

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

Гула М.Р.

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* |
| *6* | *BTree, BRIN* | *after update, insert* |

URL репозиторію з вихідним кодом: <https://github.com/HulaMR/BD>

**Завдання №1**

**Модель «сутність-зв’язок»:**

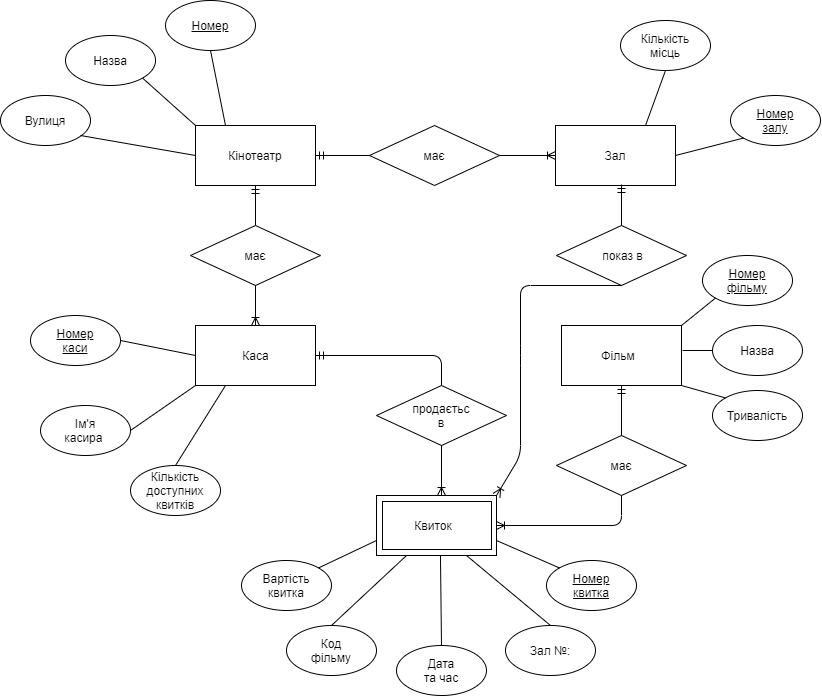


Рис1. ER-діаграма за нотацією Чена

**Схеми бази даних:**

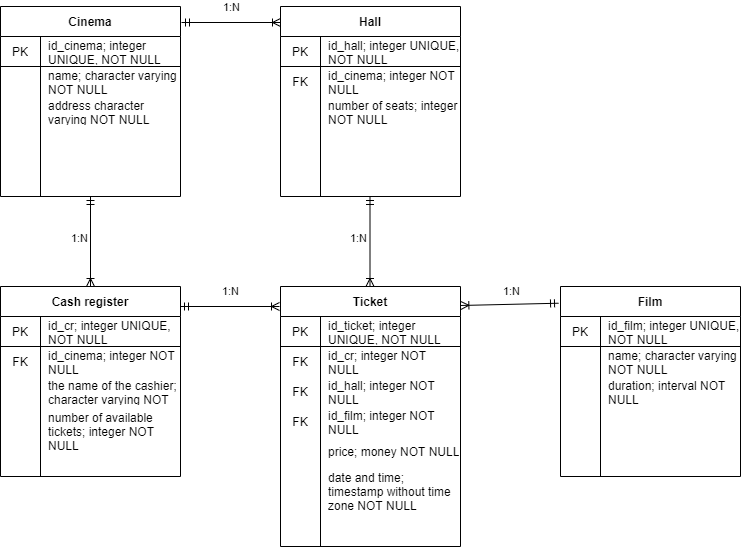


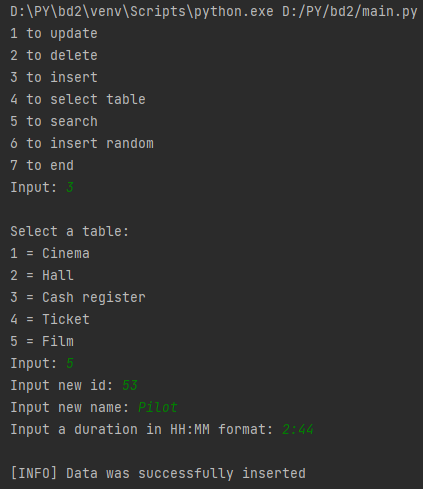
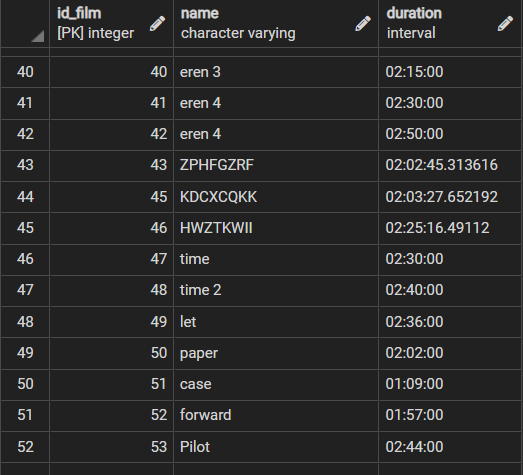
Рис2. Схема бази даних

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

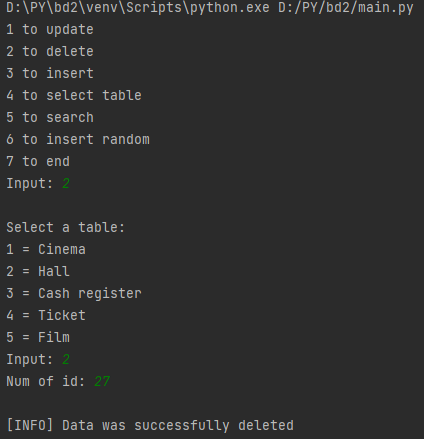
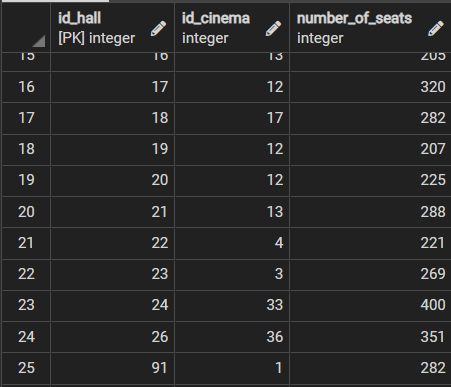
class Cinema(Base):  
 \_\_tablename\_\_ = 'Cinema'  
 id\_cinema = Column(Integer, primary\_key=True)  
 name = Column(String)  
 adress = Column(String)  
  
 crs = relationship('Cash\_register')  
 halls = relationship('Hall')  
  
 def \_\_init\_\_(self, id\_cinema: int, name: str, adress: str):  
 self.id\_cinema = id\_cinema  
 self.name = name  
 self.adress = adress  
  
  
class Cash\_register(Base):  
 \_\_tablename\_\_ = 'Cash\_register'  
 id\_cr = Column(Integer, primary\_key=True)  
 id\_cinema = Column(Integer, ForeignKey('Cinema.id\_cinema'))  
 the\_name\_of\_the\_cashier = Column(String)  
 number\_of\_available\_tickets = Column(Integer)  
  
 tick = relationship('Ticket')  
  
 def \_\_init\_\_(self, id\_cr: int, id\_cinema: int, the\_name\_of\_the\_cashier: str, number\_of\_available\_tickets: int):  
 self.id\_cr = id\_cr  
 self.id\_cinema = id\_cinema  
 self.the\_name\_of\_the\_cashier = the\_name\_of\_the\_cashier  
 self.number\_of\_available\_tickets = number\_of\_available\_tickets  
  
  
class Hall(Base):  
 \_\_tablename\_\_ = 'Hall'  
 id\_hall = Column(Integer, primary\_key=True)  
 id\_cinema = Column(Integer, ForeignKey('Cinema.id\_cinema'))  
 number\_of\_seats = Column(Integer)  
  
 tickets = relationship('Ticket')  
  
 def \_\_init\_\_(self, id\_hall: int, id\_cinema: int, number\_of\_seats: int):  
 self.id\_hall = id\_hall  
 self.id\_cinema = id\_cinema  
 self.number\_of\_seats = number\_of\_seats  
  
  
class Ticket(Base):  
 \_\_tablename\_\_ = 'Ticket'  
 id\_ticket = Column(Integer, primary\_key=True)  
 id\_cr = Column(Integer, ForeignKey('Cash\_register.id\_cr'))  
 id\_hall = Column(Integer, ForeignKey('Hall.id\_hall'))  
 id\_film = Column(Integer, ForeignKey('Film.id\_film'))  
 price = Column(Numeric)  
 date\_and\_time = Column(TIMESTAMP)  
  
 def \_\_init\_\_(self, id\_ticket: int, id\_cr: int, id\_hall: int, id\_film: int, price: float,  
 date\_and\_time: datetime.datetime):  
 self.id\_ticket = id\_ticket  
 self.id\_cr = id\_cr  
 self.id\_hall = id\_hall  
 self.id\_film = id\_film  
 self.price = price  
 self.date\_and\_time = date\_and\_time  
  
  
class Film(Base):  
 \_\_tablename\_\_ = 'Film'  
 id\_film = Column(Integer, primary\_key=True)  
 name = Column(String)  
 duration = Column(Interval)  
  
 tickets = relationship('Ticket')  
  
 def \_\_init\_\_(self, id\_film: int, name: str, duration: datetime.time):  
 self.id\_film = id\_film  
 self.name = name  
 self.duration = duration

**Результат реалізації вставки, вилучення та редагування**

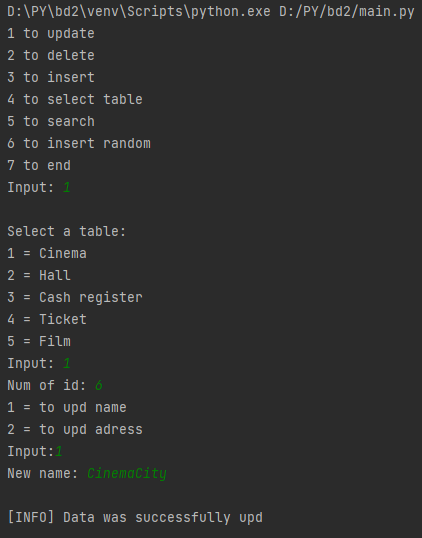
Вставка:

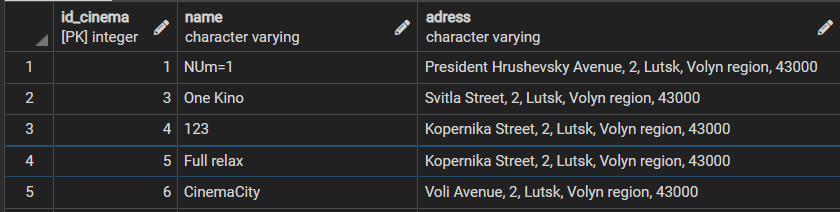
 

Вилучення:

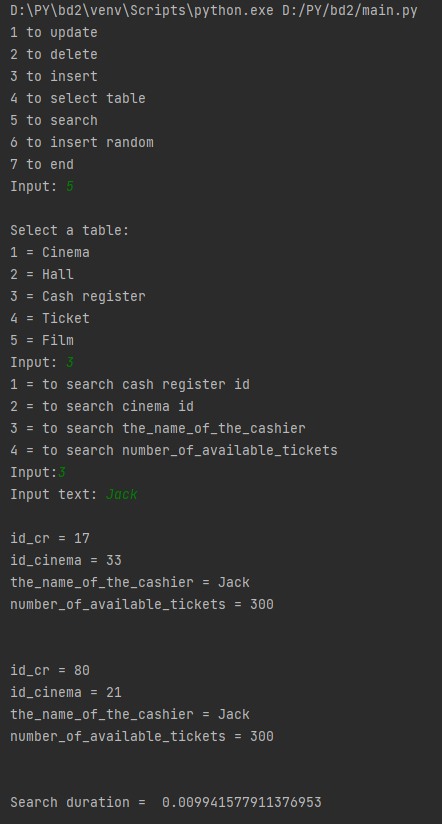
Редагування:



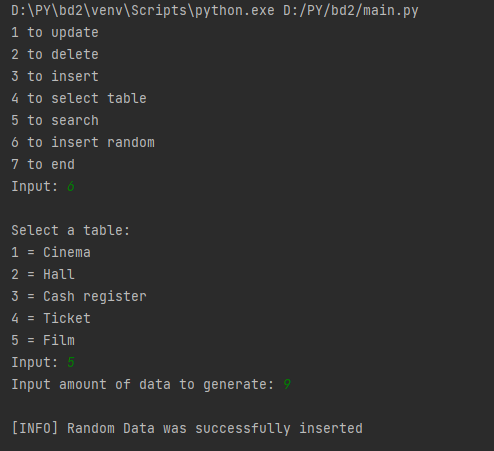
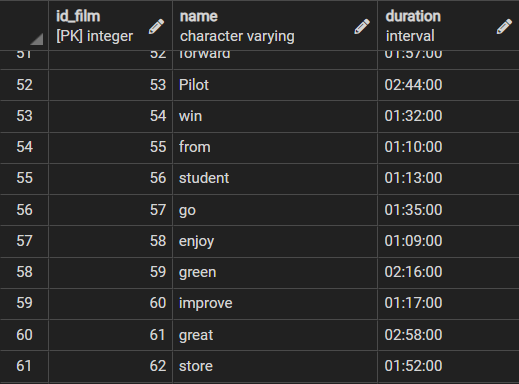


**Результат реалізації пошуку та генерації даних**

Пошук:



Генерація даних:

**Завдання №2**

**BTree**

Запити для тестування:

explain analyze select \* from “Film” where duration = ’2:06:00’ ;

explain analyze select \* from “Film” where duration < ’1:01:00’ ;

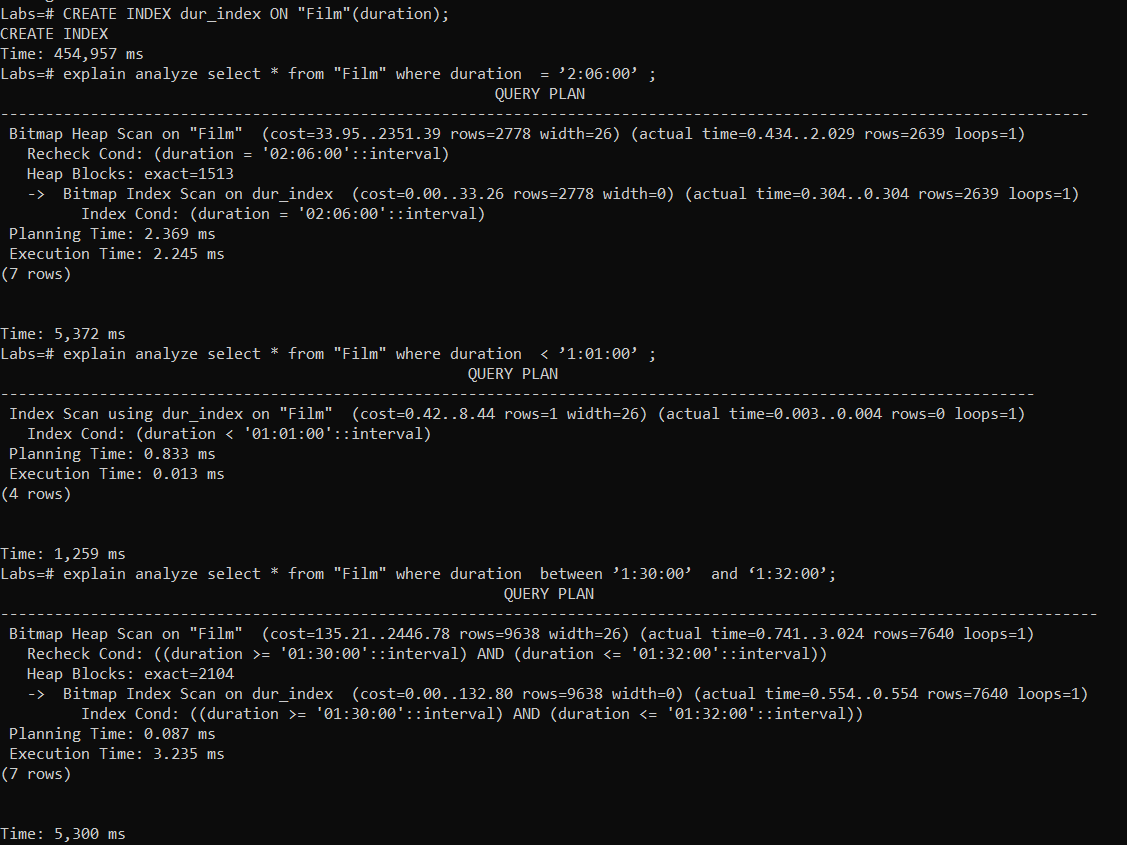
explain analyze select \* from “Film” where duration between ’1:30:00’ and ‘1:32:00’;

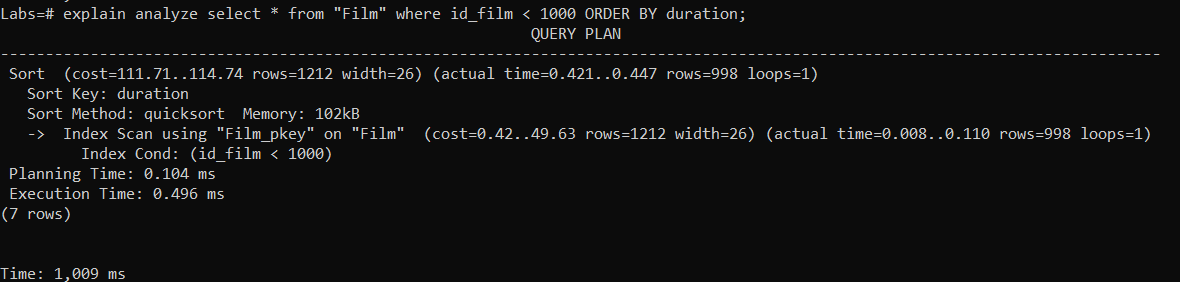
explain analyze select \* from “Film” where id\_film < 1000 ORDER BY duration;

explain analyze select \* from “Film” where duration = ’2:06:17.575575’ ;

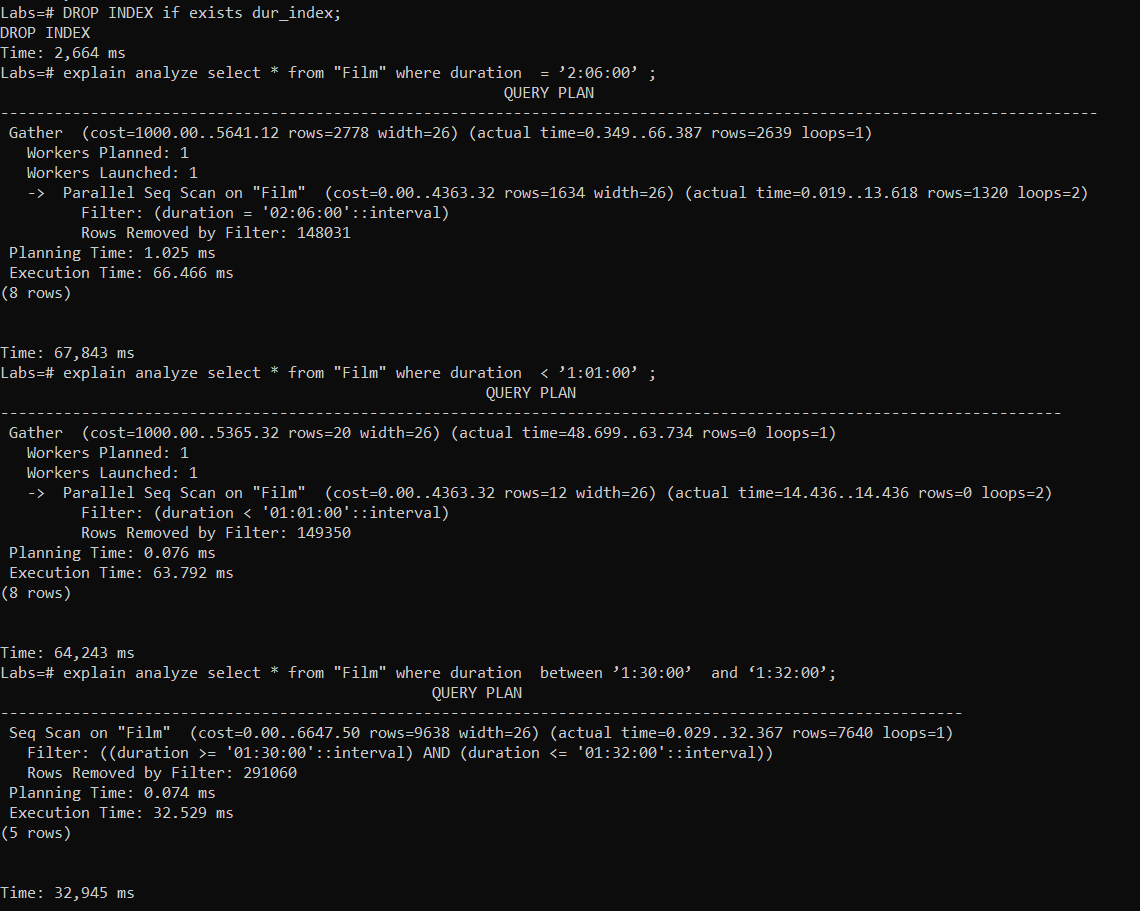
**Результати тестування:**

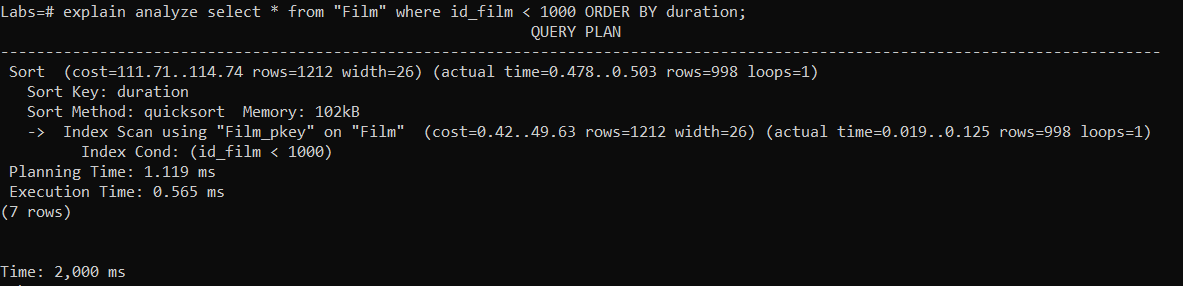
**CREATE INDEX dur\_index ON “Film”(duration);**

****

****

**DROP INDEX if exists dur\_index;**

****

****

Отже з отриманих результатів бачимо що, пошук з індексацією відбувається набагато швидше, як і без індексації. BTree має свою особливість, а саме, він показує свою ефективність тоді коли працює з невеликою кількістю даних.

При сортуванні даних з індексацією бачимо, що вона відбулась швидше, ніж без індексації, це пов’язано також з нормальною кількістю даних для індексу BTree.

**BRIN**

Запити для тестування:

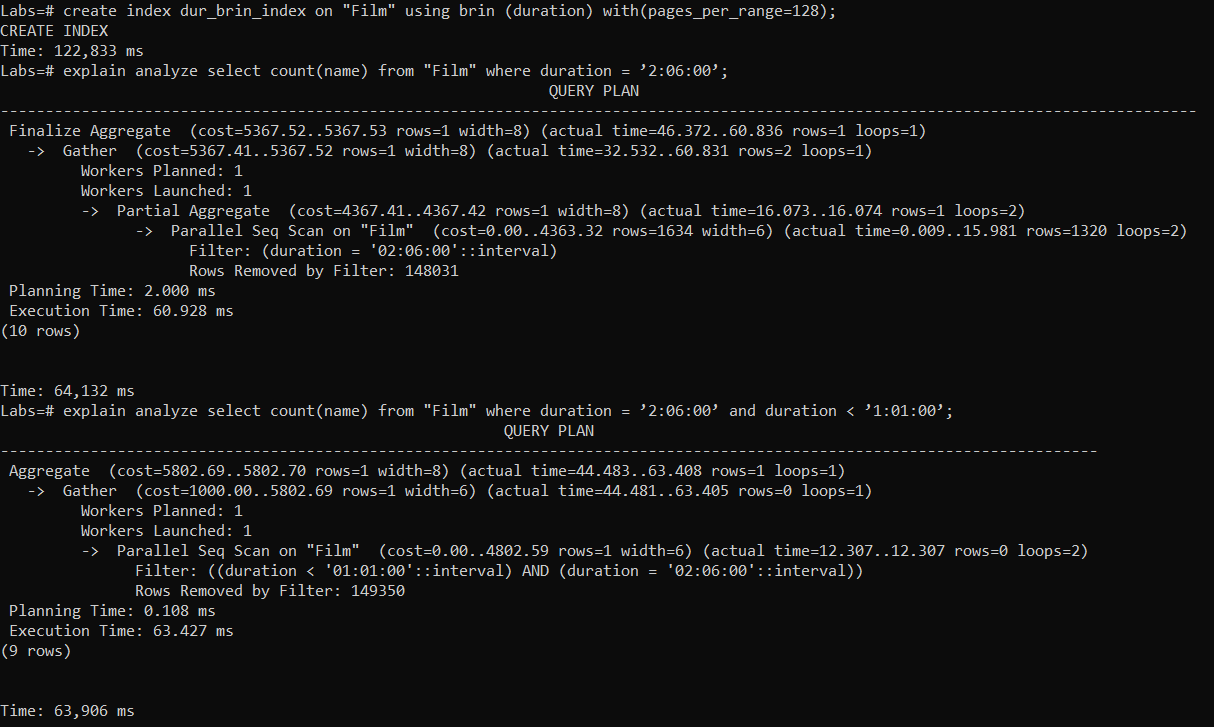
explain analyze select count(name) from “Film” where duration = ’2:06:00’;

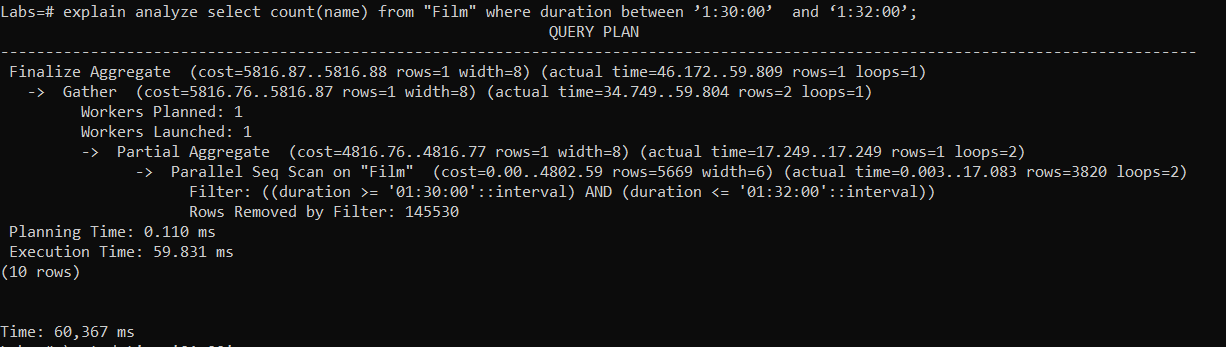
explain analyze select count(name) from “Film” where duration = ’2:06:00’ and duration < ’1:01:00’;

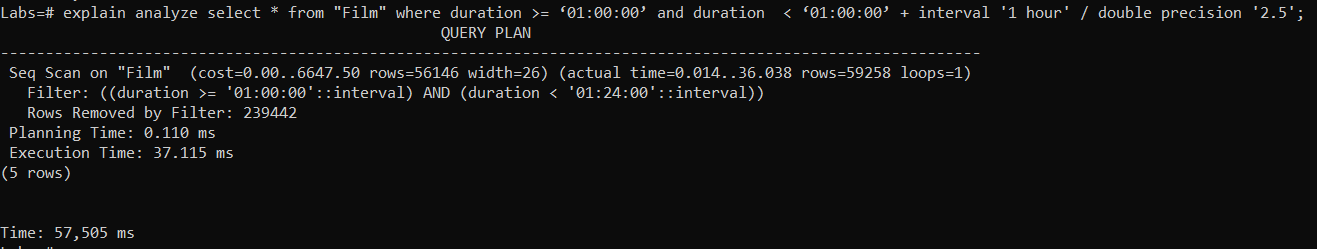
explain analyze select count(name) from “Film” where duration between ’1:30:00’ and ‘1:32:00’;  
explain analyze select \* from “Film” where duration >= ‘01:00:00’ and duration < ‘01:00:00’ + interval '1 hour' / double precision '2.5';

**Результати тестування:**

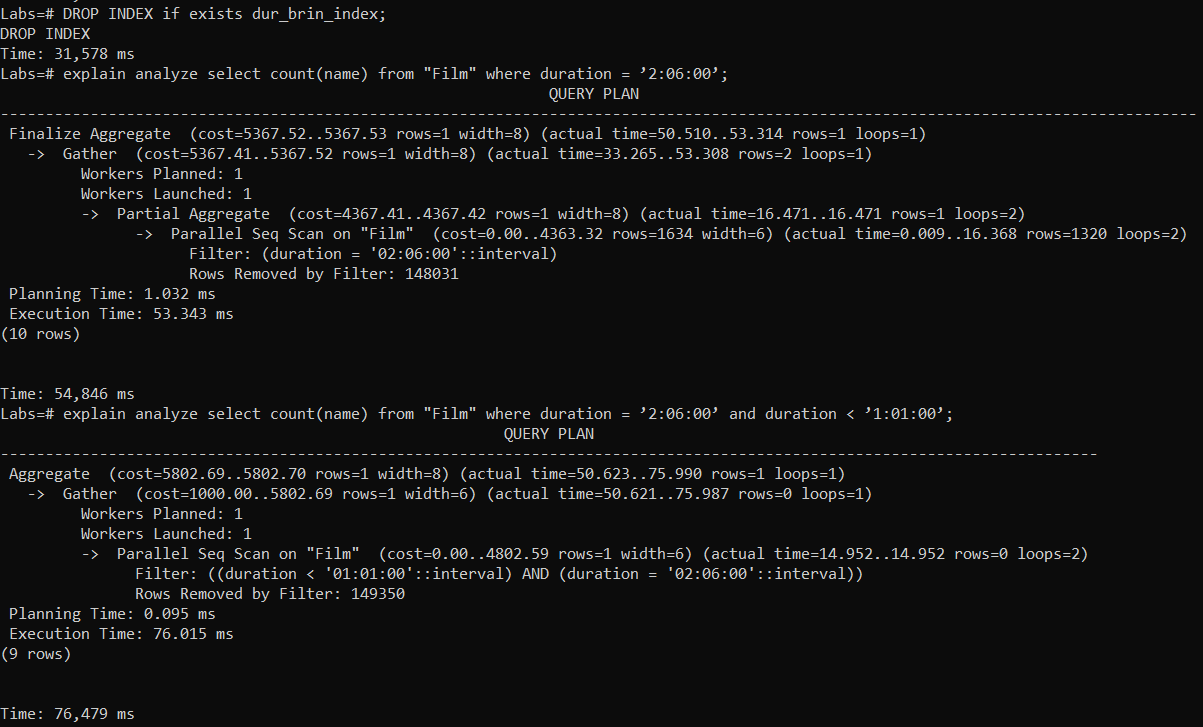
**create index dur\_brin\_index on “Film” using brin (duration) with(pages\_per\_range=128);**

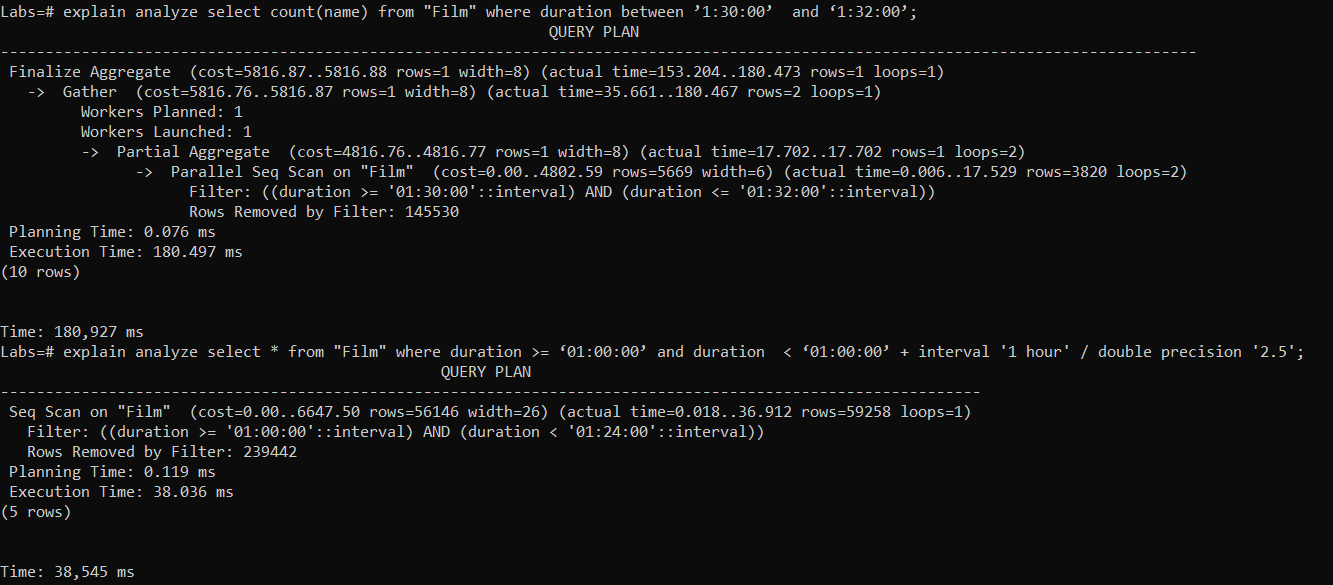






**DROP INDEX if exists dur\_brin\_index;**

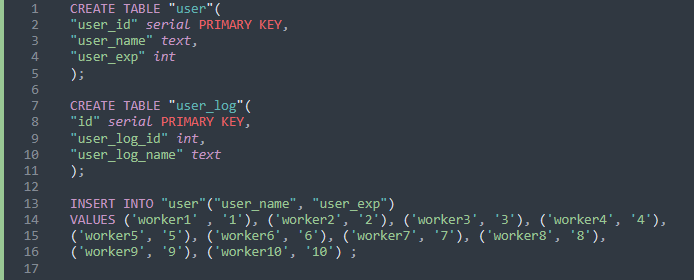




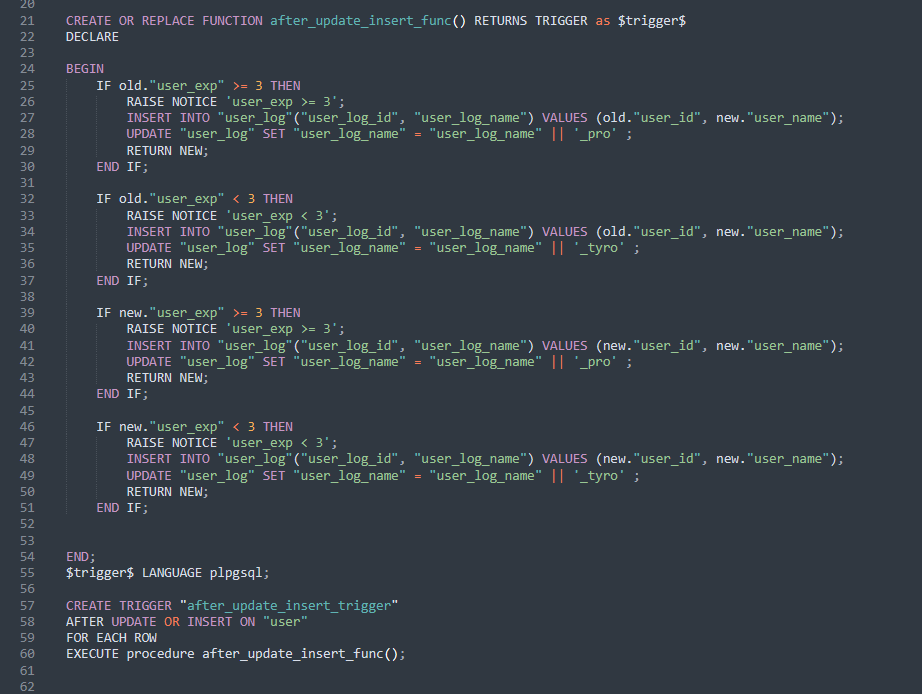
Індексування з допомогою BRIN показало не самі кращі результати, в деяких випадках воно швидше, а в деяких повільніше, ніж без індексування. Проте суть цього індексу не в швидкодії знаходження потрібних рядків, а в тому щоб уникнути перегляд зарання непотрібних рядків.

**Завдання №3**

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



**Створення тригера:**



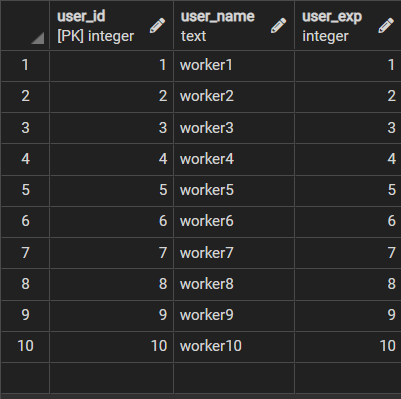
**Запити для перевірки роботи тригера:**



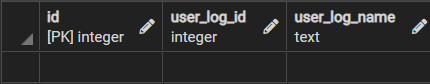
**Результати тестування:**

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

Таблиця user:



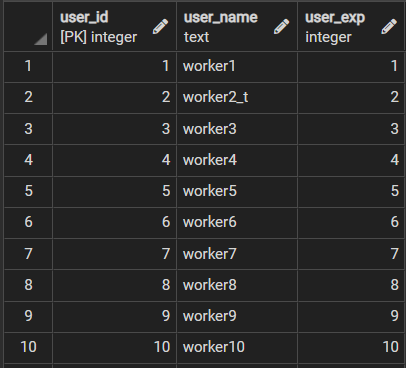
Таблиця user\_log:



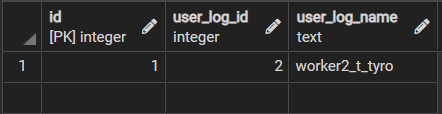
**Вміст таблиць після виконання запитів:**

UPDATE "user" SET "user\_name" = "user\_name" || '\_t' WHERE "user\_exp" = 2;

Таблиця user:



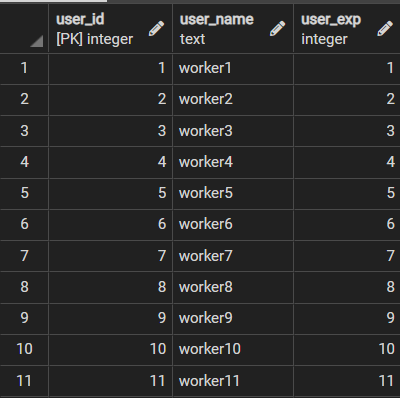
Таблиця user\_log:



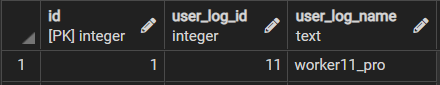
Бачимо, що запит виконався правильно та було дописано ‘\_t’ в першій таблиці, та спрацювала гілка тригерної функції, яка відповідає за оновлення та запис в таблицю user\_log, імені вже занесеного в таблицю user із стажем менше трьох років, та успішно дописала ‘\_tyro’.

INSERT INTO "user"("user\_name", "user\_exp") VALUES ('worker11', '11') ;

Таблиця user:



Таблиця user\_log:



Бачимо, що запит виконався правильно та додав дані в першу таблиці, та спрацювала гілка тригерної функції, яка відповідає за оновлення та запис в таблицю user\_log, нового імені з таблиці user із стажем більшим або рівним трьом рокам, та успішно дописала ‘\_pro’.