

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-92

Пургін В.А.

Перевірив:

Петрашенко А.В.

Київ – 2021

*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.

|  |  |  |
| --- | --- | --- |
| *№ варіанта* | *Види індексів* | *Умови для тригера* |
| *22* | *Hash, BRIN* | *after delete, insert* |

URL репозиторію з вихідним кодом: <https://github.com/VladislavPurgin/BD_lab>

**Завдання №1**

**Модель «сутність-зв’язок»:**

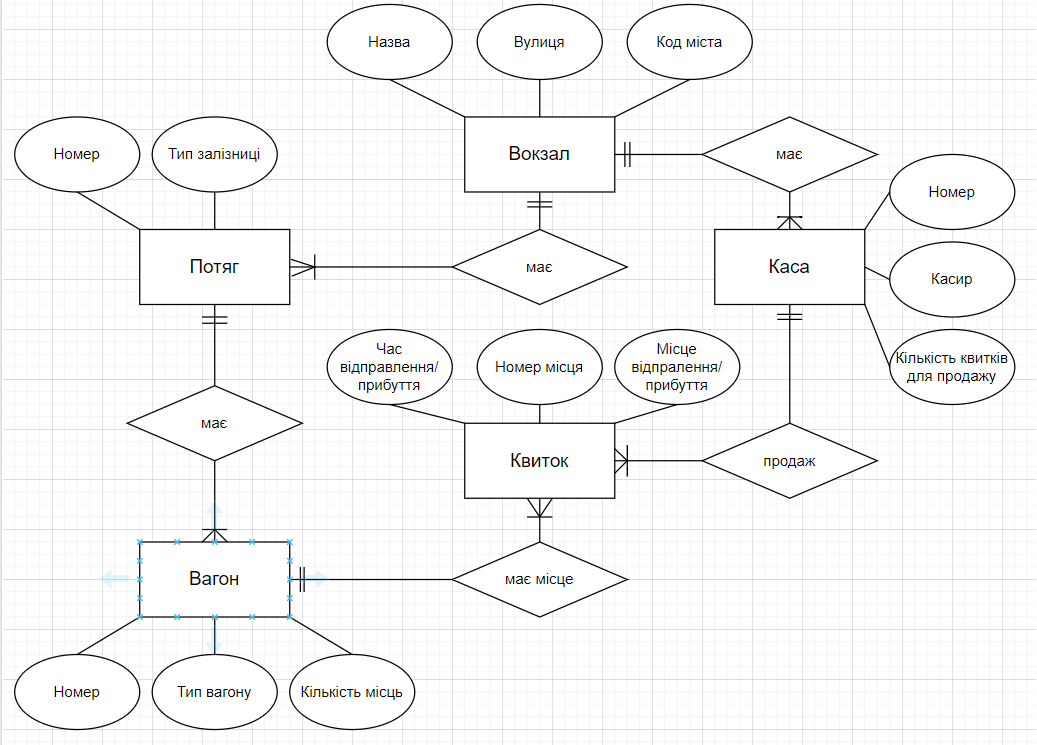
****

Рис1. ER-діаграма за нотацією Чена

**Схеми бази даних:**

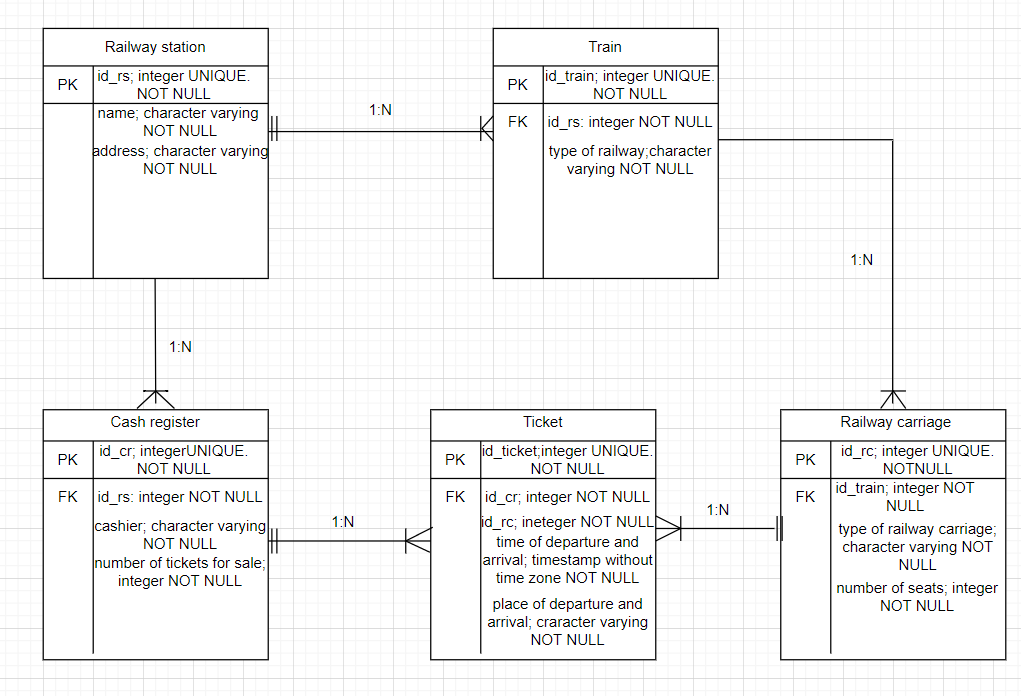


Рис2. Схема бази даних

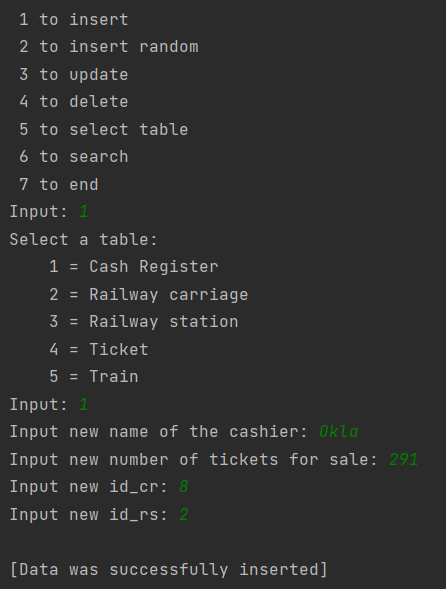
**Класи ORM, що відповідають таблицям бази даних**

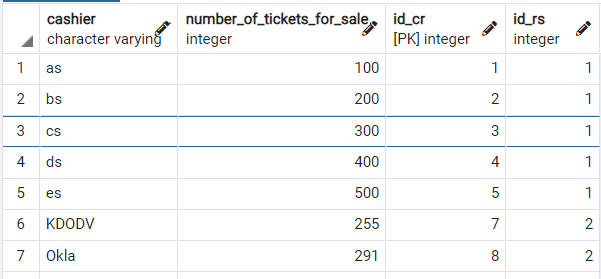
class Cash\_register(Base):  
 \_\_tablename\_\_ = 'Cash\_register'  
 cashier = Column(String)  
 number\_of\_tickets\_for\_sale = Column(Integer)  
 id\_cr = Column(Integer, primary\_key=True)  
 id\_rs = Column(Integer, ForeignKey('Railway\_station.id\_rs'))  
  
 tickets = relationship('Ticket')  
  
 def \_\_init\_\_(self, cashier: str, number\_of\_tickets\_for\_sale: int, id\_cr: int, id\_rs: int):  
 self.cashier = cashier  
 self.number\_of\_tickets\_for\_sale = number\_of\_tickets\_for\_sale  
 self.id\_cr = id\_cr  
 self.id\_rs = id\_rs  
  
  
class Railway\_carriage(Base):  
 \_\_tablename\_\_ = 'Railway\_carriage'  
 type = Column(String)  
 number\_of\_seats = Column(Integer)  
 id\_rc = Column(Integer, primary\_key=True)  
 id\_train = Column(Integer, ForeignKey('Train.id\_train'))  
  
 tickets = relationship('Ticket')  
  
 def \_\_init\_\_(self, type\_: str, number\_of\_seats: int, id\_rc: int, id\_train: int):  
 self.type = type\_  
 self.number\_of\_seats = number\_of\_seats  
 self.id\_rc = id\_rc  
 self.id\_train = id\_train  
  
  
class Railway\_station(Base):  
 \_\_tablename\_\_ = 'Railway\_station'  
 name = Column(String)  
 address = Column(String)  
 id\_rs = Column(Integer, primary\_key=True)  
  
 crs = relationship('Cash\_register')  
 trains = relationship('Train')  
  
 def \_\_init\_\_(self, name: str, address: str, id\_rs: int):  
 self.name = name  
 self.address = address  
 self.id\_rs = id\_rs  
  
  
class Ticket(Base):  
 \_\_tablename\_\_ = 'Ticket'  
 time\_of\_departure = Column(Interval)  
 place\_of\_departure\_and\_arrival = Column(String)  
 id\_ticket = Column(Integer, primary\_key=True)  
 id\_rc = Column(Integer, ForeignKey('Railway\_carriage.id\_rc'))  
 id\_cr = Column(Integer, ForeignKey('Cash\_register.id\_cr'))  
  
 def \_\_init\_\_(self, time\_of\_departure: datetime.time, place\_of\_departure\_and\_arrival: str, id\_ticket: int,  
 id\_rc: int, id\_cr: int):  
 self.time\_of\_departure = time\_of\_departure  
 self.place\_of\_departure\_and\_arrival = place\_of\_departure\_and\_arrival  
 self.id\_ticket = id\_ticket  
 self.id\_rc = id\_rc  
 self.id\_cr = id\_cr  
  
  
class Train(Base):  
 \_\_tablename\_\_ = 'Train'  
 type\_of\_railway = Column(String)  
 id\_train = Column(Integer, primary\_key=True)  
 id\_rs = Column(Integer, ForeignKey('Railway\_station.id\_rs'))  
  
 rcs = relationship('Railway\_carriage')  
  
 def \_\_init\_\_(self, type\_of\_railway: str, id\_train: int, id\_rs: int):  
 self.type\_of\_railway = type\_of\_railway  
 self.id\_train = id\_train  
 self.id\_rs = id\_rs

**Результат реалізації вставки, вилучення та редагування**

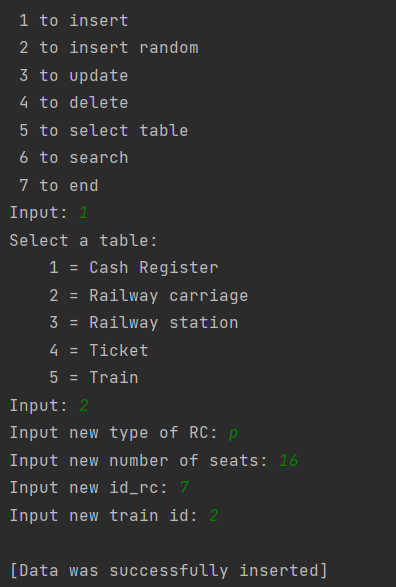
Вставка:

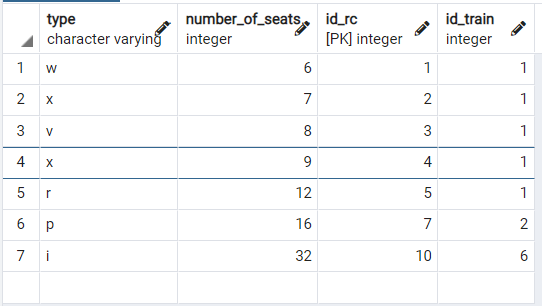
В таблицю Cash Register:



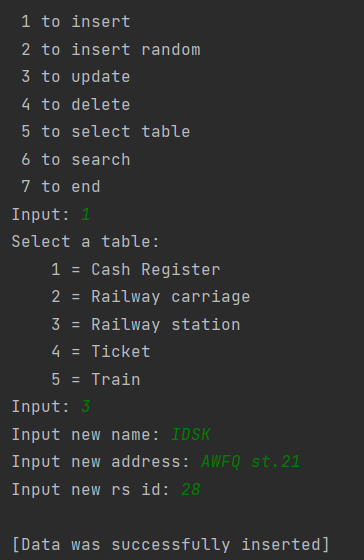


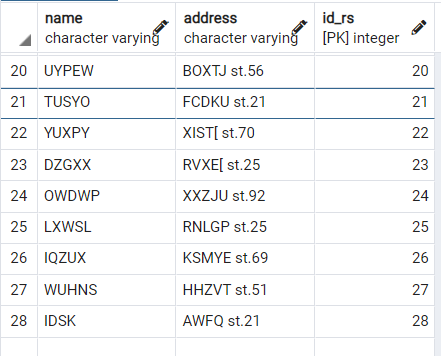
В таблицю Railway carriage:





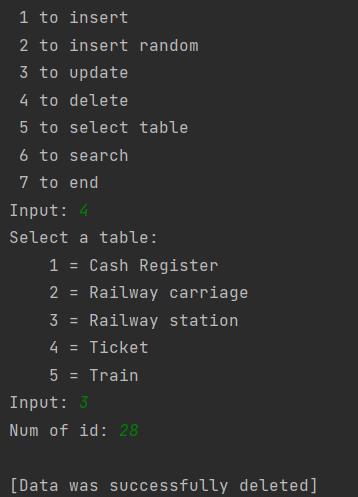
В таблицю Railway Station:

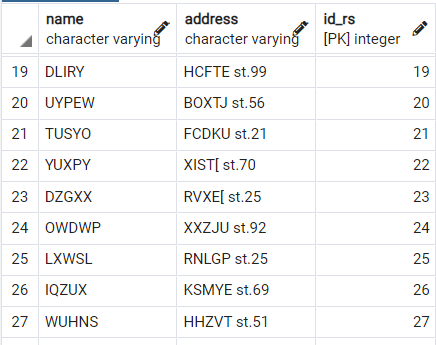




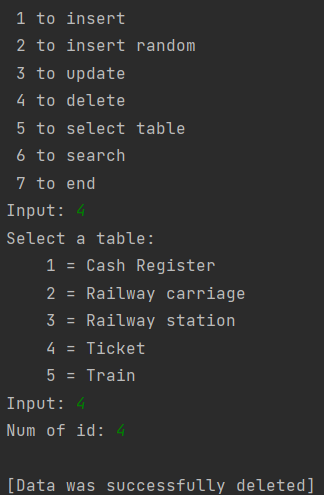
Вилучення:

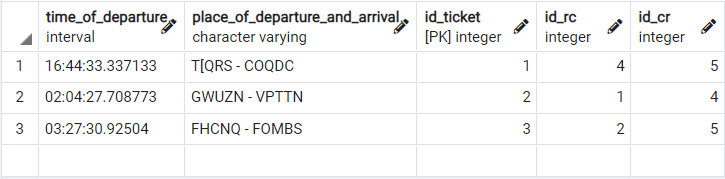
З таблиці Railway Station:



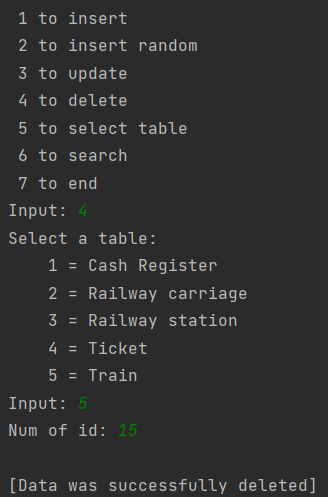


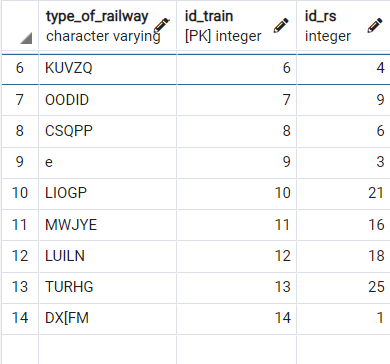
З таблиці Ticket:





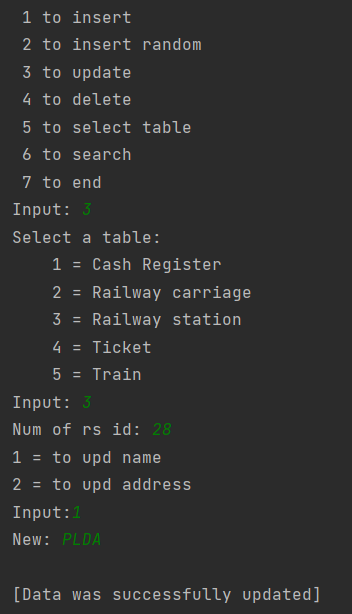
З таблиці Train:

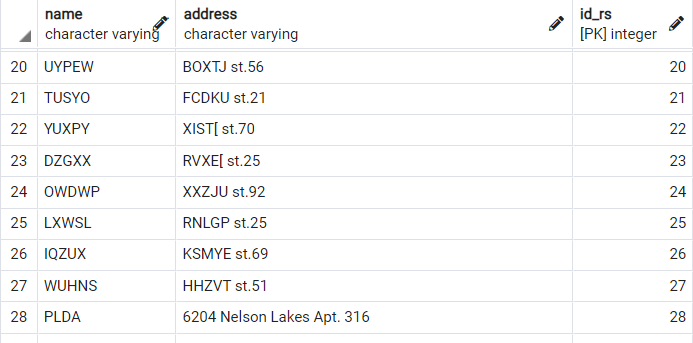




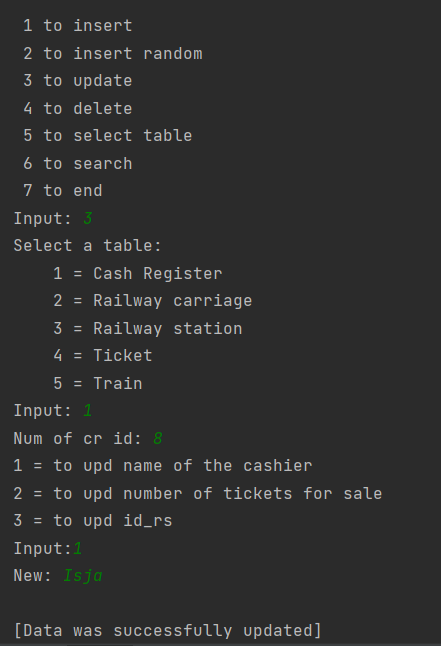
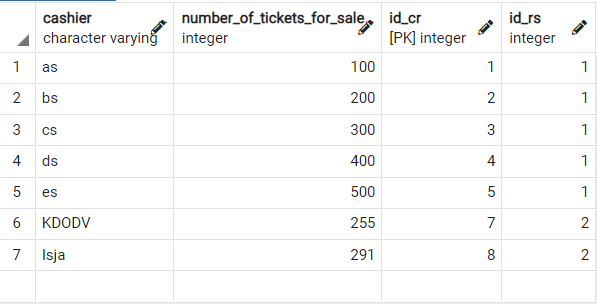
Редагування:

Таблиці Railway Station:

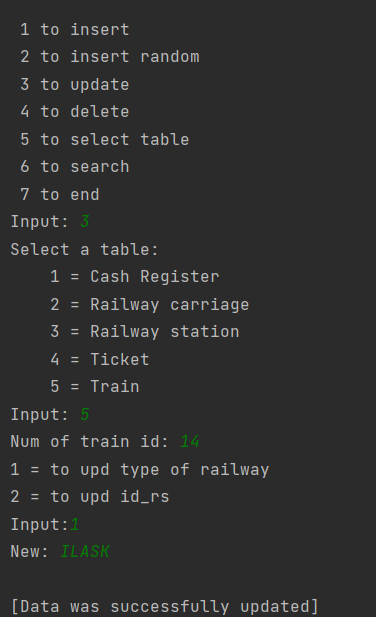


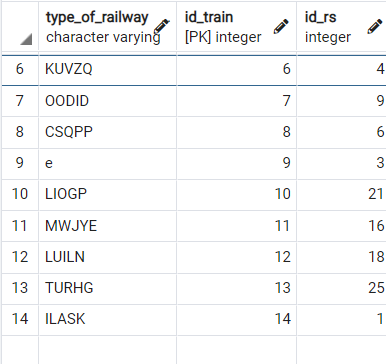


Таблицю Cash Register:

Таблицю Train:

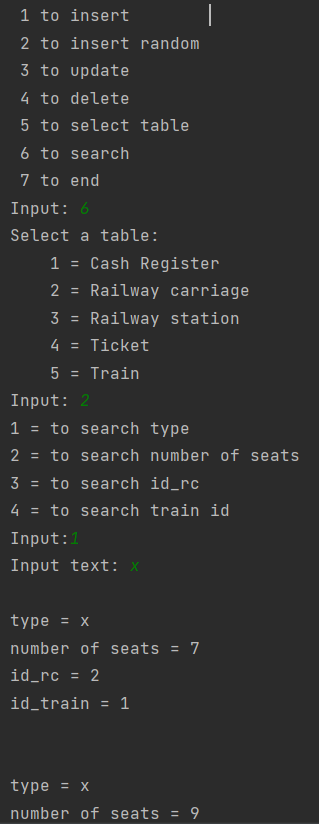


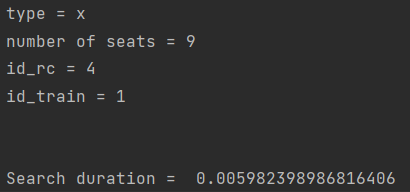


**Результат реалізації пошуку та генерації даних**

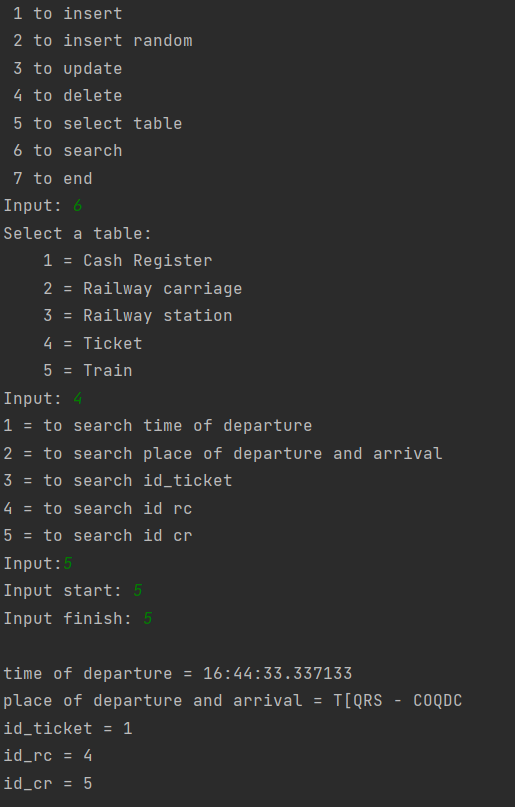
Пошук:

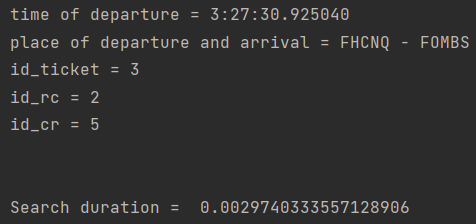
З таблиці Railway carriage:



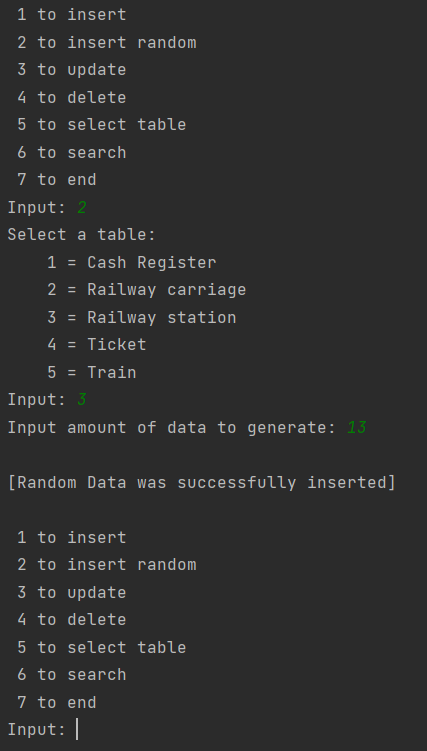


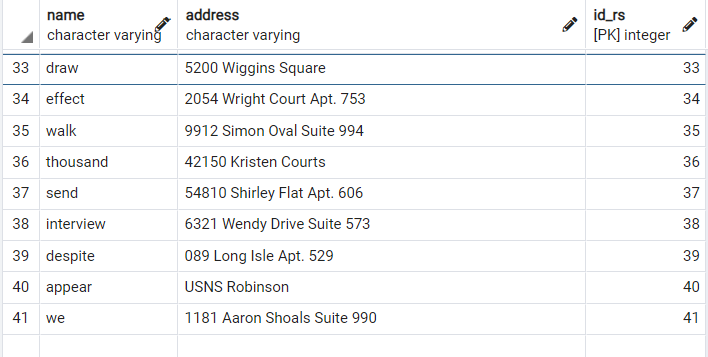
З таблиці Ticket:





Генерація даних:





**Завдання №2**

**Hash**

**Запити для тестування:**

SELECT COUNT(\*) FROM "Railway\_station" WHERE id\_rs % 2 = 0;

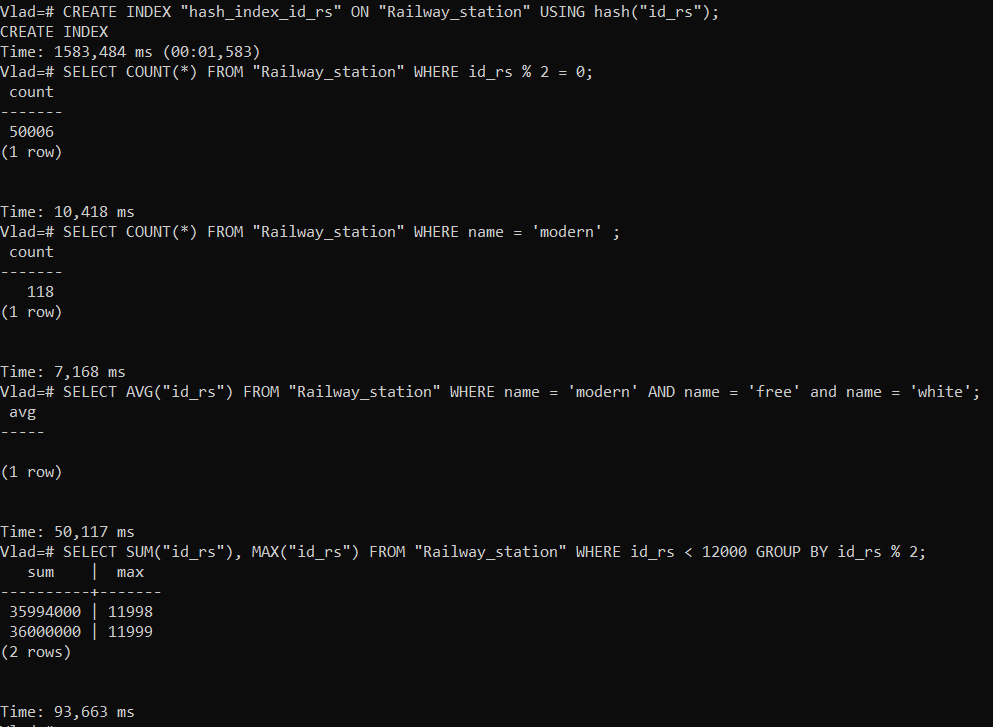
SELECT COUNT(\*) FROM "Railway\_station" WHERE name = 'modern' ;

SELECT AVG("id\_rs") FROM "Railway\_station" WHERE name = 'modern' AND name = 'free' and name = 'white';

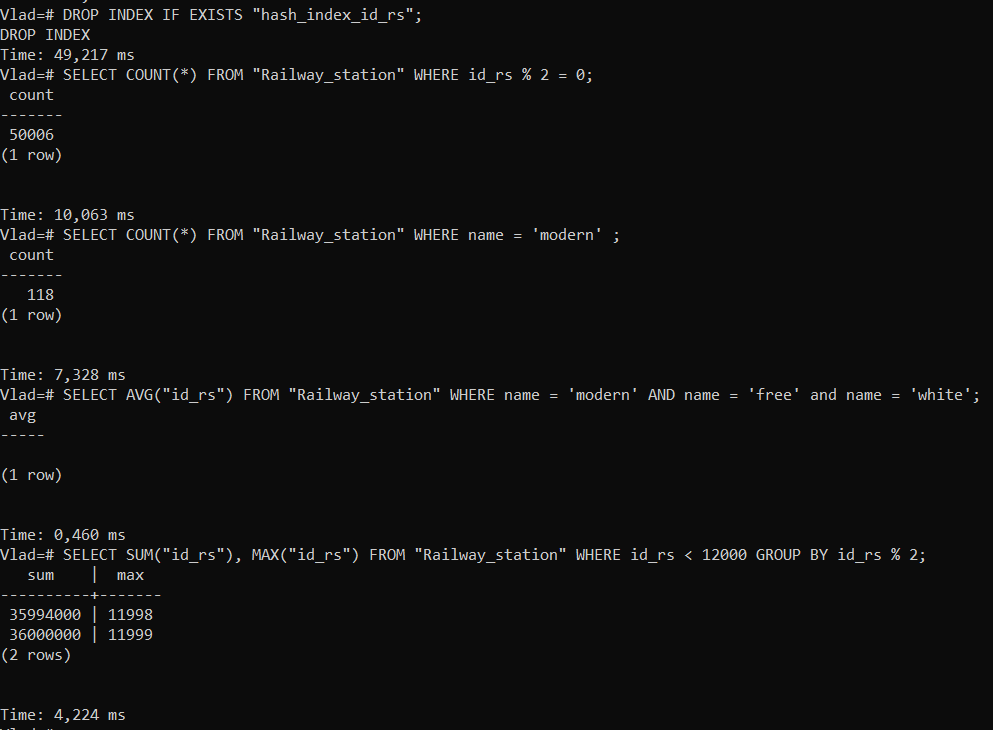
SELECT SUM("id\_rs"), MAX("id\_rs") FROM "Railway\_station" WHERE id\_rs < 12000 GROUP BY id\_rs % 2;

**Результати тестування**

**CREATE INDEX "hash\_index\_id\_rs" ON "Railway\_station" USING hash("id\_rs");**



**DROP INDEX IF EXISTS "hash\_index\_id\_rs";**



Бачим, що в деяких випадках індексування з допомогою Hash показує не значно швидші результати ніж без індексування, а в деяких випадках і значно гірші. Це пояснюється тим, що пошук з індексуванням відбувається по всій таблиці бази даних, та він показує свою ефективність тільки при роботі з числовими типами.

**BRIN**

**Запити для тестування:**

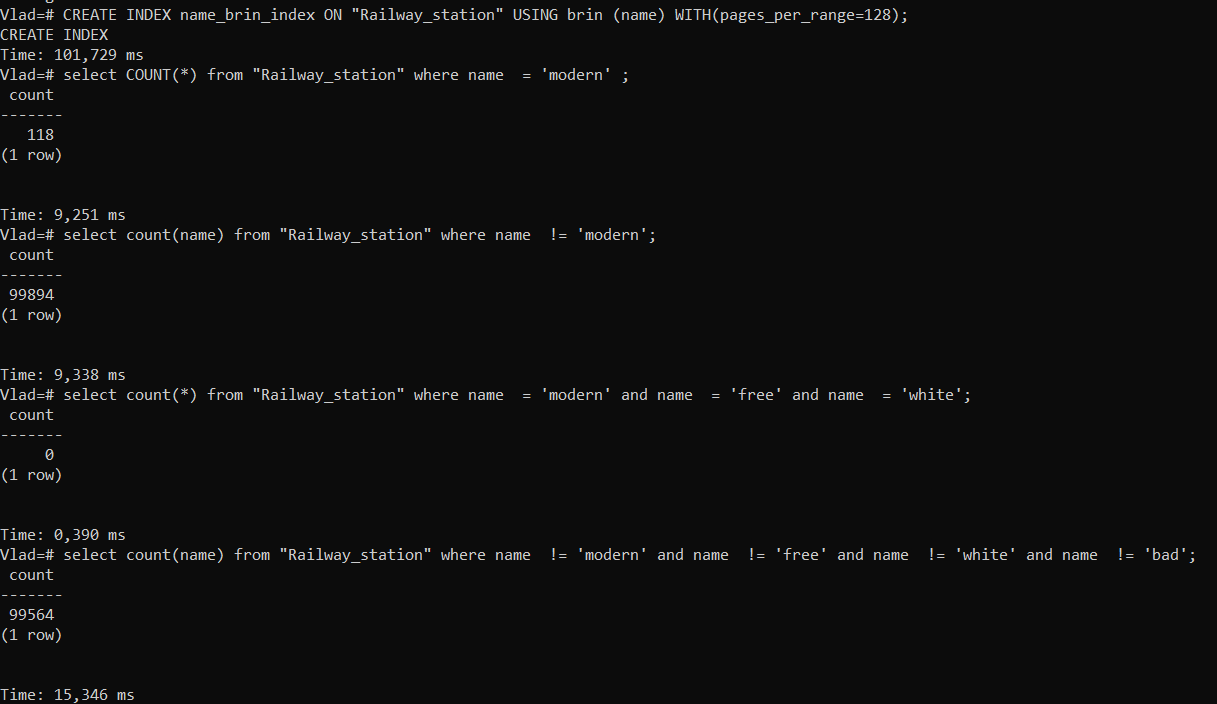
select COUNT(\*) from "Railway\_station" where name = 'modern' ;

select count(name) from "Railway\_station" where name != 'modern';

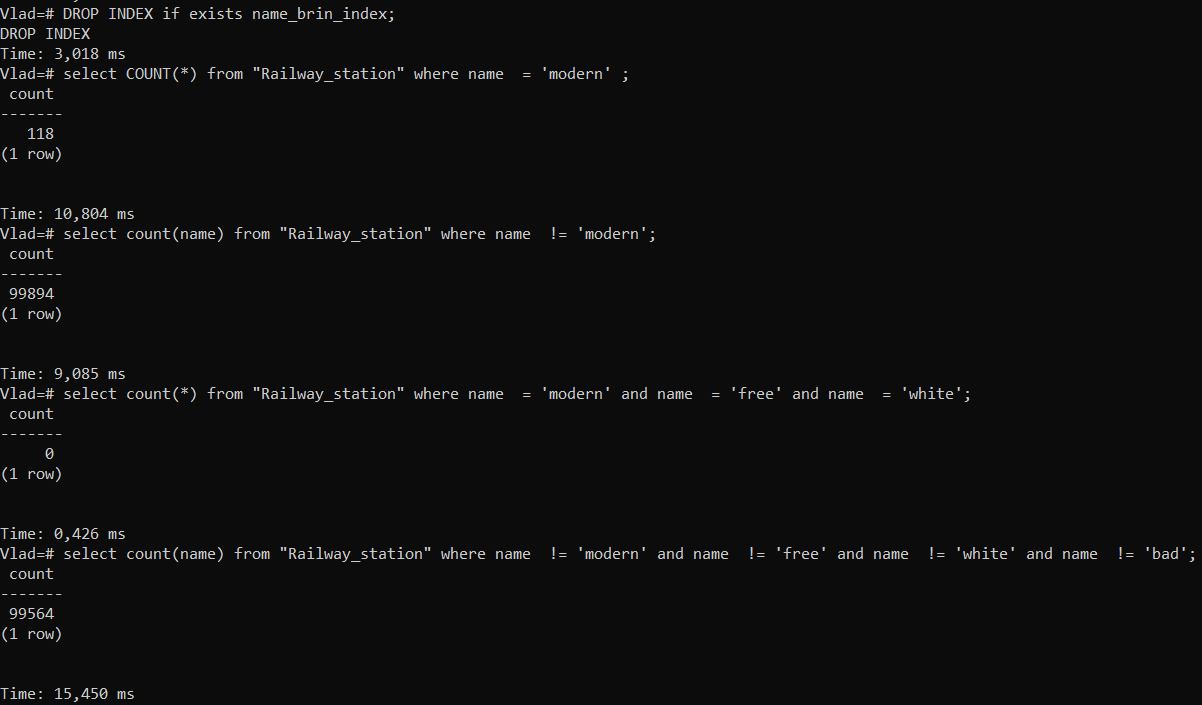
select count(\*) from "Railway\_station" where name = 'modern' and name = 'free' and name = 'white';

select count(name) from "Railway\_station" where name != 'modern' and name != 'free' and name != 'white' and name != 'bad';

**CREATE INDEX name\_brin\_index ON "Railway\_station" USING brin (name) WITH(pages\_per\_range=128);**



**DROP INDEX if exists name\_brin\_index;**



Бачим, що в деяких випадках індексування з допомогою BRIN показує на небагато швидші результати ніж без індексування, а в деяких випадках воно швидше. Це пояснюється тим, що даний індекс використовується не для швидкодії, а для уникнення перегляду непотрібних даних.

**Завдання №3**

***after delete, insert***

**Створення та заповнення таблиць баз даних для роботи з тригером:**

CREATE TABLE "train"(

"train\_id" serial PRIMARY KEY,

"train\_model" text,

"train\_count" int

);

CREATE TABLE "train\_log"(

"id" serial PRIMARY KEY,

"train\_log\_id" int,

"train\_log\_model\_name" text

);

INSERT INTO "train"("train\_model", "train\_count")

VALUES ('train1' , '10'), ('train2', '20'), ('train3', '30'), ('train4', '40'),

('train5', '50');

**Створення тригера:**

CREATE OR REPLACE FUNCTION after\_delete\_insert\_func() RETURNS TRIGGER as $trigger$

DECLARE

BEGIN

IF new."train\_count" <= 30 THEN

RAISE NOTICE 'train\_count <= 30';

INSERT INTO "train\_log"("train\_log\_id", "train\_log\_model\_name") VALUES (new."train\_id", new."train\_model" || '\_small');

RETURN NEW;

ELSE

RAISE NOTICE 'train\_count >= 30';

INSERT INTO "train\_log"("train\_log\_id", "train\_log\_model\_name") VALUES (new."train\_id", new."train\_model" || '\_big');

RETURN NEW;

END IF;

END;

$trigger$ LANGUAGE plpgsql;

CREATE TRIGGER "after\_delete\_insert\_trigger"

AFTER DELETE OR INSERT ON "train"

FOR EACH ROW

EXECUTE procedure after\_delete\_insert\_func();

**Запити для перевірки роботи тригера:**

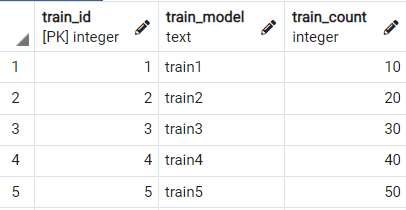
DELETE FROM "train" where "train\_id" = 4;

INSERT INTO "train"("train\_model", "train\_count") VALUES ('model6', '6') ;

**Результати тестування:**

**Вміст таблиць до початку роботи з ними:**

train:



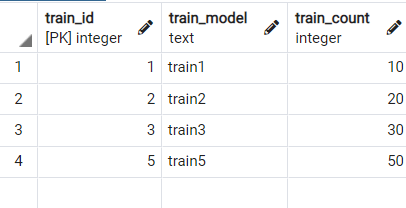
train\_log:



**Вміст таблиць після виконання запитів:**

**DELETE FROM "train" where "train\_id" = 4;**

train:

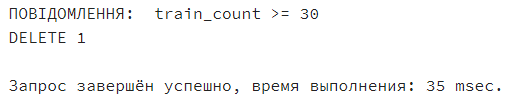


train\_log:



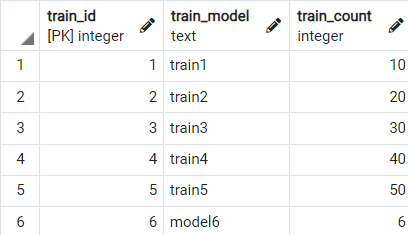
Бачимо, що запит був виконаний успішно, було видалене потрібне значення в таблиці train, в таблиці train\_log не було внесено змін так як тригер after delete має обмеження, а саме не обновляти нові та старі значення, проте бачимо, що тригерна функція спрацювала з (Додатку1).

Додаток 1.



**INSERT INTO "train"("train\_model", "train\_count") VALUES ('model6', '6') ;**

train:



train\_log:



Бачимо, що запит був виконаний успішно, нове значення було вставлене в таблицю train, та тригерна функція спрацювала успішно і також вставила нове значення в таблицю train\_log, та дописала до модель ‘\_small’, це показує, що спрацювала гілка яка відповідає за значення кількості вагонів менше 30.