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 III курсу

групи КВ-93

Гаращук Б.В

Перевірив:

Київ – 2021

Постановка задачі

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.

5	*BTree, GIN*	*before update, delete*

Посилання на репозиторій GitHub з вихідним кодом програми та звітом: https://github.com/Bodgen/database/tree/main/Lab2

Використана мова програмування: Python 3.7.2

Використані бібліотеки: psycopg2 (для зв’язку з СУБД), sqlalchemy (для реалізації ORM) та інші.

Завдання №1

Обрана предметна галузь передбачає можливість користувачеві вибрати зі списку товар і додати його у свою корзину.

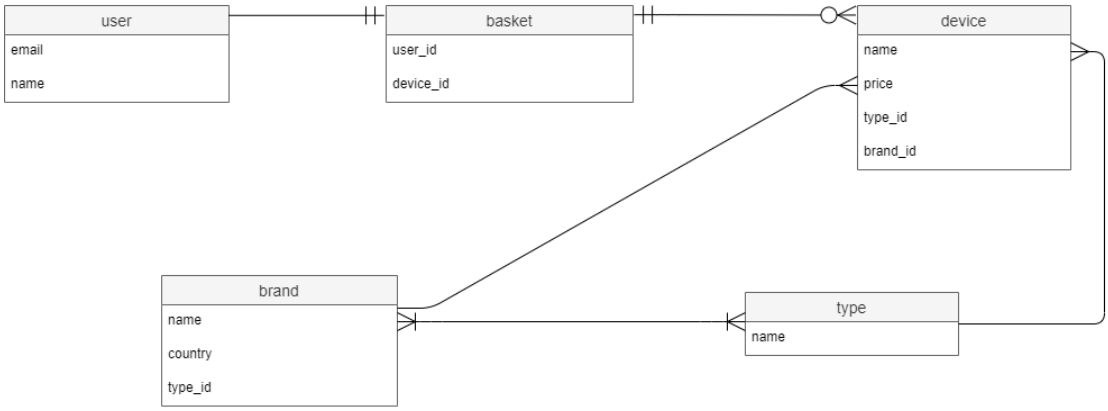


Рисунок 1. Схема бази даних

Програмний модуль «model.py», з реалізованими класами ORM:

```
import datetime
from sqlalchemy import Column, Integer, String, 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 User(Orders):
    __tablename__ = 'user'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    email = Column(String)
    baskets = relationship("Basket")

    def __init__(self, id, name, email):
        self.id = id
        self.name = name
        self.email = email

    def __repr__(self):
        return "{:>10}{:>30}{:>30}" \
            .format(self.id, self.name, self.email)


class Basket(Orders):
    __tablename__ = 'basket'
```

```
    id = Column(Integer, primary_key=True)
    device_id = Column(Integer, ForeignKey('device.id'))
    user_id = Column(Integer, ForeignKey('user.id'))

    def __init__(self, id, device_id, user_id):
        self.id = id
        self.device_id = device_id
        self.user_id = user_id

    def __repr__(self):
        return "{:>10}{:>15}{:>10}" \
            .format(self.id, self.device_id, self.user_id)


class Device(Orders):
    __tablename__ = 'device'
    id = Column(Integer, primary_key=True)
    price = Column(Integer)
    type_id = Column(Integer, ForeignKey('type.id'))
    brand_id = Column(Integer, ForeignKey('brand.id'))
    name = Column(String)
    baskets = relationship("Basket")

    def __init__(self, id, price, type_id, brand_id, name):
        self.id = id
        self.price = price
        self.type_id = type_id
        self.brand_id = brand_id
        self.name = name

    def __repr__(self):
        return "{:>10}{:>15}{:>10}{:>15}{:>25}" \
            .format(self.id, self.price, self.type_id, self.brand_id,
self.name)


class Type(Orders):
    __tablename__ = 'type'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    devices = relationship('Device')

    def __init__(self, id, name):
        self.id = id
        self.name = name

    def __repr__(self):
        return "{:>10}{:>35}" \
            .format(self.id, self.name)
```

```
class Brand(Orders):
    __tablename__ = 'brand'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    type_id = Column(Integer, ForeignKey('type.id'))
    country = Column(String)
    devices = relationship('Device')

    def __init__(self, id, name, type_id, country):
        self.id = id
        self.name = name
        self.type_id = type_id
        self.country = country

    def __repr__(self):
        return "{:>10}{:>35}{:>10}{:>35}" \
            .format(self.id, self.name, self.type_id, self.country)


class Model:
    def __init__(self):
        self.session = Session()
        self.connection = engine.connect()

    def find_pk_basket(self, key_value: int):
        return
self.session.query(Basket).filter_by(id=key_value).first()

    def find_fk_basket(self, key_value: int, table_name: str):
        if table_name == "device":
            return
self.session.query(Device).filter_by(device_id=key_value).first()
        if table_name == "user":
            return
self.session.query(User).filter_by(id=key_value).first()

    def find_pk_device(self, key_value: int):
        return
self.session.query(Device).filter_by(id=key_value).first()

    def find_fk_device(self, key_value: int, table_name: str):
        if table_name == "type":
            return
self.session.query(Type).filter_by(type_id=key_value).first()
        if table_name == "brand":
            return
self.session.query(Brand).filter_by(brand_id=key_value).first()

    def find_pk_user(self, key_value: int):
        return
self.session.query(User).filter_by(id=key_value).first()
```

```
    def find_pk_type(self, key_value: int):
        return
self.session.query(Type).filter_by(id=key_value).first()

    def find_pk_brand(self, key_value: int):
        return
self.session.query(Brand).filter_by(id=key_value).first()

    def find_fk_brand(self, key_value: int):
        return
self.session.query(Type).filter_by(type_id=key_value).first()

    def print_users(self):
        return self.session.query(User).order_by(User.id.asc()).all()

    def print_basket(self):
        return
self.session.query(Basket).order_by(Basket.id.asc()).all()

    def print_device(self):
        return
self.session.query(Device).order_by(Device.id.asc()).all()

    def print_type(self):
        return self.session.query(Type).order_by(Type.id.asc()).all()

    def print_brand(self):
        return
self.session.query(Brand).order_by(Brand.id.asc()).all()

    def delete_data_user(self, id) -> None:
        self.session.query(User).filter_by(id=id).delete()
        self.session.commit()

    def delete_data_basket(self, id) -> None:
        self.session.query(Basket).filter_by(id=id).delete()
        self.session.commit()

    def delete_data_device(self, id) -> None:
        self.session.query(Device).filter_by(id=id).delete()
        self.session.commit()

    def delete_data_type(self, id) -> None:
        self.session.query(Type).filter_by(id=id).delete()
        self.session.commit()

    def delete_data_brand(self, id) -> None:
        self.session.query(Brand).filter_by(id=id).delete()
        self.session.commit()
```

```
    def update_data_user(self, id: int, name: str, email: str) ->
None:
        self.session.query(User).filter_by(id=id) \
            .update({User.name: name, User.email: email})
        self.session.commit()

    def update_data_basket(self, id: int, device_id: int, user_id:
int) -> None:
        self.session.query(Basket).filter_by(id=id) \
            .update({Basket.device_id: device_id, Basket.user_id:
user_id})
        self.session.commit()

    def update_data_device(self, id: int, price: int, type_id: int,
brand_id: int, name: str) -> None:
        self.session.query(Device).filter_by(id=id) \
            .update({Device.price: price, Device.type_id: type_id,
Device.brand_id: brand_id, Device.name: name})
        self.session.commit()

    def update_data_type(self, id: int, name: str) -> None:
        self.session.query(Type).filter_by(id=id) \
            .update({Type.name: name})
        self.session.commit()

    def update_data_brand(self, id: int, name: str, type_id: int,
country: str) -> None:
        self.session.query(Brand).filter_by(id=id) \
            .update({Brand.name: name, Brand.type_id: type_id,
Brand.country: country})
        self.session.commit()

    def insert_data_user(self, id: int, name: str, email: str) ->
None:
        user = User(id=id, name=name, email=email)
        self.session.add(user)
        self.session.commit()

    def insert_data_basket(self, id: int, device_id: int, user_id:
int) -> None:
        basket = Basket(id=id, device_id=device_id, user_id=user_id)
        self.session.add(basket)
        self.session.commit()

    def insert_data_device(self, id: int, price: int, type_id: int,
brand_id: int, name: str) -> None:
        device = Device(id=id, price=price, type_id=type_id,
brand_id=brand_id, name=name)
        self.session.add(device)
        self.session.commit()
```

```
    def insert_data_type(self, id: int, name: str) -> None:
        types = Type(id=id, name=name)
        self.session.add(types)
        self.session.commit()

    def insert_data_brand(self, id: int, name: str, type_id: int,
country: str) -> None:
        brand = Brand(id=id, name=name, type_id=type_id,
country=country)
        self.session.add(brand)
        self.session.commit()

    def user_data_generator(self, times: int) -> None:
        for i in range(times):
            self.connection.execute("insert into public.\"user\""
                                    "select (SELECT MAX(id)+1 FROM
public.\"user\"), "
                                    "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)), ''); ")

    def basket_data_generator(self, times: int) -> None:
        for i in range(times):
            self.connection.execute("insert into public.\"basket\"
select (SELECT (MAX(id)+1) FROM public.\"basket\"), "
                                    "(SELECT id FROM public.\"device\"
LIMIT 1 OFFSET (round(random() * "
                                    "((SELECT COUNT(id) FROM
public.\"device\")-1)))),"
                                    "(SELECT id FROM public.\"user\"
LIMIT 1 OFFSET "
                                    "(round(random() * ((SELECT
COUNT(id) FROM public.\"user\")-1))));")

    def device_data_generator(self, times: int) -> None:
        for i in range(times):
            self.connection.execute("insert into public.\"device\"
select (SELECT MAX(id)+1 FROM public.\"device\"), "
                                    "FLOOR(RANDOM()*(100000-1)+1),"
                                    "(SELECT id FROM public.\"type\"
LIMIT 1 OFFSET "
                                    "(round(random() *((SELECT
COUNT(id) FROM public.\"type\")-1)))), "
                                    "(SELECT id FROM public.\"brand\"
LIMIT 1 OFFSET "
                                    "(round(random() * ((SELECT
```

```
COUNT(id) FROM public.\"brand\")-1))))),"
                                    "array_to_string(ARRAY(SELECT
chr((97 + round(random() * 25)) :: integer) "
                                    "FROM generate_series(1,
FLOOR(RANDOM()*(25-10)+10):: integer)), '')  ;")

    def type_data_generator(self, times: int) -> None:
        for i in range(times):
            self.connection.execute("insert into public.\"type\"
select (SELECT MAX(id)+1 FROM public.\"type\"), "
                                    "array_to_string(ARRAY(SELECT
chr((97 + round(random() * 25)) :: integer) "
                                    "FROM generate_series(1,
FLOOR(RANDOM()*(25-10)+10):: integer)), ''); ")

    def brand_data_generator(self, times: int) -> None:
        for i in range(times):
            self.connection.execute("insert into public.\"brand\"
select (SELECT MAX(id)+1 FROM public.\"brand\"), "
                                    "(SELECT id FROM public.\"type\"
LIMIT 1 OFFSET "
                                    "(round(random() *((SELECT
COUNT(id) FROM public.\"type\")-1)))), "
                                    "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()*(25-10)+10):: integer)), ''); ")

    def search_data_two_tables(self):
        return self.session.query(User) \
            .join(Basket) \
            .filter(and_(
            User.id.between(0, 5),
            Basket.id.between(0, 5)
        )) \
            .all()

    def search_data_three_tables(self):
        return self.session.query(Basket) \
            .join(Device).join(Type) \
            .filter(and_(
            Basket.device_id.between(0, 5),
            Device.id.between(0, 5),
            Type.id.between(0, 5)
        )) \
            .all()
```

```
    def search_data_four_tables(self):
        return self.session.query(User) \
            .join(Basket).join(Device).join(Brand) \
            .filter(and_(
            User.id.between(0, 5),
            Basket.device_id.between(0, 5),
            Device.price.between(0, 2500),
            Brand.id.between(0, 5)
        )) \
            .all()
```

Запити у вигляді ORM

Продемонструємо вставку, вилучення, редагування данних на прикладі таблиці «basket»:

Початковий стан:

```
PS C:\Users\Bohdan\Desktop\3курс\5семестр\DB\Lab3> python main.py print_table basket
basket table:
        id      device_id   user_id
         1              2         1
         2              3         4
         3              1         5
         4              3         2
```

Видалення запису:

```
PS C:\Users\Bohdan\Desktop\3курс\5семестр\DB\Lab3> python main.py delete_record basket 4
PS C:\Users\Bohdan\Desktop\3курс\5семестр\DB\Lab3> python main.py print_table basket
basket table:
        id      device_id   user_id
         1              2         1
         2              3         4
         3              1         5
```

Вставка запису:

```
PS C:\Users\Bohdan\Desktop\3курс\5семестр\DB\Lab3> python main.py insert_record basket 4 1 1
PS C:\Users\Bohdan\Desktop\3курс\5семестр\DB\Lab3> python main.py print_table basket
basket table:
        id      device_id   user_id
         1              2         1
         2              3         4
         3              1         5
         4              1         1
```

Редагування запису:

```
PS C:\Users\Bohdan\Desktop\3курс\5семестр\DB\Lab3> python main.py update_record basket 4 2 4
PS C:\Users\Bohdan\Desktop\3курс\5семестр\DB\Lab3> python main.py print_table basket
basket table:
        id      device_id   user_id
         1              2         1
         2              3         4
         3              1         5
         4              2         4
```

Запити пошуку та генерації рандомізованих даних було також реалізовано, логіку пошуку було змінено у порівнянні з лабораторною роботою №2 (усі дані для пошуку предвизначено, тепер вони не вводяться з клавіатури). Запити на пошук ті самі, що і в ЛР№2.

Запит на генерацію даних продемонтсруємо на прикладі таблиці «device».

Початковий стан:

```
PS C:\Users\Bohdan\Desktop\3курс\5семестр\DB\Lab3> python main.py print_table device
device table:
        id          price   type_id       brand_id                   name
         1            500         1              1               device_1
         2           7000         2              3               device_2
         3           2099         3              4               device_6
         5           2099         1              2               device_6
```

Вставка 3-х випадково згенерованих записів:

```
PS C:\Users\Bohdan\Desktop\3курс\5семестр\DB\Lab3> python main.py generate_randomly device 3
PS C:\Users\Bohdan\Desktop\3курс\5семестр\DB\Lab3> python main.py print_table device
device table:
        id          price   type_id       brand_id                   name
         1            500         1              1               device_1
         2           7000         2              3               device_2
         3           2099         3              4               device_6
         5           2099         1              2               device_6
         6          33047         4              3         pnxohumtlnitwbz
         7          21779         3              4           haghuldofhntx
         8          83330         5              4        gqwoztecmnrxfmpn
```

Пошук за двома атрибутами у двох таблиця, за трьома у трьох таблицях, за чотирма атрибутами у чотирьох таблицях:

```
PS C:\Users\Bohdan\Desktop\3курс\5семестр\DB\Lab3> python main.py search_records
specify the number of tables you`d like to search in: 2
search result:
        1                       Pavlo              pavlo@gmail.com
        4                        Oleg               oleg@email.com
        5                         Den                  den@ukr.net
PS C:\Users\Bohdan\Desktop\3курс\5семестр\DB\Lab3> python main.py search_records
specify the number of tables you`d like to search in: 3
search result:
        1                       Pavlo              pavlo@gmail.com
        4                        Oleg               oleg@email.com
        5                         Den                  den@ukr.net
PS C:\Users\Bohdan\Desktop\3курс\5семестр\DB\Lab3> python main.py search_records
specify the number of tables you`d like to search in: 4
search result:
        4                        Oleg               oleg@email.com
        5                         Den                  den@ukr.net
PS C:\Users\Bohdan\Desktop\3курс\5семестр\DB\Lab3>
```

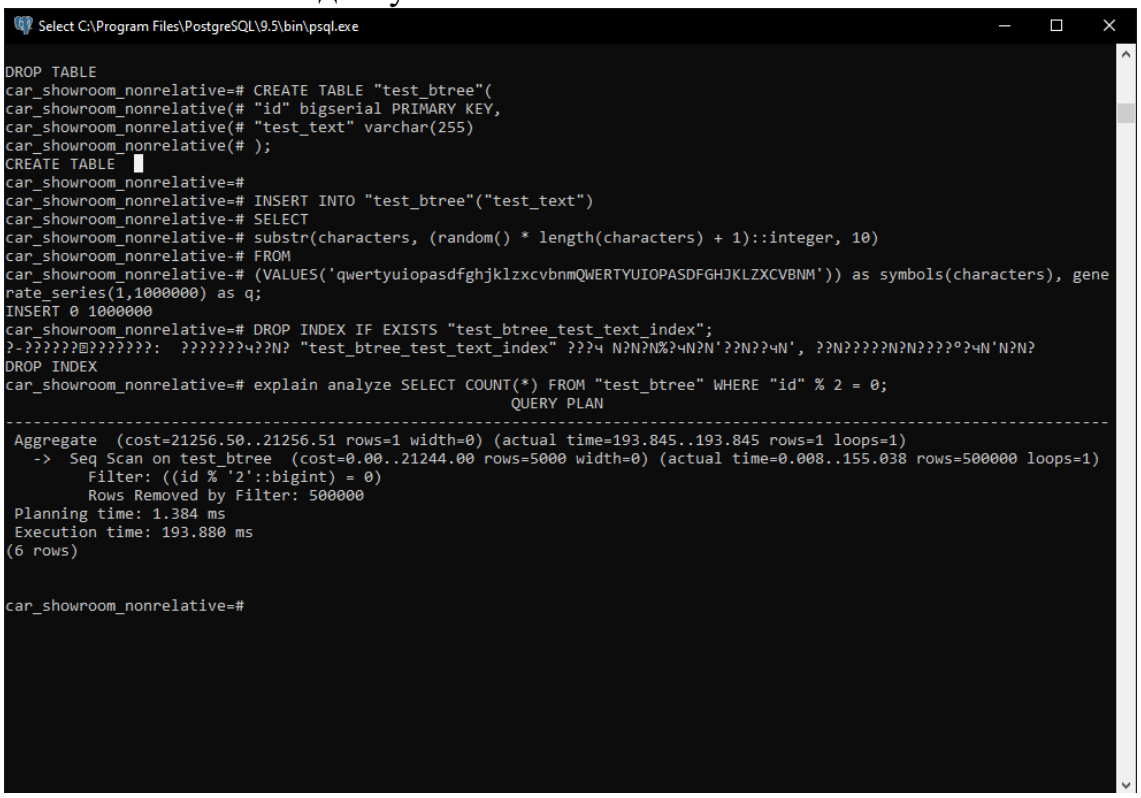
Завдання №2

Для тестування індексів було створено окремі таблиці у базі даних з 100000 записів.

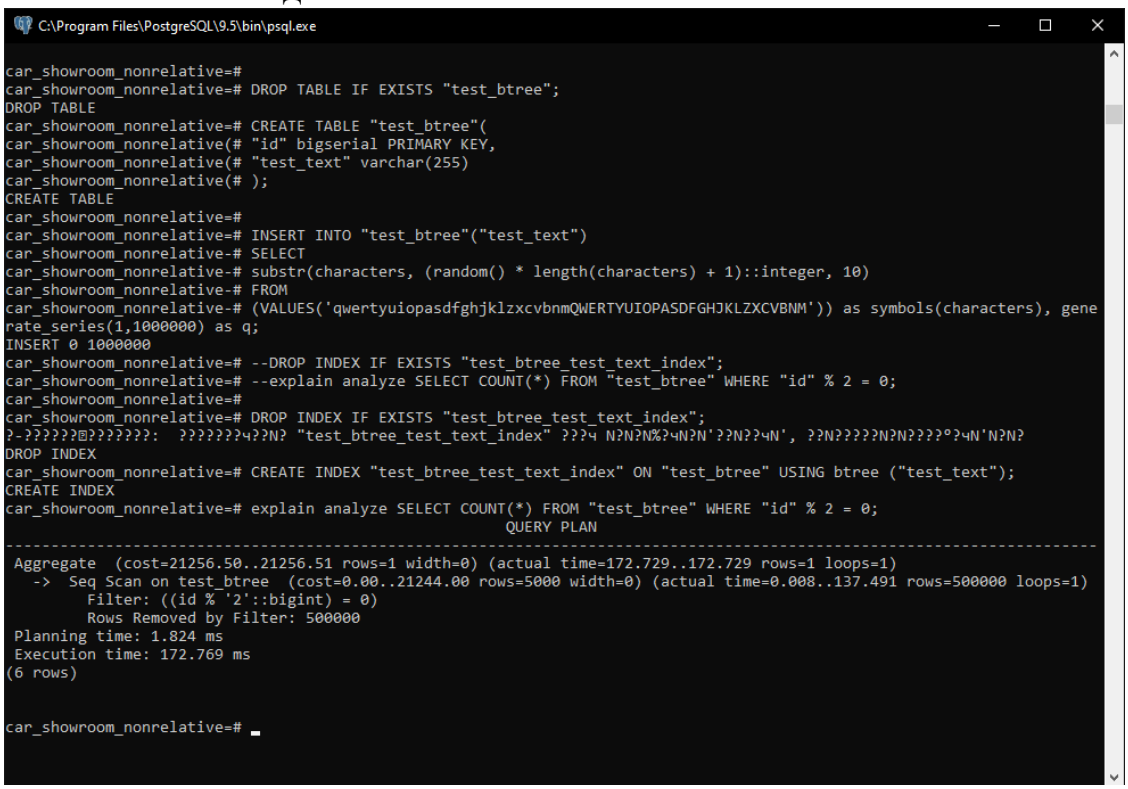
Btree

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.

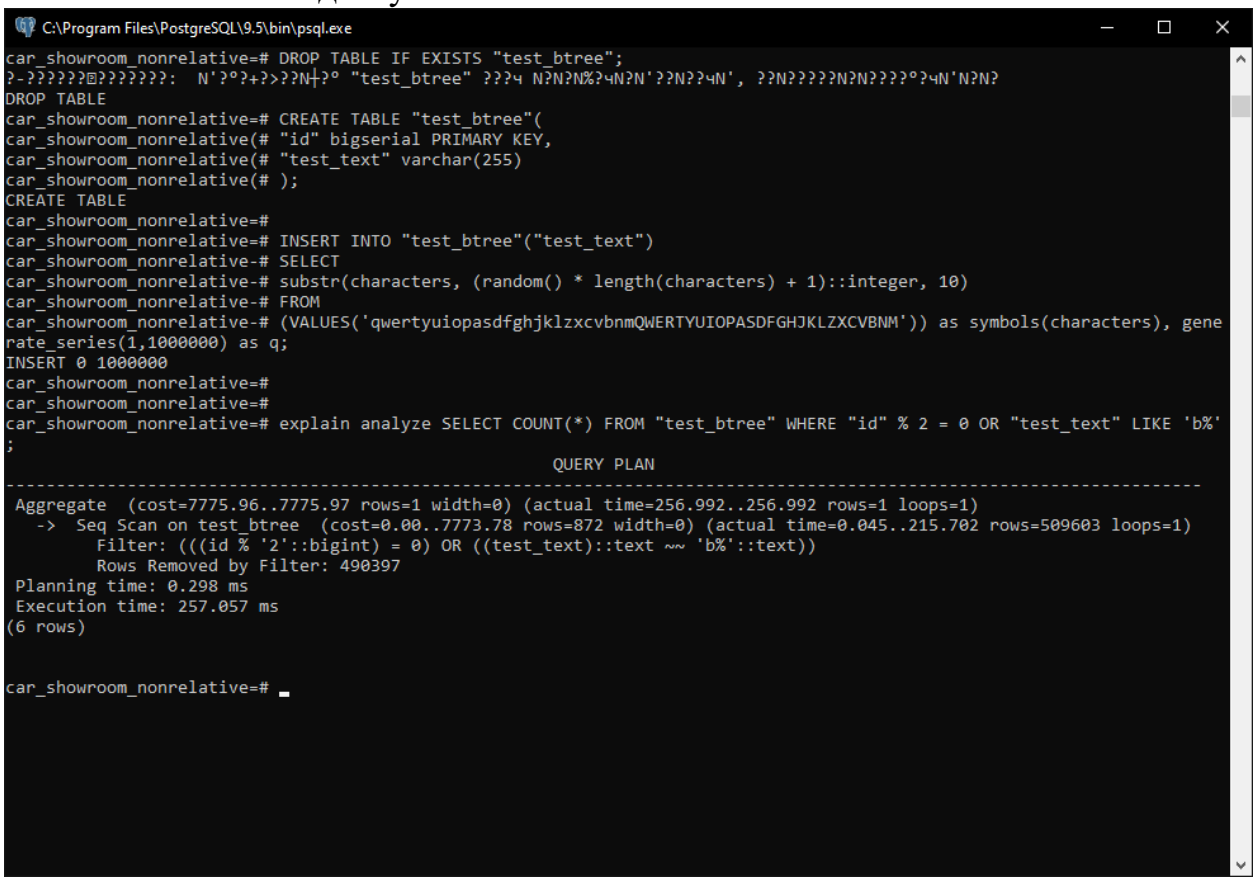
Запит 1 без індексу:



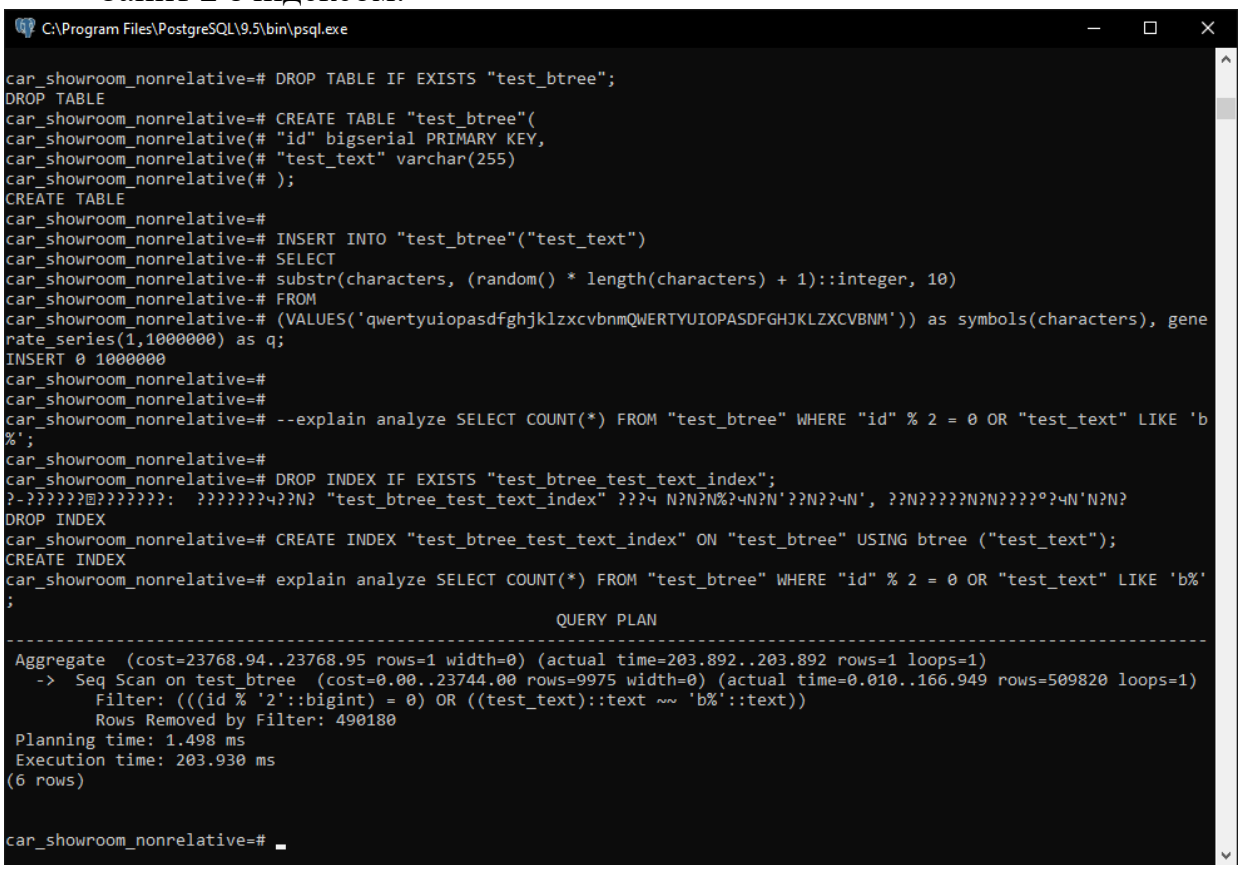
Запит 1 з індексом:



Запит 2 без індексу:



Запит 2 з індексом:



Запит 3 без індексу:

```
Select C:\Program Files\PostgreSQL\9.5\bin\psql.exe
car_showroom_nonrelative=# DROP TABLE IF EXISTS "test_btree";
DROP TABLE
car_showroom_nonrelative=# CREATE TABLE "test_btree"(
car_showroom_nonrelative(# "id" bigserial PRIMARY KEY,
car_showroom_nonrelative(# "test_text" varchar(255)
car_showroom_nonrelative(# );
CREATE TABLE
car_showroom_nonrelative=#
car_showroom_nonrelative=# INSERT INTO "test_btree"("test_text")
car_showroom_nonrelative-# SELECT
car_showroom_nonrelative-# substr(characters, (random() * length(characters) + 1)::integer, 10)
car_showroom_nonrelative-# FROM
car_showroom_nonrelative-# (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters), gene
rate_series(1,1000000) as q;
INSERT 0 1000000
car_showroom_nonrelative=#
car_showroom_nonrelative=# DROP INDEX IF EXISTS "test_btree_test_text_index";
?-??????⊡???????:  ???????ч??N? "test_btree_test_text_index" ???ч N?N?N%?чN?N'??N??чN', ??N?????N?N????°?чN'N?N?
DROP INDEX
car_showroom_nonrelative=# explain analyze SELECT COUNT(*), SUM("id") FROM "test_btree" WHERE "test_text" LIKE 'b%' GROU
P BY "id" % 2;
                                                       QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=7339.89..7346.45 rows=437 width=8) (actual time=151.431..151.432 rows=2 loops=1)
   Group Key: (id % '2'::bigint)
   ->  Seq Scan on test_btree  (cost=0.00..7336.62 rows=437 width=8) (actual time=0.017..142.403 rows=19268 loops=1)
         Filter: ((test_text)::text ~~ 'b%'::text)
         Rows Removed by Filter: 980732
 Planning time: 0.127 ms
 Execution time: 151.484 ms
(7 rows)


car_showroom_nonrelative=#
```

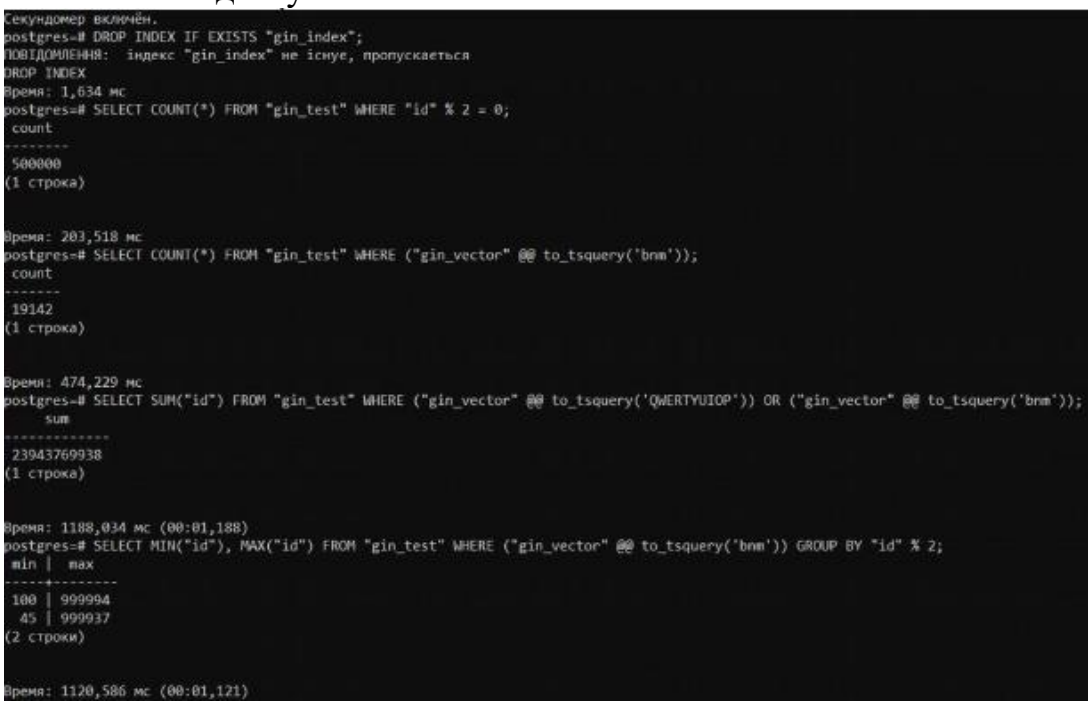
Запит 3 з індексом:

```
C:\Program Files\PostgreSQL\9.5\bin\psql.exe

car_showroom_nonrelative=# DROP TABLE IF EXISTS "test_btree";
DROP TABLE
car_showroom_nonrelative=# CREATE TABLE "test_btree"(
car_showroom_nonrelative(# "id" bigserial PRIMARY KEY,
car_showroom_nonrelative(# "test_text" varchar(255)
car_showroom_nonrelative(# );
CREATE TABLE
car_showroom_nonrelative=#
car_showroom_nonrelative=# INSERT INTO "test_btree"("test_text")
car_showroom_nonrelative-# SELECT
car_showroom_nonrelative-# substr(characters, (random() * length(characters) + 1)::integer, 10)
car_showroom_nonrelative-# FROM
car_showroom_nonrelative-# (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters), gene
rate_series(1,1000000) as q;
INSERT 0 1000000
car_showroom_nonrelative=#
car_showroom_nonrelative=#
car_showroom_nonrelative=# --explain analyze SELECT COUNT(*) FROM "test_btree" WHERE "id" % 2 = 0 OR "test_text" LIKE 'b
%';
car_showroom_nonrelative=#
car_showroom_nonrelative=# DROP INDEX IF EXISTS "test_btree_test_text_index";
?-??????⊡???????:  ???????ч??N? "test_btree_test_text_index" ???ч N?N?N%?чN?N'??N??чN', ??N?????N?N????°?чN'N?N?
DROP INDEX
car_showroom_nonrelative=# CREATE INDEX "test_btree_test_text_index" ON "test_btree" USING btree ("test_text");
CREATE INDEX
car_showroom_nonrelative=# explain analyze SELECT COUNT(*), SUM("id") FROM "test_btree" WHERE "test_text" LIKE 'b%' GROU
P BY "id" % 2;
                                                       QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=18793.00..18868.00 rows=5000 width=8) (actual time=167.804..167.819 rows=2 loops=1)
   Group Key: (id % '2'::bigint)
   ->  Seq Scan on test_btree  (cost=0.00..18755.50 rows=5000 width=8) (actual time=0.023..141.042 rows=18866 loops=1)
         Filter: ((test_text)::text ~~ 'b%'::text)
         Rows Removed by Filter: 981134
 Planning time: 21.026 ms
 Execution time: 168.044 ms
(7 rows)
```

GIN

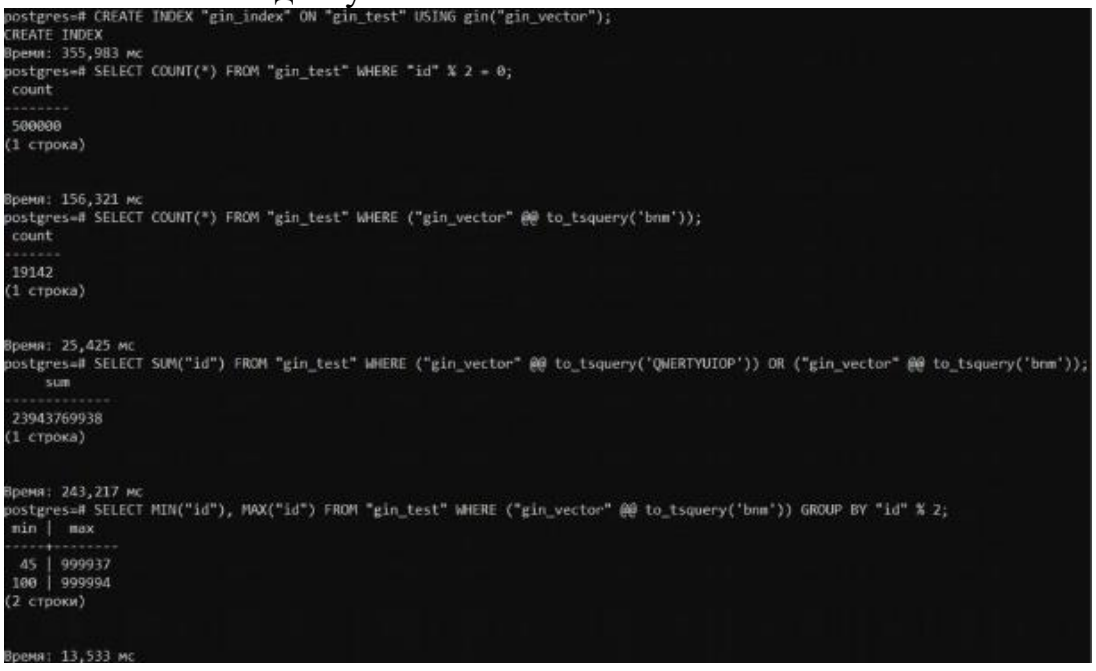
GIN призначенний для обробки, випадків, коли елементи, що підлягають індексації, є складеними значеннями (наприклад – реченнями), а запити, які обробляються індексом, мають шукати значення елементів, які з'являються в складених елементах (повторювані частини слів або речень). Індекс GIN зберігає набір пар (ключ, список появи ключа), де список появи – це набір ідентифікаторів рядків, у яких міститься ключ. Один і той самий ідентифікатор рядка може знаходитись у кількох списках, оскільки елемент може містити більше одного ключа. Кожне значення ключа зберігається лише один раз, тому індекс GIN дуже швидкий для випадків, коли один і той же ключ з'являється багато разів.

Запити без індексування:



```
Секундомер включён.
postgres=# DROP INDEX IF EXISTS "gin_index";
ПОВІДОМЛЕННЯ:  індекс "gin_index" не існує, пропускається
DROP INDEX
Время: 1,634 мс
postgres=# SELECT COUNT(*) FROM "gin_test" WHERE "id" % 2 = 0;
 count
--------
 500000
(1 строка)


Время: 203,518 мс
postgres=# SELECT COUNT(*) FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('bnm'));
 count
-------
 19142
(1 строка)


Время: 474,229 мс
postgres=# SELECT SUM("id") FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('QWERTYUIOP')) OR ("gin_vector" @@ to_tsquery('bnm'));
     sum
-------------
 23943769938
(1 строка)


Время: 1188,034 мс (00:01,188)
postgres=# SELECT MIN("id"), MAX("id") FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('bnm')) GROUP BY "id" % 2;
 min |  max
-----+--------
 100 | 999994
  45 | 999937
(2 строки)


Время: 1120,586 мс (00:01,121)
```

Запити з індексуванням:



```
postgres=# CREATE INDEX "gin_index" ON "gin_test" USING gin("gin_vector");
CREATE INDEX
Время: 355,983 мс
postgres=# SELECT COUNT(*) FROM "gin_test" WHERE "id" % 2 = 0;
 count
--------
 500000
(1 строка)


Время: 156,321 мс
postgres=# SELECT COUNT(*) FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('bnm'));
 count
-------
 19142
(1 строка)


Время: 25,425 мс
postgres=# SELECT SUM("id") FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('QWERTYUIOP')) OR ("gin_vector" @@ to_tsquery('bnm'));
     sum
-------------
 23943769938
(1 строка)


Время: 243,217 мс
postgres=# SELECT MIN("id"), MAX("id") FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('bnm')) GROUP BY "id" % 2;
 min |  max
-----+--------
  45 | 999937
 100 | 999994
(2 строки)


Время: 13,533 мс
```

Завдання №3

Для тестування тригера було створено таблицю:

```
DROP TABLE IF EXISTS "reader";
CREATE TABLE "reader"(
        "readerID" bigserial PRIMARY KEY,
        "readerName" varchar(255)
);
```

Початкові значення у таблиці:

```
INSERT INTO "reader"("readerName")

VALUES ('reader1'), ('reader2'), ('reader3'), ('reader4'),('reader5');
```

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 ’a’, 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.

Початковий стан:

	readerID [PK] bigint	readerName character varying (255)
1	1	reader1
2	2	reader2
3	3	reader3
4	4	reader4
5	5	reader5

Після виконання запиту вставки:

```
INSERT INTO "reader"("readerName") VALUES ('reader6');
```

	readerID [PK] bigint	readerName character varying (255)
1	1	reader1a
2	2	reader2a
3	3	reader3a
4	4	reader4a
5	5	reader5a
6	6	reader6a

Після виконання запиту на видалення:

```
DELETE FROM "reader" WHERE "readerID" = 6;
```

	readerID [PK] bigint	readerName character varying (255)
1	1	reader1a
2	2	reader2a
3	3	reader3a
4	4	reader4a
5	5	reader5a