

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

Проценко Владислав

Київ – 2020

**Засоби оптимізації роботи СУБД PostgreSQL**

*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).

1. Створити та проаналізувати різні типи індексів у PostgreSQL.
2. Розробити тригер бази даних PostgreSQL.

**Варіант 21**

У другому завданні проаналізувати індекси GIN, BRIN.

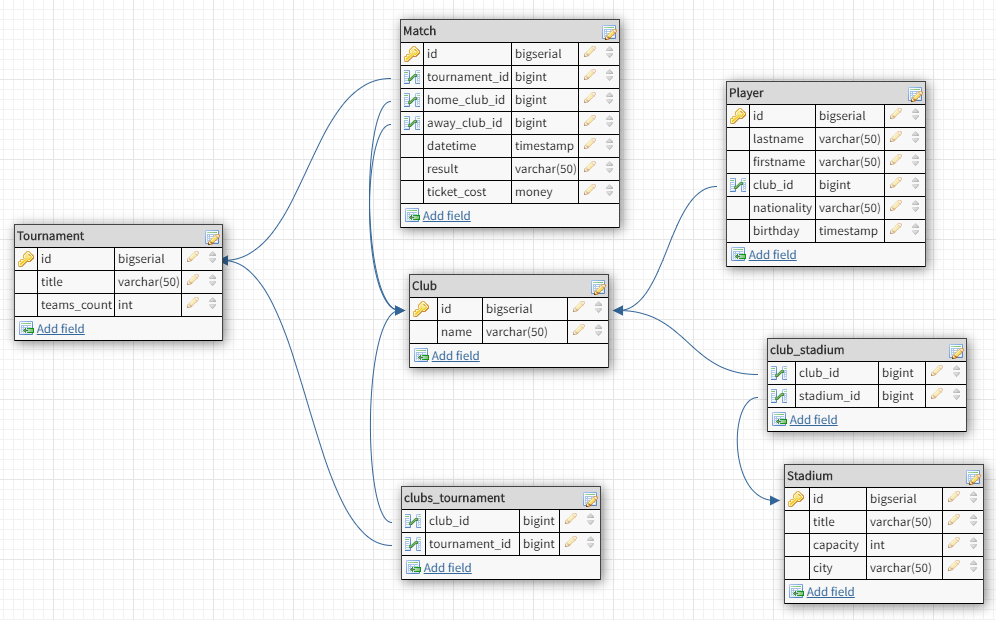
Умова для тригера – before delete, update.

Навігація по звіту:

1. [Завдання 1(перероблення програми в ORM модель)](#Завдання1);
2. [Завдання 2(створення індексів)](#Завдання2);
3. [Завдання 3(створення тригерів)](#Завдання3).

Завдання 1

Нормалізована база даних з першої лабораторної роботи



**Тепер розглянемо реалізацію даної бази даних за допомогою платформи Node.js і бібліотеки Sequelize. Далі на прикладі розберемо таблицю Club та її зв’язки. Для цього її сутність в окремому файлі.**

const { DataTypes } = require('sequelize');  
const ***db*** = require('../db');  
  
const ***Club*** = ***db***.define('club', {  
 name: {  
 type: Datatypes. ***STRING***,  
 }  
}, {  
 underscored: true,  
 freezeTableName: true,  
 timestamps: false,  
});  
  
***Club***.associate = models => {  
 ***Club***.*hasMany*(models.Player, {  
 foreignKey: 'club\_id',  
 as: 'players',  
 });  
 ***Club***.*belongsToMany*(models.Tournament, {  
 foreignKey: 'club\_id',  
 through: models.ClubsTournaments,  
 });  
}  
  
  
module.exports = ***Club***;

В цьому файлі ми створили Об’єкт/Клас який описую поля цієї таблиці та її зв’язки. А саме – клуб має багато гравців і належить багатьом турнірам, а їх зв’язок зберігає Об'єкт/Клас ClubsTournaments в іншому файлі.

Тепер, щоб отримати всіх гравців треба всього написати:

***router***.get('/club', async (req, res) => {  
 try {  
 const clubs = await ***Club***.*findAll*();  
 res.json(clubs);  
 }catch (e) {  
 res.status(400).json(e);  
 }  
});

Для пошуку за критеріями:

***router***.post('/club/search', async (req, res) => {  
 try {  
 const {name} = req.body;  
 const clubs = await ***Club***.*findAll*({  
 where: {  
 name: {  
 [***Op***.iLike]: `%${name}%`  
 }  
 }  
 });  
 res.json(clubs);  
 } catch (e) {  
 res.status(400).json(e);  
 }  
});

Для видалення:

***router***.delete('/club/:id', async (req, res) => {  
 try {  
 const response = await ***Club***.destroy({  
 where: {  
 id: req.params.id  
 }  
 });  
  
 res.json(response);  
 } catch (e) {  
 res.status(400).json(e);  
 }  
});

Для створення:

***router***.post('/club/new', async (req, res) => {  
 try {  
 const {name} = req.body;  
 const newClub = await ***Club***.create({  
 name  
 });  
 res.json(newClub);  
 } catch (e) {  
 res.status(400).json(e);  
 }  
});

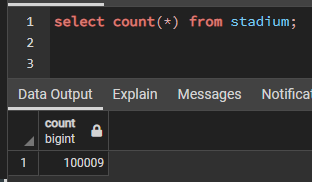
Для змінення за id:

***router***.put('/club/:id', async (req, res) => {  
 try {  
 ***console***.log(req.params.id, 'body', req.body);  
  
 const {name} = req.body;  
 let updatedClub = await ***Club***.update({  
 name: name,  
 }, {  
 where: {  
 id: req.params.id  
 }  
 });  
 if(updatedClub) {  
 updatedClub = await ***Club***.*findOne*({  
 where: {  
 id: req.params.id  
 }  
 });  
 }  
 res.json(updatedClub);  
 } catch (e) {  
 res.status(400).json(e);  
 }  
});

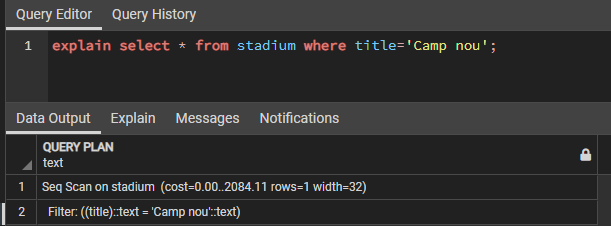
Завдання 2

**Btree**

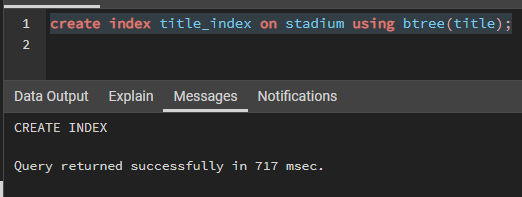
BTree індекс ефективний при пошуку по ключах, при використані умов( greater than, less than), пошуку по рядку або діапазону. Для цього завдання створимо 100000 рядків в таблиці стадіон.



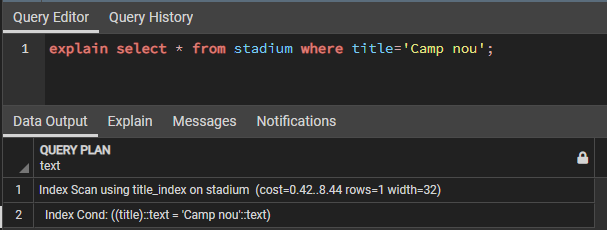
Тестувати цей приклад будемо на стовпчику title. Бо цей індекс ефективний при пошуку по ключах або діапазону. Зробимо пошук title=’Camp nou’;



Отримали cost=2084 при Seq Scan. Тепер створимо BTree індекс.



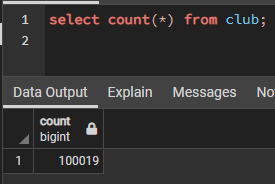
Тепер виконаємо той самий explain select запит. І отримуємо Cost = 8 при Index Scan.

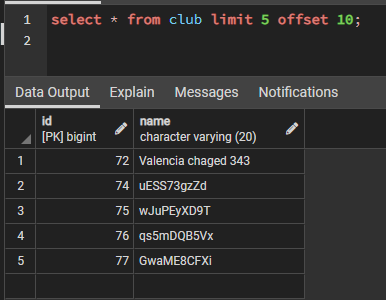


При використанні Btree індексу отримали пришвидшення в 256 раз.

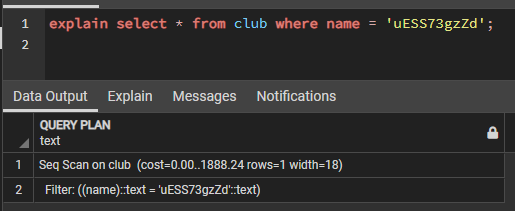
**Hash**

Для цього завдання створимо 100000 випадкових рядків в таблиці club(id, name).

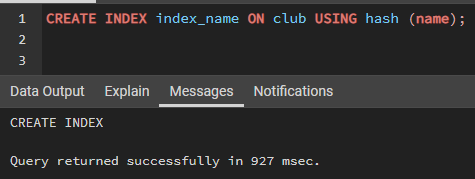




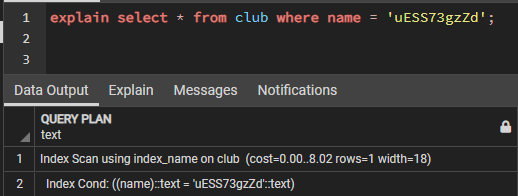
Для цього завдання вибере перше з випадково створених – uESS73gzZd. Зробимо пошук по цьому імені. Отримали 1888 при Seq Scan.



Тепер створимо індекс по колонці name.

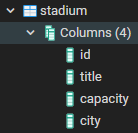


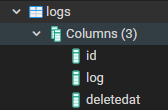
Тепер виконаємо то же запит. І отримуємо Cost = 8 при Index Scan.

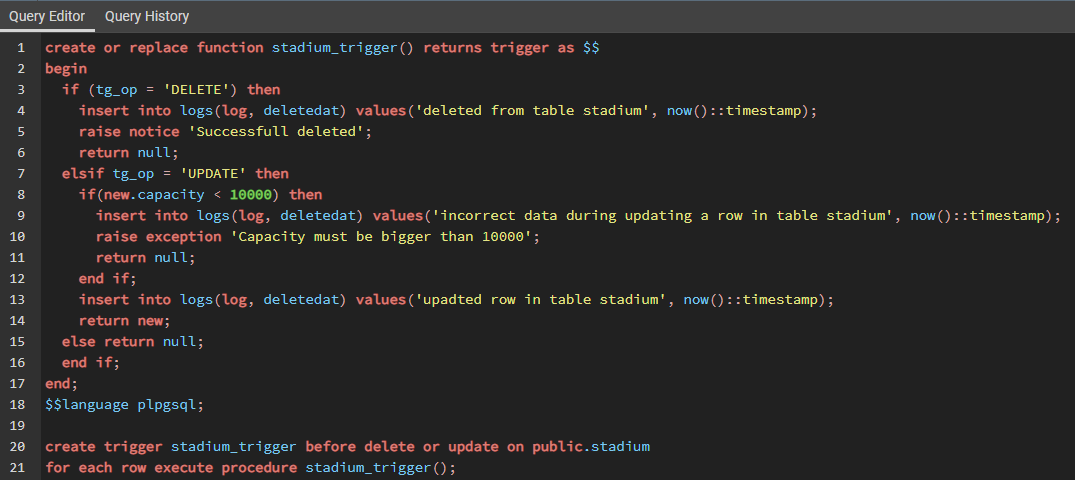


Тепер бачимо, що після створення індексу, пошук та відображення видно вже покращення результату приблизно в 236 разів(для рядка який не повторювався). Тобто бачимо, що застосування індексу HASH покращило роботу повнотекстового пошуку.

Завдання 3

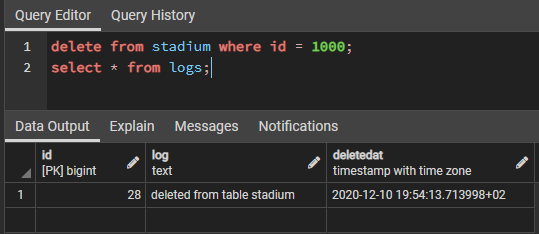
В моєму завдання необхідно створити тригери before delete та before update. Для цього я вибрав свою таблицю stadium.  


Створимо простий тригер, який при видаленні рядка записує інформацію в таблицю logs.   


  
  
  
Тут якщо, спрацював тригер delete ми записуємо в таблицю logs повідомлення та дату видалення. В випадку update тригера, виконується перевірка, щоб в оновленого рядка значення capacity було більшим за 10 тисяч, якщо воно менше то виникає помилка з відповідним повідомленням.

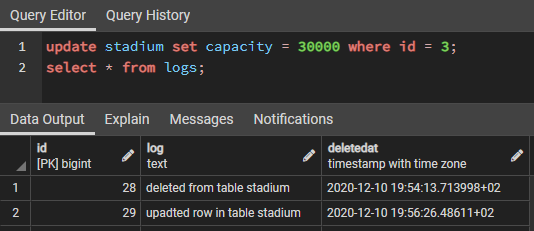
Розглянемо випадки роботи тригера.

1. Для випадку видалення.



2. При випадку оновлення рядку.

Коли дані коректні.



Та випадок коли capacity вводиться менше 10000.

