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»**

Тема: «Засоби оптимізації роботи СУБД PostgreSQL»

Виконав: студент III курсу

ФПМ групи КВ-81

Савченко В. С.

Київ – 2020

*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.

*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.

Корисні посилання: [тут](https://www.learndatasci.com/tutorials/using-databases-python-postgres-sqlalchemy-and-alembic/) і [тут](https://auth0.com/blog/sqlalchemy-orm-tutorial-for-python-developers/).

*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).

Корисні посилання: [Hash](https://habrahabr.ru/company/postgrespro/blog/328280/), [B-tree](https://habrahabr.ru/company/postgrespro/blog/330544/), [GIN](https://habrahabr.ru/company/postgrespro/blog/340978/), [BRIN](https://habrahabr.ru/company/postgrespro/blog/346460/).

*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*

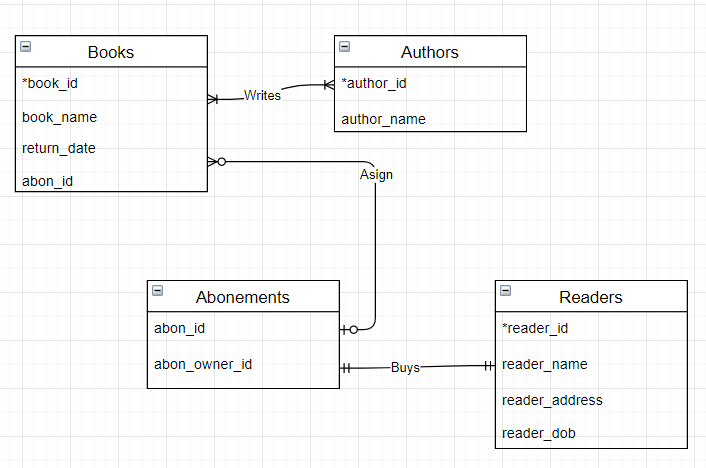
1. Бібліотека для реалізації ORM - [SQLAlchemy для Python](https://www.sqlalchemy.org/) або інша з подібною функціональністю.
2. Середовище для відлагодження SQL-запитів до бази даних –   
   pgAdmin 4.
3. СУБД - PostgreSQL 11-12.

*Варіант:*

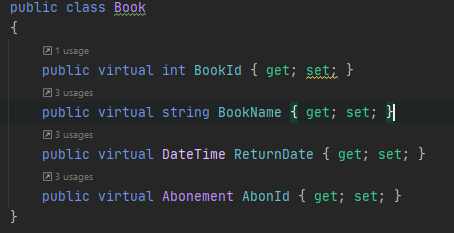
|  |  |  |
| --- | --- | --- |
| *№ варіанта* | *Види індексів* | *Умови для тригера* |
| 16 | GIN, Hash | after delete, insert |

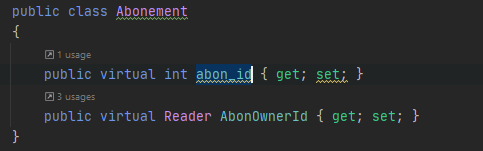
**Завдання 1**

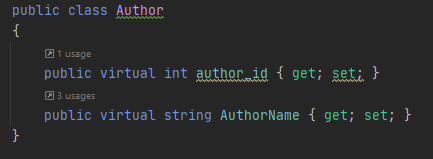
Схема бази даних «Бібліотека»

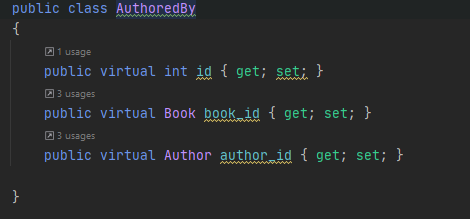


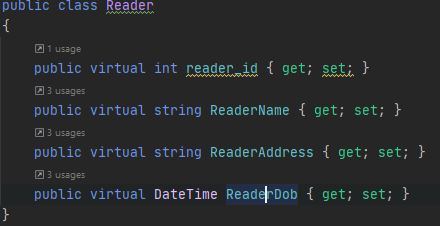
Усі таблиці у коді програми були виконані за допомогою класів, скрнішоти наводжу.





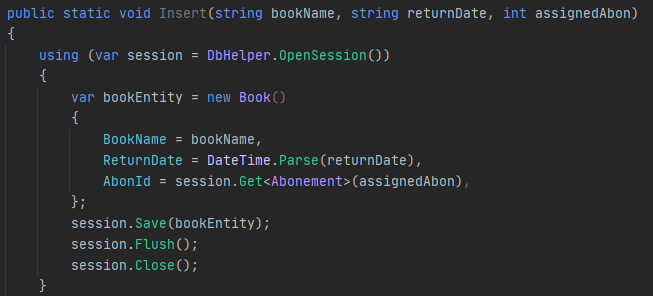






Зразок взаємодії классів у програмі

Для кожного классу був визначений набір операцій. Ось, наприклад, метод Insert для Таблиці «Книга»



Тут можна побачити, що є зв’язок із таблицею «Абонемент». Наводжу приклад виконання запиту.



Такий запит видалить із таблиці «Читач» запис, айді якого дорівнює 4.

**Завдання 2**

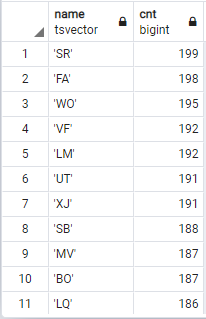
**Створення та аналіз індекса GIN**

Оскільки GIN – індекс, який використовується для повнотекстового пошуку, аналіз проводиться на текстових даних. Індекс створюється на стовпцях типу tsvector. Для дослідження використаємо таблицю *buffer* з єдиною колонкою *author\_name* типу tsvector. До таблиці було додано 100 000 рандомізованих даних та декілька рядків вручну. Для аналізу швидкодії запитів будуть використовуватись рядки, які зустрічаються найчастіше й найрідше.

Дізнаємося рядки, які зустрічаються частіше за все, виконавши такий запит:



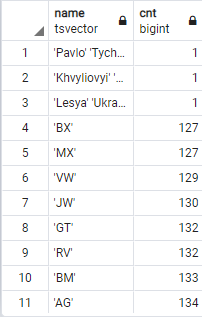
Робимо висновок, що найчастіше зустрічаються поєднання ‘SR’, ‘FA’, ‘WO’.



Далі дізнаємося рядки, що зустрічаються рідше всього. Для цього виконаємо такий запит:



За результатами виконання (див. наступну сторінку) робимо висновок, що рідше за все зустрічаються рукописні рядки (досить очікувано) та рядки ‘BX’, ‘MX’, ‘VW’.



Тепер, виконаємо пошук за цими рядками без використання індексу.





Середній час виконання запиту : 306,666(6) ms.





Середній час виконання запиту: 363,6666(6)ms.



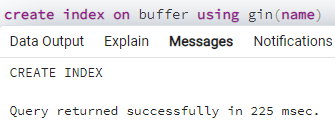
Cередній час виконання запиту: 346ms.



Середній час виконання запиту: 138,3(3)ms.

Тепер створимо індекс GIN на колонці name:



Виконаємо попередні команди знову.



Середній час виконання запиту : 295,3 ms.



Середній час виконання запиту :253 ms.





Середній час виконання запиту : 271 ms.



Середній час виконання запиту : 243,33ms.

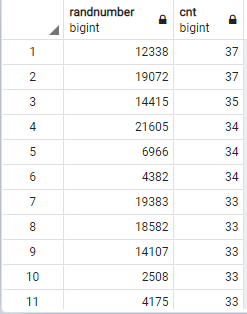
За результатами можемо зробити висновок, що використання індексу gin дає незначний приріст у найпоширеніших словах і так само у найрідше повторюваних словах.

**Створення та аналіз індекса HASH**

HASH-індекси працюють тільки з простими умовами рівності. Хеш-індекс може бути використаний лише за умови, що стовпець, який індексується, бере участь у порівнянні з оператором «=». Для дослідження використаємо таблицю *buffer* з єдиною колонкою *auh* типу integer. До таблиці було додано 500 000 рандомізованих чисел. Для аналізу швидкодії запитів будуть використовуватись рядки, які зустрічаються найчастіше й найрідше.

З’ясуємо, які числа повторюються частіше за все.

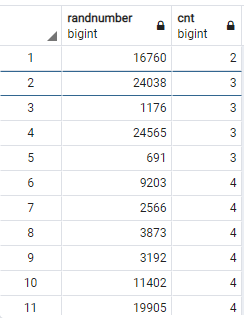




Найчастіше повторюються 12338 та 19072.

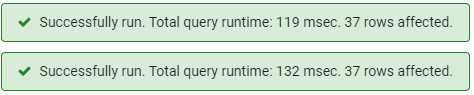
Найрідше:



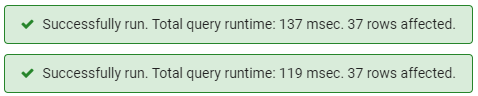


Найрідше: 16760 та 24038.

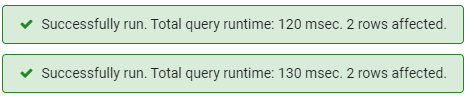
Тепер виконаємо пошук цих чисел без використання індексу



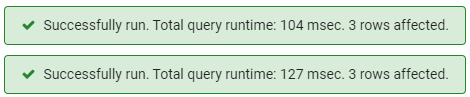




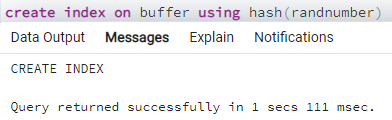




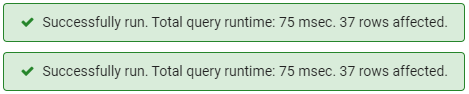


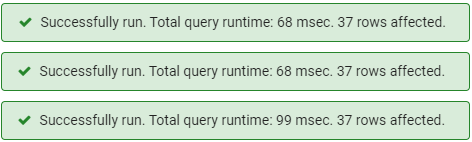


Тепер створимо індекс

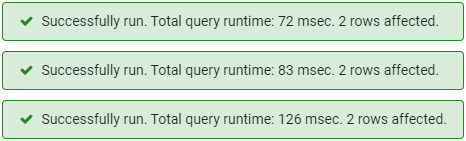


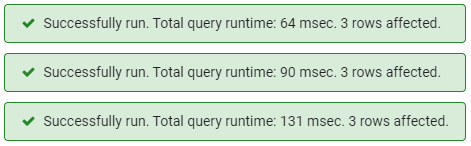
Виконаємо ті ж самі запити, але вже з індексом









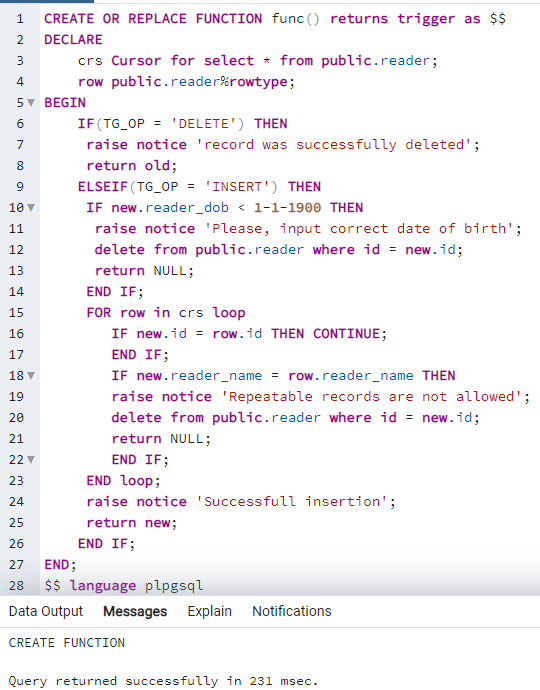


Можемо спостерігати пришвидшення виконання запитів у 2-3 рази.

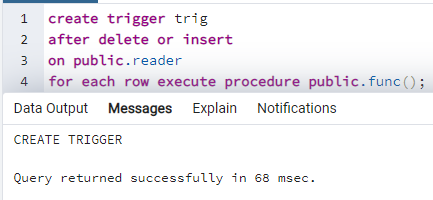
**Завдання 3**

***Логіка тригера***

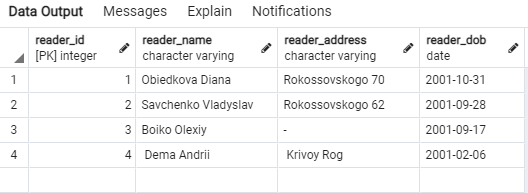
Створений тригер спрацьовує після видалення або додавання запису.



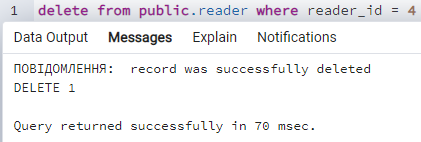
Тепер створимо тригер:



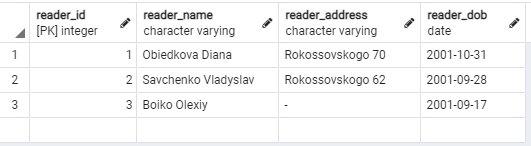
Дані таблиці до видалення запису:



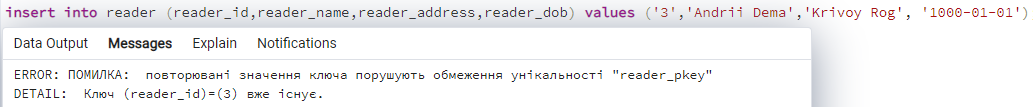
Видалимо Андрія Дему:

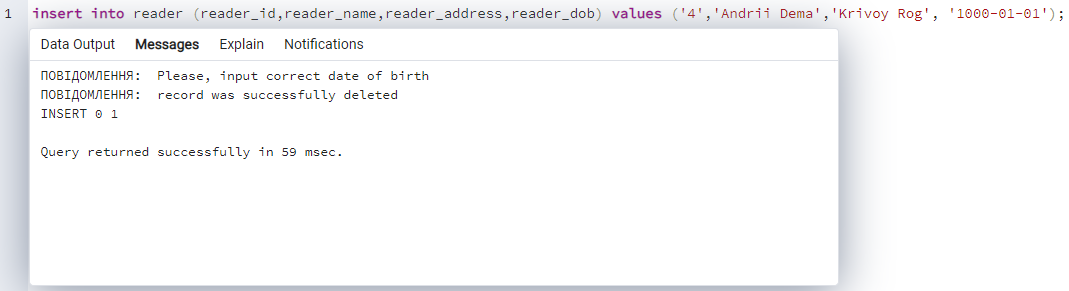


Результат:

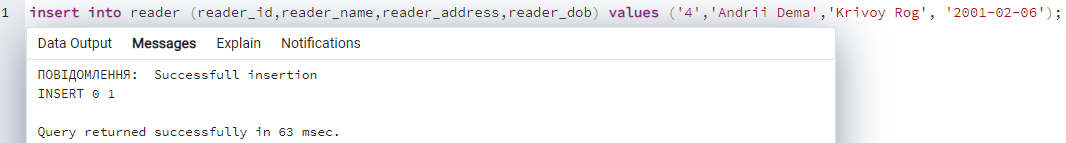


Спробуємо повернути Андрія з вже існуючим айді

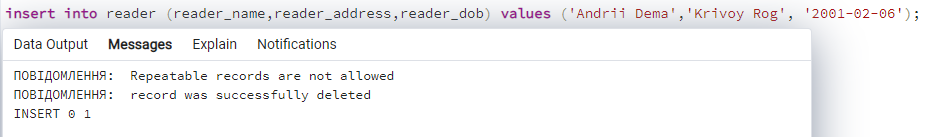


Спробуємо вписати йому не валідну дату народження

І знову помилка.

Тепер спробуємо дати вже корректний запис Андрія і потім ще один такий самий

Тепер ще один



Як бачимо, тригер працює.