id)).scalar()  
 if (check1):  
 st = staff(  
 staff\_id=staff\_id,  
 vehicle\_id=vehicle\_id,  
 name = name,  
 position = position  
 )  
 s.add(st)  
 s.commit()  
 print("Added successfully!")  
 s.close()  
  
 def update\_staff(self, staff\_id, position, name, vehicle\_id):  
 s = Session()  
 st = s.query(staff).filter\_by(staff\_id=staff\_id).first()  
 v = s.query(vehicle).filter\_by(vehicle\_id=vehicle\_id).first()  
 if st and v:  
 st.vehicle\_id = vehicle\_id  
 st.position = position  
 st.name = name  
 s.commit()  
 print("Updated successfully!")  
 else:  
 print("Error! Vehicle does not exist")  
 s.close()  
  
 def delete\_staff(self, staff\_id):  
 with session\_scope() as s:  
 st = s.query(staff).get(staff\_id)  
 if st:  
 s.delete(st)  
 s.commit()  
 print("Deleted successfully!")  
 else:  
 print("Error! Does not exist")  
  
 def get\_all\_staff(self):  
 s = Session()  
 return s.query(staff).all()  
  
  
 def add\_user(self, user\_id, phone, name, surname):  
 s = Session()  
 check = s.query(exists().where(user.user\_id == user\_id)).scalar()  
 if (check):  
 print("Error! Identifier already exists")  
 else:  
 u = user(  
 user\_id=user\_id,  
 phone = phone,  
 name=name,  
 surname=surname  
 )  
 s.add(u)  
 s.commit()  
 print("Added successfully!")  
 s.close()  
  
 def update\_user(self, user\_id, phone, name, surname):  
 s = Session()  
 u = s.query(user).filter\_by(user\_id=user\_id).first()  
 if u:  
 u.name = name  
 u.phone = phone  
 u.surname = surname  
 s.commit()  
 print("Updated successfully!")  
 else:  
 print("Error! User does not exist")  
  
 s.close()  
  
 def delete\_user(self, user\_id):  
 with session\_scope() as s:  
 u = s.query(user).get(user\_id)  
 b = s.query(booking).filter\_by(user\_id=user\_id).all()  
  
  
 if u and b:  
 for book in b:  
 s.delete(book)  
 s.commit()  
 print("User deleted from booking")  
 if u:  
 s.delete(u)  
 s.commit()  
 print("User deleted!")  
 else:  
 print(f"Error! User does not exist")  
  
 def get\_all\_user(self):  
 s = Session()  
 return s.query(user).all()  
  
 def add\_booking(self, booking\_id, user\_id, vehicle\_id, price, booking\_time):  
 s = Session()  
 check = s.query(exists().where(booking.booking\_id == booking\_id)).scalar()  
 if (check):  
 print("Error! Identifier already exists")  
 else:  
 check1 = s.query(exists().where(vehicle.vehicle\_id == vehicle\_id)).scalar()  
 check2 = s.query(exists().where(user.user\_id == user\_id)).scalar()  
 if check1 and check2:  
 b = booking(  
 booking\_id=booking\_id,  
 vehicle\_id=vehicle\_id,  
 price = price,  
 booking\_time =booking\_time,  
 user\_id=user\_id  
 )  
 s.add(b)  
 s.commit()  
 print("Booking added successfully!")  
 else:  
 print("Error! Identifier does not exist")  
 s.close()  
  
 def get\_all\_booking(self):  
 s = Session()  
 return s.query(booking).all()  
  
 def update\_booking(self, booking\_id, user\_id, vehicle\_id, price, booking\_time):  
 s = Session()  
 b = s.query(booking).filter\_by(booking\_id=booking\_id).first()  
 v = s.query(vehicle).filter\_by(vehicle\_id=vehicle\_id).first()  
 u = s.query(user).filter\_by(user\_id=user\_id).first()  
 if b and v and u:  
 b.vehicle\_id = vehicle\_id  
 b.user\_id = user\_id  
 b.price = price  
 b.booking\_time = booking\_time  
 s.commit()  
 print("Booking updated successfully!")  
 else:  
 print("Error! Booking does not exist")  
 s.close()  
  
  
 def delete\_booking(self, booking\_id):  
 s = Session()  
 b = s.query(booking).get(booking\_id)  
 if b:  
 s.delete(b)  
 s.commit()  
 print("Booking deleted successfully!")  
 else:  
 print("Error! Booking does not exist")

**Controller.py**

from Model import Model  
from View import View  
  
class Controller:  
 def \_\_init\_\_(self):  
 self.model = Model()  
 self.view = View()  
  
 def run(self):  
 while True:  
 category = self.show\_menu()  
 if category == '1':  
 self.run\_add()  
 elif category == '2':  
 self.run\_view()  
 elif category == '3':  
 self.run\_update()  
 elif category == '4':  
 self.run\_delete()  
 elif category == '5':  
 self.gener\_user()  
 elif category == '6':  
 break  
  
 def show\_menu(self):  
 self.view.show\_message("\nMenu:")  
 self.view.show\_message("1. Add")  
 self.view.show\_message("2. View")  
 self.view.show\_message("3. Update")  
 self.view.show\_message("4. Delete")  
 self.view.show\_message("5. Generate new users")  
 self.view.show\_message("6. Quit")  
 return input("Enter your choice: ")  
  
  
 def menu(self):  
 self.view.show\_message("1. Vehicle")  
 self.view.show\_message("2. Staff")  
 self.view.show\_message("3. Booking")  
 self.view.show\_message("4. User")  
 self.view.show\_message("5. Quit")  
 return input("Enter your choice: ")  
  
 def run\_view(self):  
 print("\nView:")  
 while True:  
 category2 = self.menu()  
 if category2 == '1':  
 self.view\_vehicle\_al()  
 elif category2 == '2':  
 self.view\_staff\_al()  
 elif category2 == '3':  
 self.view\_booking\_al()  
 elif category2 == '4':  
 self.view\_user\_al()  
 elif category2 == '5':  
 break  
  
 def run\_add(self):  
 self.view.show\_message("\nAdd:")  
 while True:  
 category1 = self.menu()  
 if category1 == '1':  
 self.add\_vehicle()  
 elif category1 == '2':  
 self.add\_staff()  
 elif category1 == '3':  
 self.add\_booking()  
 elif category1 == '4':  
 self.add\_user()  
 elif category1 == '5':  
 break  
  
 def run\_update(self):  
 self.view.show\_message("\nUpdate:")  
 while True:  
 category3 = self.menu()  
 if category3 == '1':  
 self.update\_vehicle()  
 elif category3 == '2':  
 self.update\_staff()  
 elif category3 == '3':  
 self.update\_booking()  
 elif category3 == '4':  
 self.update\_user()  
 elif category3 == '5':  
 break  
  
 def run\_delete(self):  
 self.view.show\_message("\nDelete:")  
 while True:  
 category4 = self.menu()  
 if category4 == '1':  
 self.delete\_vehicle()  
 elif category4 == '2':  
 self.delete\_staff()  
 elif category4 == '3':  
 self.delete\_booking()  
 elif category4 == '4':  
 self.delete\_user()  
 elif category4 == '5':  
 break  
  
 def gener\_user(self):  
 num1, num2 = self.view.input\_num()  
 self.model.gener\_add\_user(num1, num2)  
 print("Added users successfully!")  
  
 def add\_vehicle(self):  
 vehicle\_type = self.view.get\_vehicle\_input()  
 vehicle\_id = self.view.get\_vehicle\_id()  
 if vehicle\_id.isdigit():  
 self.model.add\_vehicle(vehicle\_id, vehicle\_type)  
 else:  
 print("Error! Wrong input")  
  
 def update\_vehicle(self):  
 vehicle\_id = self.view.get\_vehicle\_id()  
 vehicle\_type = self.view.get\_vehicle\_input()  
 if vehicle\_id.isdigit():  
 self.model.update\_vehicle(vehicle\_id,vehicle\_type)  
 else:  
 print("Error! Wrong input")  
  
  
 def view\_vehicle\_al(self):  
 v = self.model.get\_all\_vehicle()  
 self.view.show\_vehicle\_alchemy(v)  
  
 def delete\_vehicle(self):  
 vehicle\_id = self.view.get\_vehicle\_id()  
 if vehicle\_id.isdigit():  
 self.model.delete\_vehicle(vehicle\_id)  
 else:  
 print("Error! Wrong input")  
  
 def add\_staff(self):  
 position, name, vehicle\_id = self.view.get\_staff\_input()  
 staff\_id = self.view.get\_staff\_id()  
 if staff\_id.isdigit() and vehicle\_id.isdigit():  
 self.model.add\_staff(staff\_id, position, name, vehicle\_id)  
 else:  
 print("Error! Wrong input")  
  
 def view\_staff\_al(self):  
 s = self.model.get\_all\_staff()  
 self.view.show\_staff\_alchemy(s)  
  
 def update\_staff(self):  
 staff\_id = self.view.get\_staff\_id()  
 position, name, vehicle\_id = self.view.get\_staff\_input()  
 if staff\_id.isdigit() and vehicle\_id.isdigit():  
 self.model.update\_notes(staff\_id, position, name, vehicle\_id)  
 else:  
 print("Error! Wrong input")  
  
 def delete\_staff(self):  
 staff\_id = self.view.get\_staff\_id()  
 if staff\_id.isdigit():  
 self.model.delete\_notes(staff\_id)  
 else:  
 print("Error! Wrong input")  
  
 def add\_user(self):  
 phone, name, surname = self.view.get\_user\_input()  
 user\_id = self.view.get\_user\_id()  
 if user\_id.isdigit():  
 self.model.add\_user(user\_id, phone, name, surname)  
 else:  
 print("Error! Wrong input")  
  
 def view\_user\_al(self):  
 users = self.model.get\_all\_user()  
 self.view.show\_user\_alchemy(users)  
  
 def update\_user(self):  
 user\_id = self.view.get\_user\_id()  
 phone, name, surname = self.view.get\_user\_input()  
 if user\_id.isdigit() and name.isalpha() and surname.isalpha():  
 self.model.update\_user(user\_id, phone, name, surname)  
 self.view.show\_message("Updated successfully!")  
 else:  
 print("Error! Wrong input")  
  
 def delete\_user(self):  
 user\_id = self.view.get\_user\_id()  
 if user\_id.isdigit():  
 self.model.delete\_user(user\_id)  
 else:  
 print("Error! Wrong input")  
  
 def add\_booking(self):  
 booking\_id = self.view.get\_booking\_id()  
 user\_id, vehicle\_id, price, booking\_time = self.view.get\_booking\_input()  
 if booking\_id.isdigit() and vehicle\_id.isdigit() and user\_id.isdigit():  
 self.model.add\_booking(booking\_id, user\_id, vehicle\_id, price, booking\_time)  
 else:  
 print("Error! Incorrect id")  
  
 def view\_booking\_al(self):  
 b = self.model.get\_all\_booking()  
 self.view.show\_booking\_alchemy(b)  
  
 def update\_booking(self):  
 booking\_id = self.view.get\_booking()  
 user\_id, vehicle\_id, price, booking\_time = self.view.get\_booking\_input()  
 if booking\_id.isdigit() and vehicle\_id.isdigit() and user\_id.isdigit():  
 self.model.update\_save\_notes(booking\_id, user\_id, vehicle\_id, price, booking\_time)  
 else:  
 print("Error! Incorrect id")  
  
 def delete\_booking(self):  
 booking\_id = self.view.get\_booking\_id()  
 if booking\_id.isdigit():  
 self.model.delete\_booking(booking\_id)  
 else:  
 print("Error! Incorrect id.")

**View.py**

class View:  
 def input\_num(self):  
 num1 = int(input("Enter number 1: "))  
 num2 = int(input("Enter number 2: "))  
 return num1, num2  
  
 # catalog alchemy  
 def show\_vehicle\_alchemy(self, v):  
 if v:  
 print("vehicles:")  
 for i in v:  
 print(f"Vehicle\_id: {i.vehicle\_id}, vehicle type: {i.vehicle\_type}")  
 else:  
 print("No vehicle found.")  
  
 def show\_vehicle(self, v):  
 if v:  
 print("vehicles:")  
 for i in v:  
 print(f"Vehicle\_id: {i[0]}, vehicle type: {i[1]}")  
 else:  
 print("No vehicle found.")  
  
 def get\_vehicle\_input(self):  
 vehicle\_type = input("Enter vehicle type: ")  
 return vehicle\_type  
  
 def get\_vehicle\_id(self):  
 vehicle\_id = input("Enter vehicle ID: ")  
 return vehicle\_id  
  
 def show\_staff\_alchemy(self, s1):  
 if s1:  
 print("Staff:")  
 for s in s1:  
 print(f"Staff id: {s.staff\_id}, Vehicle id: {s.vehicle\_id}, position: {s.position}, name: {s.name}")  
 else:  
 print("No staff found.")  
  
 def show\_staff\_(self, s\_1):  
 if s\_1:  
 print("Staff:")  
 for s in s\_1:  
 print(f"Staff ID: {s[0]}, position: {s[1]}, name: {s[2]}, vehicle id: {s[3]}")  
 else:  
 print("No staff found.")  
  
 def get\_staff\_input(self):  
 position = input("Enter position:")  
 vehicle\_id = input("Enter vehicle ID: ")  
 name = input("Enter staff name: ")  
 return position, name, vehicle\_id  
  
 def get\_staff\_id(self):  
 staff\_id = input("Enter staff ID: ")  
 return staff\_id  
  
 def show\_user\_alchemy(self, users):  
 if users:  
 print("Users:")  
 for User in users:  
 print(f"ID: {User.user\_id}, Phone: {User.phone}, Name: {User.name}, Surname: {User.surname}")  
 else:  
 print("No users found")  
  
 def show\_user(self, users):  
 if users:  
 print("Users:")  
 for User in users:  
 print(f"ID: {User[0]}, Phone: {User[1]}, Name: {User[2]}, Surname: {User[3]}")  
 else:  
 print("No users found")  
  
 def get\_user\_input(self):  
 Phone = input("Enter phone number:")  
 Name = input("Enter user Name: ")  
 Surname = input("Enter user Surname: ")  
 return Phone, Name, Surname  
  
 def get\_user\_id(self):  
 user\_id = input("Enter user ID: ")  
 return user\_id  
  
 def show\_booking\_alchemy(self, b):  
 if b:  
 print("Bookings:")  
 for i in b:  
 print(  
 f"Booking ID: {i.booking\_id}, User id: {i.user\_id}, Vehicle id: {i.vehicle\_id}, price: {i.price}, Booking time: {i.booking\_time}")  
 else:  
 print("No booking found.")  
  
 def show\_booking\_(self, b):  
 if b:  
 print("Bookings:")  
 for i in b:  
 print(  
 f"booking id: {i[0]}, user id: {i[1]}, vehicle id: {i[2]}, price: {i[3]}, booking time: {i[4]}")  
 else:  
 print("No booking found.")  
  
 def get\_booking\_input(self):  
 vehicle\_id = input("Enter vehicle ID: ")  
 price = input("Enter price: ")  
 user\_id = input("Enter user ID: ")  
 booking\_time = input("Enter booking time: ")  
 return user\_id, vehicle\_id, price, booking\_time  
  
 def get\_booking\_id(self):  
 booking\_id = input("Enter booking ID: ")  
 return booking\_id  
  
 def show\_message(self, message):  
 print(message)