

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих комп’ютерних систем**

**Лабораторна робота № 2**

з дисципліни

**«Бази даних та засоби управління»**

Виконав: студент групи КВ-11

Угнівенко Ярослав

Контакт в telegram: @yar4ik4ik

Посилання на гіт: [yaro-krasav4ik228/DBLAB2 (github.com)](https://github.com/yaro-krasav4ik228/DBLAB2)

Київ

2023

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

*Завдання* роботи полягає у наступному:

1. Перетворити модуль “Модель” з шаблону MVC РГР у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.
4. Навести приклади та проаналізувати рівні ізоляції транзакцій у 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 пункту завдання №4*

Проаналізувати на прикладах використання рівнів ізоляції транзакцій READ COMMITTED, REPEATABLE READ та SERIALIZABLE, продемонструвавши феномени, які виникають, і спосіб їх уникнення завдяки встановленню відповідного рівня ізоляції транзакцій. Для виконання завдання необхідно відкрити дві транзакції у різних вікнах pgAdmin4 і виконати послідовність запитів INSERT, UPDATE або DELETE у обох транзакціях, що доводять наявність або відсутність певних феноменів.

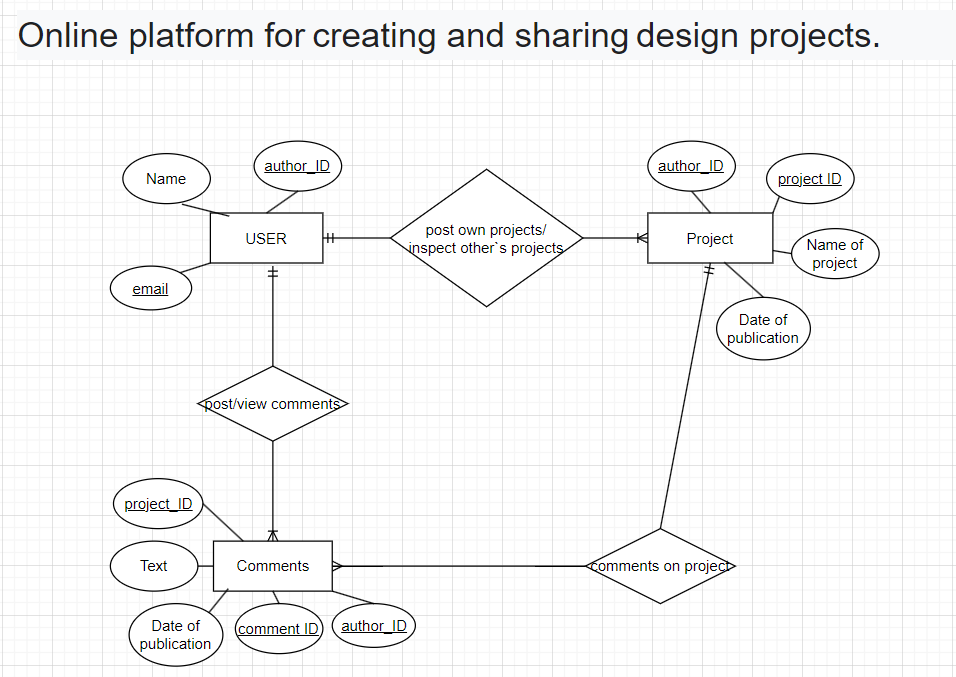
**Варіант 23**

****

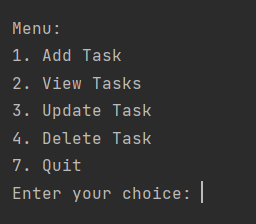
**Про базу даних**

Для даної роботи було вибрано тему “ Онлайн-платформа для зберігання та обміну дизайнерськими проектами”. Тут наявні 3 сутності: User, Project і Comments. Перша сутність – User, має 3 атрибути: нік і ключові – ID, електронна пошта. Пошта є ключовими, для того, щоб на сайт можна було заходити за її допомоги, а також, щоб не було повторень і не можна було зареєструвати більше 1 аккаунта на одну пошту. ID – унікальний ключ, який присвоюється Usery у внутрішній базі і потрібен для його ідентифікації. Ім’я не є унікальним. Друга сутність – Project, має 4 атрибути: ім’я проекту, дата публікації проекту, ID проекту, який є ключовим і по ньому можна здійснити пошук саме цього проекту, а також зовнішній ключ – author\_ID. Остання сутність – Comments, в якій є атрибути тексту(сам коментар), дата публікації і 3 ключових поля, 2 з яких – зовнішні, а саме є comment\_id, який є унікальним полем для самої сутності, author\_id, який посилається на автора коментаря і project\_id, який посилається на проект, до якого доданий коментар.

Нотація “Чена”:



**Про програму**

Так виглядає меню програми, всього 5 можливих “кнопок”. Щоб вибрати пункт з меню потрібно в графі Enter your choice вписати обрану цифру.

1. Add Task – з’являється покрокове меню з заповненням одного рядка якоїсь із трьох таблиць на вибір.

2. View Tasks – меню при виборі якого можна подивитись повністю вміст якоїсь таблиці.

3. Update Task – функція зміни даних в таблиці, працює по вибору ключа.

4. Delete Task – видаляє(якщо це можливо) рядок таблиці за вибором. Видалення не може відбутись, якщо один з параметрів є зовнішнім ключом і використовується в іншій таблиці.

7. Quit – вихід.

**Назва мови програмування та бібліотек**

IDE - PyCharm 2023.2.3 (Professional Edition)

Мова – Python 3.9

Бібліотеки:

from sqlalchemy import create\_engine  
from sqlalchemy.orm import sessionmaker  
from sqlalchemy.ext.declarative import declarative\_base – підключення sqlalchemy  
from sqlalchemy.orm import relationship  
from sqlalchemy import Column, Integer, String, Date, ForeignKey  
from datetime import date – потрібні бібліотеки з типами даних і функціями

**ЗАВДАННЯ 1**

*КЛАСИ ORM:*

USER

class USER(Base):  
 \_\_tablename\_\_ = 'USER'  
 user\_ID = Column(Integer, primary\_key=True)  
 username = Column(String)  
 email = Column(String)  
  
 projects = relationship("Projects")  
 comments = relationship("Comments")  
  
 def init(self, user\_ID, username, email):  
 self.user\_ID = user\_ID  
 self.username = username  
 self.email = email

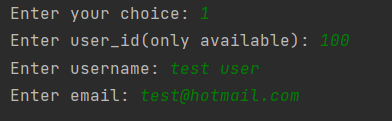
Projects

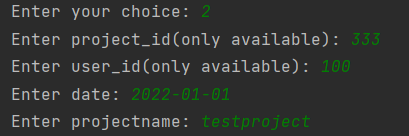
class Projects(Base):  
 \_\_tablename\_\_ = 'Projects'  
 project\_ID = Column(Integer, primary\_key=True)  
 Date\_of\_publication = Column(Date)  
 projectname = Column(String)  
  
 user\_ID = Column(Integer, ForeignKey('USER.user\_ID'))  
  
 comments = relationship("Comments")  
  
 def init(self, project\_ID, user\_ID, Date\_of\_publication, projectname):  
 self.project\_ID = project\_ID  
 self.user\_ID = user\_ID  
 self.Date\_of\_publication = Date\_of\_publication  
 self.projectname = projectname

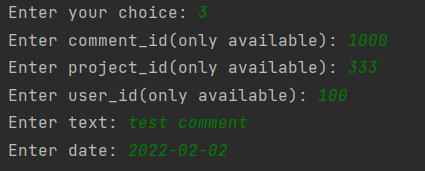
Comments

class Comments(Base):  
 \_\_tablename\_\_ = 'Comments'  
 comment\_id = Column(Integer, primary\_key=True)  
 text = Column(String)  
 date\_of\_publication = Column(Date)  
  
 project\_id = Column(Integer, ForeignKey('Projects.project\_ID'))  
 user\_id = Column(Integer, ForeignKey('USER.user\_ID'))  
  
 def init(self, comment\_id, project\_id, user\_id, text, date\_of\_publication):  
 self.comment\_id = comment\_id  
 self.project\_id = project\_id  
 self.user\_id = user\_id  
 self.text = text  
 self.date\_of\_publication = date\_of\_publication

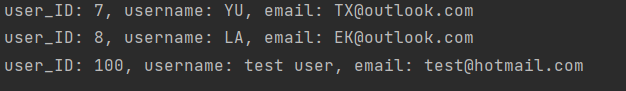
Тепер перевіримо функціонал. Для початку додамо одного користувача, створимо проект від його імені і додамо коментар до цього проекту:

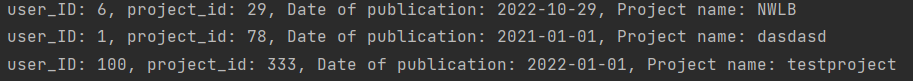


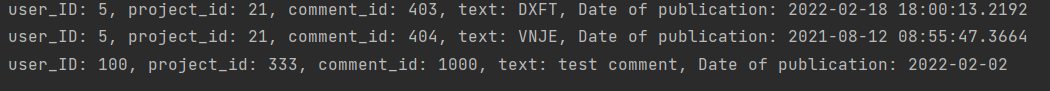




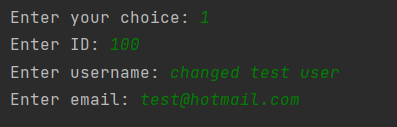
Тепер перевіримо створені рядки:

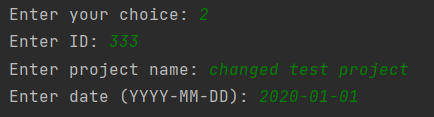


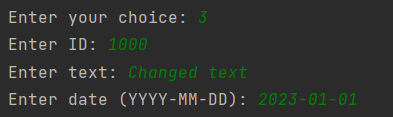




Тепер змінимо ім’я користувача, ім’я проекту і коментар:



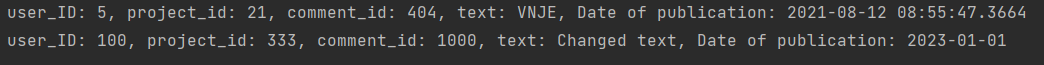




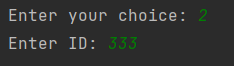
Перевіримо зміни:



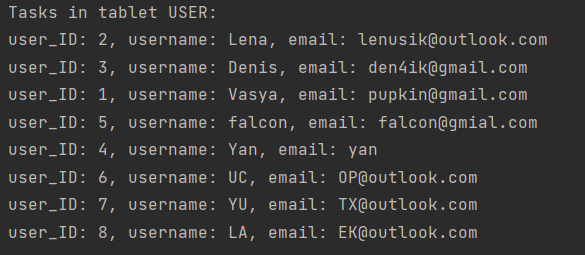




А тепер видалимо останні додані рядки. Видаляти будем з таблиці Comments і так вище за ієрархією:

І переглянемо таблицю USER, достатньо тільки її, оскільки якщо нема такого користувача, то все що було з ним пов’язане теж вже видалене:



**ЗАВДАННЯ 2**

Індекс – це спеціальна структура даних, яка зберігає групу ключових значень

та покажчиків. Індекс використовується для управління даними. Для

тестування індексів було створено окремі таблиці у базі даних test з 1000000

записів.

Індекс GIN (Generalized Inverted Index) — це тип індексу у системі

управління базами даних, який дозволяє ефективно виконувати пошук та

фільтрацію для деяких типів даних, які не обробляються традиційними Binary Tree Index.

Створемо таблицю для дослідження:

CREATE TABLE "gin"("id" bigserial PRIMARY KEY, "string" varchar(100));

INSERT INTO "gin"("id", "string")

SELECT generate\_series as "id", md5(random()::text)

FROM generate\_series(1, 1000000)

Результат:



Для початку проведемо тести без індексу:

SELECT \* FROM gin WHERE "string" = '09faea1bd809b60e7b0606c4feb8bd47';



SELECT COUNT(\*), "string" FROM gin GROUP BY "string";



SELECT \* FROM gin ORDER BY "string" DESC;



SELECT COUNT(\*), "string" FROM gin GROUP BY "string" ORDER BY COUNT(\*) DESC;



Створимо індекс:

CREATE INDEX index ON gin USING gin("string" gin\_trgm\_ops);

Перевіримо ці ж самі запити з індексом:

SELECT \* FROM gin WHERE "string" = '09faea1bd809b60e7b0606c4feb8bd47';



SELECT COUNT(\*), "string" FROM gin GROUP BY "string";



SELECT \* FROM gin ORDER BY "string" DESC;



SELECT COUNT(\*), "string" FROM gin GROUP BY "string" ORDER BY COUNT(\*) DESC;



З вищенаведених результатів можна зробити висновок, що у випадку, коли потрібно було вибрати лише один рядок оптимізація спрацювала добре, у всіх інших випадках не було відчутної різниці й іноді з GIN виходило навіть більше за часом.

Індекс Hash в базі даних використовує хеш-функцію для створення індексу. Він може бути ефективним для прискорення пошуку рівних значень в стовпцях, але має обмеження, зокрема, він не може використовуватися для операцій порівняння діапазонів або операцій LIKE.

Створимо тестову таблицю:

CREATE TABLE "hash"("id" bigserial PRIMARY KEY, "string" varchar(100));

INSERT INTO "hash"("id", "string")

SELECT generate\_series as "id", md5(random()::text)

FROM generate\_series(1, 1000000)



Приклад роботи таблиці таких самих розмірів без будь-якого індикса видно вище, коли ми проводили тестування в таблиці gin, тому зараз виконаємо оптимізацію і порівняємо ті самі запити.

SELECT \* FROM hash WHERE "string" = '09faea1bd809b60e7b0606c4feb8bd47';



SELECT COUNT(\*), "string" FROM hash GROUP BY "string";



SELECT \* FROM hash ORDER BY "string" DESC;



SELECT COUNT(\*), "string" FROM hash GROUP BY "string" ORDER BY COUNT(\*) DESC;



Як бачимо, особливого виграшу індекси не дають, в обох випадках тільки перша команда працювала з індексом швидше ніж без нього, це пов’язано з тим, що в обох випадках SELECT відбувався тільки над першим рядком. У випадках з повним проходом усіх рядків з індексами навіть було повільніше. На мою думку, це пов’язано з тим, що на опрацювання самого індексу теж потрібні ресурси, і з великою кількістю запитів це просто уповільнює операцію.