

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**»

Виконав: Шевченко Олександр

Студент групи КВ-93

Перевірив: Павловський В.І.

**Київ 2021**

**Лабораторна робота №3**

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

Загальне 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.

|  |  |  |
| --- | --- | --- |
| № варіанта | Види індексів | Умови для тригера |
| 25 | BTree, GIN | before update, delete |

Логічна модель магазину «Автосалон»

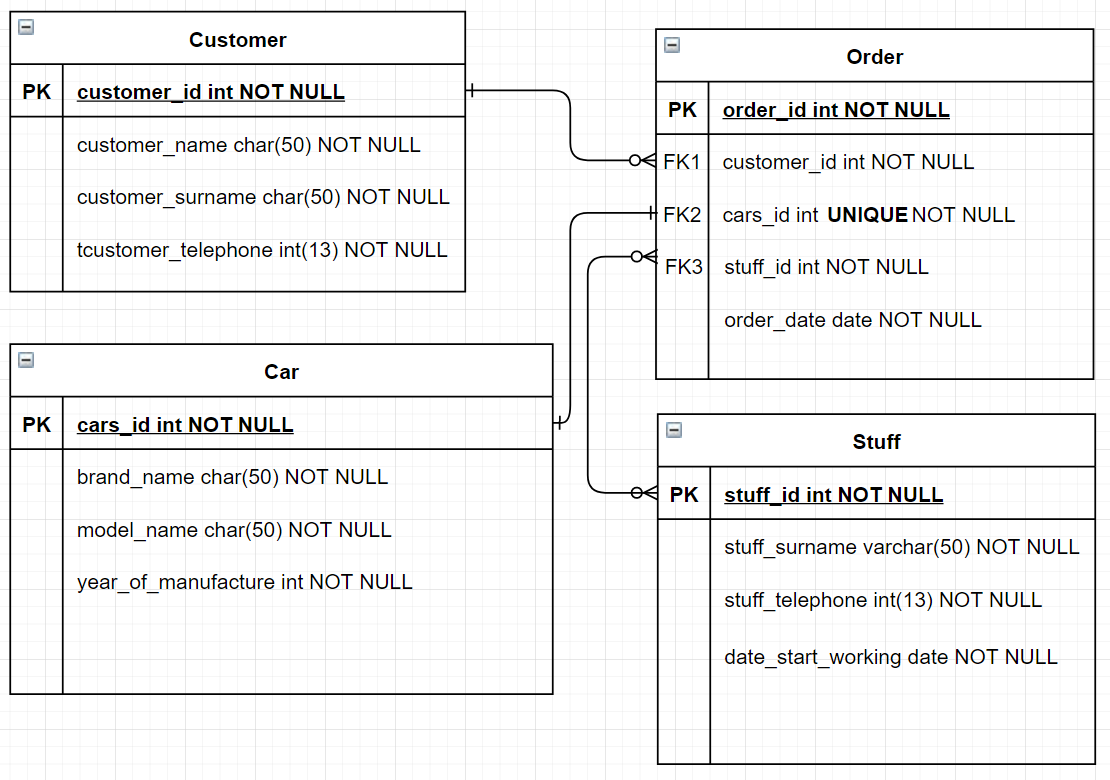


Рисунок0 – логічна модель «Автосалон»

**Середовище розробки**

Середовище розробки бази даних - PostgreSQL

Середовище розробки програми – Visual Studio. Мова програмування Python3.6.

Використані бібліотеки: бібліотека для реалізації ORM – SQLAlchemy(за рекомендацією лектора)

Посилання на репозиторій у GitHub з вихідним кодом програми та звітом: https://github.com/OleksandrShevchenkoKV93/DataBase

**Фрагменти програм внесення, редагування та вилучення даних у базі даних**

def insert\_data\_order(self, order\_id: int, customer\_id: int, car\_id: int, stuff\_id: int, order\_date: str) -> None:

order = Order(order\_id=order\_id, customer\_id=customer\_id, car\_id=car\_id, stuff\_id=stuff\_id, order\_date=order\_date)

self.session.add(order)

self.session.commit()

def insert\_data\_customer(self, customer\_id: int, customer\_name: str, customer\_surname: str, customer\_telephone: str) -> None:

customer = Customer(customer\_id=customer\_id, customer\_name=customer\_name, customer\_surname=customer\_surname, customer\_telephone=customer\_telephone)

self.session.add(customer)

self.session.commit()

def insert\_data\_car(self, car\_id: int, brand\_name: str, model\_name: str, year\_of\_manufacture: str) -> None:

car = Car(car\_id=car\_id, brand\_name=brand\_name, model\_name=model\_name, year\_of\_manufacture=year\_of\_manufacture)

self.session.add(car)

self.session.commit()

def insert\_data\_stuff(self, stuff\_id: int, stuff\_surname: str, date\_start\_working: str, stuff\_telephone: str) -> None:

stuff = Stuff(stuff\_id=stuff\_id, stuff\_surname=stuff\_surname, date\_start\_working=date\_start\_working,stuff\_telephone=stuff\_telephone)

self.session.add(stuff)

self.session.commit()

Фрагменти програми для внесення даних у таблицях

def update\_data\_order(self, order\_id: int, customer\_id: int, car\_id: int, stuff\_id: int,order\_date: str) -> None:

self.session.query(Order).filter\_by(order\_id=order\_id) \

.update({Order.customer\_id: customer\_id, Order.car\_id: car\_id,

Order.stuff\_id: stuff\_id,Order.order\_date:order\_date})

self.session.commit()

def update\_data\_customer(self, customer\_id: int, customer\_name: str, customer\_surname: str, customer\_telephone: str) -> None:

self.session.query(Customer).filter\_by(customer\_id=customer\_id) \

.update({Customer.customer\_name: customer\_name, Customer.customer\_surname: customer\_surname, Customer.customer\_telephone: customer\_telephone})

self.session.commit()

def update\_data\_car(self, car\_id: int, brand\_name: str, model\_name: str, year\_of\_manufacture: int) -> None:

self.session.query(Car).filter\_by(car\_id=car\_id) \

.update({Car.brand\_name: brand\_name, Car.model\_name: model\_name, Car.year\_of\_manufacture: year\_of\_manufacture})

self.session.commit()

def update\_data\_stuff(self, stuff\_id: int, stuff\_surname: str, date\_start\_working: str ,stuff\_telephone: str) -> None:

self.session.query(Stuff).filter\_by(stuff\_id=stuff\_id) \

.update({Stuff.stuff\_surname: stuff\_surname, Stuff.date\_start\_working: date\_start\_working, Stuff.stuff\_telephone: stuff\_telephone})

self.session.commit()

Фрагмент програми для редагування даних у таблицях

def delete\_data\_order(self, order\_id) -> None:

self.session.query(Order).filter\_by(order\_id=order\_id).delete()

self.session.commit()

def delete\_data\_customer(self, customer\_id) -> None:

self.session.query(Customer).filter\_by(customer\_id=customer\_id).delete()

self.session.commit()

def delete\_data\_car(self, car\_id) -> None:

self.session.query(Car).filter\_by(car\_id=car\_id).delete()

self.session.commit()

def delete\_data\_stuff(self, stuff\_id) -> None:

self.session.query(Stuff).filter\_by(stuff\_id=stuff\_id).delete()

self.session.commit()

Фрагмент програми для видалення даних у таблицях

Дані фрагменти програми, які наведені вижче, відповідають за функціонал додвання даних, редагування та вилучення даних у базі даних. Результати фрагментів роботи програми наведені в розділі «Результати роботи програми», які знаходяться нижче.

**Лістинги фрагментів програм з запитами пошуку**

def search\_data\_two\_tables(self):

return self.session.query(Order) \

.join(Stuff) \

.filter(and\_(

Order.order\_id.between(0, 10),

Stuff.stuff\_id.between(0, 2)

)) \

.all()

def search\_data\_three\_tables(self):

return self.session.query(Order) \

.join(Stuff).join(Car) \

.filter(and\_(

Car.car\_id.between(0, 5),

Stuff.stuff\_id.between(0, 2),

Order.order\_id.between(0, 10)

)) \

.all()

def search\_data\_all\_tables(self):

return self.session.query(Order) \

.join(Stuff).join(Customer).join(Car) \

.filter(and\_(

Customer.customer\_id.between(0, 4),

Stuff.stuff\_id.between(0, 2),

Order.order\_id.between(0, 10),

Car.car\_id.between(0, 5)

)) \

.all()

Лістинг програми для пошуку інформації з двох, трьох та чотирьох таблиць з використанням «inner join».

**Лістинги фрагментів програм генерування випадкових даних в таблицях БД**

def car\_data\_generator(self, times: int) -> None:

for i in range(times):

self.connection.execute("insert into public.\"Cars\""

"select (SELECT MAX(car\_id)+1 FROM public.\"Cars\"), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) \

FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4):: integer)), ''), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) \

FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4):: integer)), ''), "

"FLOOR(RANDOM()\*(100000-1)+1);")

def customer\_data\_generator(self, times: int) -> None:

for i in range(times):

self.connection.execute(

"insert into public.\"Customers\" select (SELECT (MAX(customer\_id)+1) FROM public.\"Customers\"), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(10-3)+3):: integer)), ''), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(10-3)+3):: integer)), ''), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(10-3)+3):: integer)), ''), "

"(SELECT to\_timestamp(1549634400+random()\*70071999));")

def order\_data\_generator(self, times: int) -> None:

for i in range(times):

self.connection.execute(

"insert into public.\"Orders\" select (SELECT MAX(order\_id)+1 FROM public.\"Orders\"), "

"(SELECT customer\_id FROM public.\"Customers\" LIMIT 1 OFFSET "

"(round(random() \*((SELECT COUNT(customer\_id) FROM public.\"Customers\")-1)))), "

"(SELECT car\_id FROM public.\"Cars\" LIMIT 1 OFFSET "

"(round(random() \* ((SELECT COUNT(car\_id) FROM public.\"Cars\")-1)))), "

"(SELECT stuff\_id FROM public.\"Stuff\" LIMIT 1 OFFSET "

"(round(random() \* ((SELECT COUNT(stuff\_id) FROM public.\"Stuff\")-1)))), "

"(SELECT to\_timestamp(1549634400+random()\*70071999));")

def stuff\_data\_generator(self, times: int) -> None:

for i in range(times):

self.connection.execute("insert into public.\"Stuff\" select (SELECT MAX(stuff\_id)+1 FROM public.\"Stuff\"), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+10):: integer)), ''), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4):: integer)), ''), "

"(SELECT to\_timestamp(1549634400+random()\*70071999));")

Лістинг програми для генерування випадкових даних для різних таблиць. В кожній таблиці обирається певне поле і заповнюється випадковими символами, які можна використовувати в данній комірці. На вхід подається параметр, який відповідає за довжину таблиці, тобто кількість записів.

**Лістинг модуля "Model"**

Моделі, які присутні у таблиці бази даних. Назви моделей і полів відповідають назвам полів у базі даних.

import datetime

import psycopg2 as ps

import datetime

from sqlalchemy import Column, Integer, String, Date, Float, ForeignKey, select, and\_

from sqlalchemy.orm import relationship

from db import Orders, Session, engine

def recreate\_database():

Orders.metadata.drop\_all(engine)

Orders.metadata.create\_all(engine)

class Order(Orders):

\_\_tablename\_\_ = 'Orders'

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

order\_date = Column(Date)

customer\_id = Column(Integer, ForeignKey('Customers.customer\_id'))

car\_id = Column(Integer, ForeignKey('Cars.car\_id'))

stuff\_id = Column(Integer, ForeignKey('Stuff.stuff\_id'))

#customer = relationship("Customers")

#car = relationship("Cars")

#stuff = relationship("Stuff")

def \_\_init\_\_(self, order\_id, order\_date,customer\_id,car\_id,stuff\_id):

self.order\_id = order\_id

self.order\_date = order\_date

self.car\_id = car\_id

self.customer = customer\_id

self.stuff\_id = stuff\_id

def \_\_repr\_\_(self):

return "{:>10}{:>10}{:>15}{:>10}\t\t{}" \

.format(self.order\_id, self.customer\_id , self.car\_id, self.stuff\_id, self.order\_date)

class Customer(Orders):

\_\_tablename\_\_ = 'Customers'

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

customer\_name = Column(String)

customer\_surname = Column(String)

customer\_telephone = Column(String)

def \_\_init\_\_(self, customer\_id, customer\_name, customer\_surname, customer\_telephone):

self.customer\_id = customer\_id

self.customer\_name = customer\_name

self.customer\_surname = customer\_surname

self.customer\_telephone = customer\_telephone

def \_\_repr\_\_(self):

return "{:>10}{:>15}{:>25}{:>45}" \

.format(self.customer\_id, self.customer\_name, self.customer\_surname, self.customer\_telephone)

class Car(Orders):

\_\_tablename\_\_ = 'Cars'

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

brand\_name = Column(String)

model\_name = Column(String)

year\_of\_manufacture = Column(String)

def \_\_init\_\_(self, car\_id, brand\_name, model\_name, year\_of\_manufacture):

self.car\_id = car\_id

self.brand\_name = brand\_name

self.model\_name = model\_name

self.year\_of\_manufacture = year\_of\_manufacture

def \_\_repr\_\_(self):

return "{:>10}{:>55}{:>65}{:>10}" \

.format(self.car\_id, self.brand\_name, self.model\_name, self.year\_of\_manufacture)

class Stuff(Orders):

\_\_tablename\_\_ = 'Stuff'

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

stuff\_surname = Column(String)

stuff\_telephone = Column(String)

date\_start\_working = Column(String)

def \_\_init\_\_(self, stuff\_id, stuff\_surname, stuff\_telephone, date\_start\_working):

self.stuff\_id = stuff\_id

self.stuff\_surname = stuff\_surname

self.stuff\_telephone = stuff\_telephone

self.date\_start\_working = date\_start\_working

def \_\_repr\_\_(self):

return "{:>10}{:>25}{:>45}{:>35}" \

.format(self.stuff\_id, self.stuff\_surname, self.stuff\_telephone, self.date\_start\_working)

class Model:

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

self.session = Session()

self.connection = engine.connect()

def find\_pk\_order(self, key\_value: int):

return self.session.query(Order).filter\_by(order\_id=key\_value).first()

def find\_fk\_order(self, key\_value: int, table\_name: str):

if table\_name == "Customers":

return self.session.query(Order).filter\_by(customer\_id=key\_value).first()

elif table\_name == "Cars":

return self.session.query(Order).filter\_by(car\_id=key\_value).first()

elif table\_name == "Stuff":

return self.session.query(Order).filter\_by(stuff\_id=key\_value).first()

def find\_pk\_customer(self, key\_value: int):

return self.session.query(Customer).filter\_by(customer\_id=key\_value).first()

def find\_pk\_car(self, key\_value: int):

return self.session.query(Car).filter\_by(car\_id=key\_value).first()

def find\_pk\_stuff(self, key\_value: int):

return self.session.query(Stuff).filter\_by(stuff\_id=key\_value).first()

def print\_orders(self):

return self.session.query(Order).order\_by(Order.order\_id.asc()).all()

def print\_customers(self):

return self.session.query(Customer).order\_by(Customer.customer\_id.asc()).all()

def print\_cars(self):

return self.session.query(Car).order\_by(Car.car\_id.asc()).all()

def print\_stuff(self):

return self.session.query(Stuff).order\_by(Stuff.stuff\_id.asc()).all()

def delete\_data\_order(self, order\_id) -> None:

self.session.query(Order).filter\_by(order\_id=order\_id).delete()

self.session.commit()

def delete\_data\_customer(self, customer\_id) -> None:

self.session.query(Customer).filter\_by(customer\_id=customer\_id).delete()

self.session.commit()

def delete\_data\_car(self, car\_id) -> None:

self.session.query(Car).filter\_by(car\_id=car\_id).delete()

self.session.commit()

def delete\_data\_stuff(self, stuff\_id) -> None:

self.session.query(Stuff).filter\_by(stuff\_id=stuff\_id).delete()

self.session.commit()

def update\_data\_order(self, order\_id: int, customer\_id: int, car\_id: int, stuff\_id: int,order\_date: str) -> None:

self.session.query(Order).filter\_by(order\_id=order\_id) \

.update({Order.customer\_id: customer\_id, Order.car\_id: car\_id,

Order.stuff\_id: stuff\_id,Order.order\_date:order\_date})

self.session.commit()

def update\_data\_customer(self, customer\_id: int, customer\_name: str, customer\_surname: str, customer\_telephone: str) -> None:

self.session.query(Customer).filter\_by(customer\_id=customer\_id) \

.update({Customer.customer\_name: customer\_name, Customer.customer\_surname: customer\_surname, Customer.customer\_telephone: customer\_telephone})

self.session.commit()

def update\_data\_car(self, car\_id: int, brand\_name: str, model\_name: str, year\_of\_manufacture: int) -> None:

self.session.query(Car).filter\_by(car\_id=car\_id) \

.update({Car.brand\_name: brand\_name, Car.model\_name: model\_name, Car.year\_of\_manufacture: year\_of\_manufacture})

self.session.commit()

def update\_data\_stuff(self, stuff\_id: int, stuff\_surname: str, date\_start\_working: str ,stuff\_telephone: str) -> None:

self.session.query(Stuff).filter\_by(stuff\_id=stuff\_id) \

.update({Stuff.stuff\_surname: stuff\_surname, Stuff.date\_start\_working: date\_start\_working, Stuff.stuff\_telephone: stuff\_telephone})

self.session.commit()

def insert\_data\_order(self, order\_id: int, customer\_id: int, car\_id: int, stuff\_id: int, order\_date: str) -> None:

order = Order(order\_id=order\_id, customer\_id=customer\_id, car\_id=car\_id, stuff\_id=stuff\_id, order\_date=order\_date)

self.session.add(order)

self.session.commit()

def insert\_data\_customer(self, customer\_id: int, customer\_name: str, customer\_surname: str, customer\_telephone: str) -> None:

customer = Customer(customer\_id=customer\_id, customer\_name=customer\_name, customer\_surname=customer\_surname, customer\_telephone=customer\_telephone)

self.session.add(customer)

self.session.commit()

def insert\_data\_car(self, car\_id: int, brand\_name: str, model\_name: str, year\_of\_manufacture: str) -> None:

car = Car(car\_id=car\_id, brand\_name=brand\_name, model\_name=model\_name, year\_of\_manufacture=year\_of\_manufacture)

self.session.add(car)

self.session.commit()

def insert\_data\_stuff(self, stuff\_id: int, stuff\_surname: str, date\_start\_working: str, stuff\_telephone: str) -> None:

stuff = Stuff(stuff\_id=stuff\_id, stuff\_surname=stuff\_surname, date\_start\_working=date\_start\_working,stuff\_telephone=stuff\_telephone)

self.session.add(stuff)

self.session.commit()

def car\_data\_generator(self, times: int) -> None:

for i in range(times):

self.connection.execute("insert into public.\"Cars\""

"select (SELECT MAX(car\_id)+1 FROM public.\"Cars\"), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) \

FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4):: integer)), ''), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) \

FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4):: integer)), ''), "

"FLOOR(RANDOM()\*(100000-1)+1);")

def customer\_data\_generator(self, times: int) -> None:

for i in range(times):

self.connection.execute(

"insert into public.\"Customers\" select (SELECT (MAX(customer\_id)+1) FROM public.\"Customers\"), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(10-3)+3):: integer)), ''), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(10-3)+3):: integer)), ''), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(10-3)+3):: integer)), ''), "

"(SELECT to\_timestamp(1549634400+random()\*70071999));")

def order\_data\_generator(self, times: int) -> None:

for i in range(times):

self.connection.execute(

"insert into public.\"Orders\" select (SELECT MAX(order\_id)+1 FROM public.\"Orders\"), "

"(SELECT customer\_id FROM public.\"Customers\" LIMIT 1 OFFSET "

"(round(random() \*((SELECT COUNT(customer\_id) FROM public.\"Customers\")-1)))), "

"(SELECT car\_id FROM public.\"Cars\" LIMIT 1 OFFSET "

"(round(random() \* ((SELECT COUNT(car\_id) FROM public.\"Cars\")-1)))), "

"(SELECT stuff\_id FROM public.\"Stuff\" LIMIT 1 OFFSET "

"(round(random() \* ((SELECT COUNT(stuff\_id) FROM public.\"Stuff\")-1)))), "

"(SELECT to\_timestamp(1549634400+random()\*70071999));")

def stuff\_data\_generator(self, times: int) -> None:

for i in range(times):

self.connection.execute("insert into public.\"Stuff\" select (SELECT MAX(stuff\_id)+1 FROM public.\"Stuff\"), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+10):: integer)), ''), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4):: integer)), ''), "

"(SELECT to\_timestamp(1549634400+random()\*70071999));")

def search\_data\_two\_tables(self):

return self.session.query(Order) \

.join(Stuff) \

.filter(and\_(

Order.order\_id.between(0, 10),

Stuff.stuff\_id.between(0, 2)

)) \

.all()

def search\_data\_three\_tables(self):

return self.session.query(Order) \

.join(Stuff).join(Car) \

.filter(and\_(

Car.car\_id.between(0, 5),

Stuff.stuff\_id.between(0, 2),

Order.order\_id.between(0, 10)

)) \

.all()

def search\_data\_all\_tables(self):

return self.session.query(Order) \

.join(Stuff).join(Customer).join(Car) \

.filter(and\_(

Customer.customer\_id.between(0, 4),

Stuff.stuff\_id.between(0, 2),

Order.order\_id.between(0, 10),

Car.car\_id.between(0, 5)

)) \

.all()

**Зауваження:** Повністю всі функції були переписані з використанням ORM шаблону, але їхнє призначення залишилося таким самим, як і в другій лаб. роботі цього предмету, тому наводжу приклад запитів у вигляді ORM на прикладі однієї таблиці та без демонстрації виводу помилок з метою економії часу.

**Завдання №1**

**Запити у вигляді** **ORM**

Продемонструємо вставку, виучення, редагування даних на прикладі таблиці Cars:

**Видалення запису:**



Рисунок1 - видалення запису

**Вставка запису:**

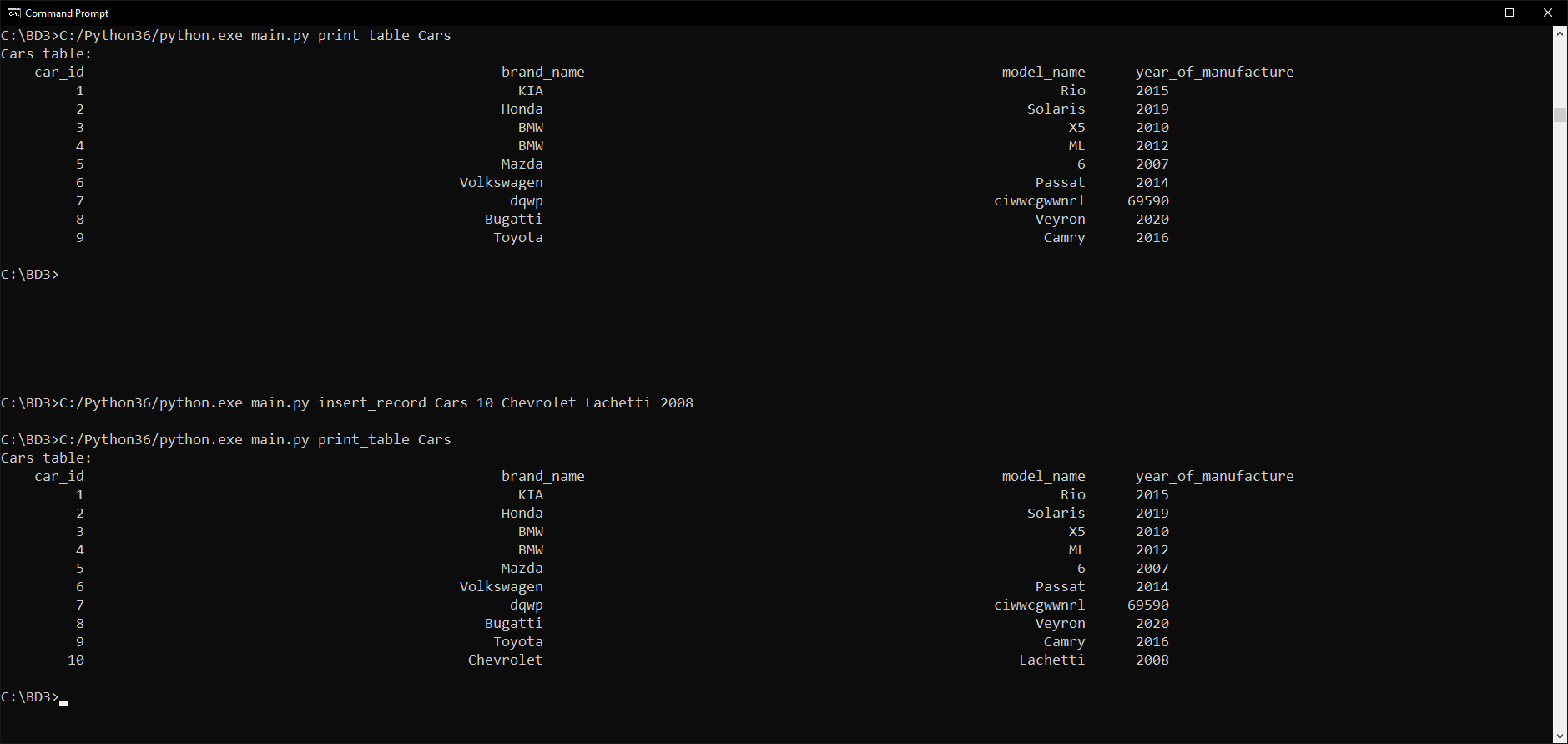


Рисунок2 - вставка запису

**Редагування запису:**

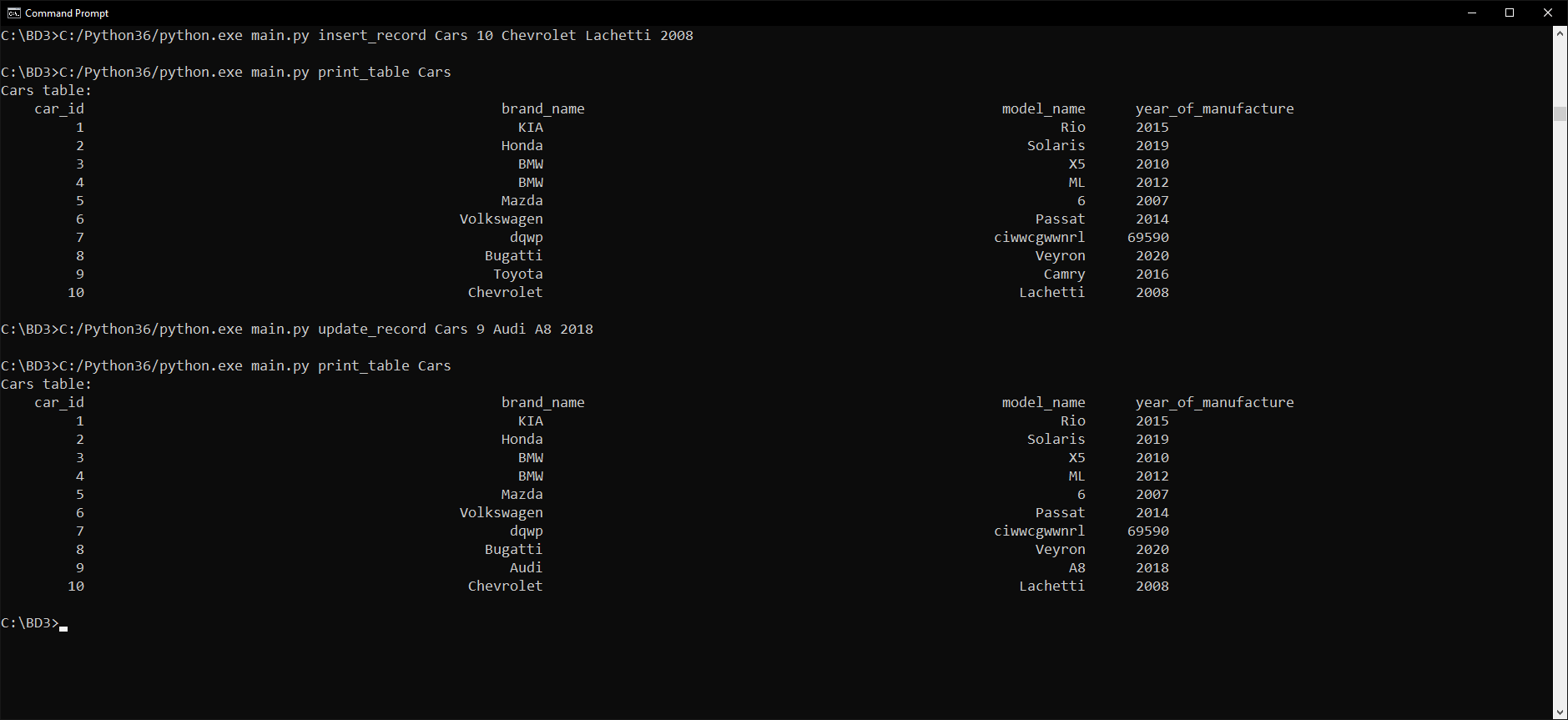


Рисунок2 - редагування запису

Запити пошуку та генерації рандомізованих даних також було реалізовано, логіку пошуку було змінено у порівнянні з лабораторною роботою №2 (усі дані для пошуку передвизначено, тепер вони не вводяться з клавіатури). Запити на пошук ті самі, що і л.р. №2. Запит на генерацію даних продемонструємо на прикладі таблиці Order.

Вставка 3-х випадково згенерованих записів:

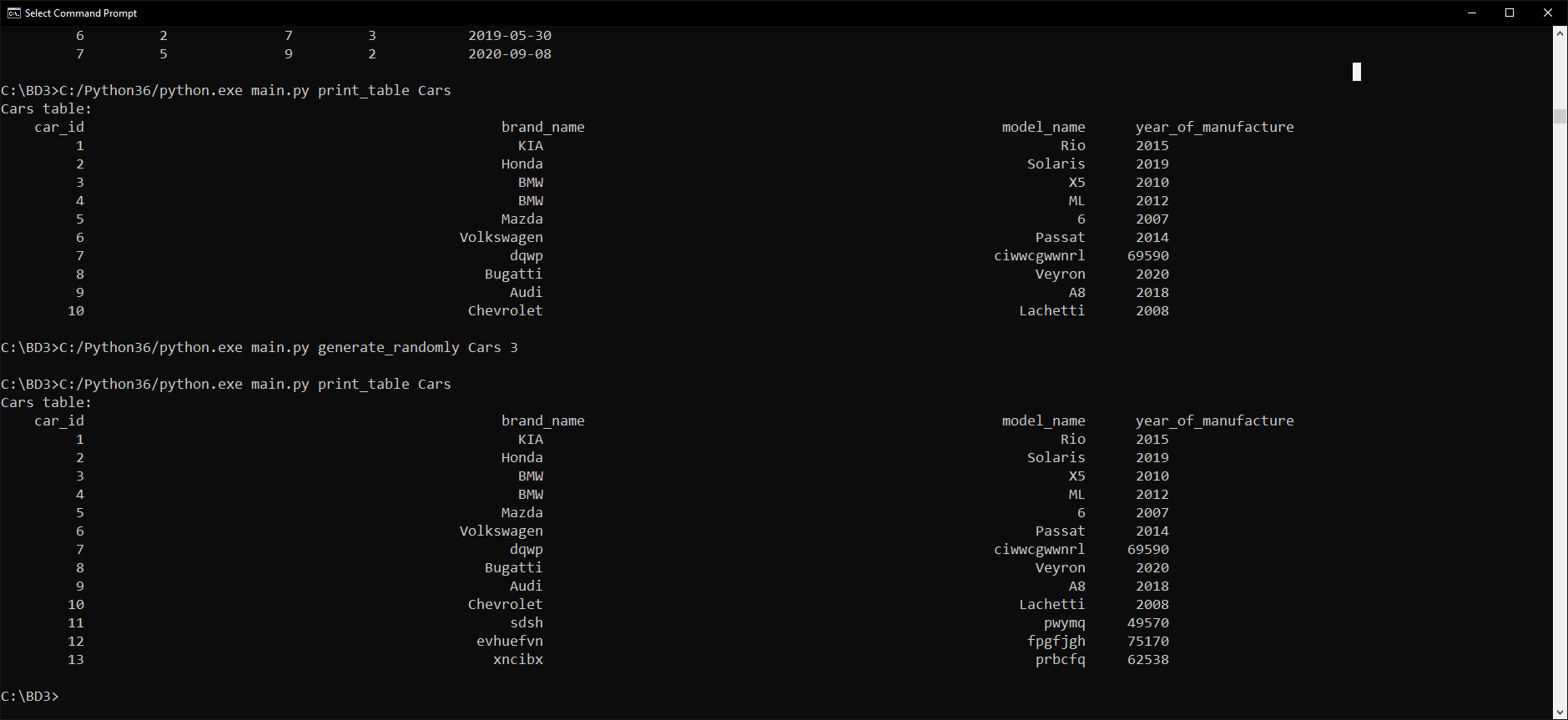


Рисунок3 – генерація записів

Пошук за трьома атрибутами у двох таблицях, за трьома атрибутами у трьох таблицях, за чотирма атрибутами у чотирьох таблицях (виводяться відповідні записи з таблиці Orders):

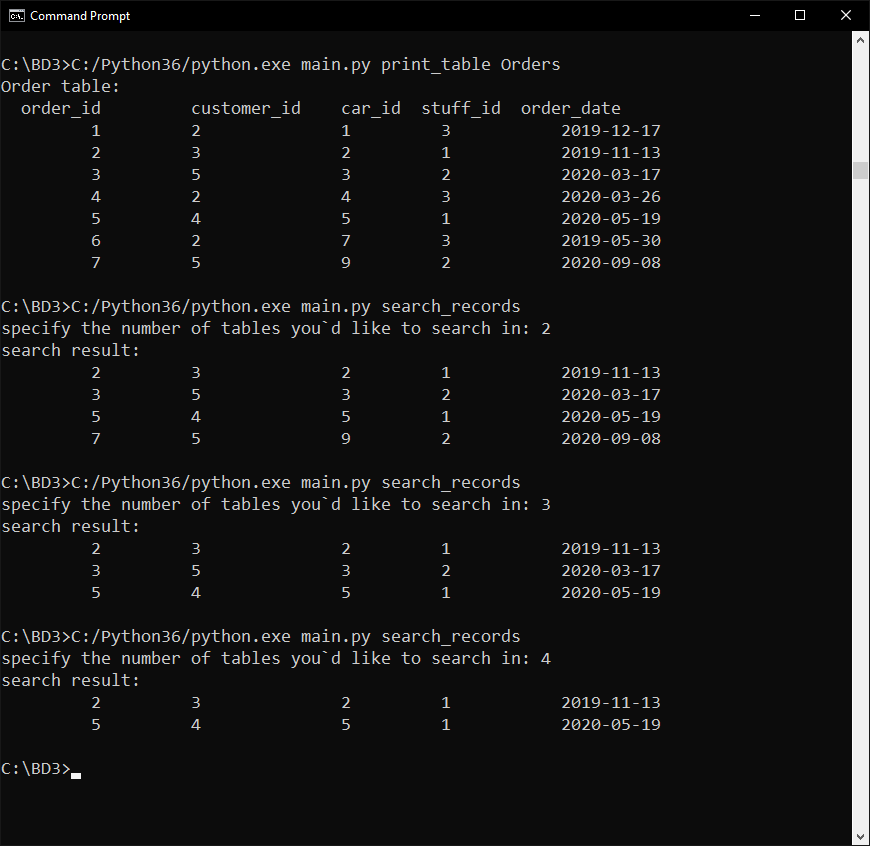


Рисунок4 - пошук запису

**Завдання № 2**

Робота з індексами

Індекс BTREE, він же B-дерево, придатний для даних, які можна відсортувати. Інакше кажучи, для типу даних мають бути визначені оператори «більше», «більше чи одно», «менше», «менше чи одно» і «рівно». Ті самі дані іноді можна сортувати різними способами, що повертає нас до концепції сімейства операторів. Як завжди, індексні записи B-дерева упаковані у сторінки. У листових сторінках ці записи містять індексовані дані (ключі) та посилання на рядки таблиці (TID-и); у внутрішніх сторінках кожен запис посилається на дочірню сторінку індексу та містить мінімальне значення ключа у цій сторінці.

Для дослідження індексу була створена таблиця, яка має дві колонки: числову і текстову. Вони проіндексовані як BTree. У таблицю було занесено 1000000 записів.

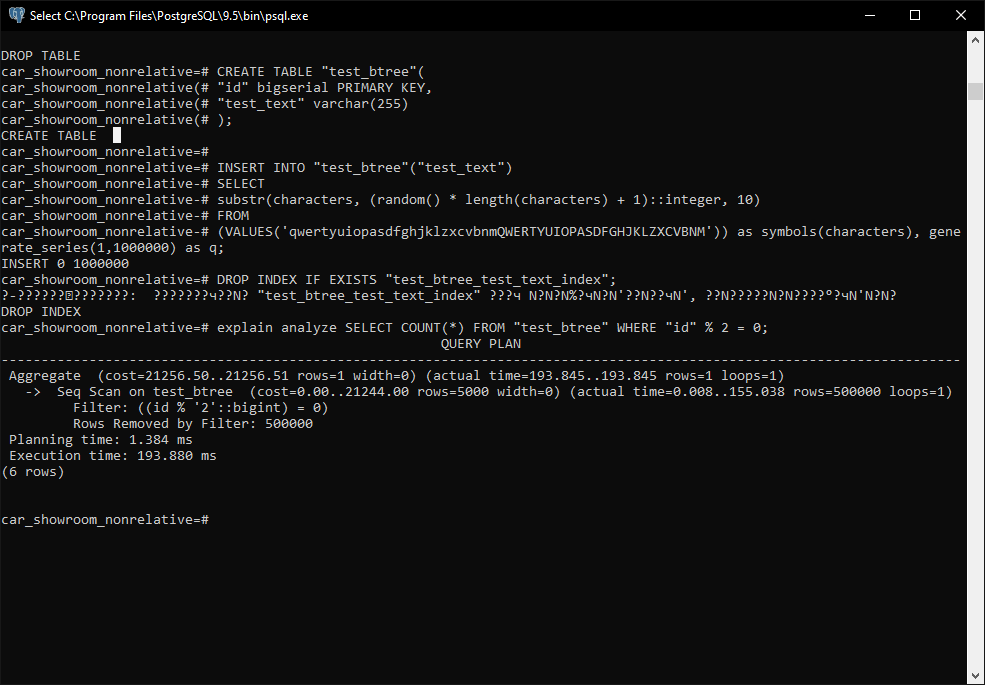


Рисунок5 - Аналіз виконання запиту1 без індексу

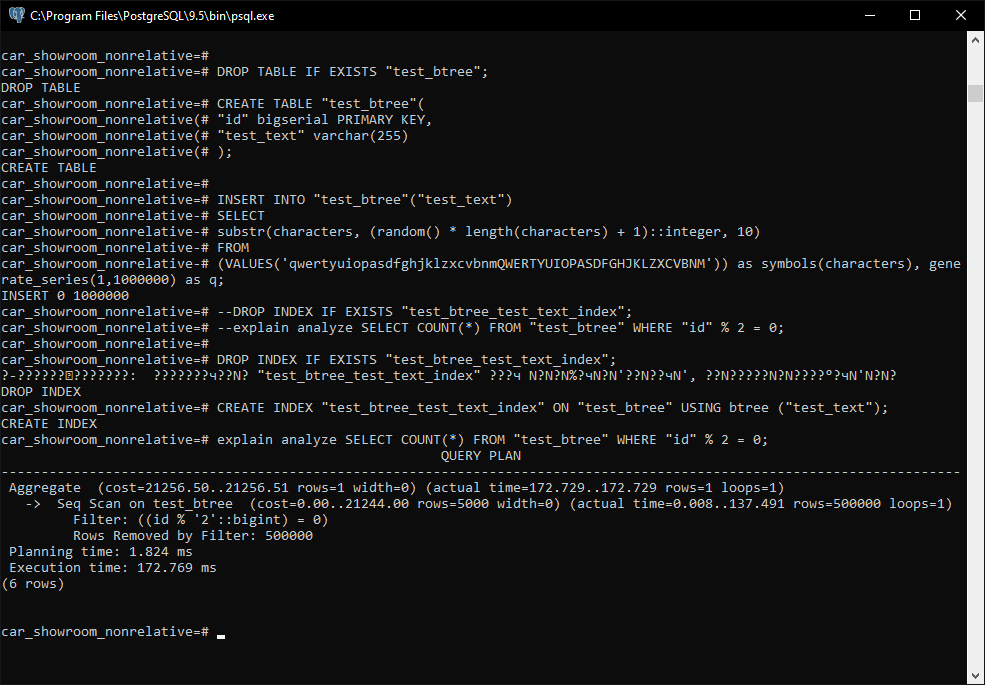


Рисунок6 - Аналіз виконання запиту1 з індексом

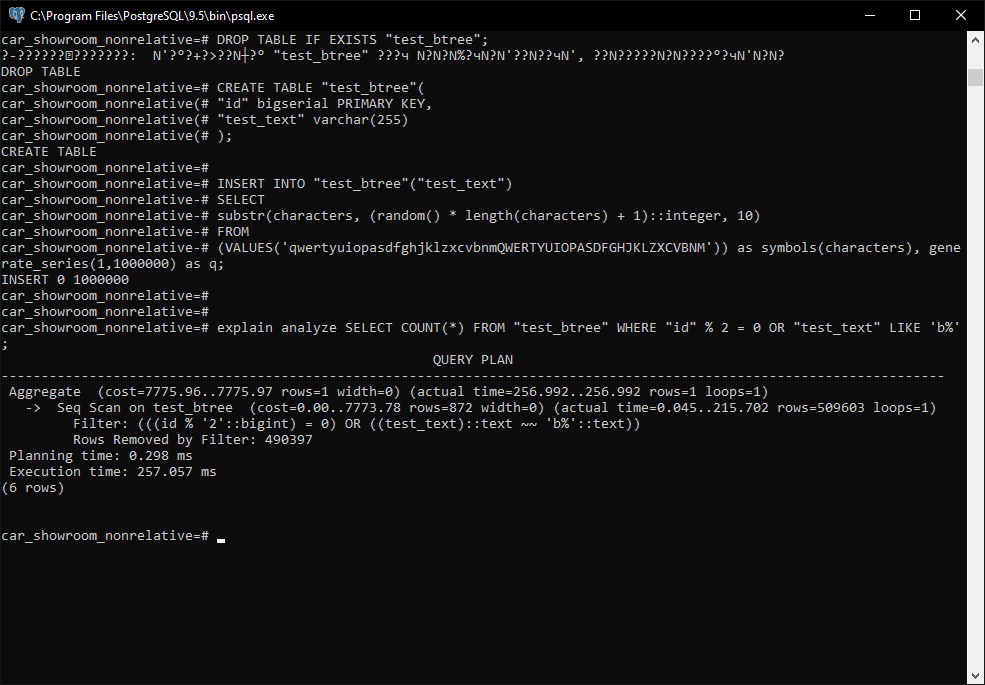


Рисунок7 - Аналіз виконання запиту2 без індексу

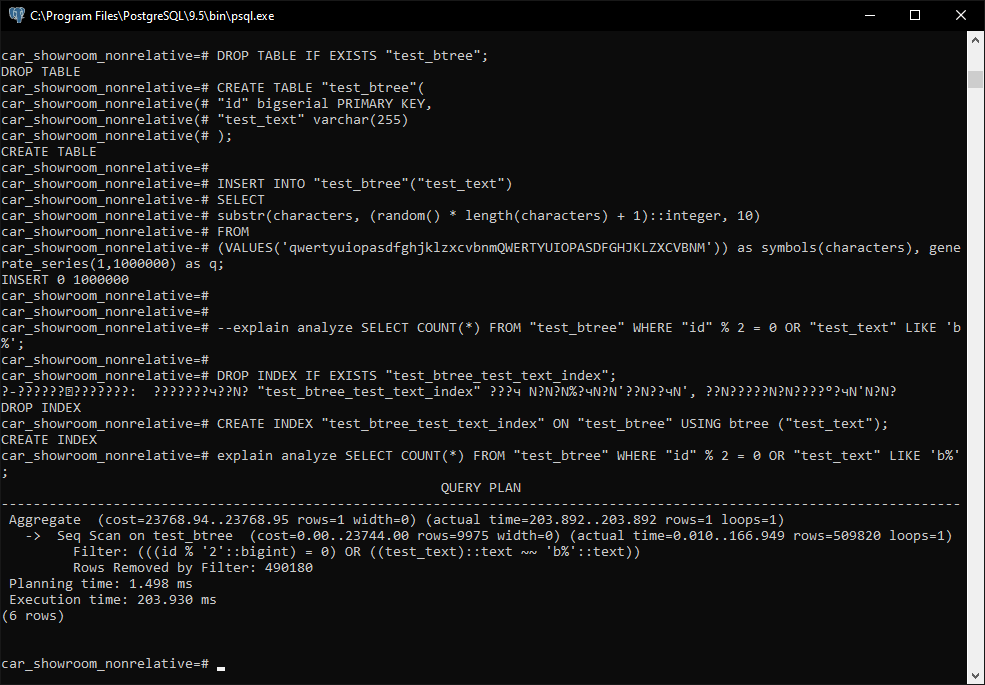


Рисунок8 - Аналіз виконання запиту2 з індексом

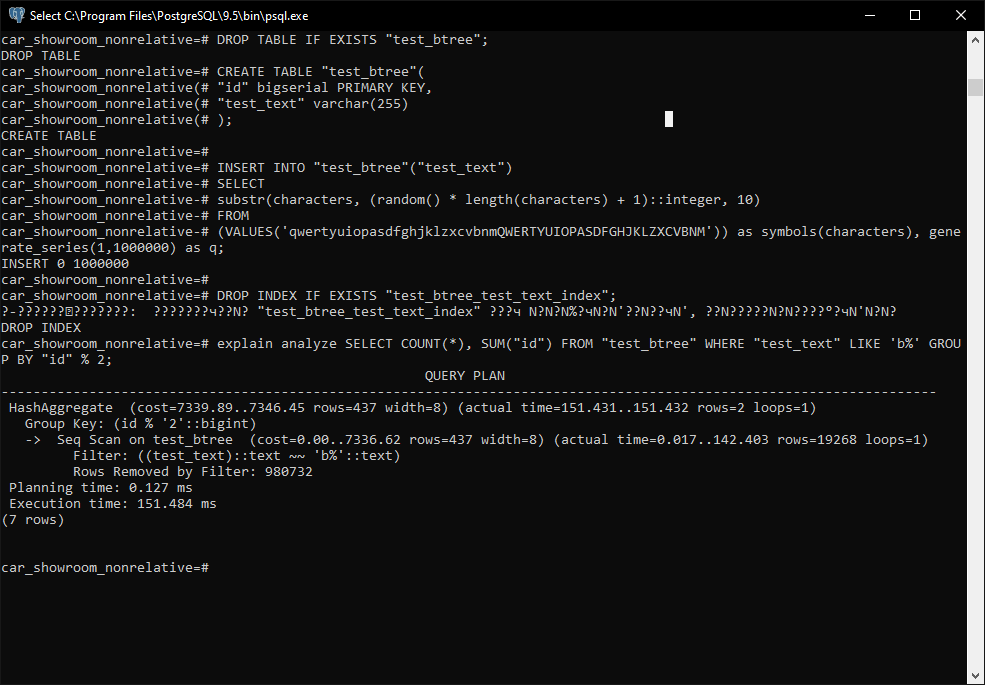


Рисунок9 - Аналіз виконання запиту3 без індексу

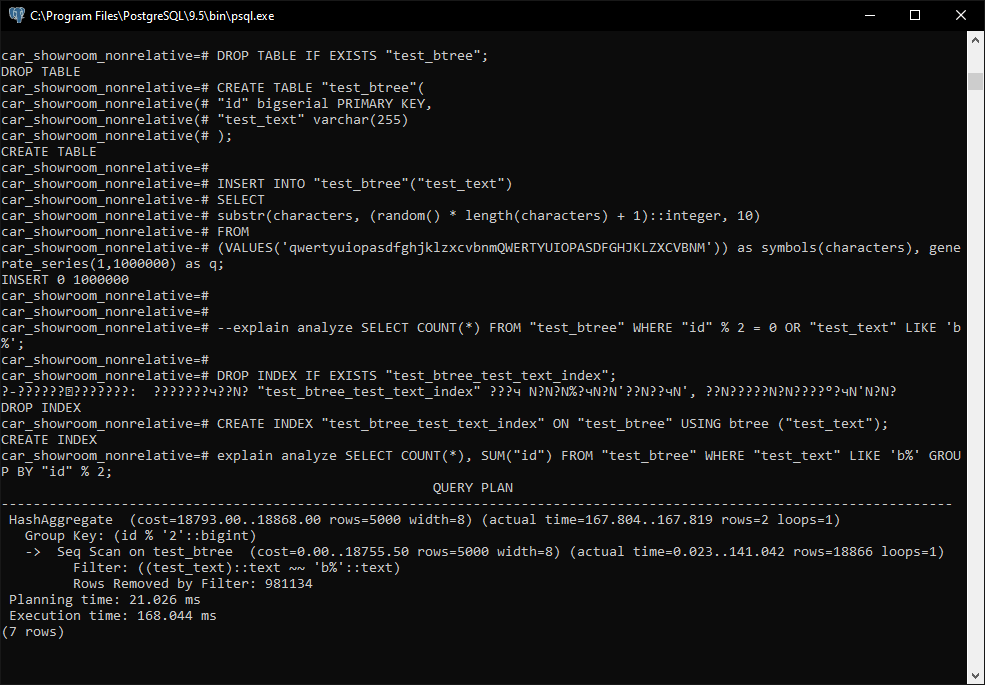


Рисунок10 - Аналіз виконання запиту3 з індексом

BRIN – це Block Range Index, головна концепція якого не знаходження необхідного значення, а уникнення перегляду свідомо непотрібних.

Він працює добре для тих стовпчиків, де значення корелюють із їх фізичним положенням в таблиці. Тобто, якщо запит без ORDER BY видає значення стовпчика практично в порядку зростання чи спадання.

Для дослідження індексу була створена таблиця, яка має дві колонки: t\_data типу timestamp without time zone (дата та час (без часового поясу)) і t\_number типу integer. Колонка t\_data проіндексована як BRIN. У таблицю занесено 1000000 записів

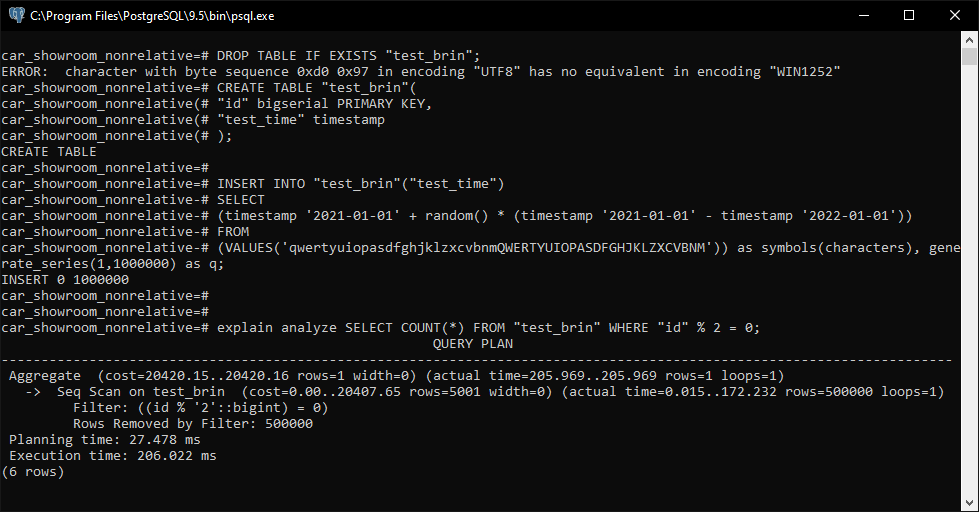


Рисунок11 – Аналіз виконання запиту1 без індексу

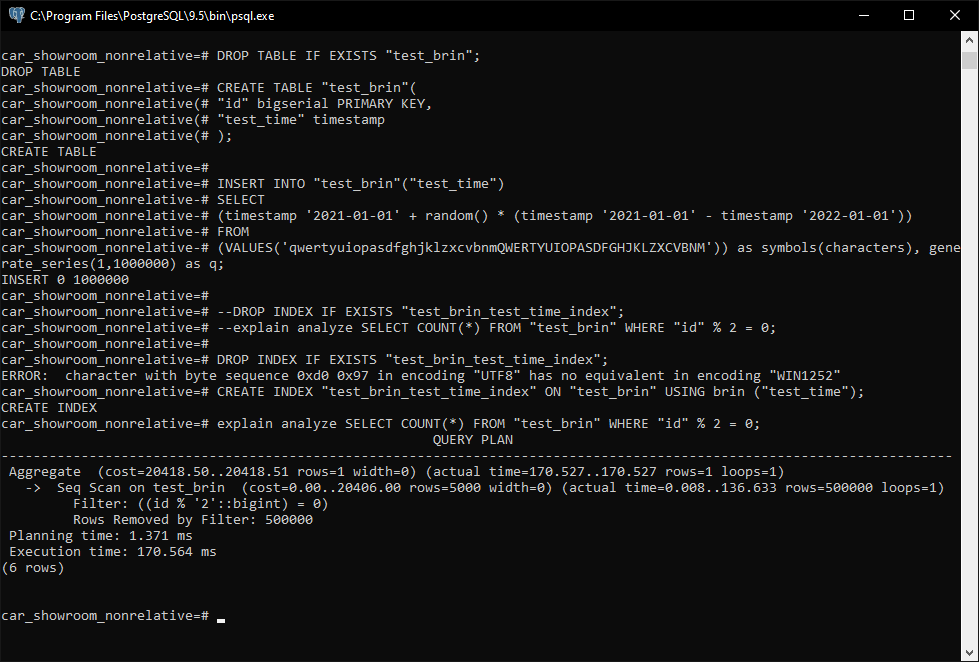


Рисунок12 – Аналіз виконання запиту1 з індексом

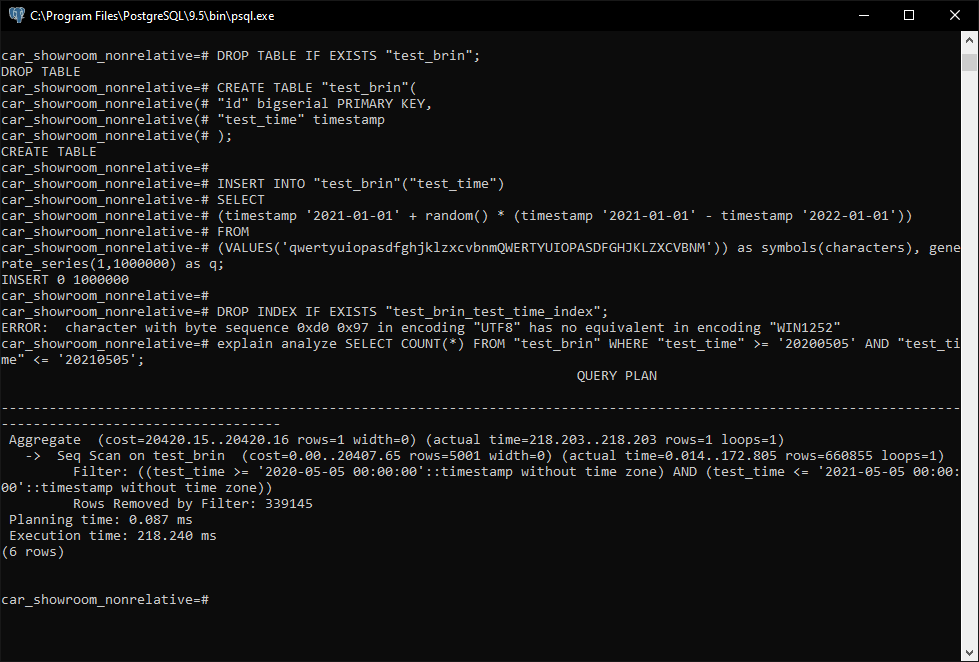


Рисунок13 – Аналіз виконання запиту2 без індексу

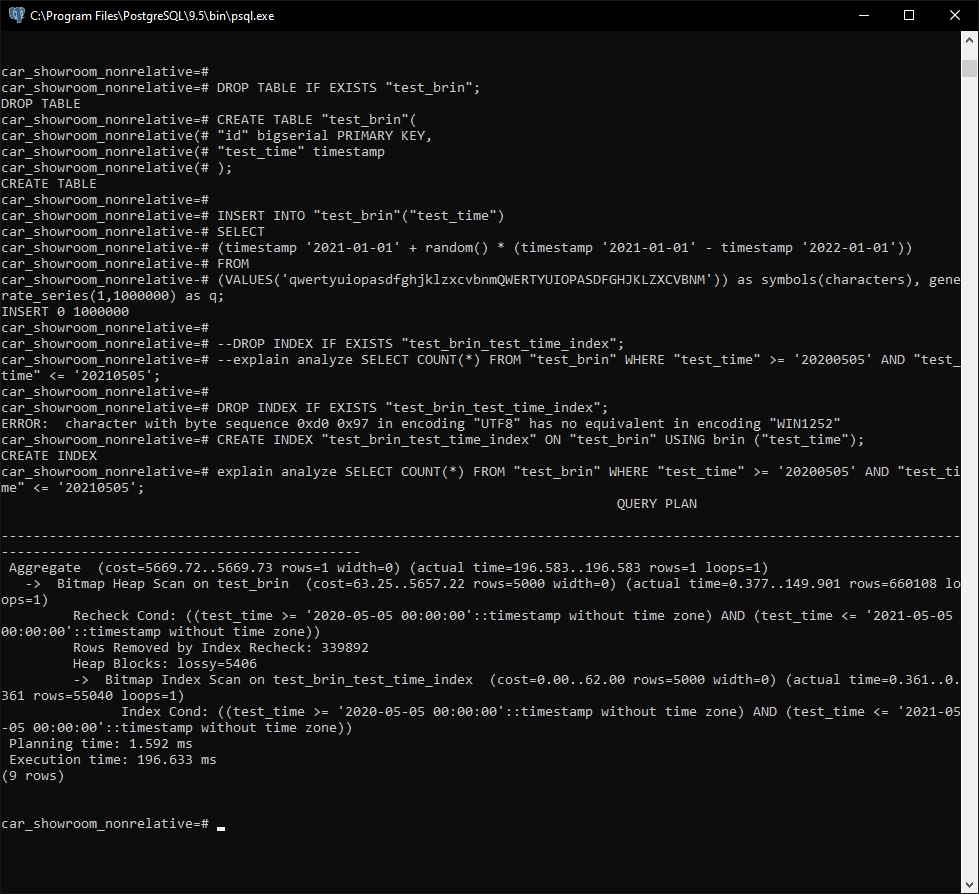


Рисунок14 – Аналіз виконання запиту2 з індексом

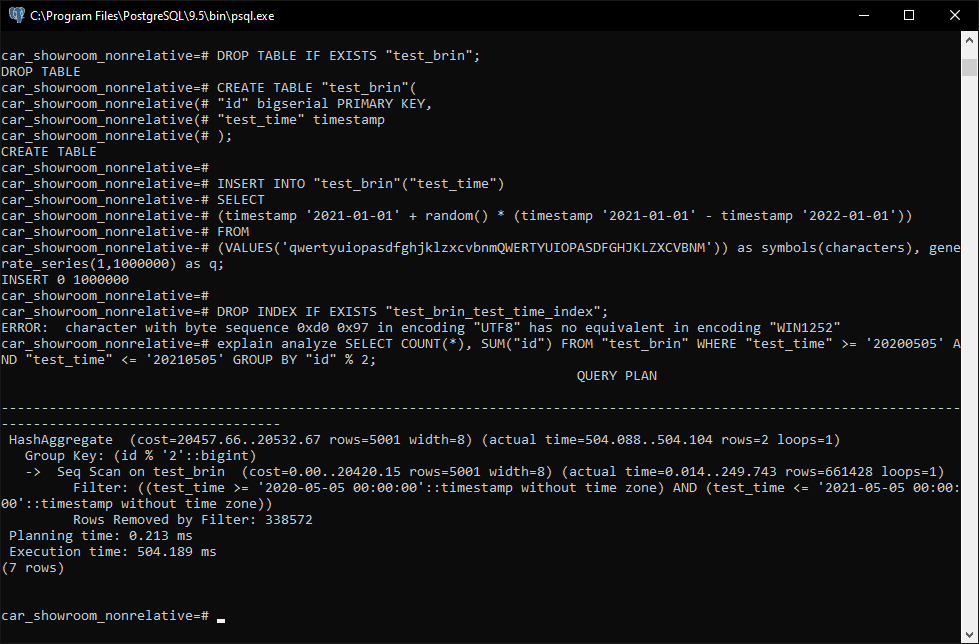


Рисунок15 – Аналіз виконання запиту3 без індексу

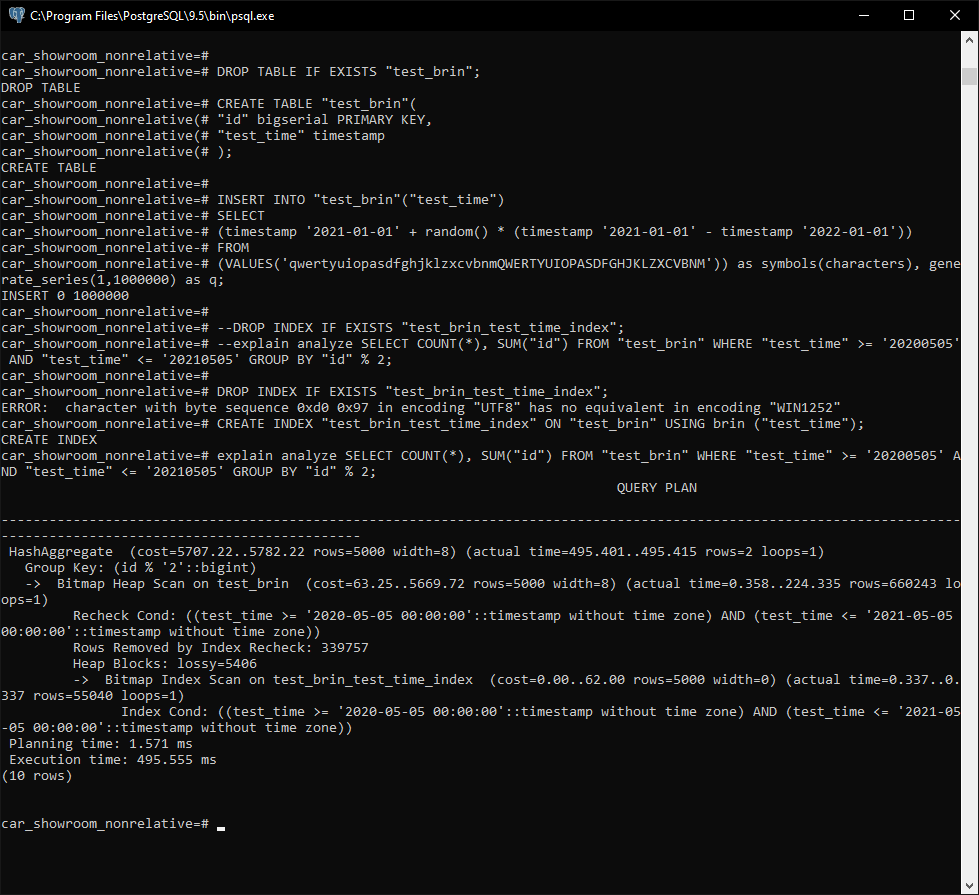


Рисунок16 – Аналіз виконання запиту3 з індексом

**Робота з тригерами**

Таблиця для якої ми будемо тестувати наш тригер

DROP TABLE IF EXISTS "reader";

CREATE TABLE "reader"(

"readerID" bigserial PRIMARY KEY,

"readerName" varchar(255)

);

Тригер

CREATE OR REPLACE FUNCTION update\_insert\_func() RETURNS TRIGGER as $$

DECLARE

curs CURSOR FOR SELECT \* FROM "reader";

m\_row "reader"%ROWTYPE;

begin

IF TG\_OP = 'INSERT' then

for m\_row in curs loop

UPDATE "reader" SET "readerName"=m\_row."readerName" || 'a' WHERE current of curs;

END LOOP;

RAISE NOTICE 'Triggered on inserting!';

return m\_row;

else

RAISE NOTICE 'Triggered on updating!';

return NULL;

END IF;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER "test\_trigger"

AFTER UPDATE OR INSERT ON "reader"

FOR EACH ROW

EXECUTE procedure update\_insert\_func();

Принцип роботи

Створений тригер update\_insert\_func стає активним після виконання вставки нового рядка,а також після оновлення вже існуючого рядка у таблиці “reader”. Якщо була виконана вставка нового рядка, то тригер проходиться по всім рядкам таблиці і додає до кінця поля Owner літеру ’а’, і після того, як він зробив дану дію над всіма рядками,то видає повідомлення - “Triggered on inserting!” та повертає зміну з останнім рядком. Якщо ж було виконане оновлення рядка таблиці, то тригер видає повідомлення “Triggered on updating!” та повертає NULL.

Занесемо тестові дані до таблиці

INSERT INTO "reader"("readerName") VALUES ('reader1'), ('reader2'), ('reader3'), ('reader4'), ('reader5');

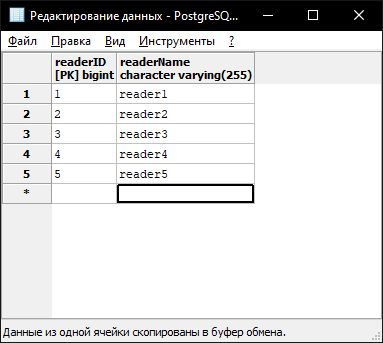


Рисунок17 – початкові дані які занесені до таблиці reader

Перевірка роботи тригера на дію вставки.

Виконаєм запит   
INSERT INTO "reader"("readerName") VALUES ('reader6');

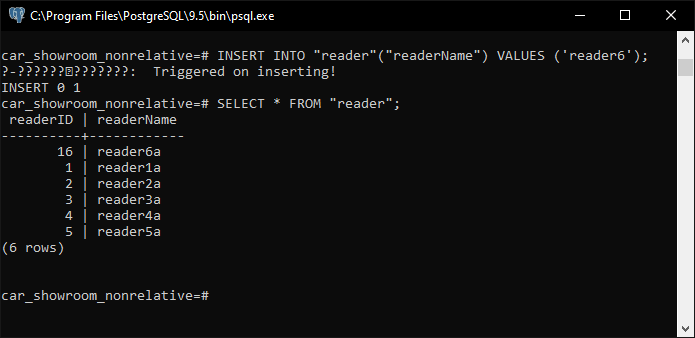


Рисунок19 – Перевірка роботи тригера після вставки нового рядка

Перевірка роботи тригера на дію видалення.

Виконаєм запит  
DELETE FROM "reader" WHERE "readerID" = 6;



Рисунок22 – виконання запиту видалення

**Завдання №4**

Для цього завдання також створювалась окрема таблиця з деякими початковими даними:

DROP TABLE IF EXISTS "transactions";

CREATE TABLE "transactions"(

"id" bigserial PRIMARY KEY,

"numeric" bigint,

"text" text

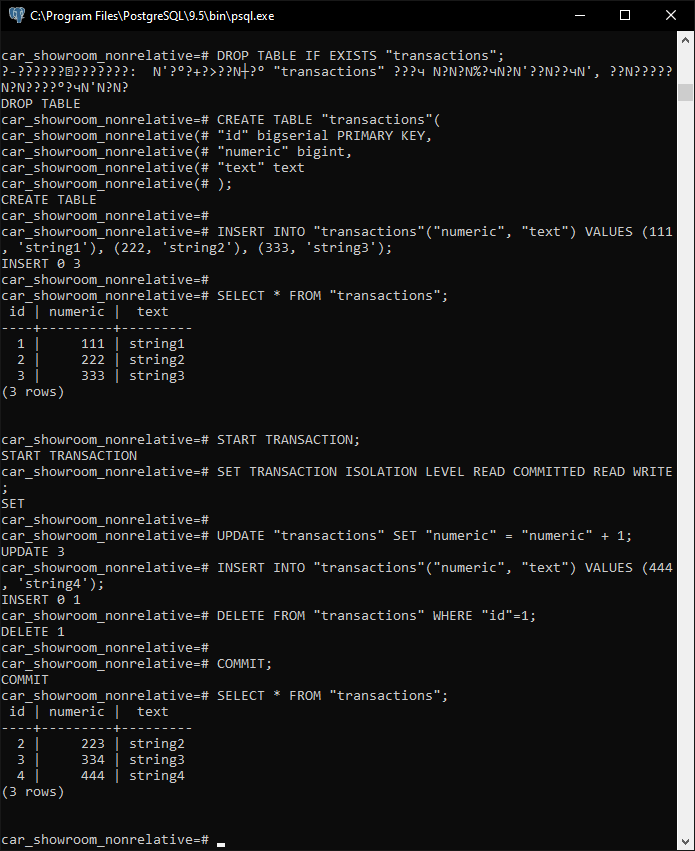
);

INSERT INTO "transactions"("numeric", "text") VALUES (111, 'string1'), (222,

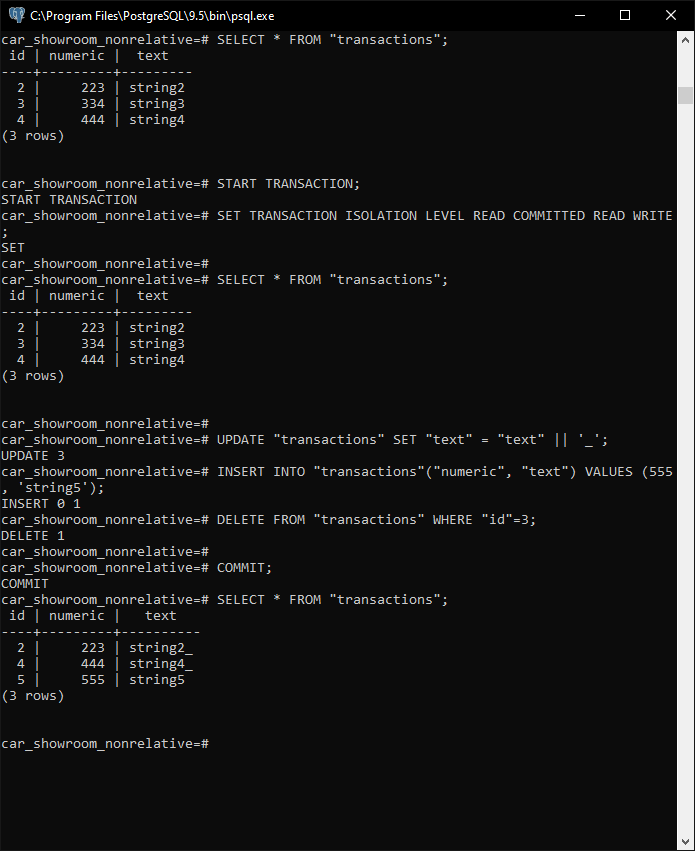
'string2'), (333, 'string3');

**READ COMMITTED**

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



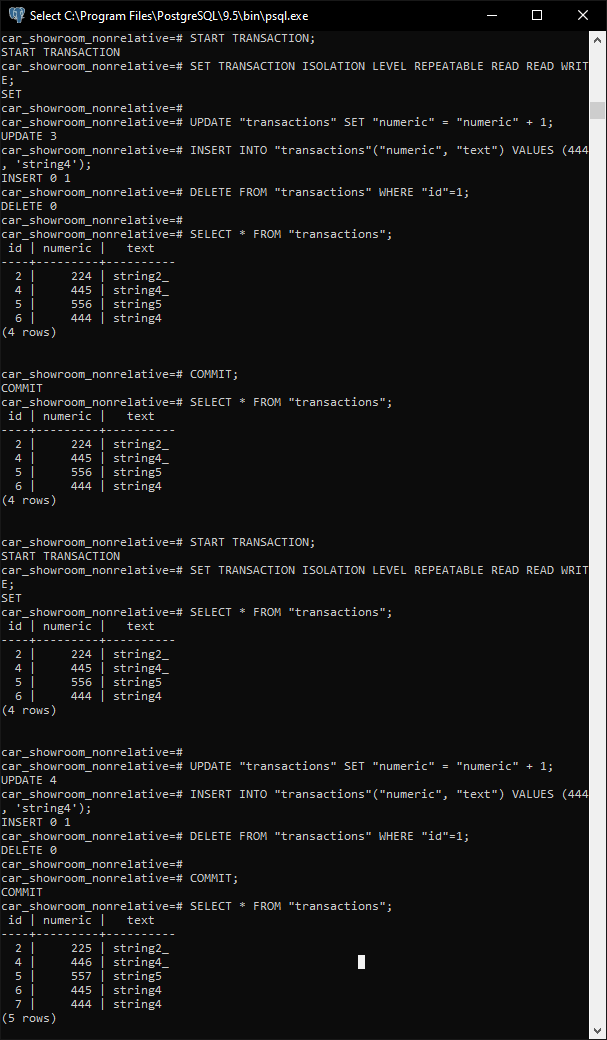
Дані після вставки та видалення так само будуть видні другій тільки після завершення першої.



Коли Тransaction2 бачить дані Тransaction1 запитів UPDATE, DELETE виникає феномен повторного читаня, а коли бачить дані запиту INSERT – читання фантомів. Цей рівень ізоляції забезпечує захист від явища брудного читання.

REPEATABLE READ

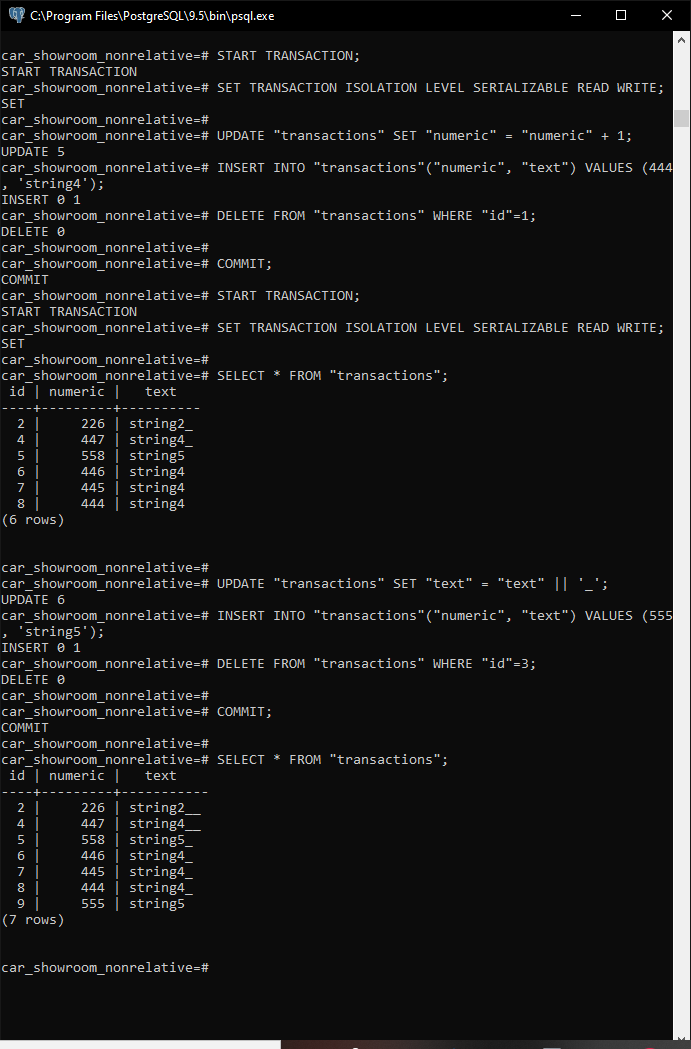
На цьому рівні ізоляції Т2 не бачитиме змінені дані транзакцією Т1, але також не зможе отримати доступ до тих самих даних. Тут видно, що друга не бачить змін з першої:



Бачимо, що не виникає читання фантомів та повторного читання, а також заборонено одночасний доступ до незбережених даних. Хоча класично цей рівень ізоляції призначений для попередження повторного читання.

SERIALIZABLE

На цьому рівні транзакіції поводять себе так, ніби вони не знають одна про одну. Вони не можуть вплинути одна на одну і одночасний доступ строго заборонений.



У попередньому випадку вдалось “відкатити” другу тразакцію і це не вплинуло на подальшу можливість роботи в терміналі. На цьому ж рівні навіть після завершення першої не вдалося зробити ні COMMIT ні ROLLBACK для другої транзакції. Взагалі, в класичному представленні цей рівень призначений для недопущення явища читання фантомів. На цьому рівні ізоляції ми отримуємо максимальну узгодженість даних і можемо бути впевнені, що зайві дані не будуть зафіксовані.

**Ілюстрації програмного коду з репозиторію Git**

Код програмного модулю “model.py”

f"where {search}")

import datetime

import psycopg2 as ps

import datetime

from sqlalchemy import Column, Integer, String, Date, Float, ForeignKey, select, and\_

from sqlalchemy.orm import relationship

from db import Orders, Session, engine

def recreate\_database():

Orders.metadata.drop\_all(engine)

Orders.metadata.create\_all(engine)

class Order(Orders):

\_\_tablename\_\_ = 'Orders'

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

order\_date = Column(Date)

customer\_id = Column(Integer, ForeignKey('Customers.customer\_id'))

car\_id = Column(Integer, ForeignKey('Cars.car\_id'))

stuff\_id = Column(Integer, ForeignKey('Stuff.stuff\_id'))

#customer = relationship("Customers")

#car = relationship("Cars")

#stuff = relationship("Stuff")

def \_\_init\_\_(self, order\_id, order\_date,customer\_id,car\_id,stuff\_id):

self.order\_id = order\_id

self.order\_date = order\_date

self.car\_id = car\_id

self.customer = customer\_id

self.stuff\_id = stuff\_id

def \_\_repr\_\_(self):

return "{:>10}{:>10}{:>15}{:>10}\t\t{}" \

.format(self.order\_id, self.customer\_id , self.car\_id, self.stuff\_id, self.order\_date)

class Customer(Orders):

\_\_tablename\_\_ = 'Customers'

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

customer\_name = Column(String)

customer\_surname = Column(String)

customer\_telephone = Column(String)

def \_\_init\_\_(self, customer\_id, customer\_name, customer\_surname, customer\_telephone):

self.customer\_id = customer\_id

self.customer\_name = customer\_name

self.customer\_surname = customer\_surname

self.customer\_telephone = customer\_telephone

def \_\_repr\_\_(self):

return "{:>10}{:>15}{:>25}{:>45}" \

.format(self.customer\_id, self.customer\_name, self.customer\_surname, self.customer\_telephone)

class Car(Orders):

\_\_tablename\_\_ = 'Cars'

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

brand\_name = Column(String)

model\_name = Column(String)

year\_of\_manufacture = Column(String)

def \_\_init\_\_(self, car\_id, brand\_name, model\_name, year\_of\_manufacture):

self.car\_id = car\_id

self.brand\_name = brand\_name

self.model\_name = model\_name

self.year\_of\_manufacture = year\_of\_manufacture

def \_\_repr\_\_(self):

return "{:>10}{:>55}{:>65}{:>10}" \

.format(self.car\_id, self.brand\_name, self.model\_name, self.year\_of\_manufacture)

class Stuff(Orders):

\_\_tablename\_\_ = 'Stuff'

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

stuff\_surname = Column(String)

stuff\_telephone = Column(String)

date\_start\_working = Column(String)

def \_\_init\_\_(self, stuff\_id, stuff\_surname, stuff\_telephone, date\_start\_working):

self.stuff\_id = stuff\_id

self.stuff\_surname = stuff\_surname

self.stuff\_telephone = stuff\_telephone

self.date\_start\_working = date\_start\_working

def \_\_repr\_\_(self):

return "{:>10}{:>25}{:>45}{:>35}" \

.format(self.stuff\_id, self.stuff\_surname, self.stuff\_telephone, self.date\_start\_working)

class Model:

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

self.session = Session()

self.connection = engine.connect()

def find\_pk\_order(self, key\_value: int):

return self.session.query(Order).filter\_by(order\_id=key\_value).first()

def find\_fk\_order(self, key\_value: int, table\_name: str):

if table\_name == "Customers":

return self.session.query(Order).filter\_by(customer\_id=key\_value).first()

elif table\_name == "Cars":

return self.session.query(Order).filter\_by(car\_id=key\_value).first()

elif table\_name == "Stuff":

return self.session.query(Order).filter\_by(stuff\_id=key\_value).first()

def find\_pk\_customer(self, key\_value: int):

return self.session.query(Customer).filter\_by(customer\_id=key\_value).first()

def find\_pk\_car(self, key\_value: int):

return self.session.query(Car).filter\_by(car\_id=key\_value).first()

def find\_pk\_stuff(self, key\_value: int):

return self.session.query(Stuff).filter\_by(stuff\_id=key\_value).first()

def print\_orders(self):

return self.session.query(Order).order\_by(Order.order\_id.asc()).all()

def print\_customers(self):

return self.session.query(Customer).order\_by(Customer.customer\_id.asc()).all()

def print\_cars(self):

return self.session.query(Car).order\_by(Car.car\_id.asc()).all()

def print\_stuff(self):

return self.session.query(Stuff).order\_by(Stuff.stuff\_id.asc()).all()

def delete\_data\_order(self, order\_id) -> None:

self.session.query(Order).filter\_by(order\_id=order\_id).delete()

self.session.commit()

def delete\_data\_customer(self, customer\_id) -> None:

self.session.query(Customer).filter\_by(customer\_id=customer\_id).delete()

self.session.commit()

def delete\_data\_car(self, car\_id) -> None:

self.session.query(Car).filter\_by(car\_id=car\_id).delete()

self.session.commit()

def delete\_data\_stuff(self, stuff\_id) -> None:

self.session.query(Stuff).filter\_by(stuff\_id=stuff\_id).delete()

self.session.commit()

def update\_data\_order(self, order\_id: int, customer\_id: int, car\_id: int, stuff\_id: int,order\_date: str) -> None:

self.session.query(Order).filter\_by(order\_id=order\_id) \

.update({Order.customer\_id: customer\_id, Order.car\_id: car\_id,

Order.stuff\_id: stuff\_id,Order.order\_date:order\_date})

self.session.commit()

def update\_data\_customer(self, customer\_id: int, customer\_name: str, customer\_surname: str, customer\_telephone: str) -> None:

self.session.query(Customer).filter\_by(customer\_id=customer\_id) \

.update({Customer.customer\_name: customer\_name, Customer.customer\_surname: customer\_surname, Customer.customer\_telephone: customer\_telephone})

self.session.commit()

def update\_data\_car(self, car\_id: int, brand\_name: str, model\_name: str, year\_of\_manufacture: int) -> None:

self.session.query(Car).filter\_by(car\_id=car\_id) \

.update({Car.brand\_name: brand\_name, Car.model\_name: model\_name, Car.year\_of\_manufacture: year\_of\_manufacture})

self.session.commit()

def update\_data\_stuff(self, stuff\_id: int, stuff\_surname: str, date\_start\_working: str ,stuff\_telephone: str) -> None:

self.session.query(Stuff).filter\_by(stuff\_id=stuff\_id) \

.update({Stuff.stuff\_surname: stuff\_surname, Stuff.date\_start\_working: date\_start\_working, Stuff.stuff\_telephone: stuff\_telephone})

self.session.commit()

def insert\_data\_order(self, order\_id: int, customer\_id: int, car\_id: int, stuff\_id: int, order\_date: str) -> None:

order = Order(order\_id=order\_id, customer\_id=customer\_id, car\_id=car\_id, stuff\_id=stuff\_id, order\_date=order\_date)

self.session.add(order)

self.session.commit()

def insert\_data\_customer(self, customer\_id: int, customer\_name: str, customer\_surname: str, customer\_telephone: str) -> None:

customer = Customer(customer\_id=customer\_id, customer\_name=customer\_name, customer\_surname=customer\_surname, customer\_telephone=customer\_telephone)

self.session.add(customer)

self.session.commit()

def insert\_data\_car(self, car\_id: int, brand\_name: str, model\_name: str, year\_of\_manufacture: str) -> None:

car = Car(car\_id=car\_id, brand\_name=brand\_name, model\_name=model\_name, year\_of\_manufacture=year\_of\_manufacture)

self.session.add(car)

self.session.commit()

def insert\_data\_stuff(self, stuff\_id: int, stuff\_surname: str, date\_start\_working: str, stuff\_telephone: str) -> None:

stuff = Stuff(stuff\_id=stuff\_id, stuff\_surname=stuff\_surname, date\_start\_working=date\_start\_working,stuff\_telephone=stuff\_telephone)

self.session.add(stuff)

self.session.commit()

def car\_data\_generator(self, times: int) -> None:

for i in range(times):

self.connection.execute("insert into public.\"Cars\""

"select (SELECT MAX(car\_id)+1 FROM public.\"Cars\"), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) \

FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4):: integer)), ''), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) \

FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4):: integer)), ''), "

"FLOOR(RANDOM()\*(100000-1)+1);")

def customer\_data\_generator(self, times: int) -> None:

for i in range(times):

self.connection.execute(

"insert into public.\"Customers\" select (SELECT (MAX(customer\_id)+1) FROM public.\"Customers\"), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(10-3)+3):: integer)), ''), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(10-3)+3):: integer)), ''), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(10-3)+3):: integer)), ''), "

"(SELECT to\_timestamp(1549634400+random()\*70071999));")

def order\_data\_generator(self, times: int) -> None:

for i in range(times):

self.connection.execute(

"insert into public.\"Orders\" select (SELECT MAX(order\_id)+1 FROM public.\"Orders\"), "

"(SELECT customer\_id FROM public.\"Customers\" LIMIT 1 OFFSET "

"(round(random() \*((SELECT COUNT(customer\_id) FROM public.\"Customers\")-1)))), "

"(SELECT car\_id FROM public.\"Cars\" LIMIT 1 OFFSET "

"(round(random() \* ((SELECT COUNT(car\_id) FROM public.\"Cars\")-1)))), "

"(SELECT stuff\_id FROM public.\"Stuff\" LIMIT 1 OFFSET "

"(round(random() \* ((SELECT COUNT(stuff\_id) FROM public.\"Stuff\")-1)))), "

"(SELECT to\_timestamp(1549634400+random()\*70071999));")

def stuff\_data\_generator(self, times: int) -> None:

for i in range(times):

self.connection.execute("insert into public.\"Stuff\" select (SELECT MAX(stuff\_id)+1 FROM public.\"Stuff\"), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+10):: integer)), ''), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4):: integer)), ''), "

"(SELECT to\_timestamp(1549634400+random()\*70071999));")

def search\_data\_two\_tables(self):

return self.session.query(Order) \

.join(Stuff) \

.filter(and\_(

Order.order\_id.between(0, 10),

Stuff.stuff\_id.between(0, 2)

)) \

.all()

def search\_data\_three\_tables(self):

return self.session.query(Order) \

.join(Stuff).join(Car) \

.filter(and\_(

Car.car\_id.between(0, 5),

Stuff.stuff\_id.between(0, 2),

Order.order\_id.between(0, 10)

)) \

.all()

def search\_data\_all\_tables(self):

return self.session.query(Order) \

.join(Stuff).join(Customer).join(Car) \

.filter(and\_(

Customer.customer\_id.between(0, 4),

Stuff.stuff\_id.between(0, 2),

Order.order\_id.between(0, 10),

Car.car\_id.between(0, 5)

)) \

.all()

Код програмного модулю “main.py”

import controller as con

from psycopg2 import Error

import sys

c = con.Controller()

try:

command = sys.argv[1]

except IndexError:

c.v.no\_command()

else:

if command == 'print\_table':

try:

name = sys.argv[2]

except IndexError:

c.v.argument\_error()

else:

c.printf(name)

elif command == 'delete\_record':

try:

args = {"name": sys.argv[2], "val": sys.argv[3]}

except IndexError:

c.v.argument\_error()

else:

c.delete(args["name"], args["val"])

elif command == 'update\_record':

try:

args = {"name": sys.argv[2], "key": sys.argv[3]}

if args["name"] == 'Cars':

args["brand\_name"], args["model\_name"], args["year\_of\_manufacture"] = \

sys.argv[4], sys.argv[5], sys.argv[6]

elif args["name"] == 'Customers':

args["customer\_name"], args["customer\_surname"], args["customer\_telephone"] = \

sys.argv[4], sys.argv[5], sys.argv[6]

elif args["name"] == 'Orders':

args["customer\_id"], args["car\_id"], args["stuff\_id"], args["order\_date"] = \

sys.argv[4], sys.argv[5], sys.argv[6], sys.argv[7]

elif args["name"] == 'Stuff':

args["stuff\_surname"], args["stuff\_telephone"], args["date\_start\_working"]= \

sys.argv[4], sys.argv[5], sys.argv[6]

else:

c.v.wrong\_table()

except IndexError:

c.v.argument\_error()

else:

if args["name"] == 'Cars':

c.update\_cars(args["key"], args["brand\_name"], args["model\_name"],

args["year\_of\_manufacture"])

elif args["name"] == 'Customers':

c.update\_customers(args["key"], args["customer\_name"], args["customer\_surname"], args["customer\_telephone"])

elif args["name"] == 'Orders':

c.update\_orders(args["key"], args["customer\_id"], args["car\_id"],args["stuff\_id"], args["order\_date"])

elif args["name"] == 'Stuff':

c.update\_stuff(args["key"], args["stuff\_surname"], args["stuff\_telephone"], args["date\_start\_working"])

elif command == 'insert\_record':

try:

args = {"name": sys.argv[2], "key": sys.argv[3]}

if args["name"] == 'Cars':

args["brand\_name"], args["model\_name"], args["year\_of\_manufacture"] = \

sys.argv[4], sys.argv[5], sys.argv[6]

elif args["name"] == 'Customers':

args["customer\_name"], args["customer\_surname"], args["customer\_telephone"] = \

sys.argv[4], sys.argv[5], sys.argv[6]

elif args["name"] == 'Orders':

args["customer\_id"], args["car\_id"],args["stuff\_id"], args["order\_date"] = \

sys.argv[4], sys.argv[5], sys.argv[6], sys.argv[7]

elif args["name"] == 'Stuff':

args["stuff\_surname"], args["stuff\_telephone"], args["date\_start\_working"] = \

sys.argv[4], sys.argv[5], sys.argv[6]

else:

c.v.wrong\_table()

except IndexError:

c.v.argument\_error()

else:

if args["name"] == 'Cars':

c.insert\_cars(args["key"], args["brand\_name"], args["model\_name"],

args["year\_of\_manufacture"])

elif args["name"] == 'Customers':

c.insert\_customer(args["key"], args["customer\_name"], args["customer\_surname"], args["customer\_telephone"])

elif args["name"] == 'Orders':

c.insert\_orders(args["key"], args["customer\_id"], args["car\_id"], args["stuff\_id"], args["order\_date"])

elif args["name"] == 'Stuff':

c.insert\_stuff(args["key"], args["stuff\_surname"], args["stuff\_telephone"], args["date\_start\_working"])

elif command == 'generate\_randomly':

try:

args = {"name": sys.argv[2], "n": int(sys.argv[3])}

except (IndexError, Exception):

print(Exception, IndexError)

else:

c.generate(args["name"], args["n"])

elif command == 'search\_records':

while True:

search\_num = c.v.get\_search\_num()

try:

search\_num = int(search\_num)

except ValueError:

c.v.invalid\_search\_num()

else:

if search\_num in [2, 3, 4]:

break

else:

c.v.invalid\_search\_num()

if search\_num == 2:

c.search\_two()

elif search\_num == 3:

c.search\_three()

elif search\_num == 4:

c.search\_all()

elif command == 'help':

c.v.print\_help()

else:

c.v.wrong\_command()

Код програмного модулю “validator.py”

import datetime

class Validator:

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

self.error = ''

self.er\_flag = False

def check\_table\_name(self, arg: str):

if arg in ['Cars', 'Customers', 'Orders', 'Stuff']:

return arg

else:

self.er\_flag = True

self.error = f'table {arg} does not exist in the database'

print(self.error)

return False

def check\_pkey\_value(self, arg: str, min\_val: int, max\_val: int):

try:

value = int(arg)

except ValueError:

self.er\_flag = True

self.error = f'{arg} is not correct primary key value'

print(self.error)

return 0

else:

if min\_val <= value <= max\_val:

return value

else:

self.er\_flag = True

self.error = f'{arg} is not existing primary key value'

print(self.error)

return 0

def check\_pk\_name(self, table\_name: str, key\_name: str):

if table\_name == 'Cars' and key\_name == 'car\_id' \

or table\_name == 'Customers' and key\_name == 'customer\_id' \

or table\_name == 'Orders' and key\_name == 'order\_id' \

or table\_name == 'Stuff' and key\_name == 'stuff\_id':

return key\_name

else:

self.er\_flag = True

self.error = f'key {key\_name} is not a primary key of table {table\_name}'

print(self.error)

return False

def check\_pk(self, val):

try:

value = int(val)

return value

except ValueError:

self.er\_flag = True

self.error = f'{val} is not correct primary key value'

print(self.error)

return 0

def check\_key\_names(self, table\_name: str, key: str):

if table\_name == 'Cars' and key in ['car\_id', 'brand\_name', 'model\_name', 'year\_of\_manufacture']:

return True

elif table\_name == 'Customers' and key in ['customer\_id', 'customer\_name', 'customer\_surname', 'customer\_telephone']:

return True

elif table\_name == 'Orders' and key in ['order\_id', 'customer\_id', 'car\_id', 'stuff\_id','order\_date']:

return True

elif table\_name == 'Stuff' and key in ['stuff\_id', 'stuff\_surname', 'stuff\_telephone','date\_start\_working']:

return True

else:

self.er\_flag = True

self.error = f'{key} is not correct name for {table\_name} table'

print(self.error)

return False

def check\_possible\_keys(self, table\_name: str, key: str, val):

if table\_name == 'Cars':

if key in ['car\_id']:

try:

value = int(val)

except ValueError:

self.er\_flag = True

self.error = f'{val} is not correct key value'

print(self.error)

return False

else:

return True

elif key in ['brand\_name', 'model\_name']:

return True

elif key == 'year\_of\_manufacture':

try:

value = float(val)

except ValueError:

self.er\_flag = True

self.error = f'{val} is not correct year\_of\_manufacture value'

print(self.error)

return False

else:

return True

else:

self.er\_flag = True

self.error = f'{key} is not correct name for Cars table'

print(self.error)

return False

elif table\_name == 'Customers':

if key in ['customer\_id']:

try:

value = int(val)

except ValueError:

self.er\_flag = True

self.error = f'{val} is not correct key value'

print(self.error)

return False

else:

return True

elif key in ['customer\_name','customer\_surname']:

return True

elif key in ['customer\_telephone']:

try:

arr = [int(x) for x in val.split(sep='.')]

datetime.datetime(arr[0], arr[1], arr[2], arr[3], arr[4], arr[5])

except TypeError:

self.er\_flag = True

self.error = f'{val} is not correct customer\_telephone value'

print(self.error)

return False

else:

return True

else:

self.er\_flag = True

self.error = f'{key} is not correct name for Customers table'

print(self.error)

return False

elif table\_name == 'Orders':

if key in ['order\_id', 'customer\_id', 'car\_id']:

try:

value = int(val)

except ValueError:

self.er\_flag = True

self.error = f'{val} is not correct key value'

print(self.error)

return False

else:

return True

elif key == 'order\_date':

return True

else:

self.er\_flag = True

self.error = f'{key} is not correct name for Orders table'

print(self.error)

return False

elif table\_name == 'Stuff':

if key == 'stuff\_id':

try:

value = int(val)

except ValueError:

self.er\_flag = True

self.error = f'{val} is not correct key value'

print(self.error)

return False

else:

return True

elif key in ['stuff\_surname', 'stuff\_telephone','date\_start\_working']:

return True

else:

self.er\_flag = True

self.error = f'{key} is not correct name for Stuff table'

print(self.error)

return False

Код програмного модулю “view.py”

import datetime

import time

import validator

class View:

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

self.valid = validator.Validator()

def cannot\_delete(self) -> None:

print('this record is connected with another table, deleting will '

'throw error')

def sql\_error(self, e) -> None:

print("[INFO] Error while working with Postgresql", e)

def insertion\_error(self) -> None:

print('Something went wrong (record with such id exists or inappropriate foreign key values)')

def updation\_error(self) -> None:

print('Something went wrong (record with such id does not exist or inappropriate foreign key value)')

def deletion\_error(self) -> None:

print('record with such id does not exist')

def invalid\_interval(self) -> None:

print('invalid interval input')

def print\_time(self, start) -> None:

print("--- %s seconds ---" % (time.time() - start))

def print\_search(self, result):

print('search result:')

for row in result:

print(row)

def print\_orders(self, table):

print('Order table:')

print("%10s%20s%10s%10s%12s" % ("order\_id", "customer\_id", "car\_id", "stuff\_id", "order\_date"))

for row in table:

print(row)

def print\_customers(self, table):

print('Customers table:')

print("%10s%25s%10s%25s" % ("customer\_id", "customer\_name", "customer\_surname", "customer\_telephone"))

for row in table:

print(row)

def print\_cars(self, table):

print('Cars table:')

print("%10s%60s%60s%25s" % ("car\_id", "brand\_name", "model\_name", "year\_of\_manufacture"))

for row in table:

print(row)

def print\_stuff(self, table):

print('Stuff table:')

print("%10s%35s%15s" % ("stuff\_id", "stuff\_surname", "stuff\_telephone","date\_start\_working"))

for row in table:

print(row)

def print\_help(self):

print('print\_table - outputs the specified table \n\targument (table\_name) is required')

print('delete\_record - deletes the specified record from table \n'

'\targuments (table\_name, key\_name, key\_value) are required')

print('update\_record - updates record with specified id in table\n'

'\tProduct args (table\_name, id\_product, title, price, category, id\_catalog, id\_order)\n'

'\tOrder args (table\_name, id\_order, customer\_name, id\_shop, date)\n'

'\tCatalog args (table\_name, id\_catalog, name, id\_shop, pid\_catalog)\n'

'\tShop args (table\_name, id\_shop, address, name)')

print('insert\_record - inserts record into specified table \n'

'\tProduct args (table\_name, id\_product, title, price, category, id\_catalog, id\_order)\n'

'\tOrder args (table\_name, id\_order, customer\_name, id\_shop, date)\n'

'\tCatalog args (table\_name, id\_catalog, name, id\_shop, pid\_catalog)\n'

'\tShop args (table\_name, id\_shop, address, name)')

print('generate\_randomly - generates n random records in table\n'

'\targuments (table\_name, n) are required')

print('search\_records - search for records in two or more tables using one or more keys \n'

'\targuments (table1\_name, table2\_name, table1\_key, table2\_key) are required, \n'

'\tif you want to perform search in more tables: \n'

'\t(table1\_name, table2\_name, table3\_name, table1\_key, table2\_key, table3\_key, table13\_key) \n'

'\t(table1\_name, table2\_name, table3\_name, table4\_name, table1\_key, table2\_key, table3\_key, table13\_key, '

'table4\_key, table24\_key)')

def get\_search\_num(self):

return input('specify the number of tables you`d like to search in: ')

def invalid\_search\_num(self):

print('should be number from 2 to 4')

def argument\_error(self):

print('no required arguments specified')

def wrong\_table(self):

print('wrong table name')

def no\_command(self):

print('no command name specified, type help to see possible commands')

def wrong\_command(self):

print('unknown command name, type help to see possible commands')

Код програмного модулю “controller.py”

import psycopg2

from psycopg2 import Error

import model

import view

import datetime

import time

class Controller:

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

self.v = view.View()

self.m = model.Model()

def printf(self, table\_name):

t\_name = self.v.valid.check\_table\_name(table\_name)

if t\_name:

if t\_name == 'Cars':

self.v.print\_cars(self.m.print\_cars())

elif t\_name == 'Customers':

self.v.print\_customers(self.m.print\_customers())

elif t\_name == 'Orders':

self.v.print\_orders(self.m.print\_orders())

elif t\_name == 'Stuff':

self.v.print\_stuff(self.m.print\_stuff())

def delete(self, table\_name, value):

t\_name = self.v.valid.check\_table\_name(table\_name)

if t\_name:

k\_val = self.v.valid.check\_pk(value)

count = 0

if t\_name == 'Orders' and k\_val:

count = self.m.find\_pk\_order(k\_val)

elif t\_name == 'Customers' and k\_val:

count = self.m.find\_pk\_customer(k\_val)

elif t\_name == 'Cars' and k\_val:

count = self.m.find\_pk\_car(k\_val)

elif t\_name == 'Stuff' and k\_val:

count = self.m.find\_pk\_stuff(k\_val)

if count:

if t\_name == 'Customers' or t\_name == 'Cars' or t\_name == 'Stuff':

count\_p = self.m.find\_fk\_order(k\_val, t\_name)

if count\_p:

self.v.cannot\_delete()

else:

try:

if t\_name == 'Customers':

self.m.delete\_data\_customer(k\_val)

elif t\_name == 'Cars':

self.m.delete\_data\_car(k\_val)

elif t\_name == 'Stuff':

self.m.delete\_data\_stuff(k\_val)

except (Exception, Error) as \_ex:

self.v.sql\_error(\_ex)

#""" elif t\_name == 'Shop':

# count\_c = self.m.find\_fk\_catalog(k\_val)

# count\_o = self.m.find\_fk\_order(k\_val)

# if count\_c or count\_o:

# self.v.cannot\_delete()

# else:

# try:

# self.m.delete\_data\_shop(k\_val)

# except (Exception, Error) as \_ex:

# self.v.sql\_error(\_ex)

#"""

else:

try:

self.m.delete\_data\_order(k\_val)

except (Exception, Error) as \_ex:

self.v.sql\_error(\_ex)

else:

self.v.deletion\_error()

def update\_customer(self, key: str, customer\_name: str, customer\_surname: str, customer\_telephone: str):

if self.v.valid.check\_possible\_keys('Customers', 'customer\_id', key):

count\_p = self.m.find\_pk\_customer(int(key))

p\_val = self.v.valid.check\_pk(key)

# p\_val=1

if count\_p and p\_val:

try:

self.m.update\_data\_customer(p\_val, customer\_name, customer\_surname,

customer\_telephone)

except (Exception, Error) as \_ex:

self.v.sql\_error(\_ex)

else:

self.v.updation\_error()

def update\_cars(self, key: str, brand\_name: str, model\_name: str, year\_of\_manufacture: int):

if self.v.valid.check\_possible\_keys('Cars', 'car\_id', key):

count\_s = self.m.find\_pk\_car(int(key))

s\_val = self.v.valid.check\_pk(key)

# s\_val = 1

if count\_s and \

s\_val and \

self.v.valid.check\_possible\_keys('Cars', 'car\_id', key):

try:

self.m.update\_data\_car(s\_val, brand\_name, model\_name, year\_of\_manufacture)

except (Exception, Error) as \_ex:

self.v.sql\_error(\_ex)

else:

self.v.updation\_error()

def update\_orders(self, key: str, customer\_id: int, car\_id: int, stuff\_id: int, order\_date: str):

if self.v.valid.check\_possible\_keys('Orders', 'order\_id', key):

count\_s = self.m.find\_pk\_order(int(key))

s\_val = self.v.valid.check\_pk(key)

if self.v.valid.check\_possible\_keys('Customers', 'customer\_id', customer\_id):

count\_c = self.m.find\_pk\_customer(int(customer\_id))

c\_val = self.v.valid.check\_pk(key)

if self.v.valid.check\_possible\_keys('Cars', 'car\_id', car\_id):

count\_pc = self.m.find\_pk\_car(int(car\_id))

pc\_val = self.v.valid.check\_pk(car\_id)

if self.v.valid.check\_possible\_keys('Stuff', 'stuff\_id', stuff\_id):

count\_p = self.m.find\_pk\_stuff(int(stuff\_id))

p\_val = self.v.valid.check\_pk(stuff\_id)

if count\_s and count\_c and count\_pc and count\_p and \

s\_val and c\_val and pc\_val and p\_val:

try:

self.m.update\_data\_order(s\_val, c\_val, pc\_val, p\_val, order\_date)

except (Exception, Error) as \_ex:

self.v.sql\_error(\_ex)

else:

self.v.updation\_error()

def update\_stuff(self, key: str, stuff\_surname: str, date\_start\_working:str ,stuff\_telephone:str):

if self.v.valid.check\_possible\_keys('Stuff', 'stuff\_id', key):

count\_s = self.m.find\_pk\_stuff(int(key))

s\_val = self.v.valid.check\_pk(key)

if count\_s and s\_val:

try:

self.m.update\_data\_stuff(s\_val, stuff\_surname, date\_start\_working,stuff\_telephone)

except (Exception, Error) as \_ex:

self.v.sql\_error(\_ex)

else:

self.v.updation\_error()

def insert\_customer(self, key: str, customer\_name: str, customer\_surname: str, customer\_telephone: str):

if self.v.valid.check\_possible\_keys('Customers', 'customer\_id', key):

count\_p = self.m.find\_pk\_customer(int(key))

if (not count\_p) \

and self.v.valid.check\_possible\_keys('Customers', 'customer\_id', key):

try:

self.m.insert\_data\_customer(int(key), title, float(price),

category, c\_val, o\_val)

except (Exception, Error) as \_ex:

self.v.sql\_error(\_ex)

else:

self.v.insertion\_error()

def insert\_cars(self, key: str, brand\_name: str, model\_name: str, year\_of\_manufacture: int):

# if self.v.valid.check\_possible\_keys('Cars ', 'car\_id', key):

count\_o = self.m.find\_pk\_car(int(key))

if (not count\_o):

try:

self.m.insert\_data\_car(int(key), brand\_name, model\_name,year\_of\_manufacture)

except (Exception, Error) as \_ex:

self.v.sql\_error(\_ex)

# else:

# self.v.insertion\_error()

def insert\_orders(self, key: str, customer\_id: int, car\_id: int, stuff\_id: int, order\_date: str):

if self.v.valid.check\_possible\_keys('Orders', 'order\_id', key):

count\_c = self.m.find\_pk\_order(int(key))

if self.v.valid.check\_possible\_keys('Customers', 'customer\_id', customer\_id):

count\_s = self.m.find\_pk\_order(int(key))

s\_val = self.v.valid.check\_pk(key)

if self.v.valid.check\_possible\_keys('Cars', 'car\_id', car\_id):

count\_pc = self.m.find\_pk\_car(int(car\_id))

pc\_val = self.v.valid.check\_pk(car\_id)

if self.v.valid.check\_possible\_keys('Stuff', 'stuff\_id', stuff\_id):

count\_p = self.m.find\_pk\_stuff(int(stuff\_id))

p\_val = self.v.valid.check\_pk(stuff\_id)

if (not count\_c) and count\_s and count\_pc and s\_val and pc\_val \

and count\_p and p\_val:

try:

self.m.insert\_data\_order(int(key), s\_val, pc\_val, p\_val,order\_date)

except (Exception, Error) as \_ex:

self.v.sql\_error(\_ex)

else:

self.v.insertion\_error()

def insert\_stuff(self, key: str, stuff\_surname: str, date\_start\_working:str ,stuff\_telephone:str):

if self.v.valid.check\_possible\_keys('Stuff', 'stuff\_id', key):

count\_s = self.m.find\_pk\_stuff(int(key))

if (not count\_s):

try:

self.m.insert\_data\_stuff(int(key), stuff\_surname, date\_start\_working, stuff\_telephone)

except (Exception, Error) as \_ex:

self.v.sql\_error(\_ex)

else:

self.v.insertion\_error()

def generate(self, table\_name: str, n: int):

t\_name = self.v.valid.check\_table\_name(table\_name)

if t\_name:

if t\_name == 'Cars':

self.m.car\_data\_generator(n)

elif t\_name == 'Customers':

self.m.customer\_data\_generator(n)

elif t\_name == 'Orders':

self.m.order\_data\_generator(n)

elif t\_name == 'Stuff':

self.m.stuff\_data\_generator(n)

def search\_two(self):

result = self.m.search\_data\_two\_tables()

self.v.print\_search(result)

def search\_three(self):

result = self.m.search\_data\_three\_tables()

self.v.print\_search(result)

def search\_all(self):

result = self.m.search\_data\_all\_tables()

self.v.print\_search(result)