

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

# «Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»

Тема:

Засоби оптимізації роботи СУБД PostgreSQL

Виконала:

Студенка групи КВ-04

Устименко І.В.

Перевірив:

Git: https://github.com/illya-U/bd\_lab\_2022/tree/develop/lab\_3

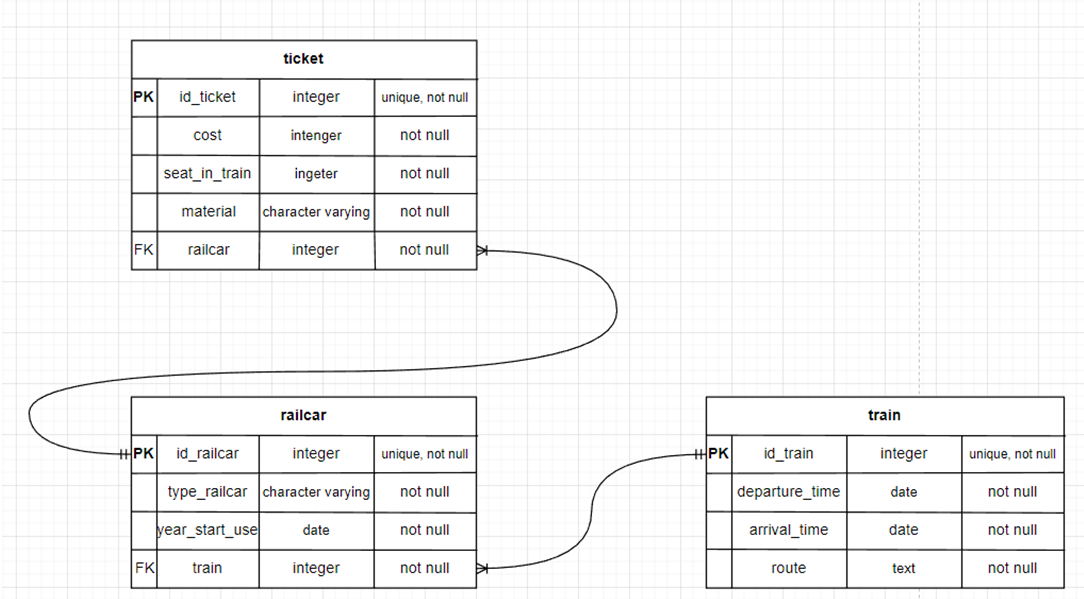
Петрашенко А.В.

**Київ – 2022**

**Варіант 23**

|  |  |  |
| --- | --- | --- |
| *23* | *GIN, Hash* | *before update, delete* |

**Завдання 1**



Для роботи з sqlaclemy було імпортовано такі засоби:

from sqlalchemy import create\_engine

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

from sqlalchemy.orm import relationship

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

from sqlalchemy import text

Підключення до бази та створення сесії:

from sqlalchemy.orm import relationship

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

from base import Base

class Train(Base):

tablename = 'train'

id\_train = Column('id\_train', Integer, primary\_key=True, autoincrement=True)

departure\_time = Column('departure\_time', Date)

arrival\_time = Column('arrival\_time', Date)

route = Column('route', String(100))

train\_railcar = relationship('Railcar', cascade='delete-orphan')

def repr(self):

return "<Train(id\_train='{}', departure\_time='{}', arrival\_time='{}', route='{}')>".format(self.id\_train,

self.departure\_time,

self.arrival\_time,

self.route)

class Railcar(Base):

tablename = 'railcar'

id\_railcar = Column('id\_train', Integer, primary\_key=True, autoincrement=True)

type\_railcar = Column('type\_railcar', String(50))

year\_start\_use\_railcar = Column('year\_start\_use\_railcar', Date)

train = Column('train', Integer, ForeignKey('Train.id\_train', onupdate='cascade'), primary\_key=True)

railcar\_ticket = relationship('Ticket', cascade='delete-orphan')

def repr(self):

use\_railcar = self.year\_start\_use\_railcar

return "<Railcar(id\_railcar='{}', type\_railcar='{}', year\_start\_use\_railcar='{}')>".format(self.id\_railcar,

self.type\_railcar,

use\_railcar

)

class Ticket(Base):

tablename = 'ticket'

id\_ticket = Column('id\_ticket', Integer, primary\_key=True, autoincrement=True)

cost = Column('cost', Integer)

seat\_in\_the\_train = Column('seat\_in\_the\_train', Integer)

material = Column('material', String(50))

railcar = Column('railcar', Integer, ForeignKey('Railcar.id\_railcar', onupdate='cascade'), primary\_key=True)

def repr(self):

ticket = self.id\_ticket

cost = self.cost

seat = self.seat\_in\_the\_train

material = self.material

return "Ticket(id\_ticket='{}', cost='{}', seat\_in\_the\_train='{}', material='{}'".format(ticket,

cost,

seat,

material)

Функції, що реалізують вставки, вилучення та редагування екземплярів класів-сутностей.

def random(self, table\_name, n):

field\_type = {additional\_inf.parametrs[table\_name][1:]}

        try:

**for** i in range(**int**(n)):

self.add\_inf(table\_name, ***{***field: additional\_inf.randomvalue\_type[type] **for** field, type in field\_type***}***)

except (Exception, sqlalchemy.exc.SQLAlchemyError) as err:

print(err)

self.session.rollback()

print(f***'WARNING:Error {err}'***)

def delete(self, table\_name, field\_value):

        try:

table\_class = orm.get\_class\_by\_tablename(table\_name)

filter\_txt = dict\_to\_sql\_str(field\_value)

self.session.query(table\_class).filter(text(filter\_txt)).delete(synchronize\_session=False)

self.session.commit()

except (Exception, sqlalchemy.exc.SQLAlchemyError) as err:

print(err)

self.session.rollback()

print(f***'WARNING:Error {err}'***)

def add\_inf(self, table\_name, field\_value):

        try:

obj = orm.get\_class\_by\_tablename(table\_name)(\*\*field\_value)

self.session.add(obj)

self.session.commit()

except (Exception, sqlalchemy.exc.SQLAlchemyError) as err:

print(err)

self.session.rollback()

print(f***'WARNING:Error {err}'***)

def updt(self, table\_name, field\_value, based\_field\_value):

        try:

table\_class = orm.get\_class\_by\_tablename(table\_name)

filter\_txt = dict\_to\_sql\_str(based\_field\_value)

self.session.query(table\_class).filter(text(filter\_txt)).update(field\_value, synchronize\_session=False)

self.session.commit()

except (Exception, sqlalchemy.exc.SQLAlchemyError) as err:

print(err)

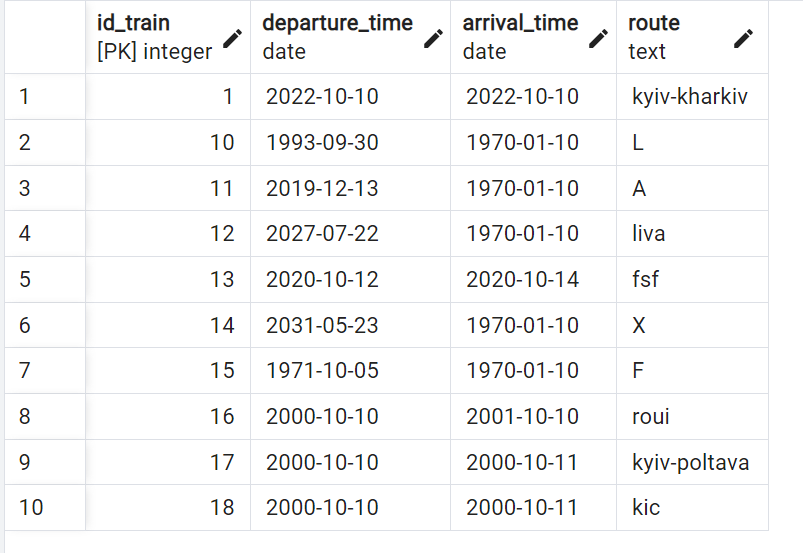
self.session.rollback()

print(f***'WARNING:Error {err}'***)

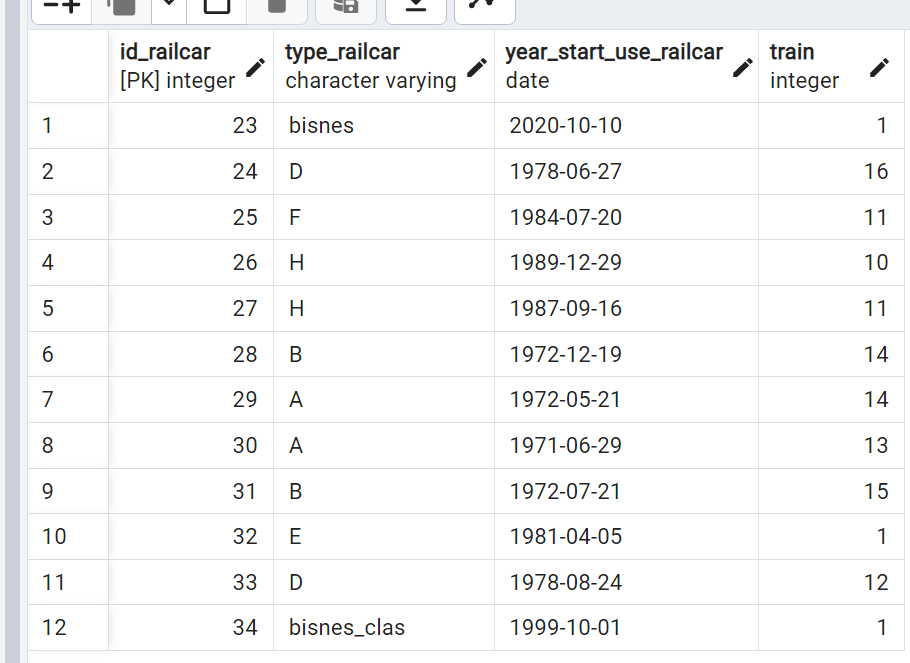
Приклади використання:

Видалення данних

Даннi потяга до видалення:



Даннi вагона до видалення:



Update press 1

Add press 2

Delete press 3

Random press 4

Search press 5

Info about tables press 6

Please print all not numeric types like int and float,like this 'value'

Enter command : 3

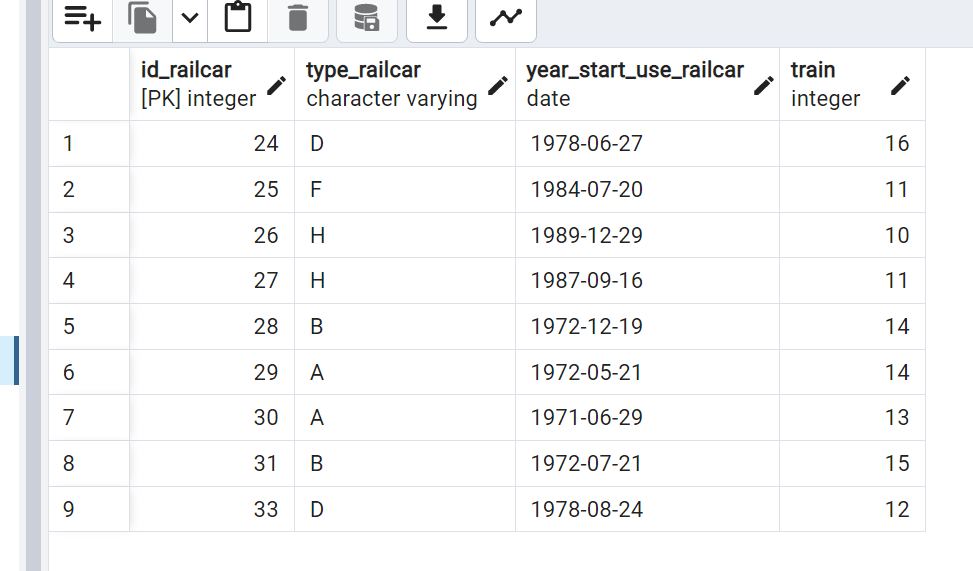
Your table\_name name: train, ticket, railcar

Enter table\_name name train

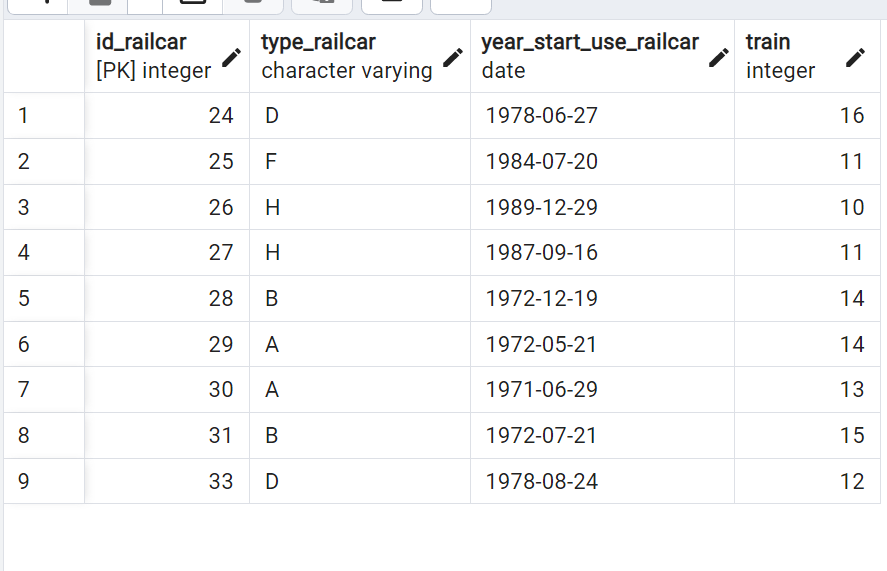
Enter column atribute,based on which you want to change this fileld like 'Column\_name = 'value'' and press Enter,when you want to stop enter columns press '-' and press Enter:id\_train = 1

Enter column atribute,based on which you want to change this fileld like 'Column\_name = 'value'' and press Enter,when you want to stop enter columns press '-' and press Enter:-

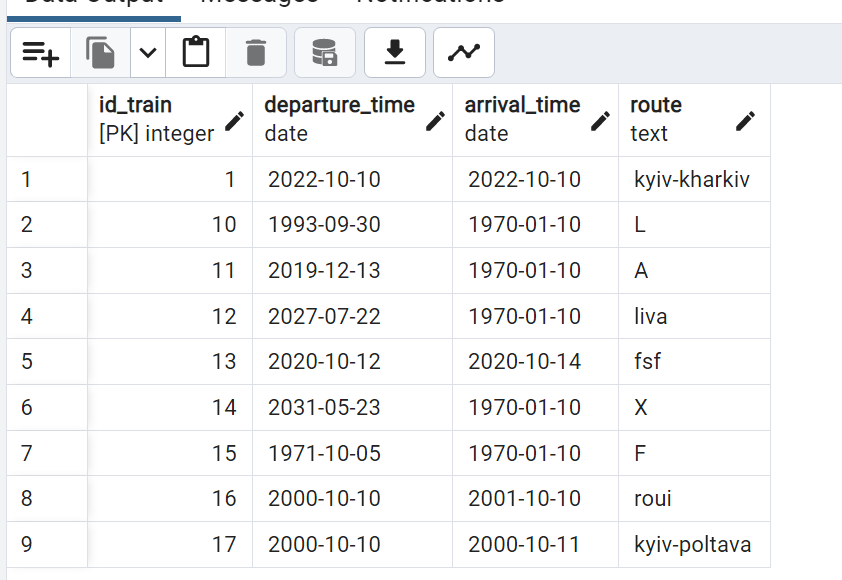
Потяг після видалення:

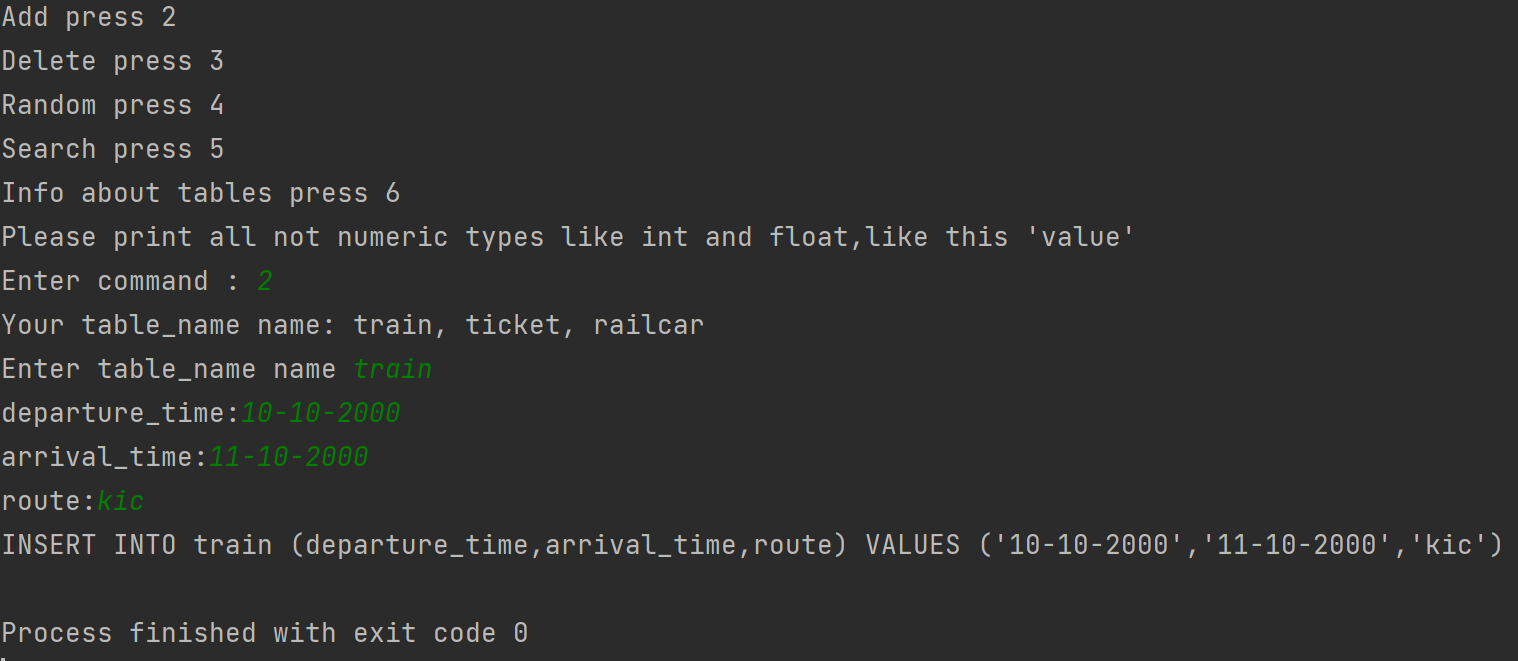


Вагон після видалення:

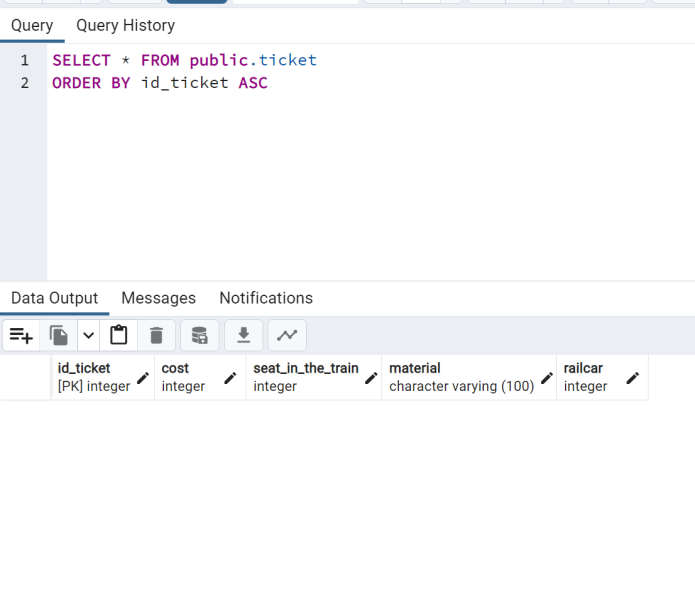


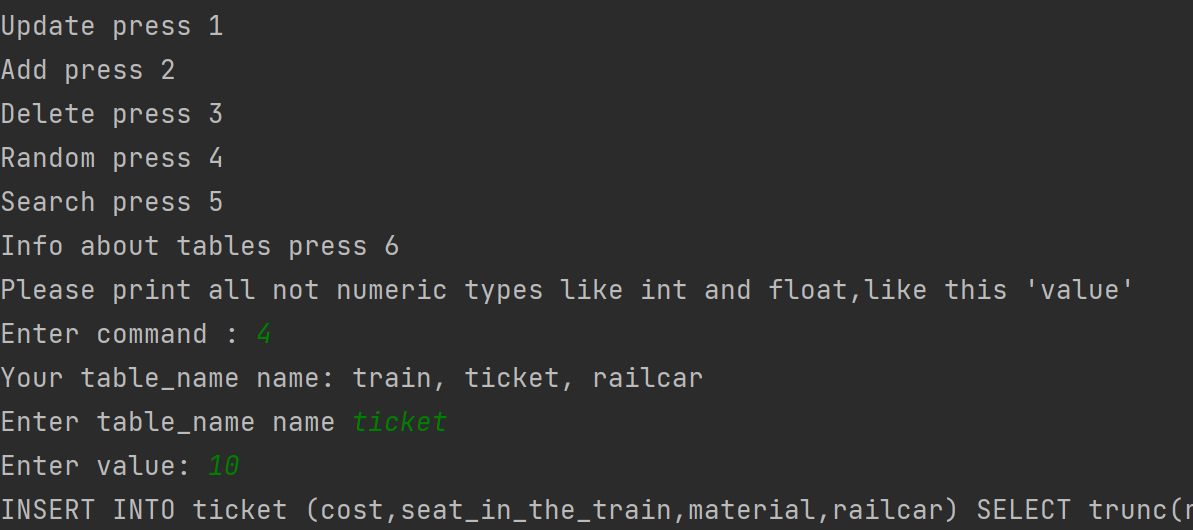
Додавання данних:

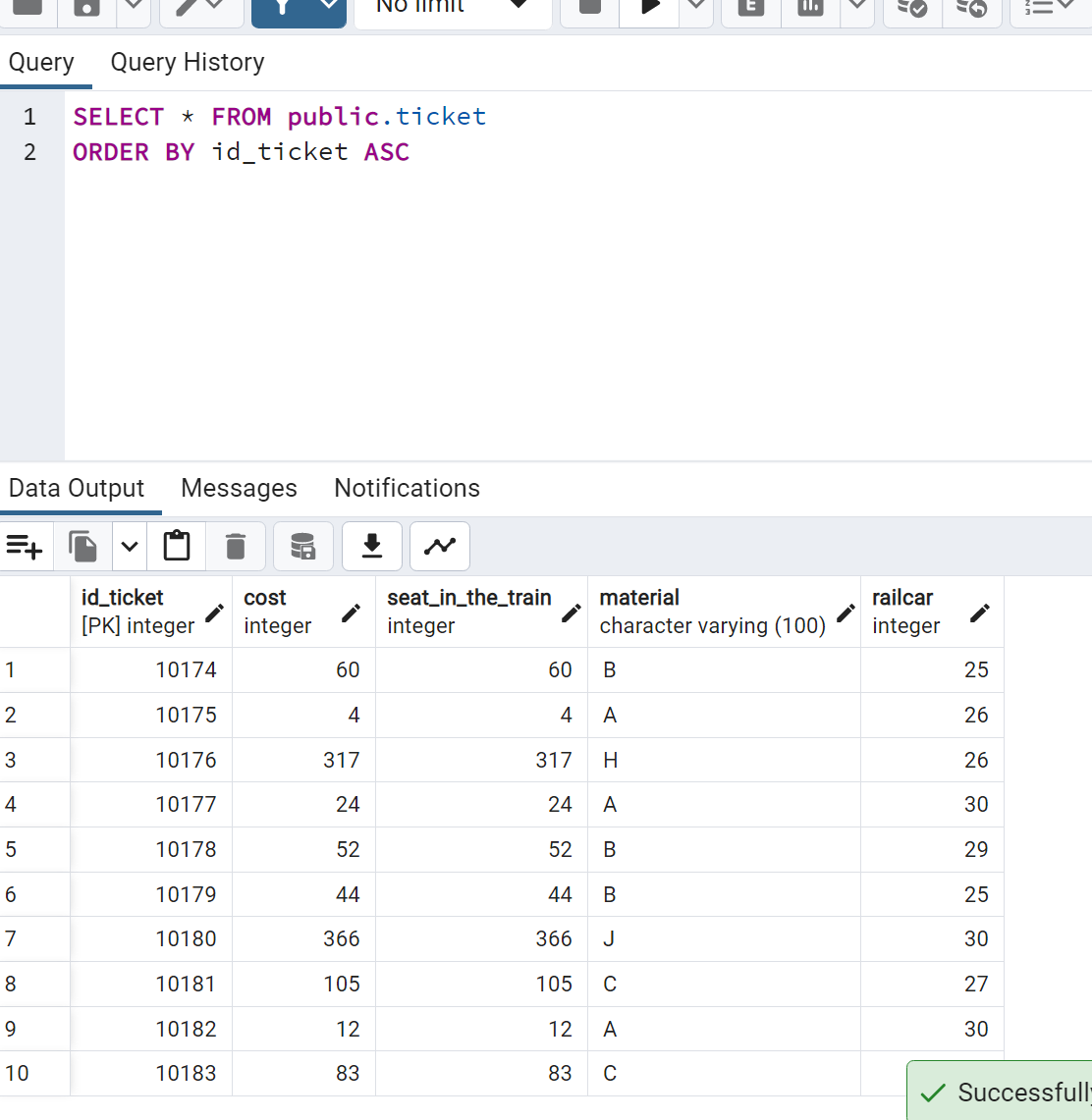


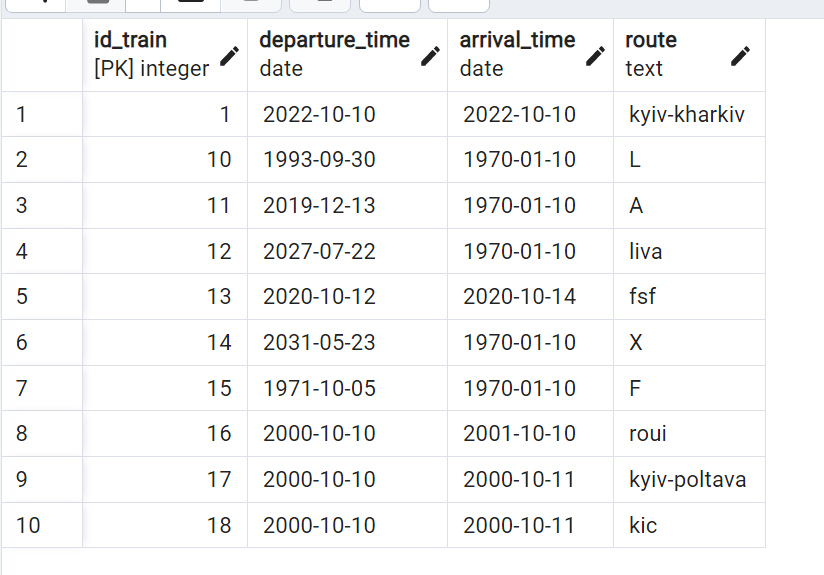


Меню генерації:









**Завдання 2**

В даному завданні розглянемо використання індексів GIN, Hash. Для створення індексів для текстового поля використовувалися такі команди:

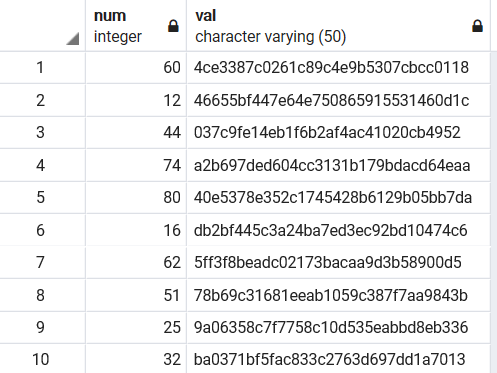
CREATE INDEX txt\_idx ON "Test" USING GIN (val gin\_trgm\_ops);

CREATE INDEX txt\_idx ON "Test" USING Hash (val);

Для створення індексів по числовому полю для Hash можливо скористатися командою:

CREATE INDEX num\_idx ON "Test" USING Hash (num);

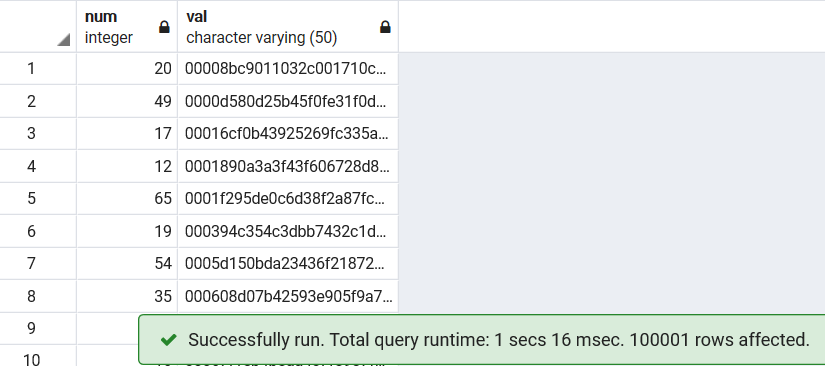
Результат запиту SELECT \* FROM public."Test"



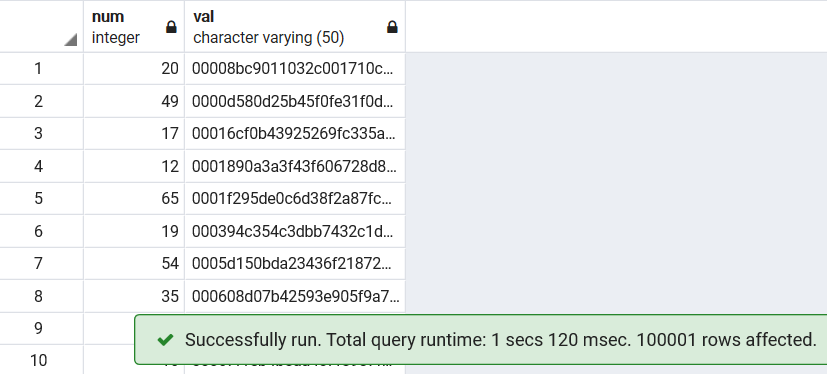
**Запит №1**

SELECT num, val FROM "Test" ORDER BY val;

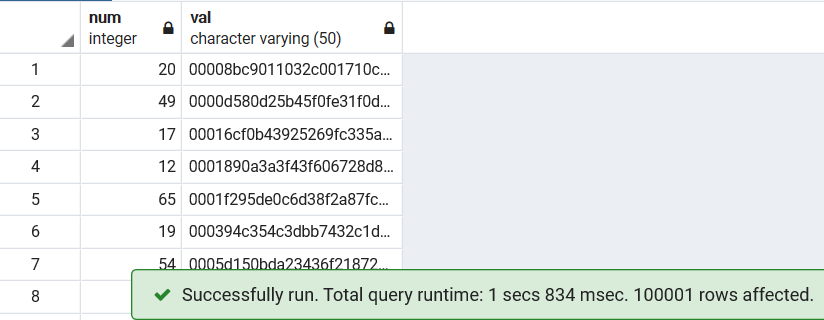
Без індексування.



Hash



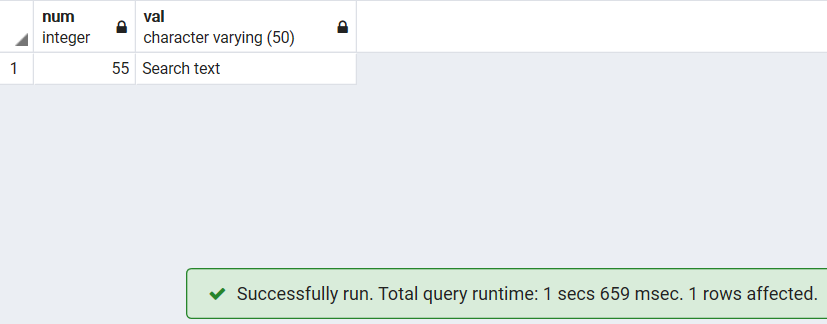
GIN



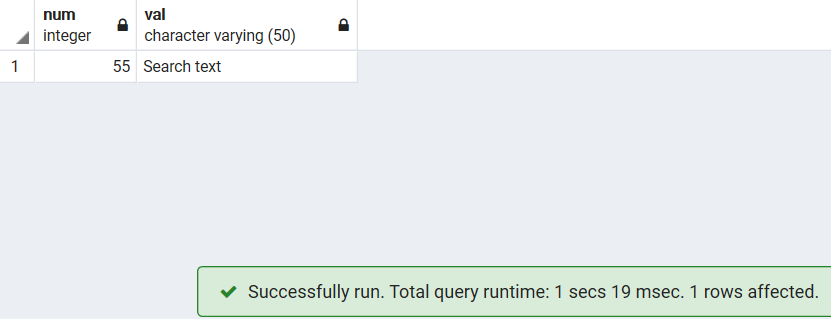
**Запит №2**

SELECT \* FROM "Test" WHERE val = 'Search text';

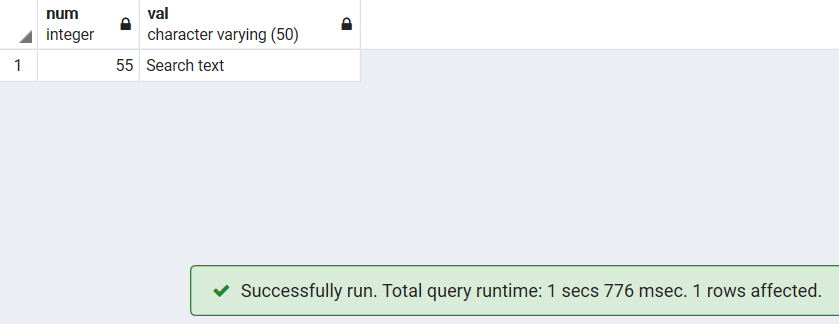
Без індексування



Hash



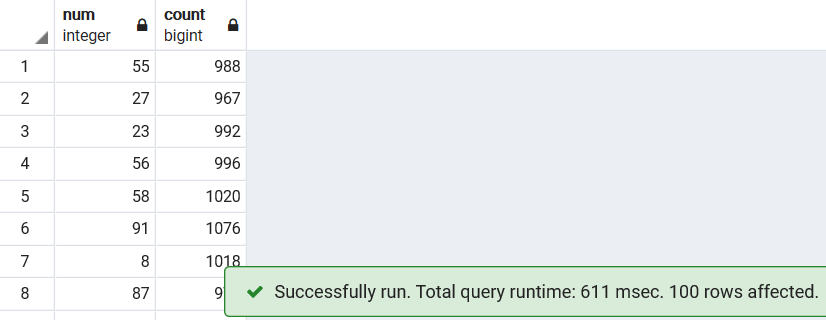
GIN



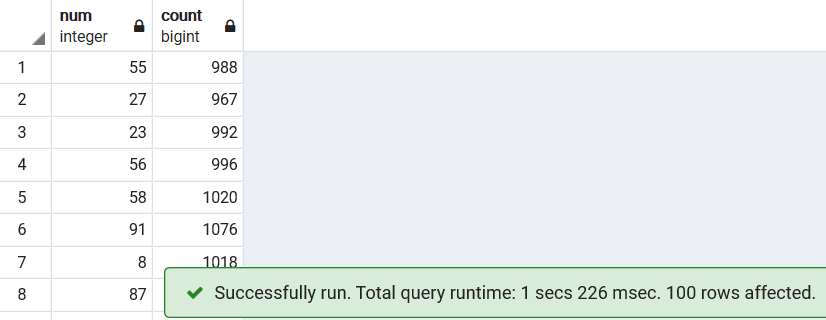
**Запит №3**

SELECT num, count(num) FROM "Test" GROUP BY num

Без індексування



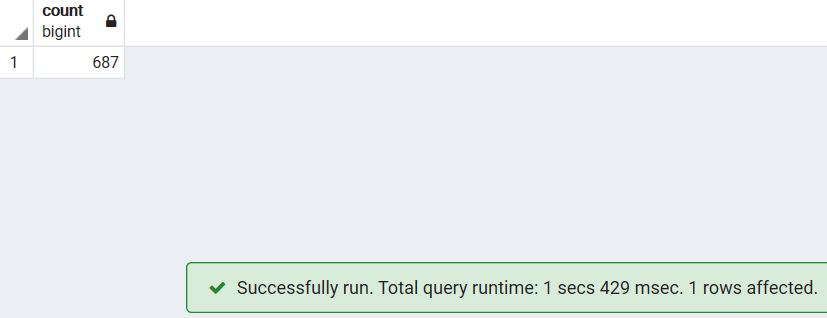
Hash



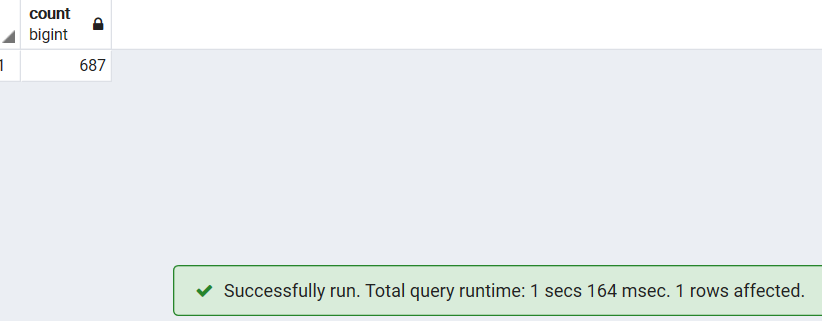
**Запит №4**

SELECT count(\*) FROM "Test" WHERE val ILIKE '%aaa%';

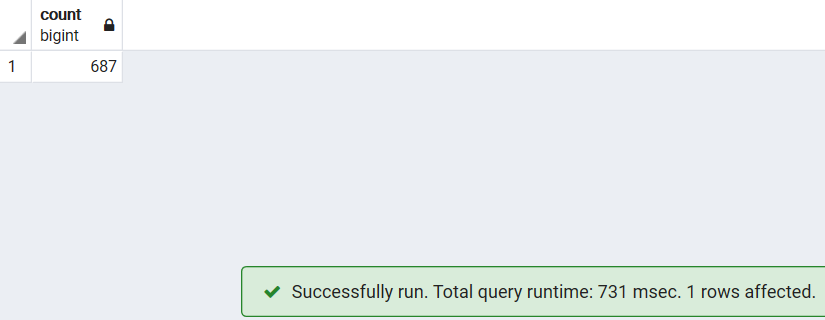
Без індексування



Hash



GIN



**Висновок**

Індекс GIN добре використовувати коли необхідно виконувати пошук символів у тексті, або масиві. Основним недоліком цього індексу є його обмеженість у використання (не може наприклад використовуватися з числовими типами даних) і досить тривале створення. Індекс GIN краще використовувати у запитах в яких присутні такі операції як LIKE, ILIKE.

Індекс Hash добре використовувати коли необхідно прискорити пошук і також важливим є розмір індексу. Що до розміру індексу Hash, то він є «плоским» в порівнянні з індексом Btree, тому є значно меншим за розміром.

Найбільшою проблемою Hash є те що він обмежений лише оператором рівності, тому треба шукати тільки точні збіги. І він краще працює з числами ніж з рядками символів.

**Завдання 3**

***Створення тригерної функції.***

CREATE OR REPLACE FUNCTION trig\_func() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'UPDATE') THEN

IF NEW.subscription\_number < 0 THEN

RAISE EXCEPTION 'NOT CORRECT NUMBER. SHOULD NOT BE UPDATED';

END IF;

IF NEW.name is NULL THEN

RAISE EXCEPTION 'NOT CORRECT NAME. SHOULD NOT BE UPDATED';

END IF;

RETURN NEW;

ELSIF (TG\_OP = 'DELETE') THEN

IF NEW.subscription\_number < 0 THEN

RAISE EXCEPTION 'NOT CORRECT NUMBER. SHOULD NOT BE DELETED';

END IF;

RETURN OLD;

END IF;

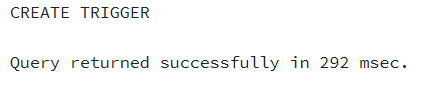
RETURN NEW;

END;

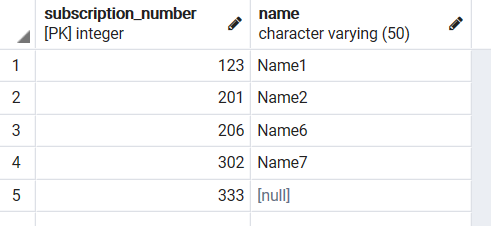
$$;

CREATE TRIGGER trig\_func BEFORE DELETE OR UPDATE ON "Readers"

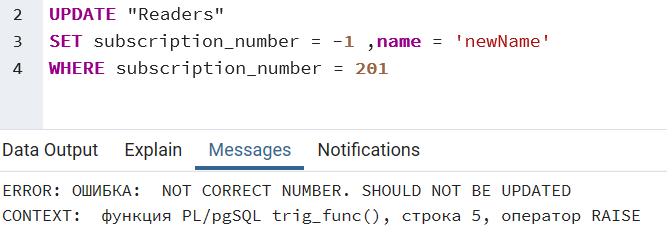
FOR EACH ROW EXECUTE FUNCTION trig\_func();

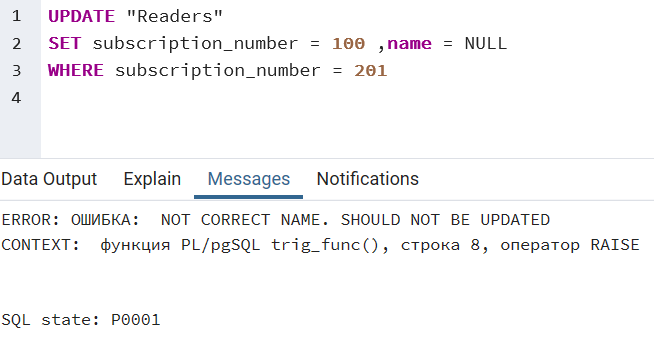


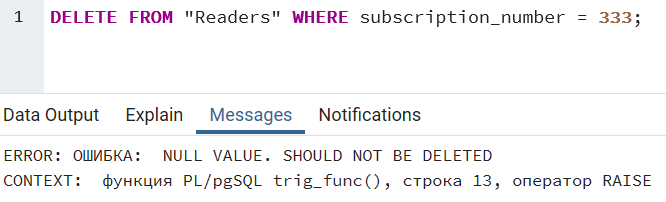
Зміст таблиці "Readers"

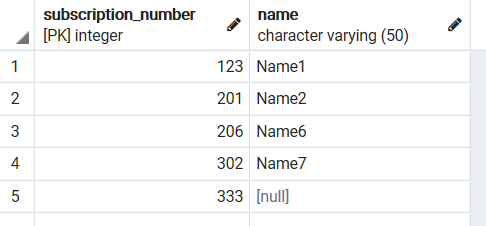


Приклади використання тригеру

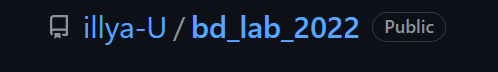








**Завдання 4**

****