

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

Левчук Євген

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.

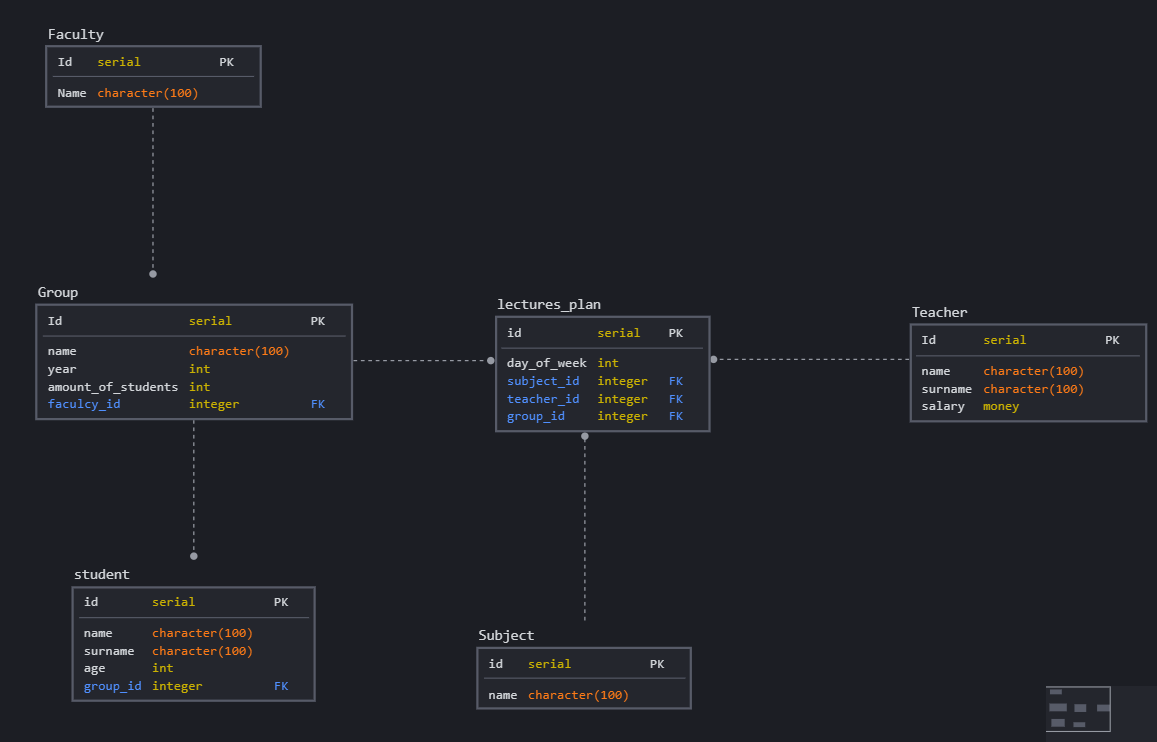
Вимоги до інструментарію:

1. Бібліотека для реалізації ORM - [SQLAlchemy для Python](https://www.sqlalchemy.org/) або інша з подібною функціональністю.
2. Середовище для відлагодження SQL-запитів до бази даних – pgAdmin 4.
3. СУБД - PostgreSQL 11-12.

Варіант 13: Btree, BRIN. Before insert, delete.

**Завдання 1**

Логічна схема бази даних ‘Університет’



Моделі ORM

public class Faculty

{

public int id { get; set; }

public string name { get; set; }

}

public class Group

{

public int id { get; set; }

public string name { get; set; }

public int year { get; set; }

public int amount\_of\_students { get; set; }

public Faculty faculty { get; set; }

public int faculty\_id { get; set; }

}

public class LecturesPlan

{

public int id { get; set; }

public int day\_of\_week { get; set; }

public int group\_id { get; set; }

public Group group { get; set; }

public int subject\_id { get; set; }

public Subject subject { get; set; }

public int teacher\_id { get; set; }

public Teacher teacher { get; set; }

}

public class Student

{

public int id { get; set; }

public string name { get; set; }

public string surname { get; set; }

public int age { get; set; }

public int group\_id { get; set; }

public Group group { get; set; }

}

public class Subject

{

public int id { get; set; }

public string name { get; set; }

}

public class Teacher

{

public int id { get; set; }

public string name { get; set; }

public string surname { get; set; }

public decimal salary { get; set; }

}

Для того, щоб додати зовнішній ключ, достатньо додати два поля:

1. Поле типу int яке містить назву зовнішнього ключа і суфікс \_id (наприклад для таблиці Student з її зовнішнім ключем group – group\_id);
2. Поле типу моделі таблиці, на яку зовнішній ключ зсилається.

Також програма містить так званий ‘клас контексту’, який відповідає за створення бази даних. В ньому ''регіструються'' всі моделі таблиць бази даних. Відношення створюються автоматично.

public class ApplicationContext : DbContext

{

public DbSet<Faculty> faculties { get; set; }

public DbSet<Group> groups { get; set; }

public DbSet<LecturesPlan> lectures\_plan { get; set; }

public DbSet<Student> students { get; set; }

public DbSet<Subject> subjects { get; set; }

public DbSet<Teacher> teachers { get; set; }

public ApplicationContext()

{

Database.EnsureCreated();

}

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseNpgsql("Host=localhost;Username=postgres;Password=p;Database=bdlz3");

}

}

Приклади запитів у вигляді ORM

Всі запити містяться в контролерах. Для прикладу візьмемо контролер FacultyController.cs який використовується для факультету.

Приклад запиту get (і відразу виведення його на екран):

foreach(var i in context.faculties)

{

Console.WriteLine("Id: {0}", i.id);

Console.WriteLine("Name: {0}", i.name);

Console.WriteLine();

}

Приклад запиту put:

context.faculties.Add(new Faculty { name = name });

context.SaveChanges();

Приклад запиту delete:

context.faculties.Remove(entity);

context.SaveChanges();

Приклад запиту update:

context.faculties.Update(entity);

context.SaveChanges();

**Завдання 2**

Для того, щоб перевірити швидкодію різних типів індексів, нам необхідна велика кількість записів. Згенеруємо їх запитом з другої лабораторної роботи.

insert into students(name, surname, age, group\_id)

(select chr(trunc(65 + random() \* 50)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int),

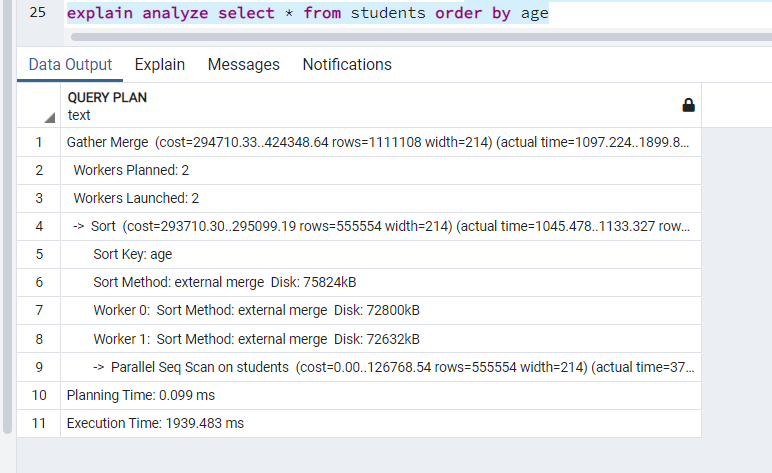
chr(trunc(65 + random() \* 50)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int),

10 + trunc(random() \* 30)::int,

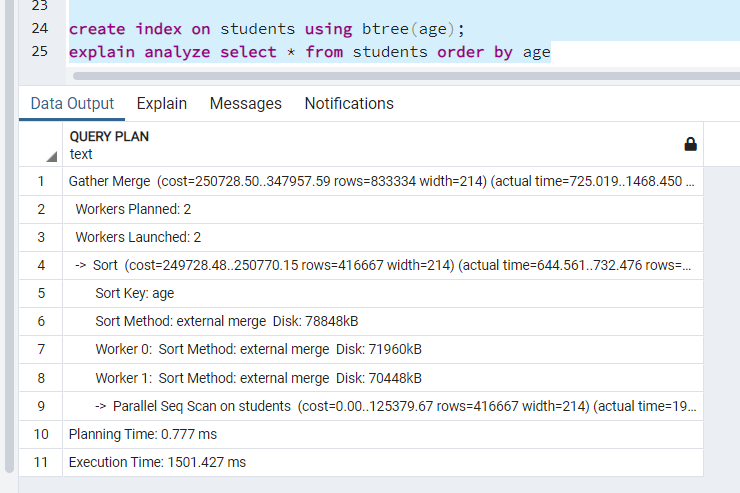
groups.id

from generate\_series(1, 1000000), groups limit(1000000))

Без використання індекса

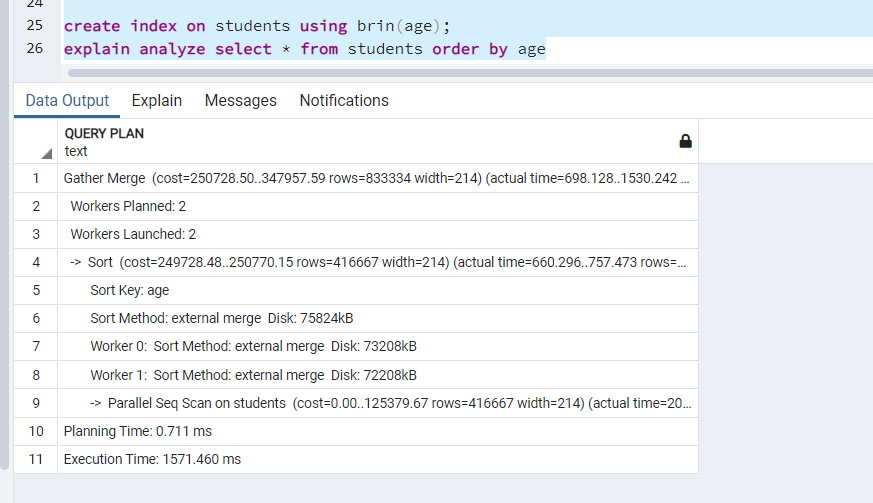


Btree індекс



Як видно, з Btree індексом execution time значно 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.

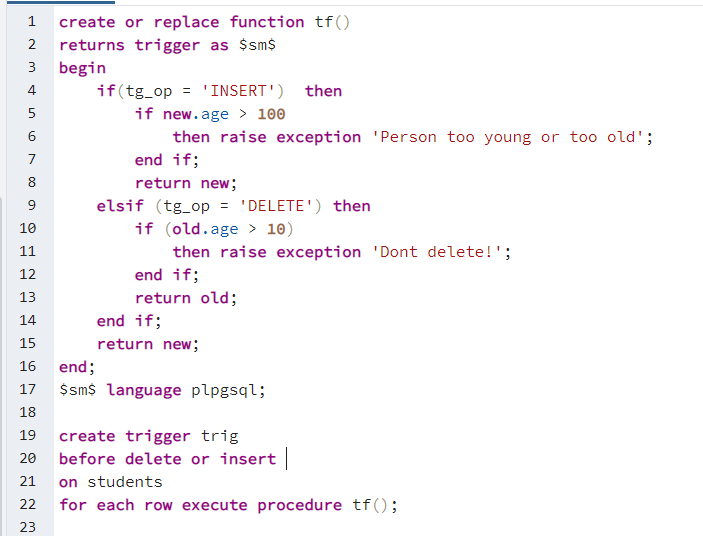
BRIN індекс



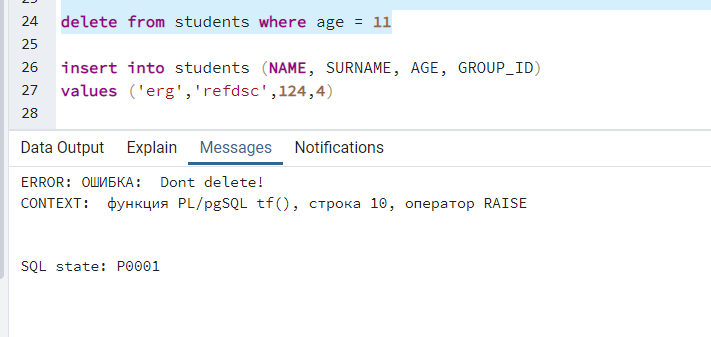
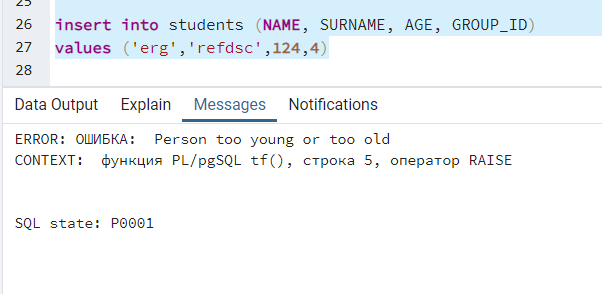
Як видно на зображенні, сортування з BRIN індексом працює швидше, ніж без нього. BRIN індекс працює так: всі дані діляться на секції, і кожного разу, коли ми шукаємо мінімальне число, ми дивимось на метадані кожної секції. Зазвичай там зберігається мінімальне і максимальне число секції, але може бути й по іншому. Це дозволяє не проглядати зайвий раз деякі ділянки пам’яті.

**Завдання 3**

**Before delete, insert**



Даний тригер не дає видалити записи в яких age > 10 а також не дозволяє вставити записи в яких age < 10 або age > 100. Перевіримо.



Все працює вірно.