

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»**

**Тема:**

«Засоби оптимізації роботи СУБД PostgreSQL»

Виконав: студент ІІI курсу ФПМ групи КВ-04

Пригоцький А.П.

Перевірив:

Київ – 2023

**Лабораторна робота №3**

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

*Завдання* роботи полягає у наступному:

1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.
4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

**Завдання № 1**

**Код БД:**

CREATE TABLE "products" (

"product\_id" SERIAL,

"product\_name" varchar(30) NOT NULL,

PRIMARY KEY ("product\_id")

);

CREATE TABLE "customers" (

"customer\_id" SERIAL,

"name" varchar(30) NOT NULL,

PRIMARY KEY ("customer\_id")

);

CREATE TABLE "orders" (

"order\_id" SERIAL,

"customer\_id" int NOT NULL,

"date" timestamptz default now(),

"product\_id" int NOT NULL,

PRIMARY KEY ("order\_id"),

CONSTRAINT "FK\_orders.customer\_id"

FOREIGN KEY ("customer\_id")

REFERENCES "customers"("customer\_id"),

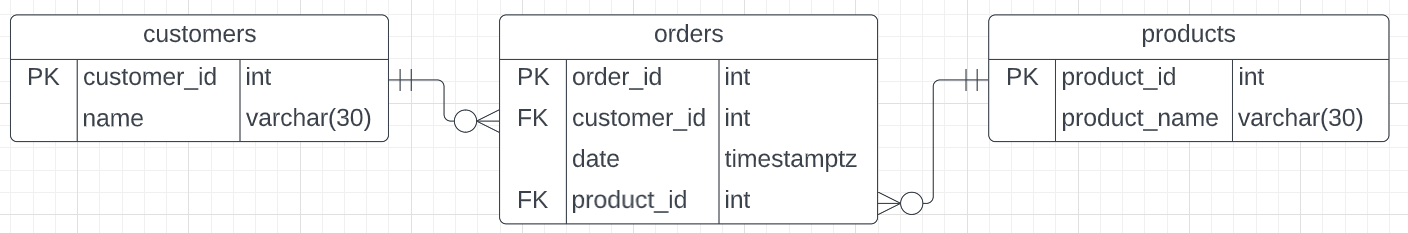
CONSTRAINT "FK\_orders.product\_id"

FOREIGN KEY ("product\_id")

REFERENCES "products"("product\_id")

);

**Схема БД:**

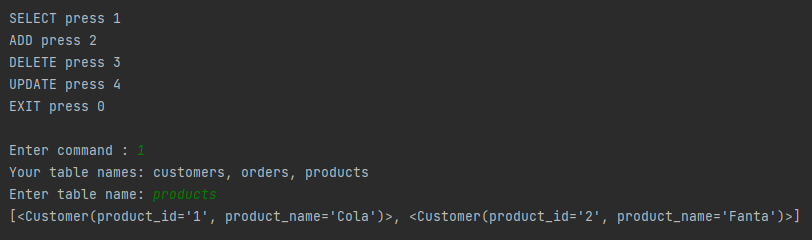
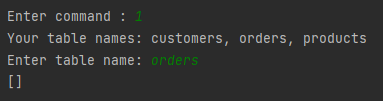
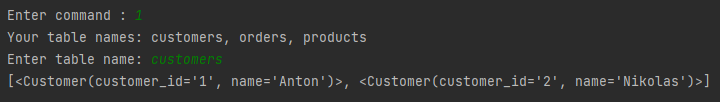


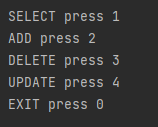
**Перетворення функцій:**

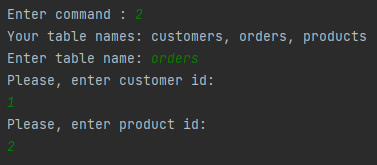
Виклики запитів замінені засобами SQLAlchemy по роботі з об’єктами. Реалізована вставка, вилучення та редагування екземплярів класів-сутностей.

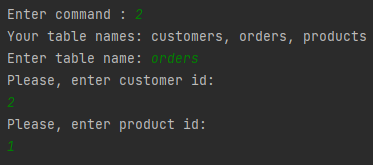
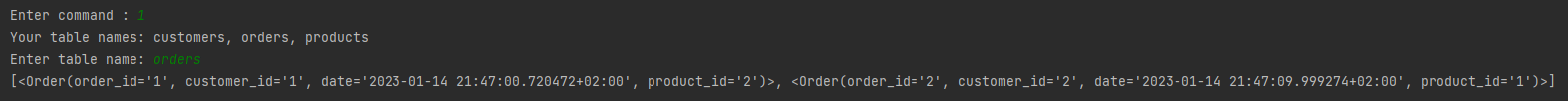
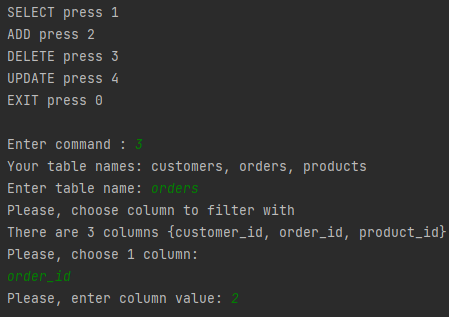
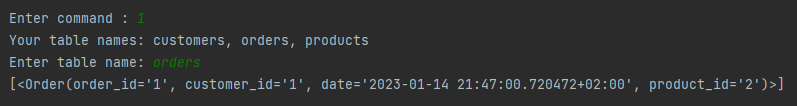
**Демонстрація роботи:**

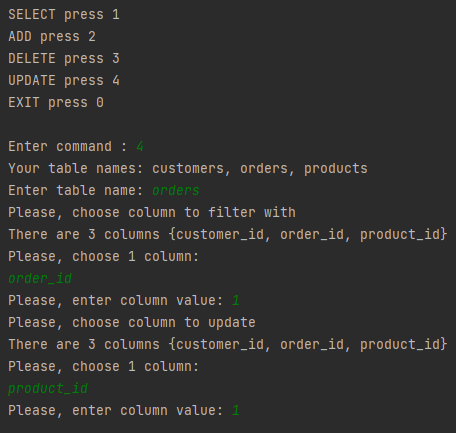
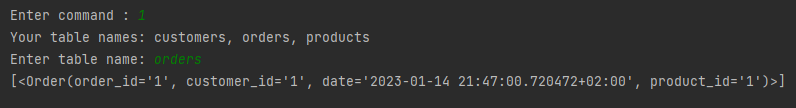
Скріншоти будуть містити меню тільки коли виконується нова команда (так воно виводиться після кожної команди).

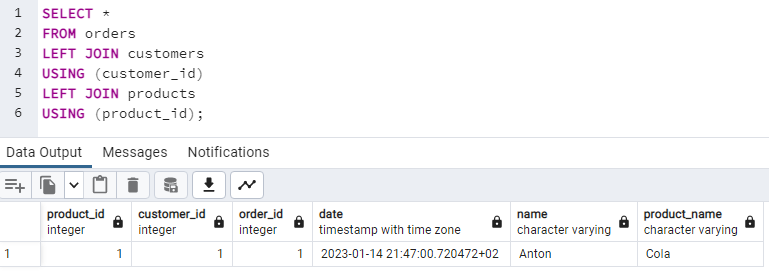
  



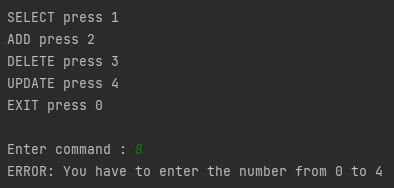


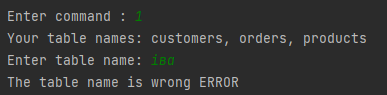
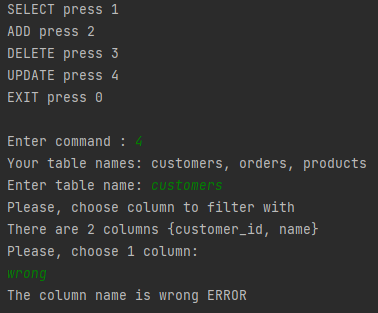
   



**Демонстрація опрацювання помилок:**



**Код програми:**

Структура кода була змінена, через те що створення класів SQLAlchemy по всім стандартам має відбуватися в окремому файлі (це дуже сильно збільшує читабельність коду).  
  
- control\_func – без змін.

- view – без змін.

- crud – файл models тепер називається саме так, щоб не було плутатини з наступним файлом.

- model – реалізація класів-таблиць БД.  
  
Функції які реалізують запроси до БД були перероблені майже повністю (код функцій), через то що використання SQLAlchemy достатньо сильно змінює реалізацію однієї й тієї ж задачі, при цьому, інтерфейси функцій залишилися без змін.

**control\_func.py**

**from** crud **import** \*

**from** view **import** \*

**def** request():

input\_command = command\_identification()

# Select function.

**if** input\_command == '1':

table\_name = table()

select(table\_name)

# Add function.

**elif** input\_command == '2':

table\_name = table()

add(values\_to\_add(table\_name))

# Delete function.

**elif** input\_command == '3':

table\_name = table()

filter\_column, filter\_value = filter\_column\_data(table\_name)

delete(table\_name, filter\_column, filter\_value)

# Update function.

**elif** input\_command == '4':

table\_name = table()

filter\_column, filter\_value = filter\_column\_data(table\_name)

update\_column, update\_value = column\_to\_update\_data(table\_name)

update(table\_name, filter\_column, filter\_value, update\_column, update\_value)

# Exit from menu.

**elif** input\_command == '0':

exiting()

**else**:

command\_error()

main()

main()

**def** main():

menu()

request()

**if** \_\_name\_\_ == '\_\_main\_\_':

# Uncomment to create new database.

# recreate\_database()

main()

**view.py**

**import** sys

**from** models **import** \*

**def** table\_invalid():

**print**('The table name is wrong ERROR')

sys.exit()

**def** column\_invalid():

**print**("The column name is wrong ERROR")

sys.exit()

**def** exiting():

**print**('Exiting')

sys.exit()

**def** command\_error():

**print**("ERROR: You have to enter the number from 0 to 4")

**def** command\_identification():

**return** input('Enter command : ')

**def** table():

**print**('Your table names: customers, orders, products')

table\_name = input('Enter table name: ')

tables = {'customers': Customer, 'orders': Order, 'products': Product}

**if** table\_name **not** **in** tables.keys():

**return** table\_invalid()

**return** tables[table\_name]

**def** values\_to\_add(table\_name):

**if** table\_name == Order:

**print**("Please, enter customer id:")

customer\_id = input()

**print**("Please, enter product id:")

product\_id = input()

**if** **not** customer\_id.isdigit() **or** **not** product\_id.isdigit():

**print**("ERROR: Id should be a number")

sys.exit()

**return** Order(int(customer\_id), int(product\_id))

**else**:

**print**(f"Please, enter {'product\_name' if table\_name == Product else 'name'}")

**return** table\_name(input())

**def** column(table\_name):

**if** table\_name == Order:

**return** Order.order\_id

**elif** table\_name == Customer:

**return** Customer.customer\_id

**else**:

**return** Product.product\_id

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

**if** table\_name == Order:

**print**('There are 3 columns {customer\_id, order\_id, product\_id}')

**print**('Please, choose 1 column:')

column = input()

columns = {'customer\_id': Order.customer\_id, 'order\_id': Order.order\_id, 'product\_id': Order.product\_id}

**if** column **not** **in** columns.keys():

**return** column\_invalid()

**return** columns[column]

**elif** table\_name == Customer:

**print**('There are 2 columns {customer\_id, name}')

**print**('Please, choose 1 column:')

column = input()

columns = {'customer\_id': Customer.customer\_id, 'name': Customer.name}

**if** column **not** **in** columns.keys():

**return** column\_invalid()

**return** columns[column]

**else**:

**print**('There are 2 columns {product\_id, product\_name}')

**print**('Please, choose 1 column:')

column = input()

columns = {'product\_id': Product.product\_id, 'product\_name': Product.product\_name}

**if** column **not** **in** columns.keys():

**return** column\_invalid()

**return** columns[column]

**def** get\_data():

**return** input('Please, enter column value: ')

**def** filter\_column\_data(table\_name):

**print**("Please, choose column to filter with")

filter\_column = get\_column(table\_name)

filter\_value = get\_data()

**return** filter\_column, filter\_value

**def** column\_to\_update\_data(table\_name):

**print**("Please, choose column to update")

update\_column = get\_column(table\_name)

update\_value = get\_data()

**return** update\_column, update\_value

**def** menu():

**print**()

**print**("SELECT press 1")

**print**("ADD press 2")

**print**("DELETE press 3")

**print**("UPDATE press 4")

**print**('EXIT press 0')

**print**()

**models.py**

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

**from** sqlalchemy.orm **import** relationship

**from** sqlalchemy.sql **import** func

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

Base = declarative\_base()

**class** Customer(Base):

\_\_tablename\_\_ = 'customers'

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

name = Column(String, nullable=False)

orders = relationship("Order", cascade="all,delete", passive\_deletes=True)

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

self.name = name

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

**return** "<Customer(customer\_id='{}', name='{}')>" \

.format(self.customer\_id, self.name)

**class** Order(Base):

\_\_tablename\_\_ = 'orders'

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

customer\_id = Column(Integer, ForeignKey('customers.customer\_id', ondelete='CASCADE'))

date = Column(DateTime(timezone=True), server\_default=func.now())

product\_id = Column(Integer, ForeignKey('products.product\_id', ondelete='CASCADE'), nullable=False)

**def** \_\_init\_\_(self, customer\_id, product\_id):

self.customer\_id = customer\_id

self.product\_id = product\_id

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

**return** "<Order(order\_id='{}', customer\_id='{}', date='{}', product\_id='{}')>" \

.format(self.order\_id, self.customer\_id, self.date, self.product\_id)

**class** Product(Base):

\_\_tablename\_\_ = 'products'

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

product\_name = Column(String, nullable=False)

products = relationship("Order", cascade="all,delete", passive\_deletes=True)

**def** \_\_init\_\_(self, product\_name):

self.product\_name = product\_name

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

**return** "<Customer(product\_id='{}', product\_name='{}')>" \

.format(self.product\_id, self.product\_name)

**crud.py**

**import** sys

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

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

**from** sqlalchemy.exc **import** SQLAlchemyError

**from** view **import** \*

engine = create\_engine('postgresql://postgres:password@localhost:5432/test')

Session = sessionmaker(bind=engine)

**def** db\_error(err):

**print**("WARNING: Error has occurred\n")

**print**(err)

sys.exit(-1)

**def** table\_nf(table\_name):

**print**(f"ERROR: Table {table\_name} was not found in the database")

sys.exit(-1)

**def** recreate\_database():

Base.metadata.drop\_all(engine)

Base.metadata.create\_all(engine)

**def** delete(table, column, row):

s = Session()

**try**:

s.query(table).filter(column == row).delete()

s.commit()

**except** SQLAlchemyError **as** err:

db\_error(err)

s.close()

**def** add(element):

s = Session()

**try**:

s.add(element)

s.commit()

**except** SQLAlchemyError **as** err:

db\_error(err)

s.close()

**def** select(table):

s = Session()

**try**:

# Change to return if needed.

**print**(s.query(table).all())

**except** SQLAlchemyError **as** err:

db\_error(err)

**def** update(table, filter\_column, filter\_column\_value, column\_to\_upd, updated\_value):

s = Session()

**try**:

s.query(table).filter(filter\_column == filter\_column\_value).update({column\_to\_upd: updated\_value})

s.commit()

**except** SQLAlchemyError **as** err:

db\_error(err)

s.close()

**Завдання №2**

**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:**

DROP TABLE IF EXISTS "test\_btree";

CREATE TABLE "test\_btree"(

"id" bigserial PRIMARY KEY, "test\_text" varchar(255)

);

**Заповнення таблиці:**

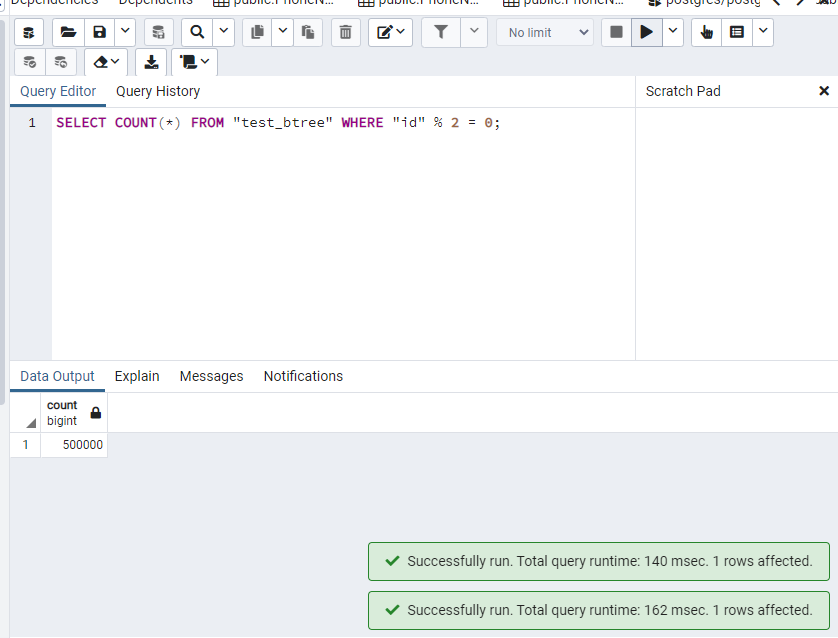
INSERT INTO "test\_btree"("test\_text") SELECT substr(characters, (random() \*

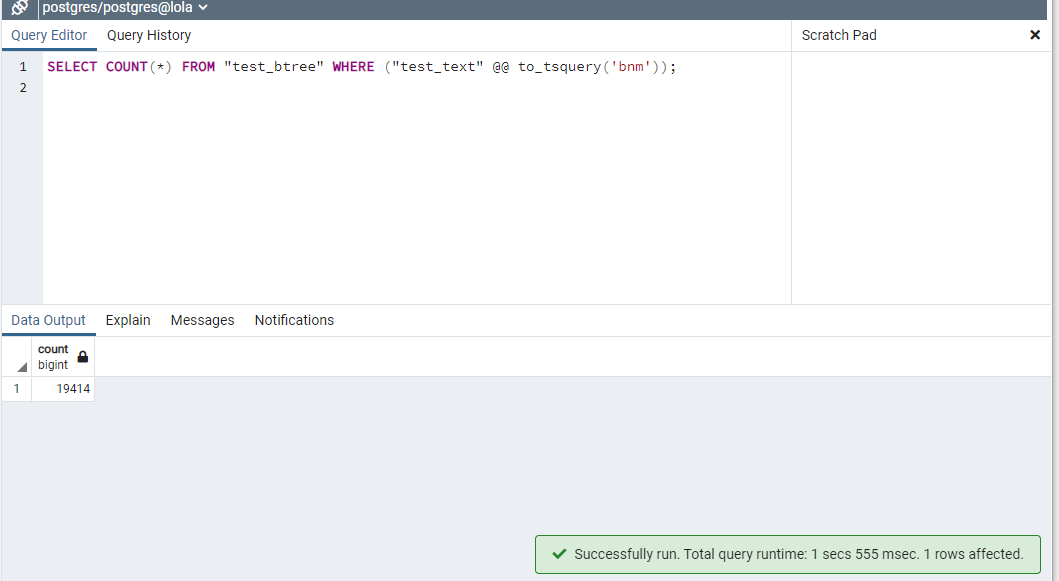
length(characters) + 1)::integer, 10) FROM

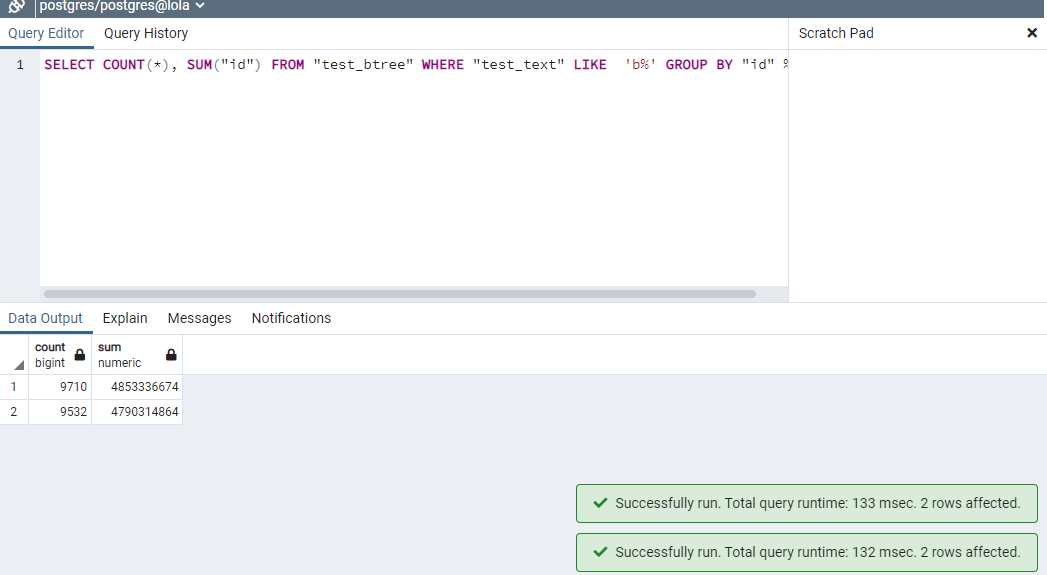
(VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as

symbols(characters), generate\_series(1, 1000000) as q;

**Вибір даних без індексу:**





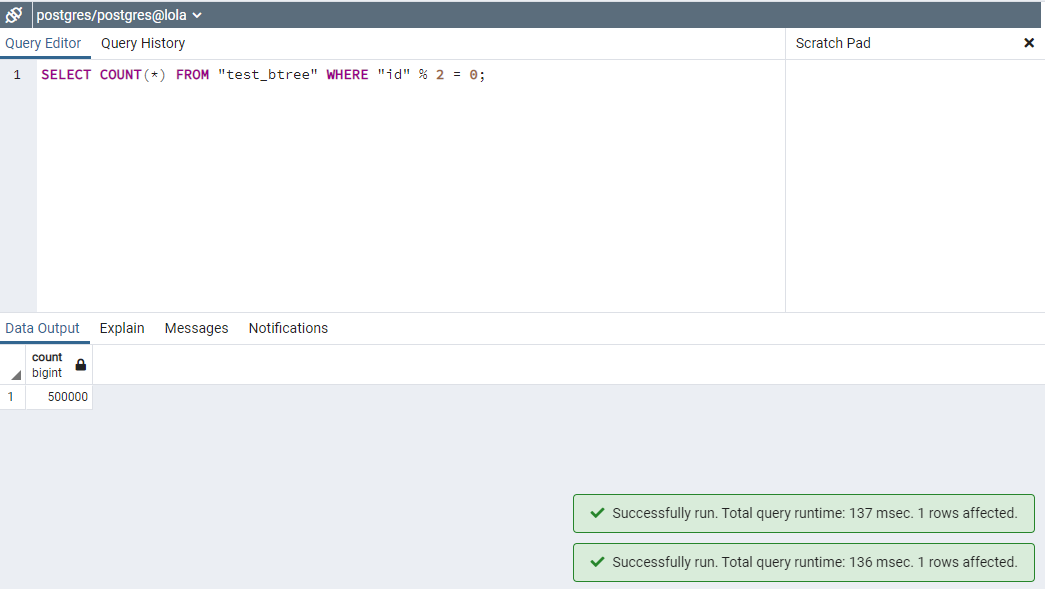


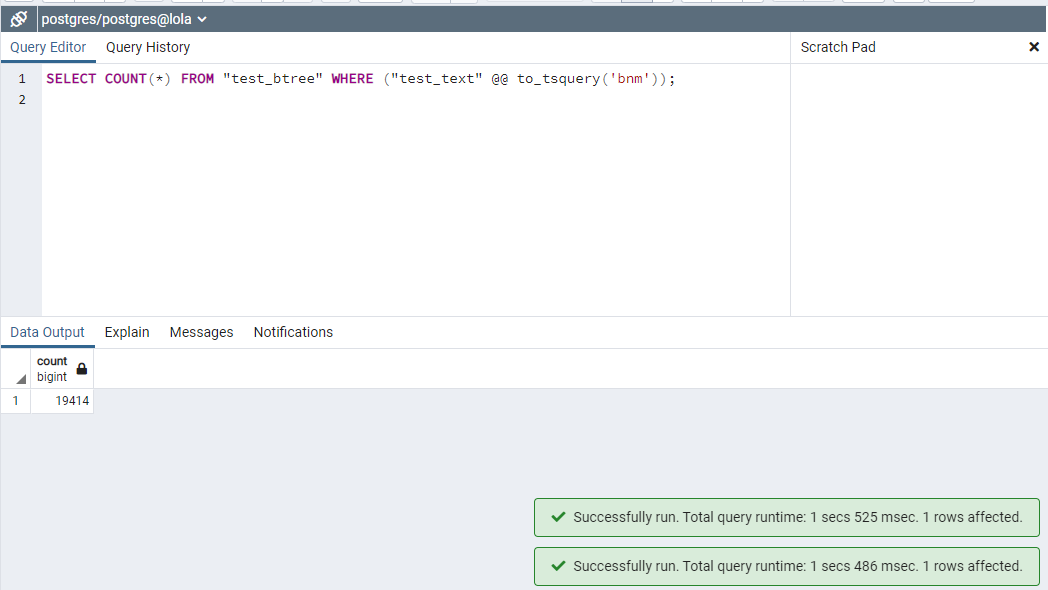
**Створюємо індекс:**

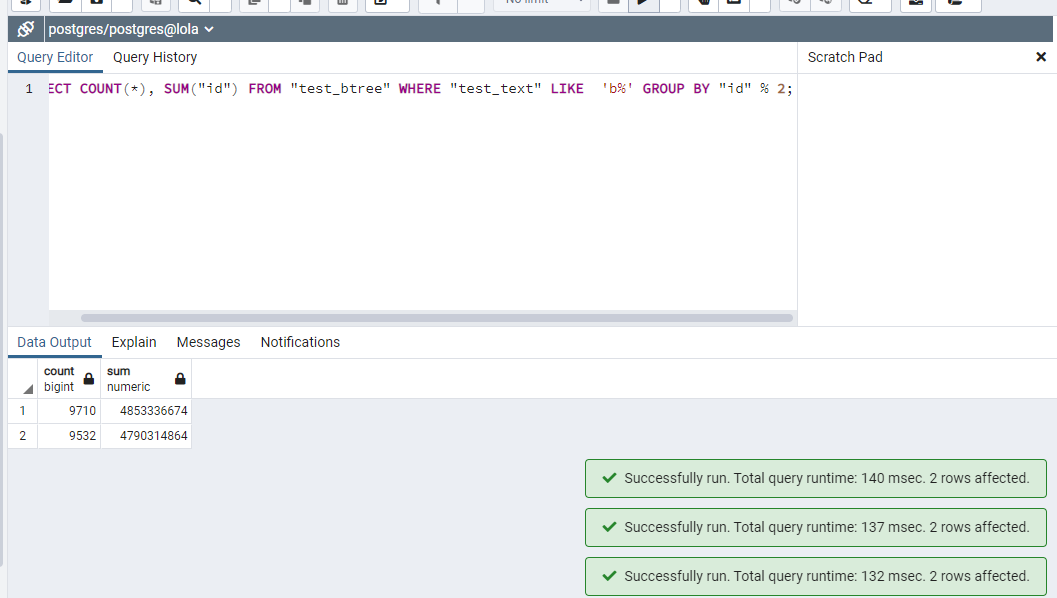
DROP INDEX IF EXISTS "test\_btree\_test\_text\_index";

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

**Вибір даних з створеним індексом:**







**BRIN**

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.

**Створення таблиці:**

DROP TABLE IF EXISTS "test\_brin";

CREATE TABLE "test\_brin"(

"id" bigserial PRIMARY KEY, "test\_time" timestamp

);

**Заповнення таблиці:**

INSERT INTO "test\_brin"("test\_time")

SELECT

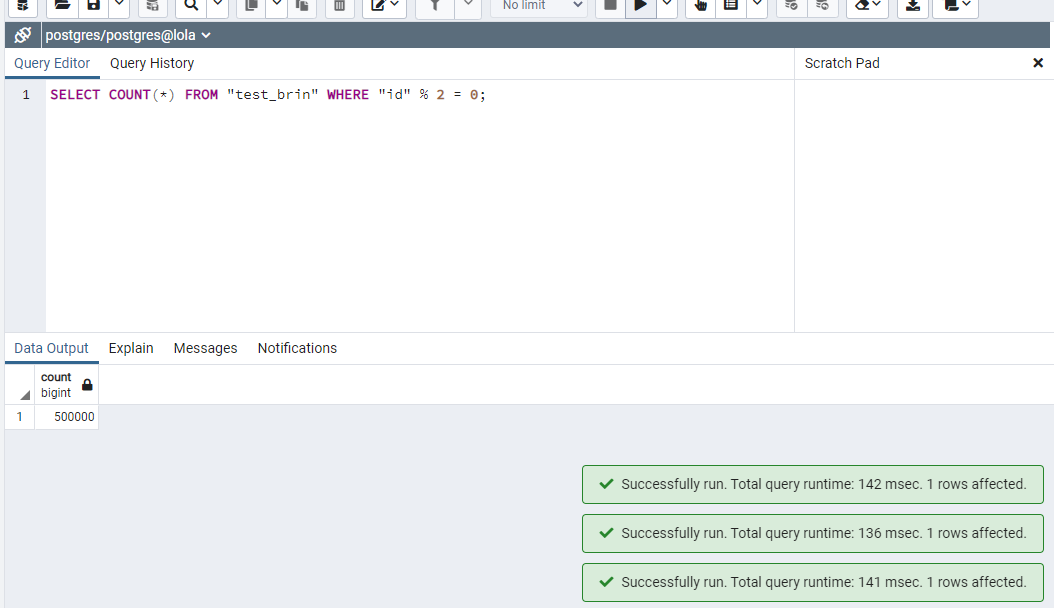
FROM

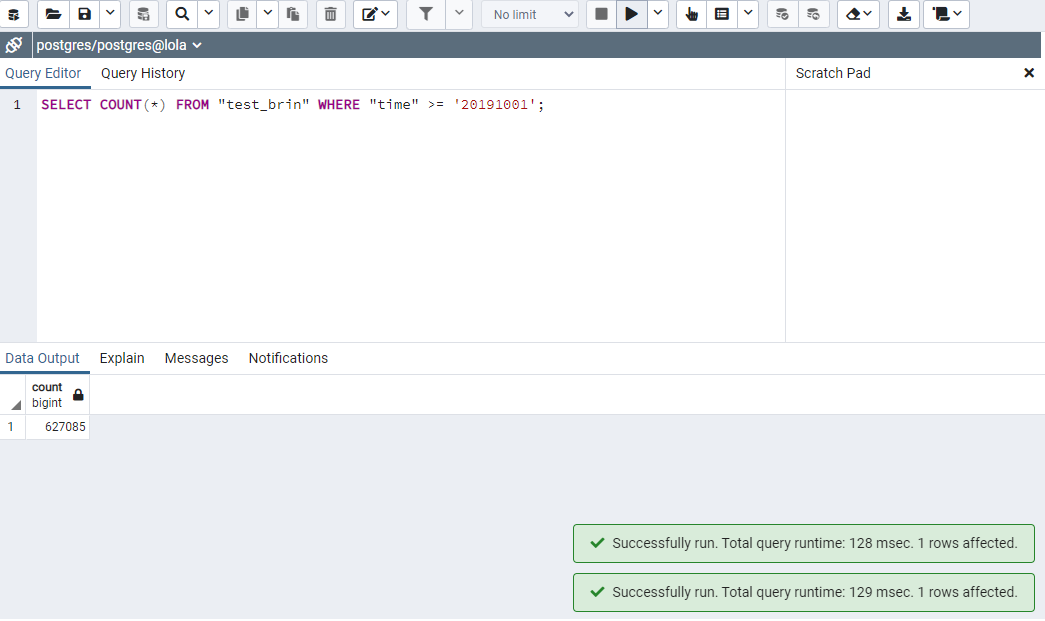
(timestamp '2021-01-01' + random() \* (timestamp '2020-01-01' - timestamp '2022-01-01'))

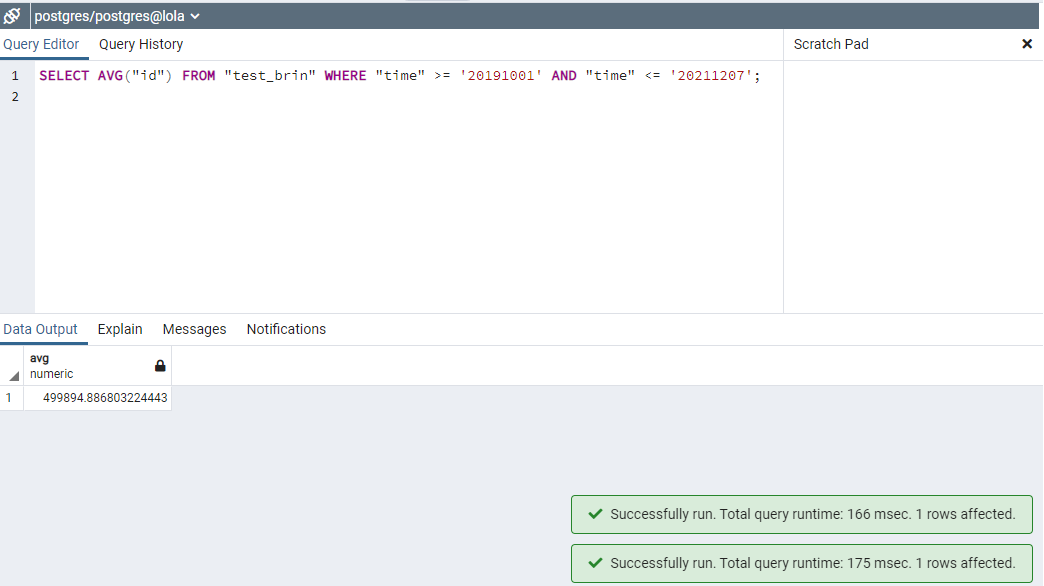
(VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),

generate\_series(1, 1000000) as q;

**Вибір даних без індексу:**





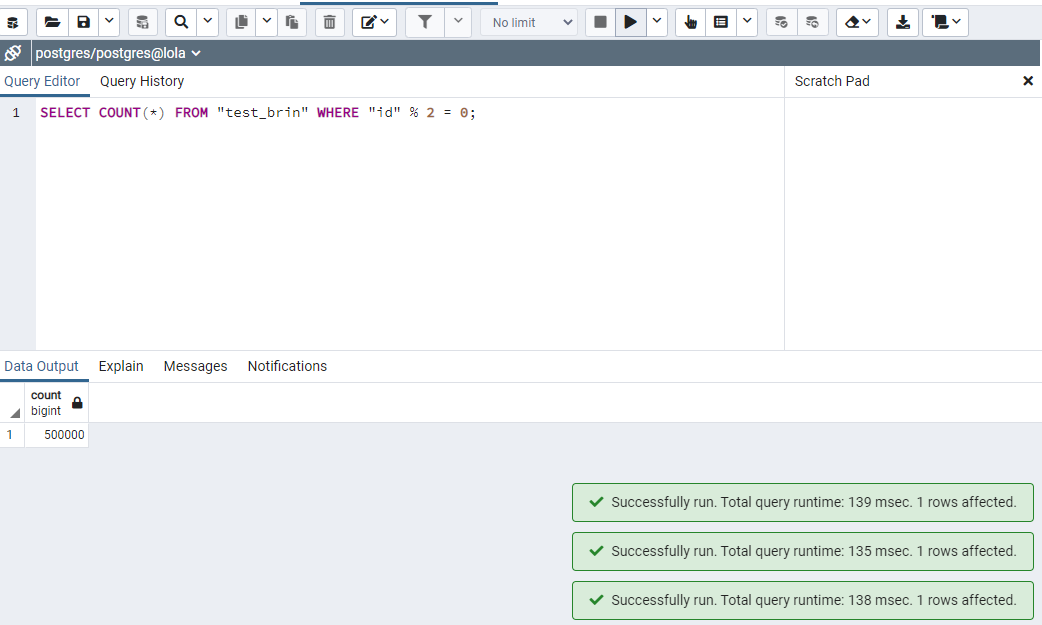


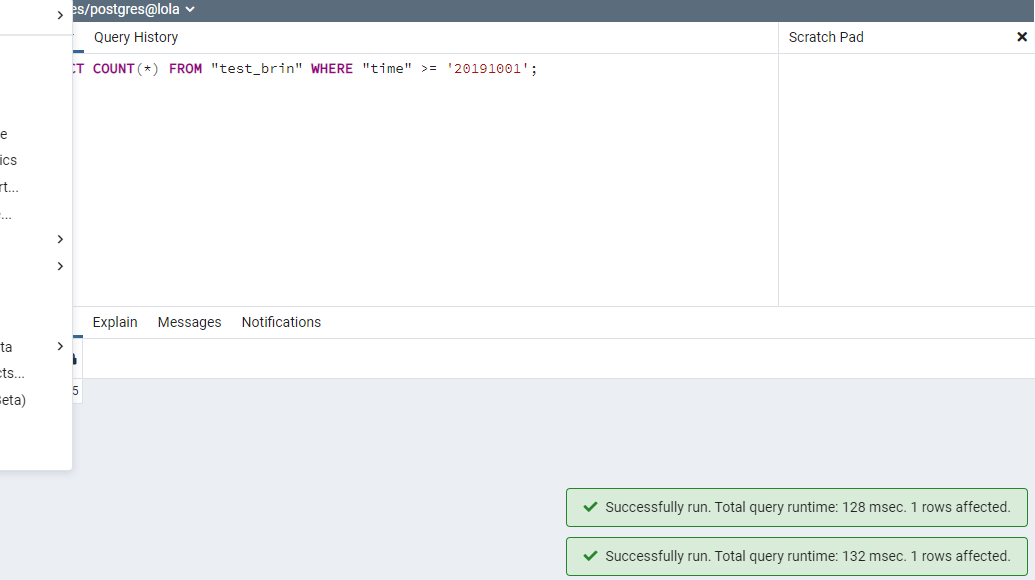
**Створюємо індекс:**

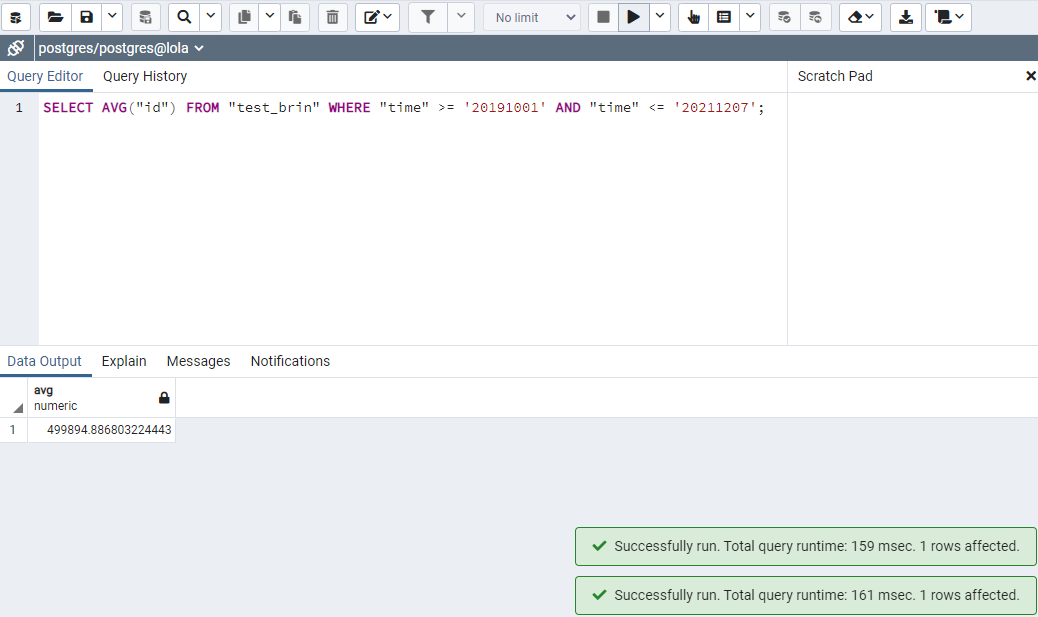
DROP INDEX IF EXISTS "test\_brin\_time\_index";

CREATE INDEX "test\_brin\_time\_index" ON "test\_brin" USING brin("time")

**Вибір даних з створеним індексом:**







**Завдання №3**

Розробити тригер бази даних PostgreSQL.

Умова для тригера – before update, delete.

**Таблиці:**

DROP TABLE IF EXISTS "reader";

CREATE TABLE "reader"(

"readerID" bigserial PRIMARY KEY, "readerName" varchar(255)

);

DROP TABLE IF EXISTS "readerLog";

CREATE TABLE "readerLog"(

"id" bigserial PRIMARY KEY, "readerLogID" bigint, "readerLogName" varchar(255)

);

**Тригер:**

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

DECLARE

CURSOR\_LOG CURSOR FOR SELECT \* FROM "readerLog";

row\_Log "readerLog"%ROWTYPE;

begin

IF old."readerID" % 2 = 0 THEN

INSERT INTO "readerLog"("readerLogID", "readerLogName") VALUES (old."readerID",

old."readerName");

UPDATE "readerLog" SET "readerLogName" = trim(BOTH 'x' FROM "readerLogName");

RETURN NEW;

ELSE

RAISE NOTICE 'readerID is odd';

FOR row\_log IN cursor\_log LOOP

UPDATE "readerLog" SET "readerLogName" = 'x' || row\_Log."readerLogName" || 'x' WHERE "id" = row\_log."id";

END LOOP;

RETURN NEW;

END IF;

END;

$$ language plpgsql;

**Ініціалізація тригера:**

create trigger test\_trigger before update or delete on reader for each row

execute procedure update\_delete\_func();

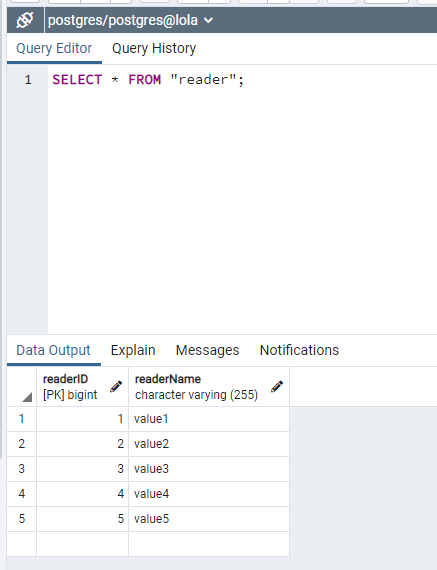
**Принцип тригера:**

Спрацьовує коли проходить оновлення чи видалення рядка з парним номером і заноситься у таблицю Logs.

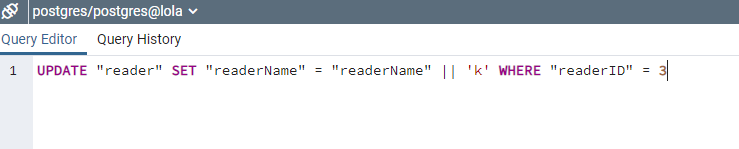
**Ініцалізуємо таблицю:**

INSERT INTO "reader"("readerName")

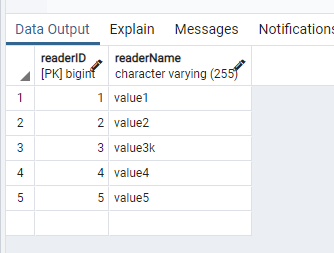
VALUES ('value1'), ('value2'), ('value3'), ('value4'), ('value5');

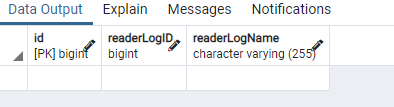


Робимо запит:

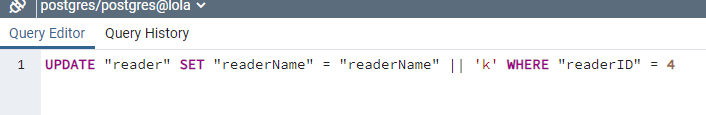


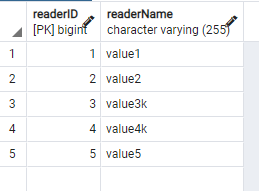
Після виконання запиту, бачимо, що у рядок за номером 3 було додано “k” та не був доданий рядок у logs.

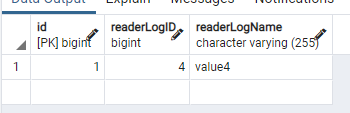




Виконуємо такий же запит для 4 рядка:



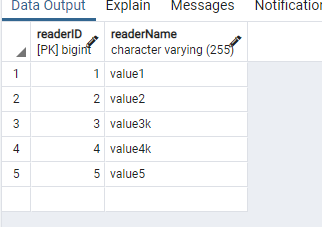


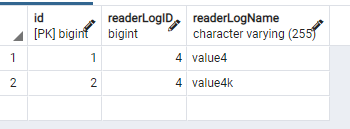


Як бачимо, тригер спрацював і зробив запис до Logs.

Робиму запит на видалення за номером рядка 4:







Тригер спрацював на видалення також.