

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»**

Виконав: студент ІII курсу

ФПМ групи КВ-83

Лазуткін Олег

Київ – 2020

**Лабораторна робота № 3.**

Засоби оптимізації роботи СУБД PostgreSQL

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

Завдання роботи полягає у наступному:

Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM).

Створити та проаналізувати різні типи індексів у PostgreSQL.

Розробити тригер бази даних PostgreSQL.

**Вимоги до пункту завдання №1**

Для перетворення функцій, що реалізують запити до об’єктної бази даних, необхідно встановити бібліотеку sqlAlchemy, налаштувати програму на роботу з ORM, розробити класи-сутності для об’єктів-сутностей, представлених відповідними таблицями БД та пов’язаних зв’язками 1:М, М:М та 1:1 виконати опис схеми бази даних. Особливу увагу приділити контролю зовнішніх зв’язків між таблицями засобами ORM.

Замінити виклики запитів мовою SQL на відповідні запити засобами SQLAlchemy по роботі з об’єктами. Обов’язковим є реалізація вставки, вилучення та редагування екземплярів класів-сутностей. Розробка запитів на генерацію даних та пошук екземплярів класів-сутностей вітається, але не є обов’язковою.

Інтерфейси функцій (вхідні та вихідні аргументи функцій модуля “Модель”) мають залишитись без змін.

**Вимоги до пункту завдання №2**

Відповідно до варіанту індексування продемонструвати на прикладах запитів SQL SELECT підвищення швидкодії їх виконання з використанням індексів, а також пояснити чому для деяких випадків індексування використовувати недоцільно. При цьому для наочного представлення слід використати функцію генерування рандомізованих даних з лабораторної роботи №2, створивши необхідну кількість тестових даних. Навести 4-5 прикладів запитів SELECT (із виведенням результуючих даних), що містять фільтрацію, агрегатні функції, групування та сортування (у необхідних комбінаціях).

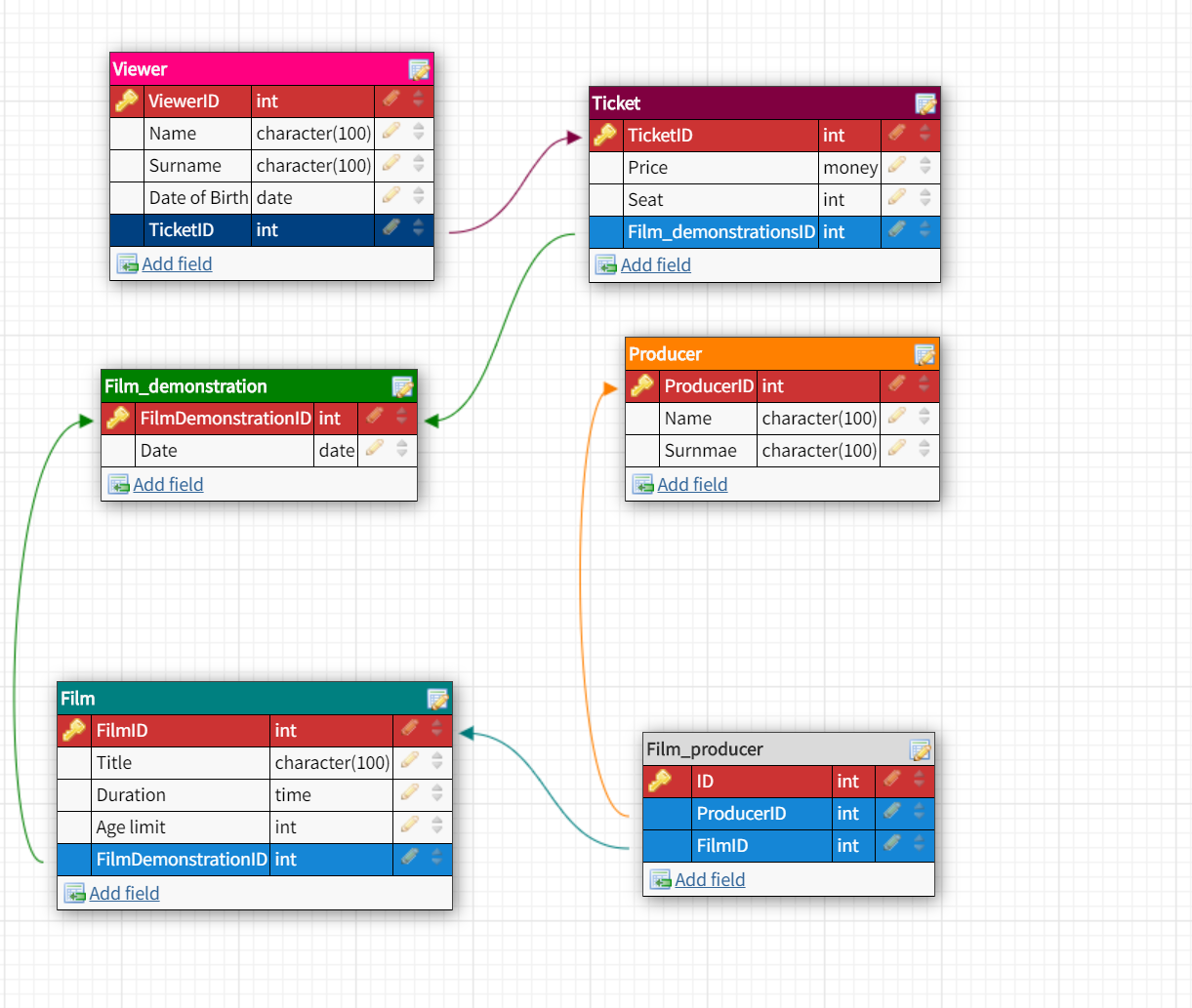
**Вимоги до пункту завдання №3**

Створити тригер бази даних PostgreSQL відповідно до варіанта. Тригерна функція має включати обробку запису, що модифікується (вставляється або вилучається), умовні оператори, курсорні цикли та обробку виключних ситуацій. Виконати відлагодження тригера при різних вхідних даних, навівши 2-3 приклади його використання.

Варіант 12. Індекси: Btree, GIN. Тип тригерів: *after update, insert.*

**Завдання 1**

Логічна схема бази даних “Кінотеатр”



Моделі ORM

public class Film

{

public int id { get; set; }

public string title { get; set; }

public int age\_limit { get; set; }

public int film\_demonstration\_id { get; set; }

public FilmDemonstration film\_demonstration { get; set; }

}

public class FilmDemonstration

{

public int id { get; set; }

public DateTime date { get; set; }

}

public class FilmProducer

{

public int id { get; set; }

public int producer\_id { get; set; }

public Producer producer { get; set; }

public int film\_id { get; set; }

public Film film { get; set; }

}

public class Producer

{

public int id { get; set; }

public string name { get; set; }

public string surname { get; set; }

}

public class Ticket

{

public int id { get; set; }

public decimal price { get; set; }

public int seat { get; set; }

public int film\_demonstration\_id { get; set; }

public FilmDemonstration film\_demonstration { get; set; }

}

public class Viewer

{

public int id { get; set; }

public string name { get; set; }

public string surname { get; set; }

public DateTime date\_of\_birth { get; set; }

public int ticker\_id { get; set; }

public Ticket ticket { get; set; }

}

Приклади запитів у вигляді ORM

Read:

context.film

Create:

context.film.Add(film);

Delete:

context.film.Remove(entity);

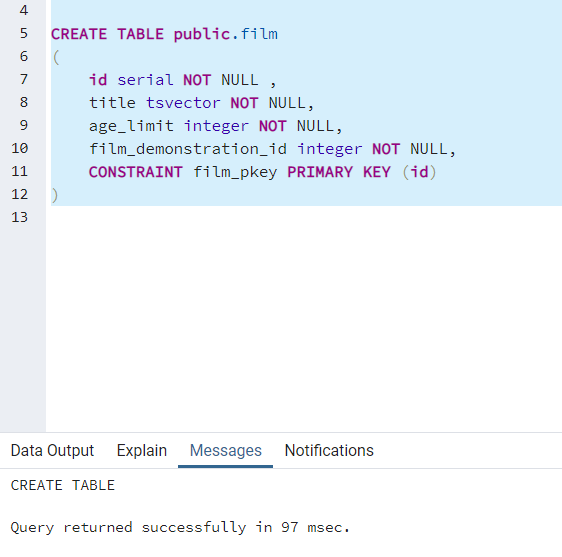
Update:

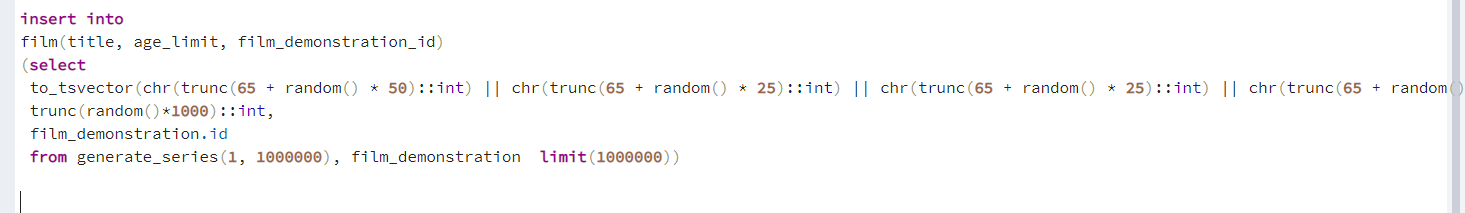
context.film.Update(entity);

Find:

context.film.Find(findId);

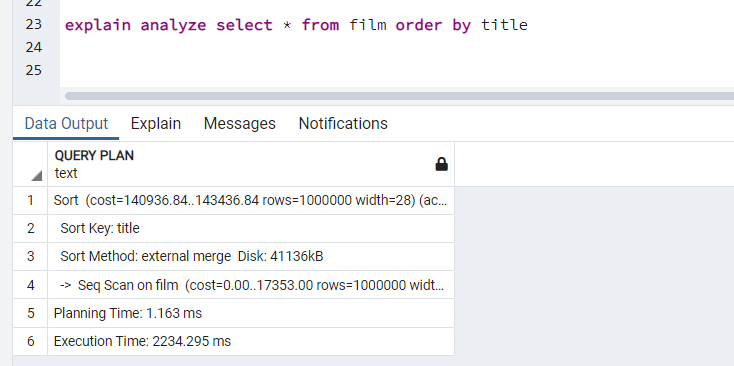
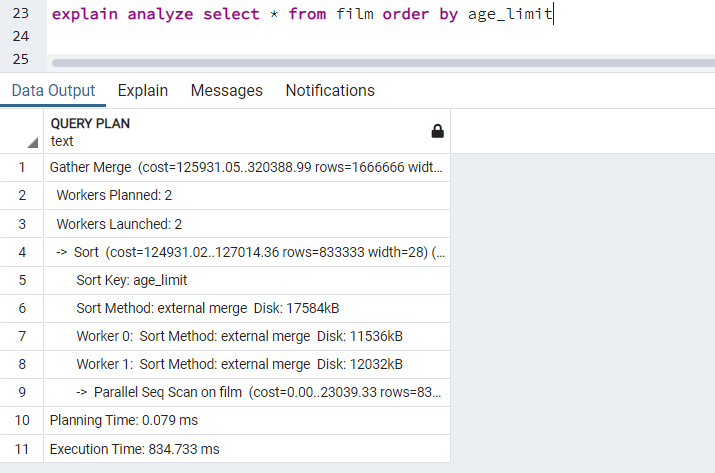
**Завдання 2**





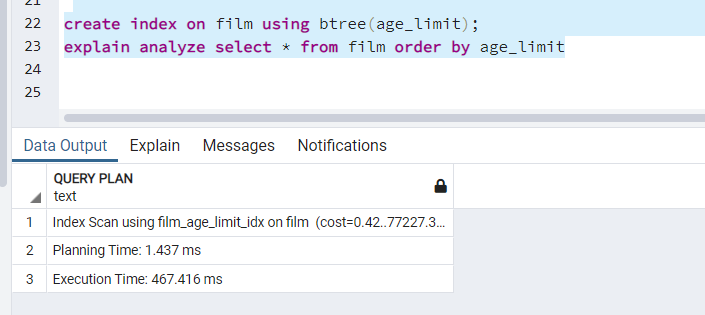
Перш за все я змінив тип даних поля title на tsvector для того, щоб можна було працювати з індексом GIN. Згенеруємо випадкові записи.

Без використання індекса

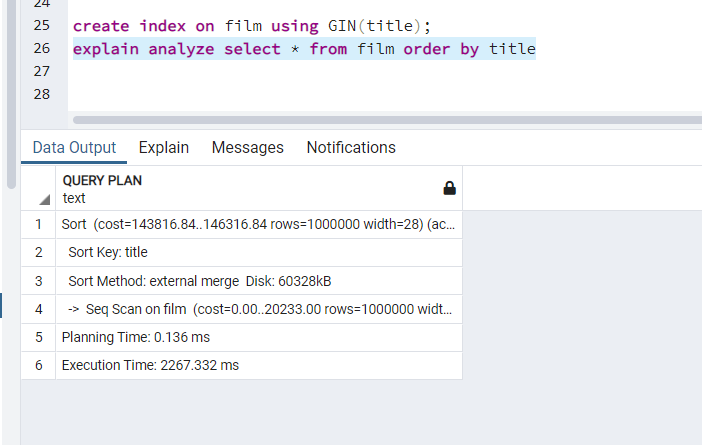
Сортування без індексу відбувається дуже повільно, особливо для текстового поля title.

BTREE індекс



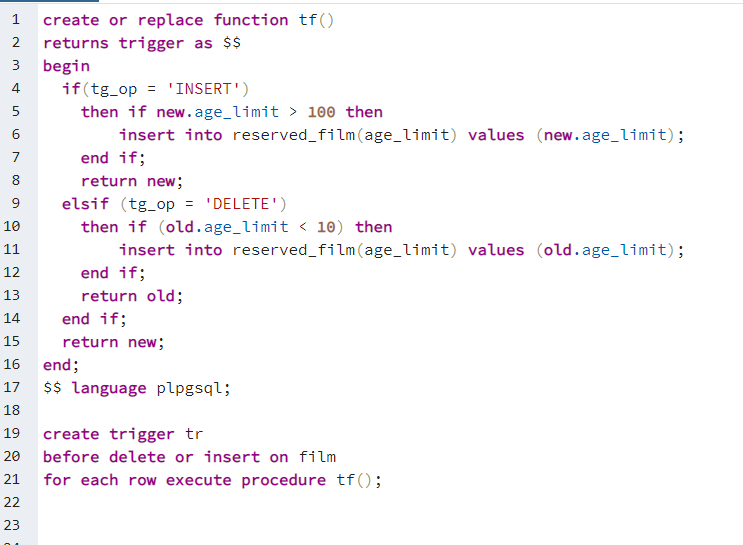
Як видно, з Btree індексом execution time вдвічі зменшився. Це не дивно, адже цей індекс фактично зберігає в собі уже відсортовані дані, тому сортування займає значно менше часу. Для того щоб кожного разу знаходити найменший елемент в масиві тих, що ще не використовувались, достатньо просто рекурсивно проходити по дереву, кожного разу переходити до вузла з найменшим значенням. Коли вузол уже не ссилається ні на один інший вузол, повертаємо його значення і повертаємося на рівень вище.

GIN індекс



Даний індекс створений для повнотекстового пошуку, тому дуже добре оптимізований для великих рядків.

**Завдання 3**



Перевірю його роботу:

