

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»**

**Тема: «Засоби оптимізації роботи СУБД PostgreSQL»**

Виконав: студент 3 курсу

ФПМ групи КВ-13

Абраменко Д. О.

Київ – 2024

**Telegram - <https://t.me/SaigeNick>**

**GitHub - <https://github.com/SaigeNick/BD_Library>**

**Проектування бази даних та ознайомлення з базовими операціями 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 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.

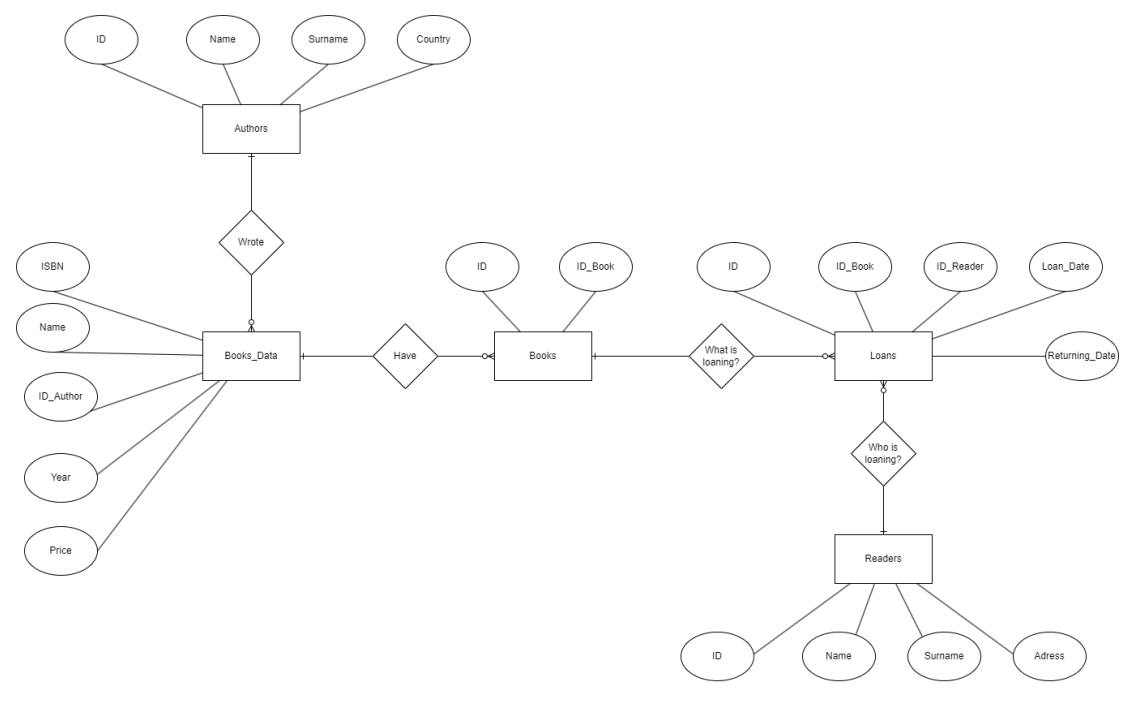
*Вимоги до пункту завдання №4*

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 №1

|  |  |  |
| --- | --- | --- |
| *№ варіанта* | *Види індексів* | *Умови для тригера* |
| *1* | *Btree, Hash* | *before insert, delete* |

**Концептуальна модель предметної області "Бібліотека"**



В концептуальній моделі предметної області "Бібліотека" виділяються наступні сутності та зв'язки між ними:

Сутність “Authors” з атрибутами: ID, Surname, Name, Country.

Сутність "Books\_Data" з атрибутами : ID, Name, ID\_Author, Year, Price;

Сутність “Books” з атрибутами: ID, ID\_Book.

Сутність “Loans” з атрибутами: ID, ID\_Book, ID\_Reader, Loan\_Date,

Returning\_Date.

Сутність “Readers” з атрибутами: ID, Name, Surname, Adress.

Між сутностями “Loans” та "Books" зв’язок R(1:N) тому, що одну книгу

може бути видано багато разів, і кожній видачі відповідає одна книга.

Між сутностями “Books” та "Books\_Data" зв’язок R(1:N) тому, що кожна

книга може мати разні дані, але тільки один номер.

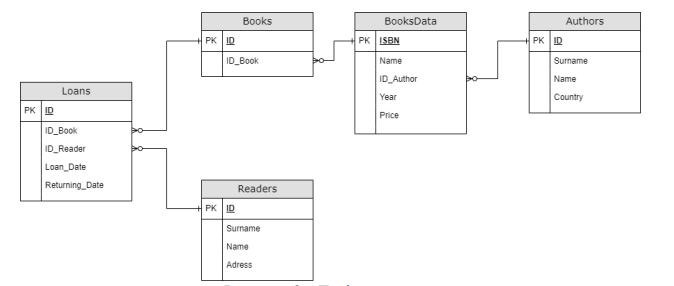
Між сутностями “Books\_Data” та "Authors" зв’язок R(1:N) тому, що кожен

автор може мати декілька книг, але у книги може бути один автор.

Між сутностями “Loans” та "Readers" зв’язок R(1:N) тому, що кожній

видачі відповідає один читач, але читач може мати багато видач.

**Схема бази даних у графічному вигляді**



**Завдання №1**

У даній лабораторній роботі було реалізовано 5 класів відповідно до 5 існуючих таблиць у розробленій базі даних, а саме:

1. Author
2. BookData
3. Book
4. Loan
5. Reader

**Author**

Таблиця Authors має такі стовпці: ID (ідентифікатор автора), Surname (прізвище автора), Name (ім’я автора), Country (країна автора). Також наявний зв’язок із таблицею Books\_Data, тому в цьому класі встановлений зв’язок relationship("BookData").

Програмна реалізація класу Author:

**class** Author(Base):

\_\_tablename\_\_ = 'Authors'

ID = Column(BigInteger, primary\_key=True)

Surname = Column(String)

Name = Column(String)

Country = Column(String)

book\_data = relationship("BookData")

**def** \_\_init\_\_(self, id, surname, name, country):

self.ID = id

self.Surname = surname

self.Name = name

self.Country = country

**def** \_\_repr\_\_(self):

**return** f"<Authors(ID={self.ID}, Surname={self.Surname}, Name={self.Name}, Country={self.Country})>"

**BookData**

Таблиця BookData має такі стовпці: ID (ідентифікатор поля), Name (назва книги), ID\_Author (зовнішній ключ, який посилається на автора), Year (рік написання), Price (ціна). Також наявний зв’язок із таблицею Books, тому в цьому класі встановлений зв’язок relationship("Book").

Програмна реалізація класу BookData:

**class** BookData(Base):

\_\_tablename\_\_ = 'Books\_Data'

ISBN = Column(BigInteger, primary\_key=True)

Name = Column(String)

Year = Column(DateTime)

Price = Column(Numeric)

ID\_Author = Column(Integer, ForeignKey('Authors.ID'))

book = relationship("Book")

**def** \_\_init\_\_(self, isbn, id\_author, name, year, price):

self.ISBN = isbn

self.ID\_Author = id\_author

self.Name = name

self.Year = year

self.Price = price

**def** \_\_repr\_\_(self):

**return** f"<BooksData(ISBN={self.ISBN}, ID\_Author={self.ID\_Author}, Name={self.Name}, Year={self.Year}, Price={self.Price})>"

**Book**

Таблиця Book має такі стовпці: ID (ідентифікатор поля), ID\_Book (зовнішній ключ, який посилається на IBSN книги). Також наявний зв’язок із таблицею Loan, тому в цьому класі встановлений зв’язок relationship("Loan").

Програмна реалізація класу Book:

**class** Book(Base):

\_\_tablename\_\_ = 'Books'

ID = Column(BigInteger, primary\_key=True)

ID\_Book = Column(Integer, ForeignKey('Books\_Data.ISBN'))

loan = relationship("Loan")

**def** \_\_init\_\_(self, id, id\_book):

self.ID = id

self.ID\_Book = id\_book

**def** \_\_repr\_\_(self):

**return** f"<Books(ID={self.ID}, ID\_Book={self.ID\_Book})>"

**Loan**

Таблиця Loan має такі стовпці: ID (ідентифікатор рядка), ID\_Book (зовнішній ключ, який посилається на IBSN книги), ID\_Reader (зовнішній ключ, який посилається на користувача), Loan\_Date (дата та час взяття книги), Returning\_Date (дата та час повернення книги).

Програмна реалізація класу Loan:

**class** Loan(Base):

\_\_tablename\_\_ = 'Loan'

ID = Column(BigInteger, primary\_key=True)

Loan\_Date = Column(DateTime)

Return\_Date = Column(DateTime)

ID\_Book = Column(Integer, ForeignKey('Books.ID'))

ID\_User = Column(Integer, ForeignKey('Readers.ID'))

**def** \_\_init\_\_(self, id, id\_book, id\_user, loan\_date, return\_date):

self.ID = id

self.ID\_Book = id\_book

self.ID\_User = id\_user

self.Loan\_Date = loan\_date

self.Return\_Date = return\_date

**def** \_\_repr\_\_(self):

**return** f"<Loans(ID={self.ID}, ID\_Book={self.ID\_Book}, ID\_User={self.ID\_User}, Loan\_Date={self.Loan\_Date}, Return\_Date={self.Return\_Date})>"

**Reader**

Таблиця Reader має такі стовпці: ID (ідентифікатор читача), Name (ім’я читача), Surname (прізвище читача), Adress (адреса читача). Також наявний зв’язок із таблицею Loan, тому в цьому класі встановлений зв’язок relationship("Loan").

Програмна реалізація класу Reader:

**class** Reader(Base):

\_\_tablename\_\_ = 'Readers'

ID = Column(BigInteger, primary\_key=True)

Surname = Column(String)

Name = Column(String)

Address = Column(String)

loan = relationship("Loan")

**def** \_\_init\_\_(self, id, surname, name, address):

self.ID = id

self.Surname = surname

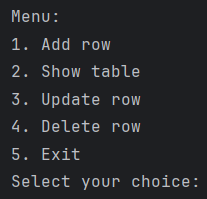
self.Name = name

self.Address = address

**def** \_\_repr\_\_(self):

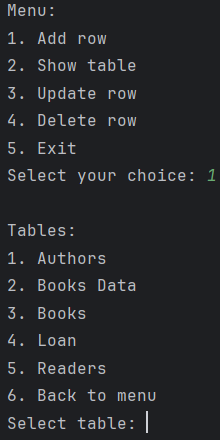
**return** f"<Readers(ID={self.ID}, Surname={self.Surname}, Name={self.Name}, Address={self.Address})>"

Меню складається із 5 пунктів, кожен з яких буде розглянуто далі.



1. **Add row (Додати рядок)**

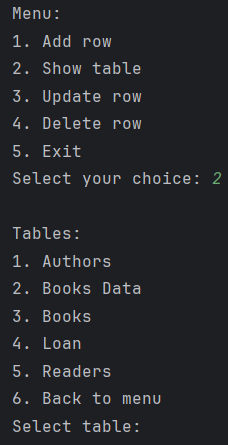
Цей пункт створений для додавання рядка у таблицю. Після його вибору, відкривається список всіх таблиць БД, де потрібно обрати таблицю, до якої хочемо додати рядок:



Після вибору таблиці, користувачу потрібно ввести всі необхідні дані для нового рядка.

1. **Show table (Показ таблиці)**

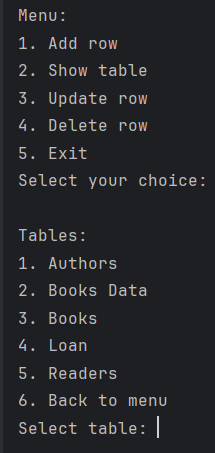
Цей пункт створений для показу таблиць. Після його вибору, відкривається список доступних таблиць БД, де потрібно вибрати таблицю, яку бажаємо побачити.



Після вибору таблиці, мають вивестися всі рядки і стовпці з обраної таблиці БД.

1. **Update row (Редагувати рядок)**

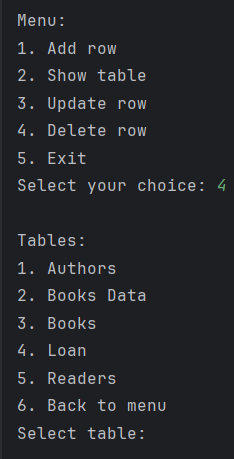
Цей пункт створений для редагування рядків у таблицях. Після вибору цього пункту, відкривається список доступних таблиць, де потрібно вибрати таблицю, в якій бажаємо зробити зміну.



Після вибору таблиці, користувачу потрібно ввести ідентифікатор існуючого рядка в таблиці. Потім записати нові дані для обраного рядка.

1. **Delete row (Видалити рядок)**

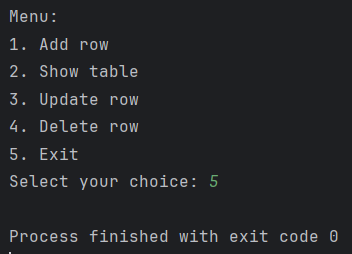
Цей пункт створений для видалення рядків у таблицях. Після вибору цього пункту, відкривається список доступних таблиць БД, де потрібно вибрати таблицю, в якій бажаємо видалити рядок.



Після вибору таблиці, користувачу потрібно ввести ідентифікатор існуючого рядка в таблиці для видалення.

1. **Exit (Вихід)**

Пункт виходу з програми: закривається з’єднання і програма завершується.

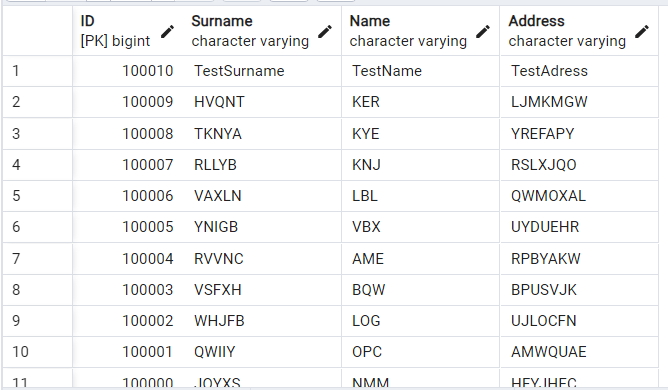


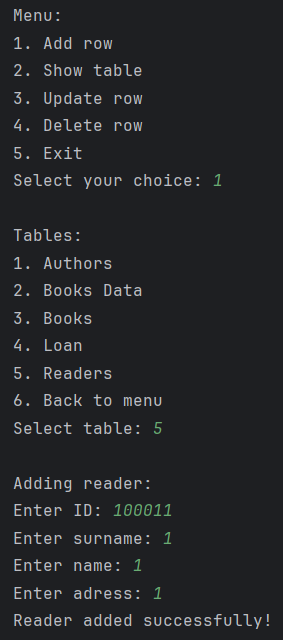
**Приклади запитів у вигляді ORM**

Для демонстрації запитів виберемо по 1-2 таблиці до кожного.

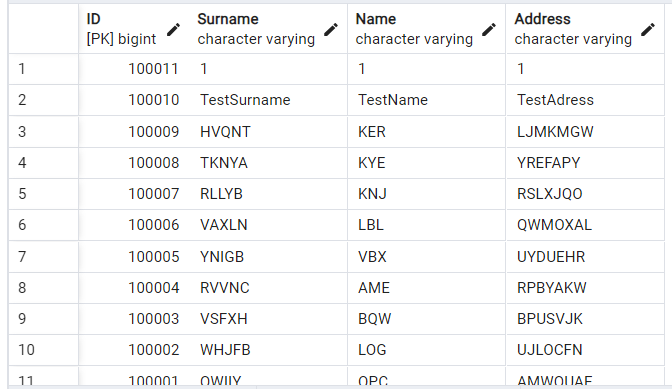
***Запити вставки*** реалізовані за допомогою функцій insert. Спочатку в меню користувач обирає опцію додавання, далі обирає таблицю, до якої хоче додати рядок і вводить необхідні дані.

Таблиця “Readers” до вставки:





Таблиця “Readers” після вставки:



Лістинг функцій insert для кожної таблиці:

**def** insert\_author(self, id: int, surname: str, name: str, country: str) -> None:

author = Author(id=id, surname=surname, name=name, country=country)

s.add(author)

s.commit()

**def** insert\_book\_data(self, isbn: int, id\_author: int, name: str, year: DateTime, price: Numeric) -> None:

book\_data = BookData(isbn=isbn, id\_author=id\_author, name=name, year=year, price=price)

s.add(book\_data)

s.commit()

**def** insert\_book(self, id: int, id\_book: int) -> None:

book = Book(id=id, id\_book=id\_book)

s.add(book)

s.commit()

**def** insert\_loan(self, id: int, id\_book: int, id\_user: int, loan\_date: DateTime, return\_date: DateTime) -> None:

loan = Loan(id=id, id\_book=id\_book, id\_user=id\_user, loan\_date=loan\_date, return\_date=return\_date)

s.add(loan)

s.commit()

**def** insert\_reader(self, id: int, surname: str, name: str, address: str) -> None:

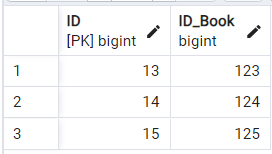
reader = Reader(id=id, surname=surname, name=name, address=address)

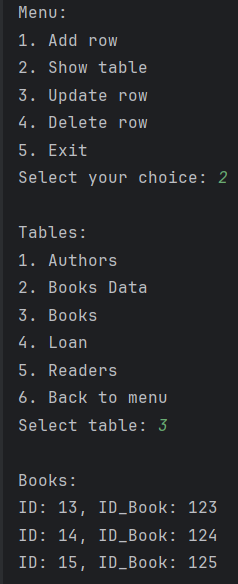
s.add(reader)

s.commit()

***Запити показу*** реалізовані за допомогою функцій show. Спочатку в меню користувач обирає опцію показу, далі обирає таблицю, яку хоче побачити.

Таблиця “Books”:





Лістинг функцій show для кожної таблиці:

**def** show\_authors(self):

**return** s.query(Author.ID, Author.Surname, Author.Name, Author.Country).all()

**def** show\_books\_data(self):

**return** s.query(BookData.ISBN, BookData.ID\_Author, BookData.Name, BookData.Year, BookData.Price).all()

**def** show\_books(self):

**return** s.query(Book.ID, Book.ID\_Book).all()

**def** show\_loans(self):

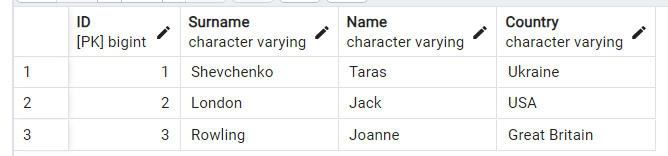
**return** s.query(Loan.ID, Loan.ID\_Book, Loan.ID\_User, Loan.Loan\_Date, Loan.Return\_Date).all()

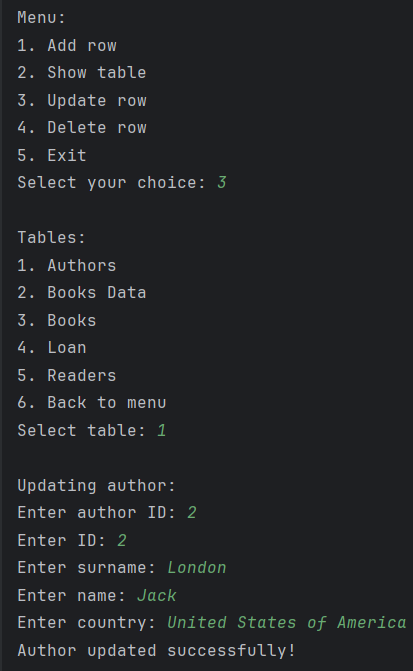
**def** show\_readers(self):

**return** s.query(Reader.ID, Reader.Surname, Reader.Name, Reader.Address).all()

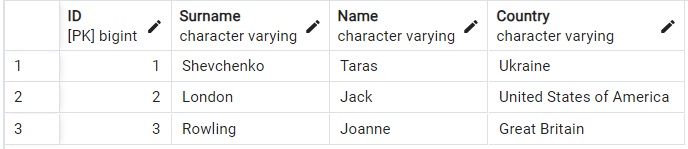
***Запит редагування*** реалізовано за допомогою функції update. Спочатку користувач обирає, у якій таблиці потрібно змінити запис і за яким ідентифікатором. Потім треба ввести всі необхідні дані для реадгування рядка.

Таблиця “Authors” до редагування:





Таблиця “Authors” після редагування:



Лістинг функцій update для кожної таблиці:

**def** update\_author(self, author\_id: int, surname: str, name: str, country: str, id: int) -> None:

s.query(Author).filter\_by(ID=id).update(

{Author.ID: author\_id, Author.Surname: surname, Author.Name: name, Author.Country: country})

s.commit()

**def** update\_book\_data(self, isbn: int, id\_author: int, name: str, year: DateTime, price: Numeric, id: int) -> None:

s.query(BookData).filter\_by(ISBN=id).update(

{BookData.ISBN: isbn, BookData.ID\_Author: id\_author, BookData.Name: name, BookData.Year: year, BookData.Price: price})

s.commit()

**def** update\_book(self, book\_id: int, id\_book: int, id: int) -> None:

s.query(Book).filter\_by(ID=id).update(

{Book.ID: book\_id, Book.ID\_Book: id\_book})

s.commit()

**def** update\_loan(self, loan\_id: int, id\_book: int, id\_user: int, loan\_date: DateTime, return\_date: DateTime, id: int) -> None:

s.query(Loan).filter\_by(ID=id).update(

{Loan.ID: loan\_id, Loan.ID\_Book: id\_book, Loan.ID\_User: id\_user, Loan.Loan\_Date: loan\_date, Loan.Return\_Date: return\_date})

s.commit()

**def** update\_reader(self, reader\_id: int, surname: str, name: str, address: str, id: int) -> None:

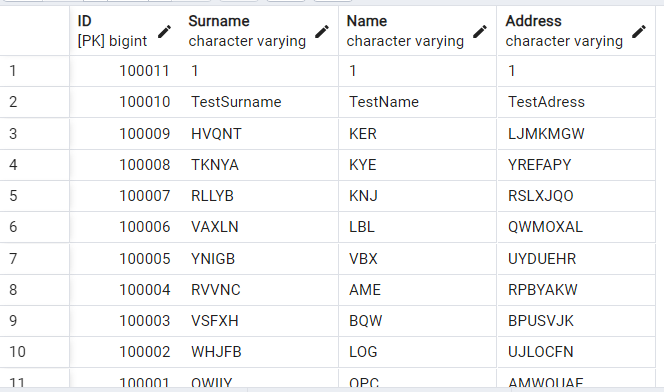
s.query(Reader).filter\_by(ID=id).update(

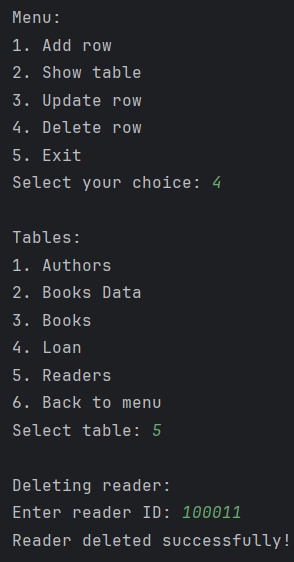
{Reader.ID: reader\_id, Reader.Surname: surname, Reader.Name: name, Reader.Address: address})

s.commit()

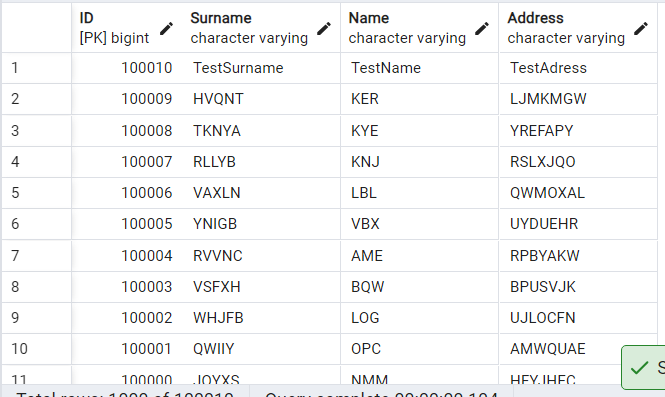
***Запити видалення*** реалізовані за допомогою функцій delete. Спочатку користувач обирає таблицю, з якої потрібно видалити дані. Потім потрібно ввести номер ідентифікатора рядка для видалення.

Таблиця “Readers” до видалення:





Таблиця “Readers” після видалення:



Лістинг функцій delete для кожної таблиці:

**def** delete\_author(self, id) -> None:

author = s.query(Author).filter\_by(ID=id).one()

s.delete(author)

s.commit()

**def** delete\_book\_data(self, isbn) -> None:

book\_data = s.query(BookData).filter\_by(ISBN=isbn).one()

s.delete(book\_data)

s.commit()

**def** delete\_book(self, id) -> None:

book = s.query(Book).filter\_by(ID=id).one()

s.delete(book)

s.commit()

**def** delete\_loan(self, id) -> None:

loan = s.query(Loan).filter\_by(ID=id).one()

s.delete(loan)

s.commit()

**def** delete\_reader(self, id) -> None:

reader = s.query(Reader).filter\_by(ID=id).one()

s.delete(reader)

s.commit()

**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 записами.

***Btree***

Індекс B-дерева (B-tree) - це структура даних, яка використовується в базах даних для ефективного зберігання та пошуку великої кількості впорядкованих даних. Це дерево, в якому кожен вузол може мати кілька дітей (відоме як "B"), і його глибина оптимізована для швидкого доступу до даних.

Індекс B-дерева зазвичай використовується для покращення швидкості пошуку в базах даних. Він може бути використаний для індексації ключів у таблицях баз даних, що дозволяє системі ефективно виконувати операції пошуку, сортування та об'єднання.

Основна ідея полягає в тому, що кожен вузол має ключі та посилання на його дочірні вузли. Велика кількість ключів дозволяє зменшити глибину дерева, що призводить до ефективного використання пам'яті та швидшого доступу до даних. Індекс B-дерева широко використовується в системах управління базами даних для оптимізації роботи з індексами та швидкого виконання запитів.

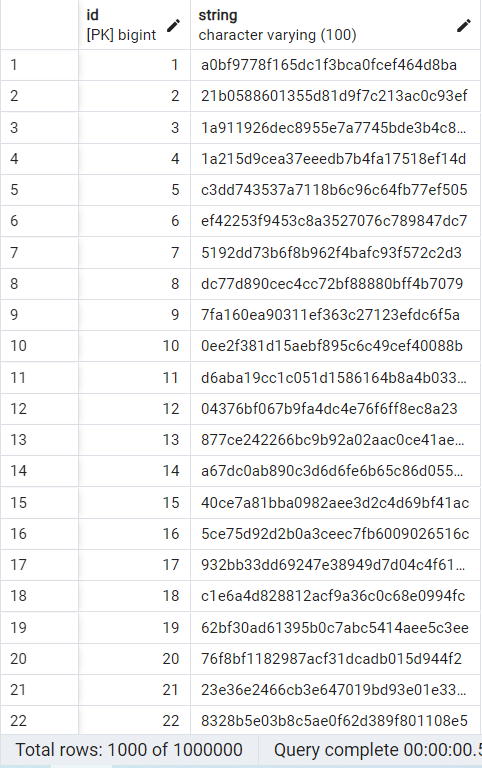
Для дослідження індексу була створена таблиця btree\_test, яка має дві колонки: “id” та “string”:

**CREATE** **TABLE** "btree\_test"("id" bigserial **PRIMARY** **KEY**, "string" **varchar**(100));

**INSERT** **INTO** "btree\_test"("id", "string")

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

**FROM** generate\_series(1, 1000000)



Для тестування візьмемо 4 запити:

**SELECT** \* **FROM** btree\_test **WHERE** "string" = '0ee2f381d15aebf895c6c49cef40088b';

**SELECT** "string", COUNT(id) **FROM** btree\_test **GROUP** **BY** "string";

**SELECT** \* **FROM** btree\_test **ORDER** **BY** "string" **ASC**;

**SELECT** "string", COUNT(id) **as** total **FROM** btree\_test **WHERE** id **BETWEEN** 10000 AND 50000 **GROUP** **BY** "string" **ORDER** **BY** total **DESC**;

Створення індексу:

**CREATE** **INDEX** btree\_index **ON** btree\_test (string);

Результати виконання запитів

Без індекса BTree

Запит №1



Запит №2



Запит №3



Запит №4



З індексом BTree

Запит №1



Запит №2



Запит №3



Запит №4



Отже, з отриманих результатів можна побачити, що лише запит №2 виконався швидше без використання індексу BTree.

Є кілька можливих причин, чому запит з агрегатною функцією та групуванням може виконатися повільніше при використанні індексу btree.

1. Якщо таблиця "btree\_test" дуже мала (наприклад, кілька десятків рядків), то використання індексу може виявитися менш ефективним, оскільки витрати на індексацію можуть бути вищими, ніж простий прогін.
2. Якщо кількість унікальних значень у стовпці "string" дуже велика, то вартість використання індексу для групування та агрегації може зрости. Індексація може бути менш ефективною в таких випадках.
3. Оптимізатор запитів може вирішити, що використання індексу не є оптимальним для даного запиту через різні оптимізації та вагомі фактори.

***Hash***

Хеш-індекс (hash) використовує хеш-функцію для створення внутрішнього представлення ключа і визначення позиції (адреси) у віртуальному масиві чи таблиці. Основна ідея полягає в тому, що за допомогою хеш-функції можна ефективно призначити унікальний код (хеш) кожному ключевому значенню. Потім цей хеш використовується для знаходження місця зберігання в пам'яті чи на диску.

Одна з особливостей хеш-індексу - це швидкий пошук за значенням ключа, оскільки хеш-функція дозволяє швидко визначити адресу в пам'яті чи на диску, де повинно знаходитися відповідне значення.

Проте, є деякі обмеження та проблеми при використанні хеш-індексів. Наприклад, можливі колізії, коли різні ключі мають однаковий хеш-код і відображаються на одну і ту ж саму адресу. Для вирішення цього можливі проблеми використовуються різні техніки, такі як використання відкритої адресації, використання ланцюгів для управління колізіями тощо.

Хеш-індекси ефективні при швидкому пошуку великої кількості даних за ключем, але їх використання обмежується конкретними властивостями хеш-функції та можливими колізіями.

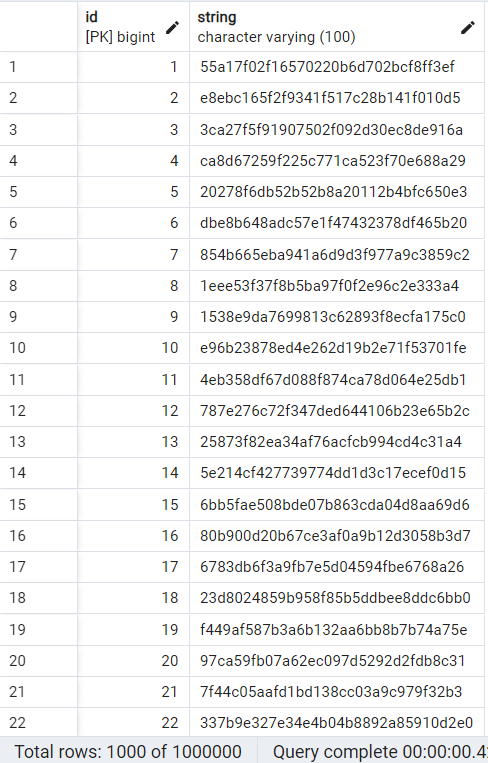
Для дослідження індексу була створена таблиця hash\_test, яка має дві колонки: “id” та “string”:

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

**INSERT** **INTO** "hash\_test"("id", "string")

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

**FROM** generate\_series(1, 1000000)



Для тестування візьмемо 4 запити:

**SELECT** \* **FROM** hash\_test **WHERE** "string" = 'e96b23878ed4e262d19b2e71f53701fe';

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

**SELECT** \* **FROM** hash\_test **ORDER** **BY** "string" **ASC**;

**SELECT** "string", COUNT(id) **as** total **FROM** hash\_test **WHERE** id **BETWEEN** 10000 AND 50000 **GROUP** **BY** "string" **ORDER** **BY** total **DESC**;

Створення індексу:

**CREATE** **INDEX** hash\_index **ON** hash\_test **USING** hash("string");

Результати виконання запитів

Без індекса Hash

Запит №1



Запит №2



Запит №3



Запит №4



З індексом Hash

Запит №1



Запит №2



Запит №3



Запит №4



Отже, з отриманих результатів можна побачити, що виконання обраних запитів з використанням індексу Hash у всіх випадках є швидшим.