

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

Дембіцький Максим

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.

Корисні посилання: [тут](https://www.enterprisedb.com/postgres-tutorials/everything-you-need-know-about-postgresql-triggers), [тут](https://www.postgresqltutorial.com/postgresql-triggers/).

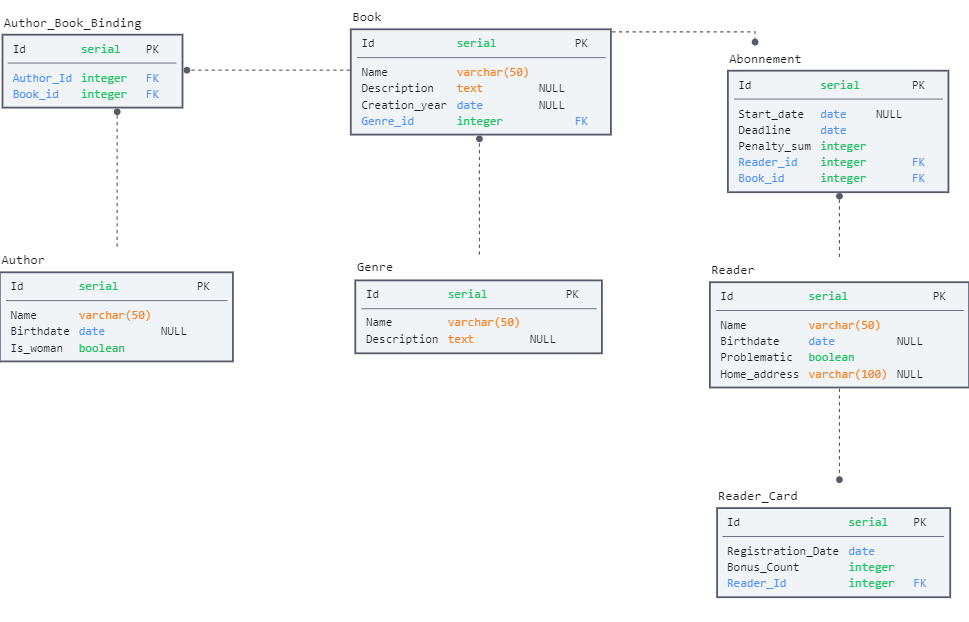
Вимоги до інструментарію:

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.

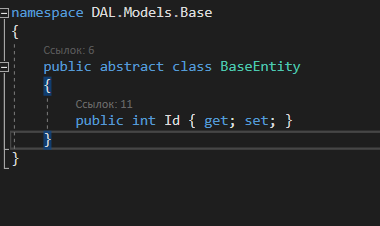
Варіант 9: Btree, BRIN. Before update, delete.

**Завдання 1**

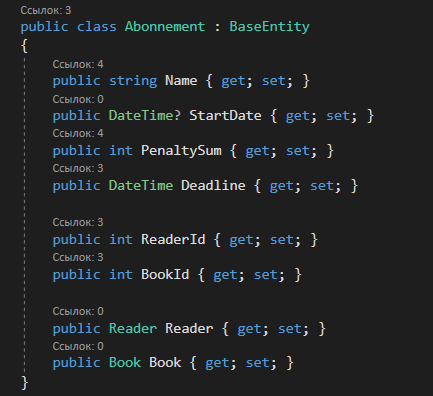
Логічна схема бази даних



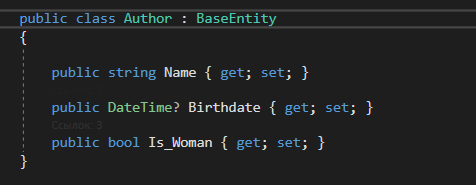
Моделі ORM

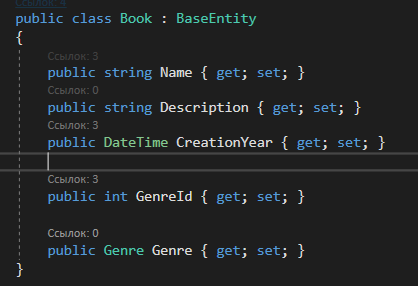


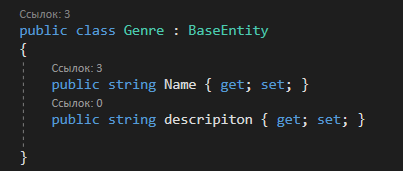
Від абстрактного класу BaseEntity наслідуються всі моделі бази даних, для того, щоб додати до них первинний ключ Id.

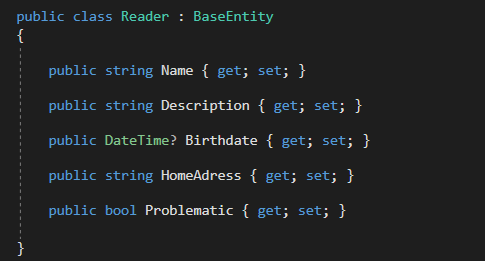


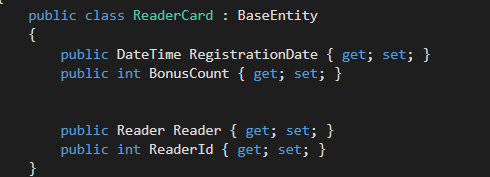
Таблиця Abonnement містить два зв’язки – з таблицею Reader і з таблицею Book.







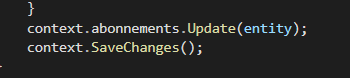






Також програма містить так званий ‘клас контексту’, який відповідає за створення бази даних. В ньому ''регіструються'' всі моделі таблиць бази даних. Відношення створюються автоматично.

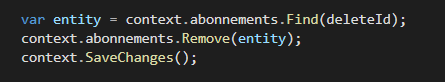
Приклади запитів у вигляді ORM



На рисунку зображений запит ‘оновлення’ для таблиці Abonnements.



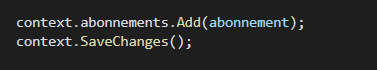
На рисунку зображений запит ‘пошук за первинним ключем’ для таблиці Abonnements.



На рисунку зображений запит 'видалення'. Для цього спершу знаходимо потрібний запис у таблиці Abonnements.



На рисунку зображений запит `Select all`



На рисунку зображений запит ‘Insert` для таблиці Abonnenement.

**Завдання 2**

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

insert into abonnements

("Name", "PenaltySum", "Deadline", "ReaderId", "BookId")

(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),

trunc(random() \* 1000)::int,

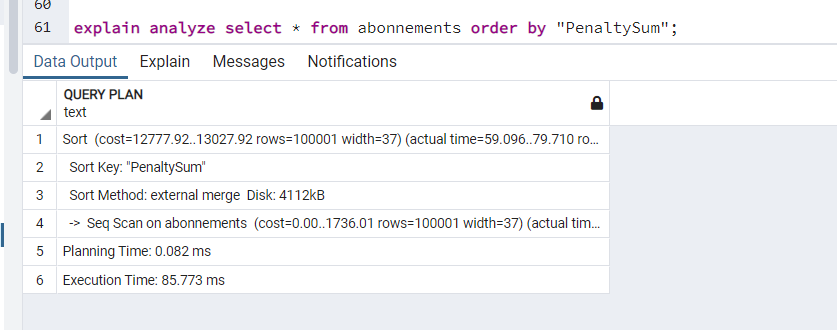
timestamp '2014-01-10 20:00:00' + random() \* (timestamp '2014-01-20 20:00:00' - timestamp '2014-01-10 10:00:00'),

1,1

from generate\_series(1,1000000), readers, books

limit(100000))

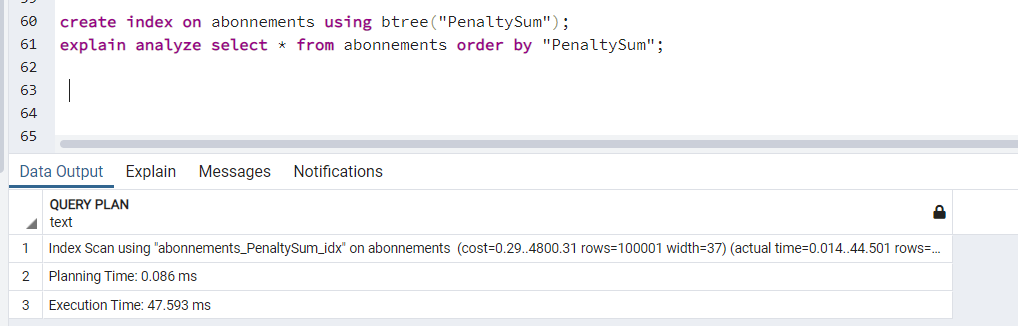
Без використання індекса



Planning Time: 0.082 ms

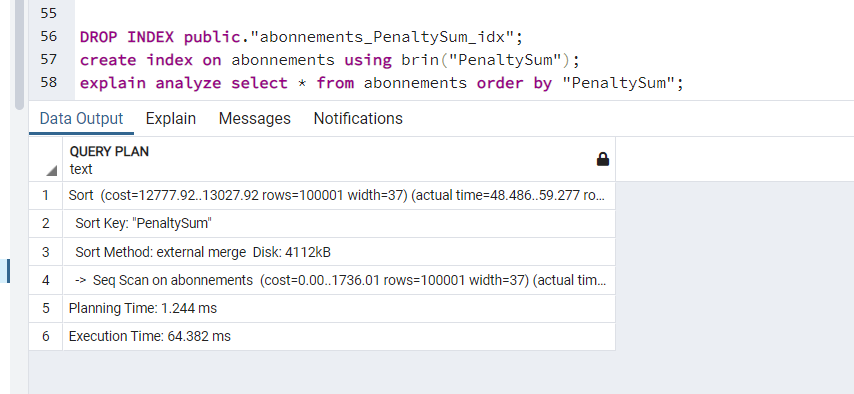
Execution Time: 85.773 ms

Btree індекс



Як видно, з Btree індексом execution time значно зменшився (на 25%). 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.

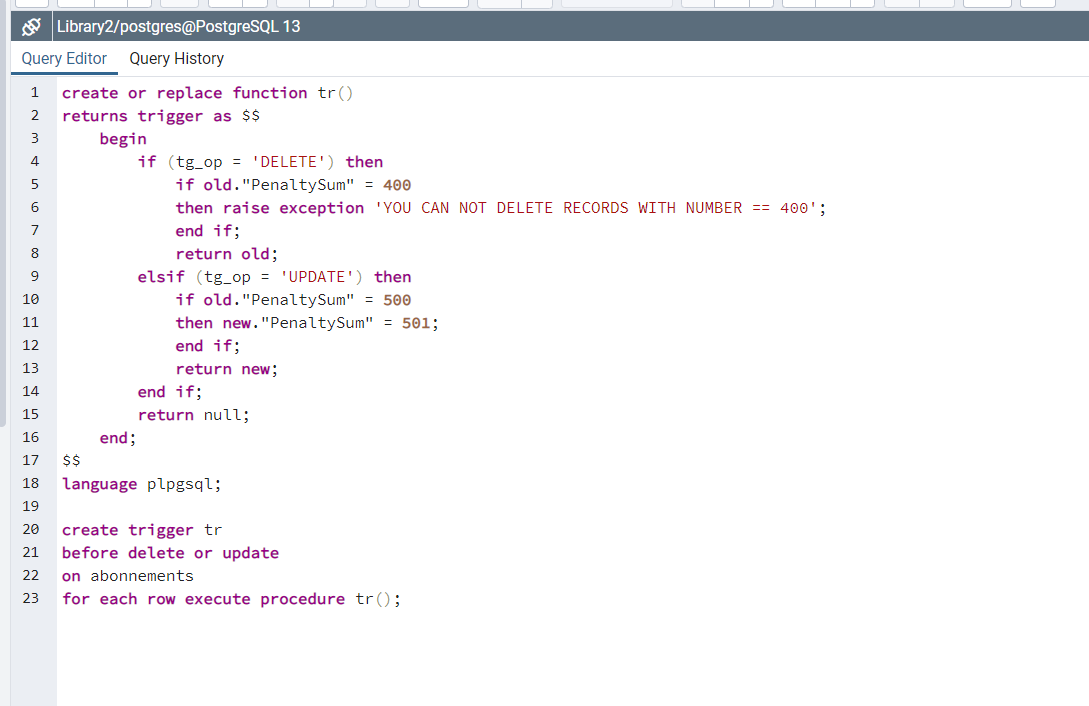
BRIN індекс



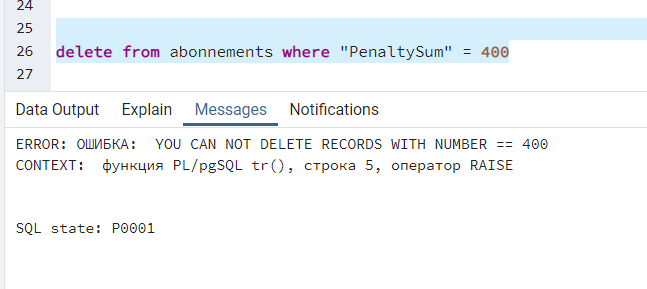
Як видно на зображенні, сортування з BRIN індексом працює швидше, ніж без нього, але повільніше, ніж з Btree індексом. Повільніше, тому що Btree – завжди найшвидший варіант, адже дані відразу відсортовані. BRIN індекс працює так: всі дані діляться на секції, і кожного разу, коли ми шукаємо мінімальне 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**

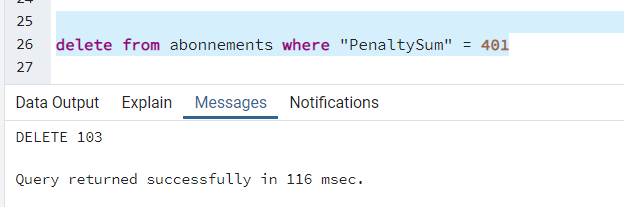
**Before delete, update**



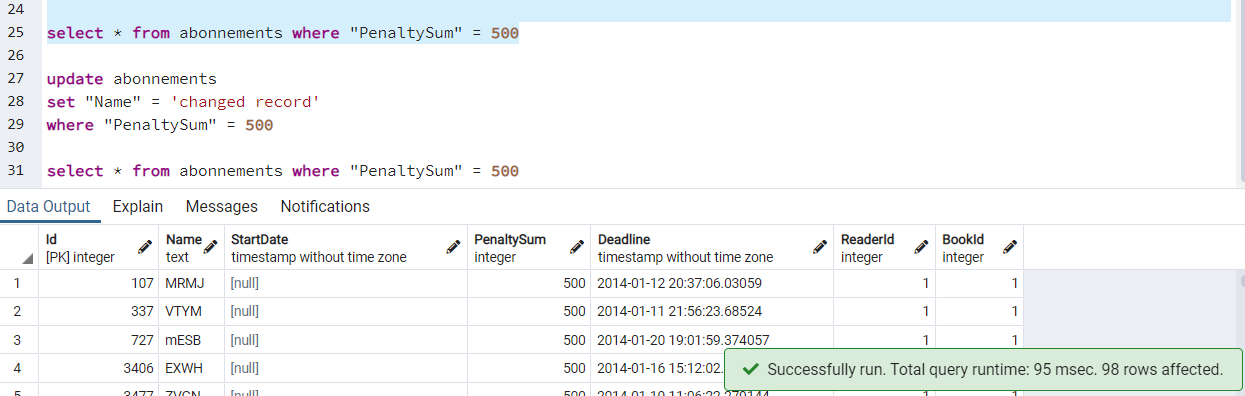
Даний тригер при видаленні перевіряє чи запис, який видаляється, має “PenaltySum” = 400. Якщо так, вибиває помилку. Якщо при оновленні “PenaltySum” = 500, то ставимо PenaltySum = 501.



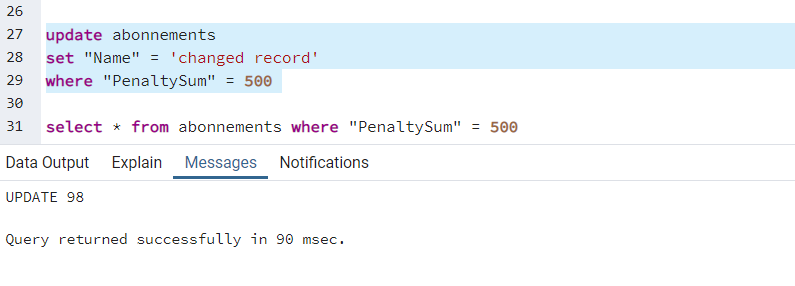
Якщо спробувати видалити, то з’явиться повідомлення про помилку, як ми й описали в тригері. В іншому випадку все спрацює:



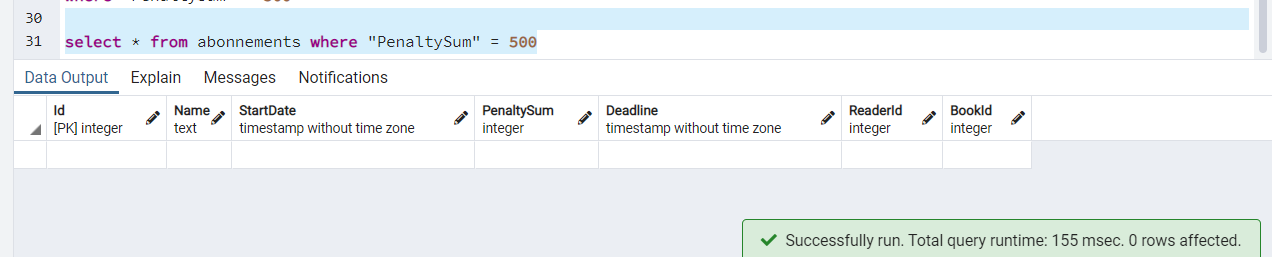
Тепер спробуємо оновити всі записи, в яких PenaltySum = 500. Після цього в таблиці не має бути жодного запису з таким PenaltySum. Пробуємо, але спершу переконаємось, що до виконання цих операцій, в таблиці є хоча б один запис з PenaltySum = 500:



Знайшли 98 записів.



І рівно 98 записів зачепило. Все правильно. Перевіряємо, чи працює тригер:



Отже, все працює, як повинно працювати.