

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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

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

**Лабораторна робота №3**

з дисципліни **Бази даних і засоби управління**

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

Виконав:

студент ІII курсу

групи КВ-02

Єгоров М. А.

Перевірив: Павловський В. І.

Київ – 2023

**Варіант 8**

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.

**Логічна модель бази даних**

Нижче (Рисунок 1) наведено логічну модель бази даних:

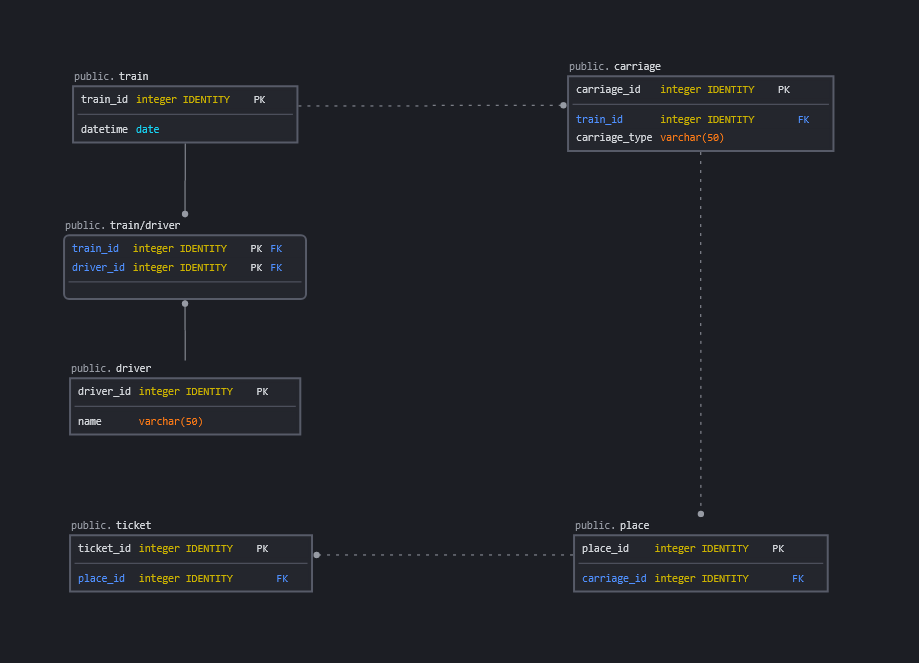
****

Рисунок 1 – Логічна модель бази даних

Зміни у порівнянні з першою лабораторною роботою відсутні.

**Середовище розробки** – PyCharm

**Налаштування підключення до БД** – З використанням sqlAlchemy

engine = create\_engine('postgresql://postgres:111@localhost/lab', echo = True)

**Посилання на гіт із текстом програми:**

**Структура програми та її опис**

Програма складається з трьох файлів-модулів: Controller, View та Model.

Controller відповідає за обробку запитів користувача.

Model за з’єднання та роботу з БД.

View за оформлення та вивід інформації.

**Модулі моделі ORM**

**Controller.py**

**import** View

**import** Model

**import** time

**while** 1==1:

View.hello()

choice = input("Оберіть варіант: ")

*#model = Model.db\_model("lab", "postgres", "111", "")*

match choice:

case "1":

table = input("Введіть назву таблиці: ")

val = input("Введіть відповідні значення: ").split(' ')

match table:

case "driver":

s = Model.Session()

driver = Model.Driver(name=val[0])

s.add(driver)

s.commit()

s.close()

case "train":

s = Model.Session()

train = Model.Train(datetime=val[0])

s.add(train)

s.commit()

s.close()

case "carriage":

s = Model.Session()

carriage = Model.Carriage(train\_id=val[0], carriage\_type=val[1])

s.add(carriage)

s.commit()

s.close()

case "place":

s = Model.Session()

place = Model.Place(carriage\_id=val[0])

s.add(place)

s.commit()

s.close()

case "ticket":

s = Model.Session()

ticket = Model.Ticket(place\_id=val[0])

s.add(ticket)

s.commit()

s.close()

time.sleep(4)

case "2":

table = input("Введіть назву таблиці: ")

val = input("Введіть id: ").split(' ')

znach = input("Введіть відповідні значення: ").split(' ')

match table:

case "driver":

s = Model.Session()

driver = s.query(Model.Driver).filter(Model.Driver.drier\_id == val[0]).first()

driver.name = znach[0]

s.commit()

s.close()

case "train":

s = Model.Session()

train = s.query(Model.Train).filter(Model.Train.train\_id == val[0]).first()

train.datetime = znach[0]

s.commit()

s.close()

case "carriage":

s = Model.Session()

carriage = s.query(Model.Carriage).filter(Model.Carriage.carriage\_id == val[0]).first()

carriage.carriage\_type = znach[0]

carriage.train\_id = znach[1]

s.commit()

s.close()

case "place":

s = Model.Session()

place = s.query(Model.Place).filter(Model.Place.place\_id == val[0]).first()

place.carriage\_id = znach[0]

s.commit()

s.close()

case "ticket":

s = Model.Session()

ticket = s.query(Model.Ticket).filter(Model.Ticket.ticket\_id == val[0]).first()

ticket.place\_id = znach[0]

s.commit()

s.close()

time.sleep(4)

case "3":

table = input("Введіть назву таблиці: ")

val = input("Введіть відповідні значення: ").split(' ')

match table:

case "driver":

s = Model.Session()

driver = s.query(Model.Driver).filter(Model.Driver.driver\_id == val[0]).first()

s.delete(driver)

s.commit()

s.close()

case "train":

s = Model.Session()

train = s.query(Model.Train).filter(Model.Train.train\_id == val[0]).first()

s.delete(train)

s.commit()

s.close()

case "carriage":

s = Model.Session()

carriage = s.query(Model.Carriage).filter(Model.Carriage.carriage\_id == val[0]).first()

s.delete(carriage)

s.commit()

s.close()

case "place":

s = Model.Session()

place = s.query(Model.Place).filter(Model.Place.place\_id == val[0]).first()

s.delete(place)

s.commit()

s.close()

case "ticket":

s = Model.Session()

ticket = s.query(Model.Ticket).filter(Model.Ticket.ticket\_id == val[0]).first()

s.delete(ticket)

s.commit()

s.close()

time.sleep(4)

case \_:

**print**("Error")

**Model.py**

**import** sqlalchemy

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

engine = create\_engine('postgresql://postgres:111@localhost/lab', echo = True)

**from** sqlalchemy **import** MetaData, Table, Column, Integer, String, ForeignKey, Date, Text

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

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

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

**from** sqlalchemy.orm **import** sessionmaker

meta = MetaData()

Base = declarative\_base()

**class** Train(Base):

\_\_tablename\_\_ = 'train'

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

datetime = Column(Date)

drivers = relationship("Driver", secondary='traindriver')

**class** Driver(Base):

\_\_tablename\_\_ = 'driver'

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

name = Column(String)

trains = relationship("Train", secondary='traindriver')

**class** TrainDriverAssoc(Base):

\_\_tablename\_\_ = 'traindriver'

train\_id = Column(Integer, ForeignKey('train.train\_id'), primary\_key=True)

driver\_id = Column(Integer, ForeignKey('driver.driver\_id'), primary\_key=True)

**class** Carriage(Base):

\_\_tablename\_\_ = 'carriage'

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

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

carriage\_type = Column(String)

place = relationship("Place", order\_by="Place.place\_id", back\_populates="carriage")

**class** Place(Base):

\_\_tablename\_\_ = 'place'

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

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

ticket = relationship("Ticket", order\_by="Ticket.ticket\_id", back\_populates="place")

carriage = relationship("Carriage", back\_populates="place")

**class** Ticket(Base):

\_\_tablename\_\_ = 'ticket'

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

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

place = relationship("Place", back\_populates="ticket")

Session = sessionmaker(bind=engine)

**View.py**

**import** sqlalchemy

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

engine = create\_engine('postgresql://postgres:111@localhost/lab', echo = True)

**from** sqlalchemy **import** MetaData, Table, Column, Integer, String, ForeignKey, Date, Text

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

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

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

**from** sqlalchemy.orm **import** sessionmaker

meta = MetaData()

Base = declarative\_base()

**class** Train(Base):

\_\_tablename\_\_ = 'train'

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

datetime = Column(Date)

drivers = relationship("Driver", secondary='traindriver')

**class** Driver(Base):

\_\_tablename\_\_ = 'driver'

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

name = Column(String)

trains = relationship("Train", secondary='traindriver')

**class** TrainDriverAssoc(Base):

\_\_tablename\_\_ = 'traindriver'

train\_id = Column(Integer, ForeignKey('train.train\_id'), primary\_key=True)

driver\_id = Column(Integer, ForeignKey('driver.driver\_id'), primary\_key=True)

**class** Carriage(Base):

\_\_tablename\_\_ = 'carriage'

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

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

carriage\_type = Column(String)

place = relationship("Place", order\_by="Place.place\_id", back\_populates="carriage")

**class** Place(Base):

\_\_tablename\_\_ = 'place'

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

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

ticket = relationship("Ticket", order\_by="Ticket.ticket\_id", back\_populates="place")

carriage = relationship("Carriage", back\_populates="place")

**class** Ticket(Base):

\_\_tablename\_\_ = 'ticket'

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

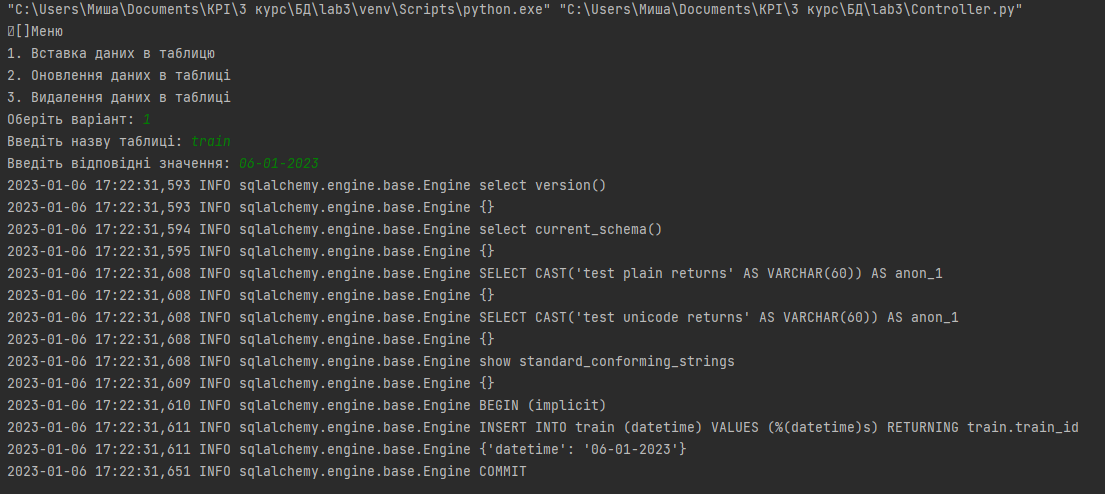
place\_id = Column(Integer, ForeignKey('place.place\_id'))

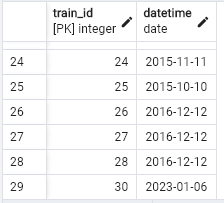
place = relationship("Place", back\_populates="ticket")

Session = sessionmaker(bind=engine)

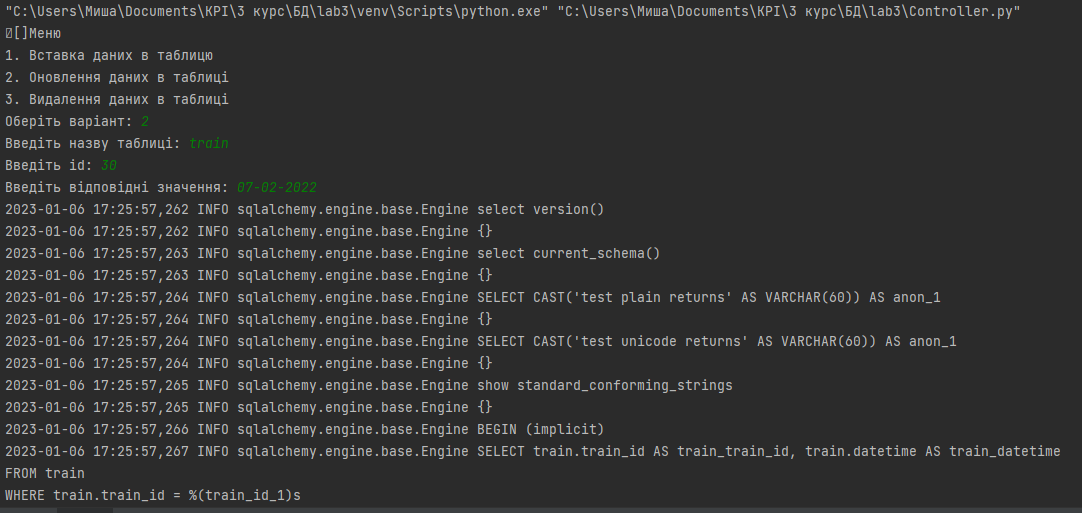
**Приклади запитів ORM:**

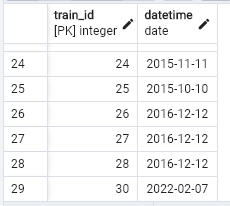
**Створення:**



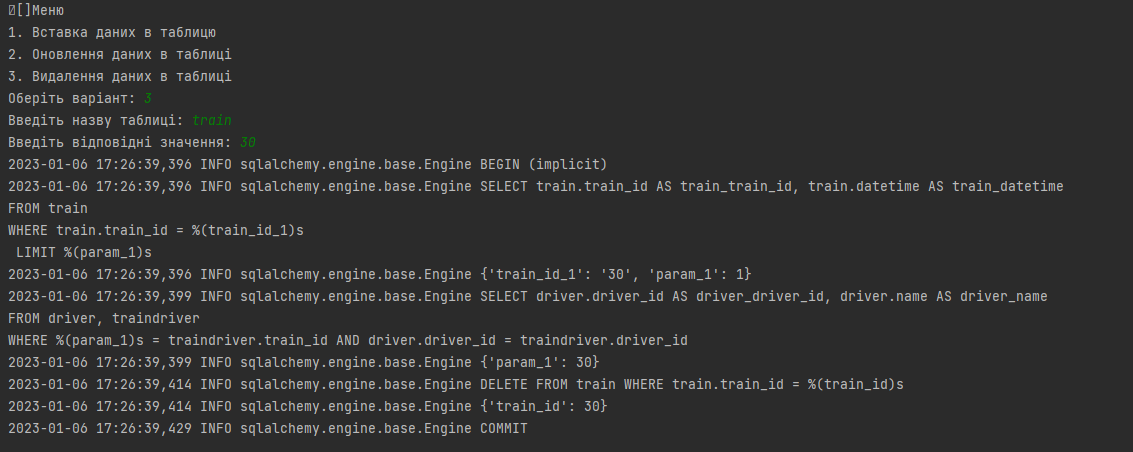


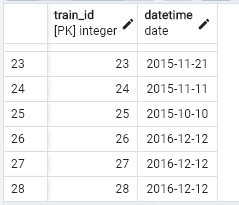
**Оновлення:**





**Видалення:**





**BTree**

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

**Створення таблиці та її заповнення:**

DROP TABLE IF EXISTS "test\_btree";

CREATE TABLE "test\_btree"(

"id" bigserial PRIMARY KEY,

"test\_text" varchar(255)

);

INSERT INTO "test\_btree"("test\_text")

SELECT

substr(characters, (random() \* length(characters) + 1)::integer, 10)

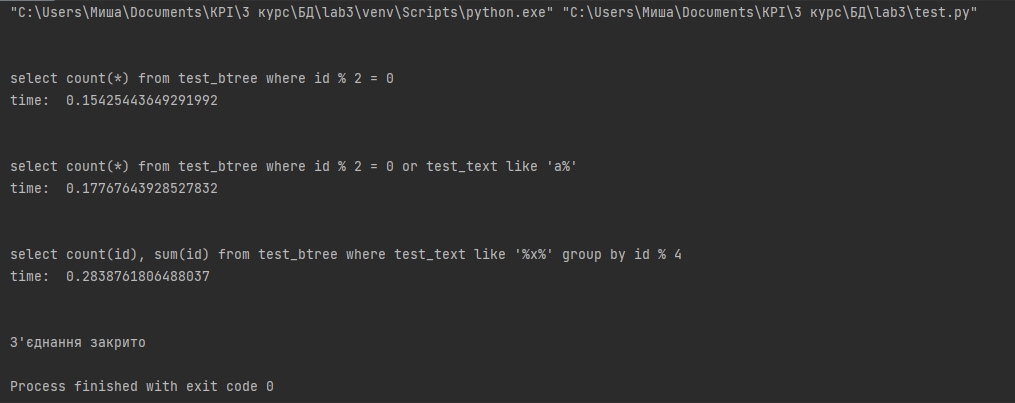
FROM

(VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM'))

as symbols(characters),

generate\_series(1, 1000000) as q;

**Вибір даних без індексу:**

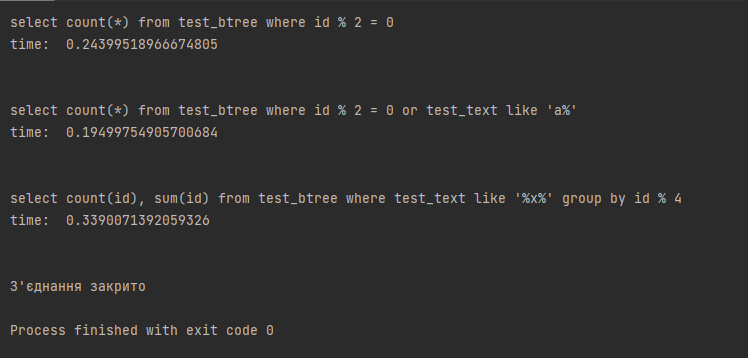


**Створюємо індекс:**

DROP INDEX IF EXISTS "test\_btree\_test\_text\_index";

CREATE INDEX "test\_btree\_test\_text\_index" ON "test\_btree" USING btree ("test\_text");

**Вибір даних з створеним індексом:**



**Код програми для тесту**

**import** time

**import** psycopg2

**from** psycopg2 **import** Error

**try**:

connection = psycopg2.connect(user="postgres",

password="111",

host="localhost",

port="5432",

database="test")

cursor = connection.cursor()

query1 = "select count(\*) from test\_btree where id % 2 = 0"

query2 = "select count(\*) from test\_btree where id % 2 = 0 or test\_text like 'a%'"

query3 = "select count(id), sum(id) from test\_btree where test\_text like '%x%' group by id % 4"

start1 = time.time()

cursor.execute(query1)

end1 = time.time() - start1

start2 = time.time()

cursor.execute(query2)

end2 = time.time() - start2

start3 = time.time()

cursor.execute(query3)

end3 = time.time() - start3

*#for row in result:*

*#print("id =", row[0])*

*#print("test\_text =", row[1], "\n")*

**print**("**\n**")

**print**("select count(\*) from test\_btree where id % 2 = 0")

**print**("time: ", end1)

**print**("**\n**")

**print**("select count(\*) from test\_btree where id % 2 = 0 or test\_text like 'a%'")

**print**("time: ", end2)

**print**("**\n**")

**print**("select count(id), sum(id) from test\_btree where test\_text like '%x%' group by id % 4")

**print**("time: ", end3)

**print**("**\n**")

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

**print**("Помилка PostgreSQL", error)

**finally**:

**if** connection:

cursor.close()

connection.close()

**print**("З'єднання закрито")

**GIN**

Для дослідження індексу була створена таблиця, яка має дві колонки: test\_text типу text[] (масив рядків) і id типу integer. Колонка test\_text проіндексована як GIN. У таблицю занесено 1000000 записів.

Створення таблиці та її заповнення:

DROP TABLE IF EXISTS "test\_gin";

CREATE TABLE "test\_gin"(

"id" bigserial PRIMARY KEY,

"test\_text" text[5]

);

DO $$

begin

for i IN 1..1000000 LOOP

INSERT INTO "test\_gin"("test\_text")

values(array[(SELECT

substr(characters, (random() \* length(characters) + 1)::integer, 10)

FROM

(VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters))

,(SELECT

substr(characters, (random() \* length(characters) + 1)::integer, 10)

FROM

(VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters))

, (SELECT

substr(characters, (random() \* length(characters) + 1)::integer, 10)

FROM

(VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters))

, (SELECT

substr(characters, (random() \* length(characters) + 1)::integer, 10)

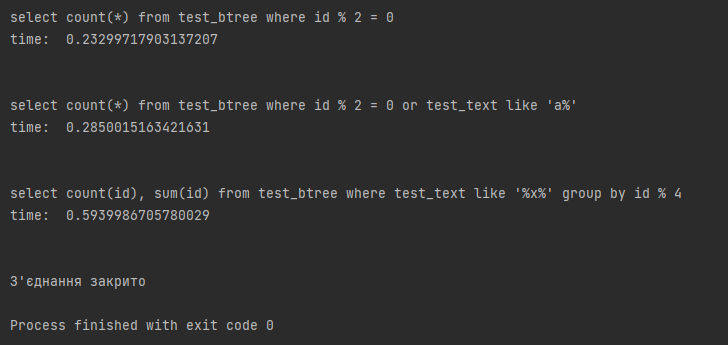
FROM

(VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters))]);

END LOOP;

end$$;

**Вибір даних без індексу:**

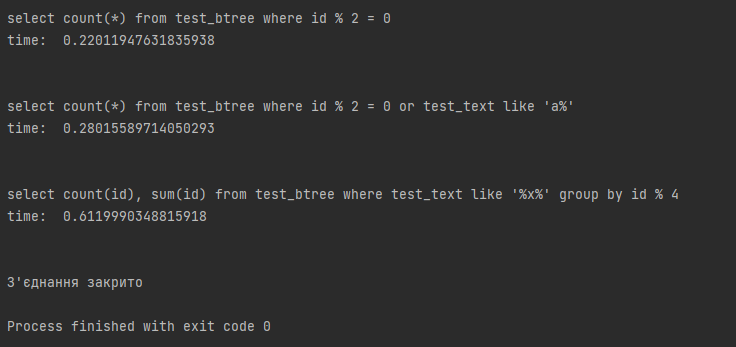


**Сворюємо індекс:**

DROP INDEX IF EXISTS "test\_gin\_test\_text\_index";

CREATE INDEX "test\_gin\_test\_text\_index" ON test\_gin USING gin (test\_text);

**Вибір даних з створеним індексом:**



**Код програми для тесту**

**import** time

**import** psycopg2

**from** psycopg2 **import** Error

**try**:

connection = psycopg2.connect(user="postgres",

password="111",

host="localhost",

port="5432",

database="test")

cursor = connection.cursor()

query1 = "select count(\*) from test\_gin where id % 2 = 0"

query2 = "select count(\*) from test\_gin where id % 2 = 0 or test\_text[1] like '%f%'"

query3 = "select count(id), sum(id) from test\_gin where test\_text[2] like '%x%' group by id % 2"

start1 = time.time()

cursor.execute(query1)

end1 = time.time() - start1

start2 = time.time()

cursor.execute(query2)

end2 = time.time() - start2

start3 = time.time()

cursor.execute(query3)

end3 = time.time() - start3

*#for row in result:*

*#print("id =", row[0])*

*#print("test\_text =", row[1], "\n")*

**print**("**\n**")

**print**("select count(\*) from test\_btree where id % 2 = 0")

**print**("time: ", end1)

**print**("**\n**")

**print**("select count(\*) from test\_btree where id % 2 = 0 or test\_text like 'a%'")

**print**("time: ", end2)

**print**("**\n**")

**print**("select count(id), sum(id) from test\_btree where test\_text like '%x%' group by id % 4")

**print**("time: ", end3)

**print**("**\n**")

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

**print**("Помилка PostgreSQL", error)

**finally**:

**if** connection:

cursor.close()

connection.close()

**print**("З'єднання закрито")

В моєму випадку індекси в деяких запитах пришвидшили роботу пошуку, але не на багато. Побачити значну відмінність можна на великих базах даних з багатьма елементами. Тому для відносно малих баз даних немає необхідності у використанні індексів.

**Тригер бази даних PostgreSQL. Умова для тригера – after insert, update**

**after insert**

CREATE OR REPLACE FUNCTION train\_insert\_fnc()

RETURNS trigger AS

$$

BEGIN

INSERT INTO "carriage" ( "train\_id", "carriage\_type")

VALUES(NEW.current\_train\_id,NEW."P");

RETURN NEW;

END;

$$

LANGUAGE 'plpgsql';

CREATE TRIGGER train\_insert

AFTER INSERT

ON "train"

FOR EACH ROW

EXECUTE PROCEDURE train\_insert\_fnc();

**Принцип роботи тригеру**

Тригер спрацьовує після вставки в таблицю “train”. При додаванні будь-якого потяга у таблиці “carriage” буде створений вагон з таким самим значенням у полі “train\_id” як і у останнього доданого потяга.

**Приклад**

