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 і засоби управління**

на тему: "Засоби оптимізації роботи СУБД PostgreSQL"

Виконав:
студент III курсу
групи КВ-21
Кузнецов Д. С.
Перевірив:
Павловский В. І.

Київ – 2024

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.

Варіант 8

8	*BTree, GIN*	*after insert, update*

Хід роботи

Графічне подання логічної моделі «Сутність-зв’язок» зображено на рисунку 1.

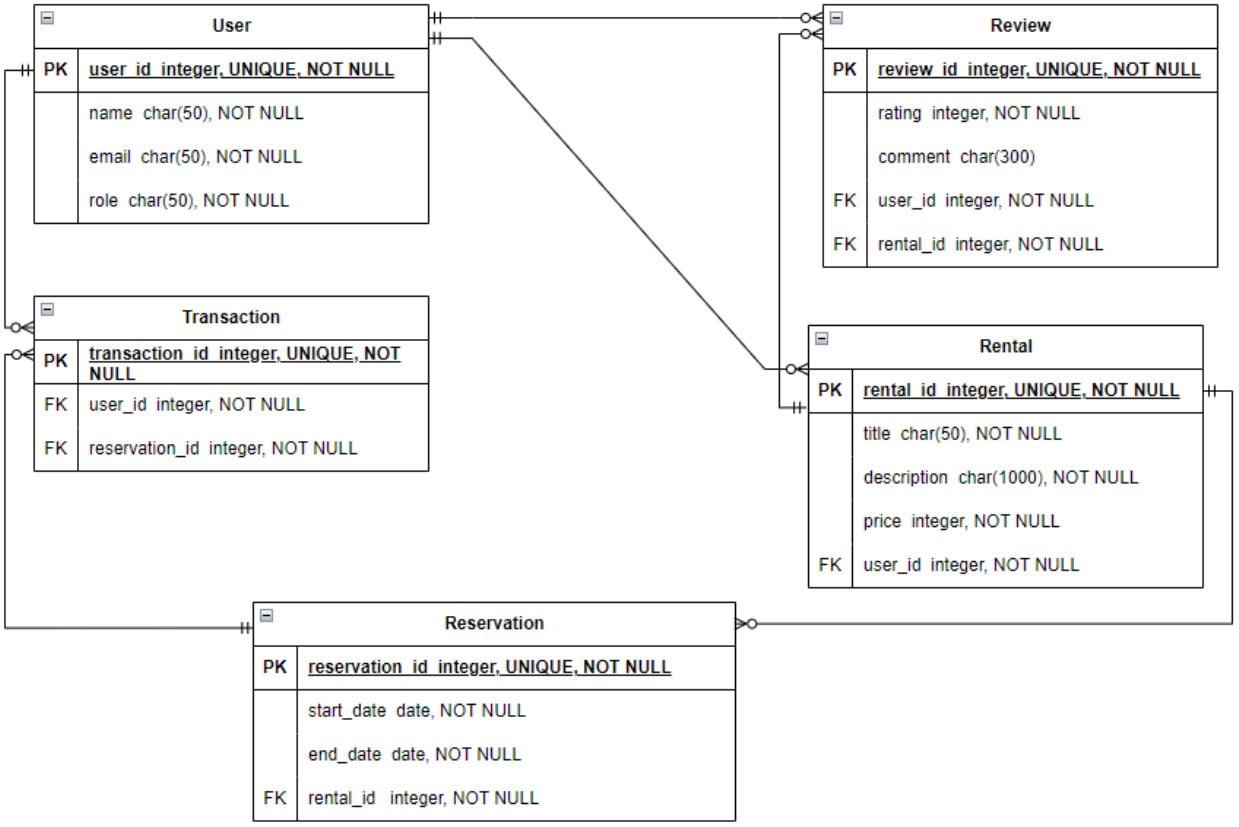


Рисунок 1 – Логічна модель бази даних

Індекси

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. Також тестування обох таблиць буде відбуватися за 4 запитами:

1. Запит з фільтрацією та сортуванням за індексом:

```
SELECT * FROM table

WHERE string LIKE 'a%'

ORDER BY id ASC

LIMIT 10;
```

2. Запит з агрегатною функцією та індексом:

```
SELECT COUNT(id) AS total_records, AVG(id) AS average_id
FROM table;
```

3. Запит з групуванням та індексом:

```
SELECT string, COUNT(*) AS record_count FROM table

GROUP BY string;
```

4. Запит з фільтрацією, групуванням та індексом:

```
SELECT LEFT(string, 1) AS first_letter, AVG(id) AS
average_id FROM table

WHERE string LIKE 'a%'

GROUP BY first_letter;
```

BTree

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

Query | Query History

```
1  CREATE TABLE "btree"("id" bigserial PRIMARY KEY, "string" varchar(100));
2  INSERT INTO "btree"("id", "string")
3  SELECT generate_series as "id", md5(random()::text)
4  FROM generate_series(1, 1000000)
5
```

Data Output | Messages | Notifications

```
INSERT 0 1000000

Query returned successfully in 3 secs 185 msec.
```

Query | Query History

```
1  SELECT * from btree
2
```

Data Output | Messages | Notifications

	id [PK] bigint	string character varying (100)
1	1	4f5293c73f52accf7e23cc46ff2c66ab
2	2	54c7bd7ee33ce817090cd775e57de3…
3	3	ca97f20cf32308eee06a1126039748f8
4	4	74118fed1c00066d40541bdae1b6f5ce
5	5	c23bad3eef4abd0b092b2a2e97369ca8
6	6	929f75805e1de30c9535cc253ca58649

Створення індексів

Query | Query History

```
1  CREATE INDEX btree_index ON btree (string);
2
3
4
5
```

Data Output | Messages | Notifications

```
CREATE INDEX

Query returned successfully in 5 secs 290 msec.
```

Результати виконання запитів

Запит 1

Query Query History

```
1 SELECT * FROM btree
2 WHERE string LIKE 'a%'
3 ORDER BY id ASC
4 LIMIT 10;
5
6
7
```

Data Output Messages Notifications

SQL

	id [PK] bigint	string character varying (100)
1	7	aaac3482d1e2a0844e30b817917c6...
2	8	ad5a8818ee258044b1f618b062d1e...
3	24	aa8a85395615bdf701787e27d9f766...
4	67	abd5408c7e3adbb1d52f4888828d6...
5	74	aa20be42b5241c5e1855f48eae16b5...
6	83	a5dcee9879e931bc9e495a7392229...
7	132	aa1d1654894bdaef4328e94fff53ae0f
8	156	a98b7465d45c10d14af84ba5eae8ff...
9	193	acac3c9a3a02eb645c91a45f55d25a...
10	198	a6c16aee6766f323531afc7cf5d0fa82

Без індекса BTree

✓ Successfully run. Total query runtime: 99 msec. 10 rows affected. ×

З індексом BTree

✓ Successfully run. Total query runtime: 74 msec. 10 rows affected. ×

Запит 2

Query Query History

```
1  SELECT COUNT(id) AS total_records, AVG(id) AS average_id FROM btree;
2
```

Data Output Messages Notifications

	total_records bigint	average_id numeric
1	1000000	500000.500000000000

Без індекса BTree

Successfully run. Total query runtime: 139 msec. 1 rows affected.

З індексом BTree

Successfully run. Total query runtime: 267 msec. 1 rows affected.

Запит 3

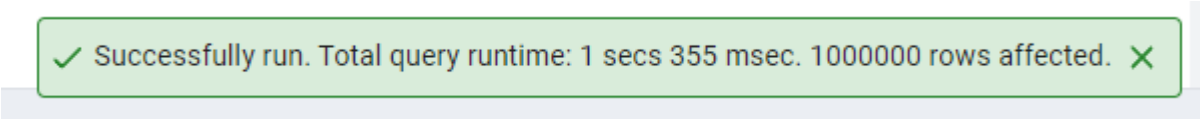
Query Query History

```
1  SELECT string, COUNT(*) AS record_count
2  FROM btree
3  GROUP BY string;
4
5
```

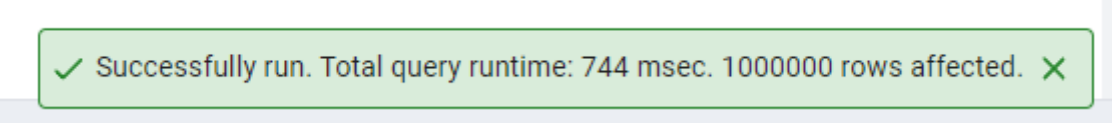
Data Output Messages Notifications

	string character varying (100)	record_count bigint
1	b8f01a2862f6f51e907d1c4f71ea7546	1
2	98e98a990ebd6516e28aad95a04635...	1
3	a2b1e3510ff8ee1b3f7795c88ec57514	1
4	0eb600e76388aafbec7e0146aaf8a906	1
5	caefc545469ff328801fda01bc58ec64	1
6	af8fbc80bd1b063209d08ff2465fe41f	1

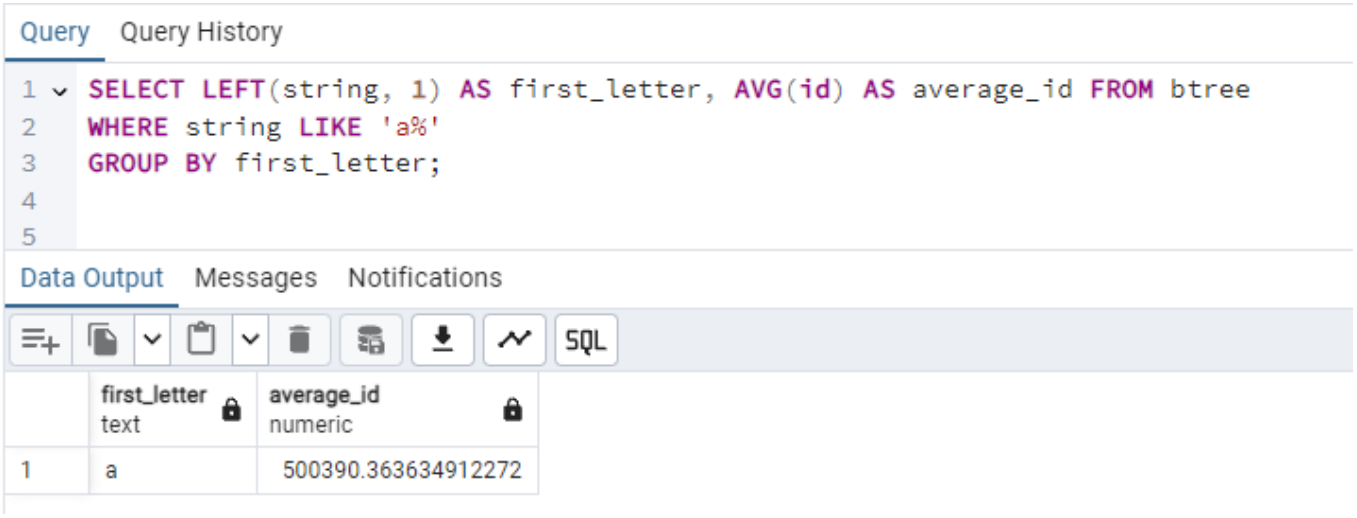
Без індекса BTree



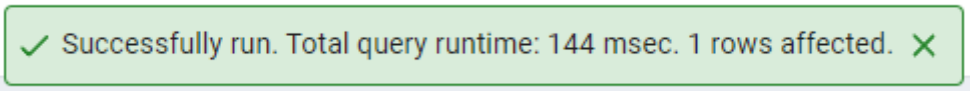
З індексом BTree



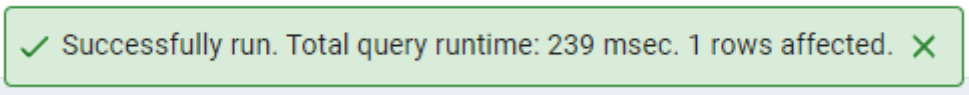
Запит 4



Без індекса BTree



З індексом BTree



Як бачимо, запити 2 та 4 виконуються швидше без використання індексу btree.

Це може бути обумовлено тим, що:

1. Запит, який виконує підрахунок або обчислює середнє значення по всій таблиці, зазвичай не використовує індекс, оскільки для виконання агрегатних функцій потрібно опрацювати всі записи.
2. Запит з умовою string LIKE 'a%' може обійтися без використання індексу, оскільки вибірка рядків, що починаються на 'a', може містити значення, для яких індекс не дає переваг. У таких випадках база даних може вирішити, що ефективніше виконати повне сканування стовпця string, ніж звертатися до індексу.

GIN

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

Query Query History

```
1  CREATE TABLE "gin"("id" bigserial PRIMARY KEY, "string" varchar(100));
2  INSERT INTO "gin"("id", "string")
3  SELECT generate_series as "id", md5(random()::text)
4  FROM generate_series(1, 1000000)
5
```

Data Output Messages Notifications

```
INSERT 0 1000000

Query returned successfully in 3 secs 355 msec.
```

Query Query History

```
1  SELECT * FROM gin
2
```

Data Output Messages Notifications

	id [PK] bigint	string character varying (100)
1	1	ea1b4ca5b15b579a8113619536e26d…
2	2	bb3c0e33da1f4041c8b34f43ba3480bd
3	3	47dc79c183c0952880bec6962236f17e
4	4	4e2e854211ebb255bea1ebe8410ea4…
5	5	1c19a0b1310081d86f1977d8ad431cbf
6	6	69e216c4c914ac17c66702a8eaf3b457
7	7	b07308d9386b63940d001044c03403…
8	8	cc82cda06afa5165193ce7fd7bfe4466
9	9	fc500910c67c880b0934432904cc259

Створення індексів

Query Query History

```
1  CREATE INDEX gin_index ON gin USING gin("string" gin_trgm_ops);
```

Data Output Messages Notifications

```
CREATE INDEX

Query returned successfully in 13 secs 588 msec.
```

Результати виконання запитів

Запит 1

Query Query History

```
1 SELECT * FROM gin
2 WHERE string LIKE 'a%'
3 ORDER BY id ASC
4 LIMIT 10;
5
```

Data Output Messages Notifications

	id [PK] bigint	string character varying (100)
1	11	a9da3aa07d4475c2f1809004ea844b...
2	33	a0586cbd366f6ade37ef298c0c6650bd
3	35	a76fe8589e79d94ebe6edc7b5327bf0a
4	68	a98a490696802c386fffd5e497031861
5	84	a0c968d8bb85c4b709e6550b7150e6...
6	131	a8d5b45e3ef56e79962d0de7319265...
7	171	ae7903eb3e726b4c747f3294a1231c...
8	173	aabc195dfcaa2a16d200359a43d64f...
9	178	a314e717ab90ad647157ceada3cce8...
10	180	a23b5599fdb7be7e37da1aacdbfae4f3

Без індекса BTree

✓ Successfully run. Total query runtime: 92 msec. 10 rows affected. ×

З індексом BTree

✓ Successfully run. Total query runtime: 70 msec. 10 rows affected. ×

Запит 2

Query Query History

```
1  SELECT COUNT(id) AS total_records, AVG(id) AS average_id FROM gin;
```

Data Output Messages Notifications

SQL

	total_records bigint	average_id numeric
1	1000000	500000.500000000000

Без індекса BTree

З індексом BTree

Запит 3

Query Query History

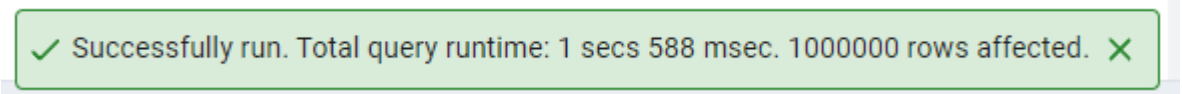
```
1  SELECT string, COUNT(*) AS record_count FROM gin
2  GROUP BY string;
3
```

Data Output Messages Notifications

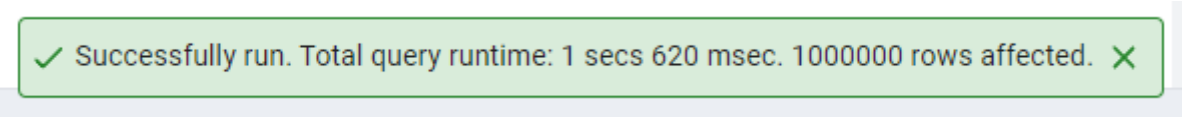
SQL

	string character varying (100)	record_count bigint
1	1264305e244ccd5257b8189bd4f62ebf	1
2	2ef75d0264a4d5674dd67f981f73d603	1
3	cbc7c76dc3e9af513e6fd70848b2278b	1
4	4f1e952b00e8dd7fdf3e994eaba3e7f3	1
5	96f923b71da3929fdf6d9fbd8905c295	1
6	1b86ecfdc89319844051df449cff0970	1
7	e6157693ff8bae37fd49afd8e6ef6ea8	1

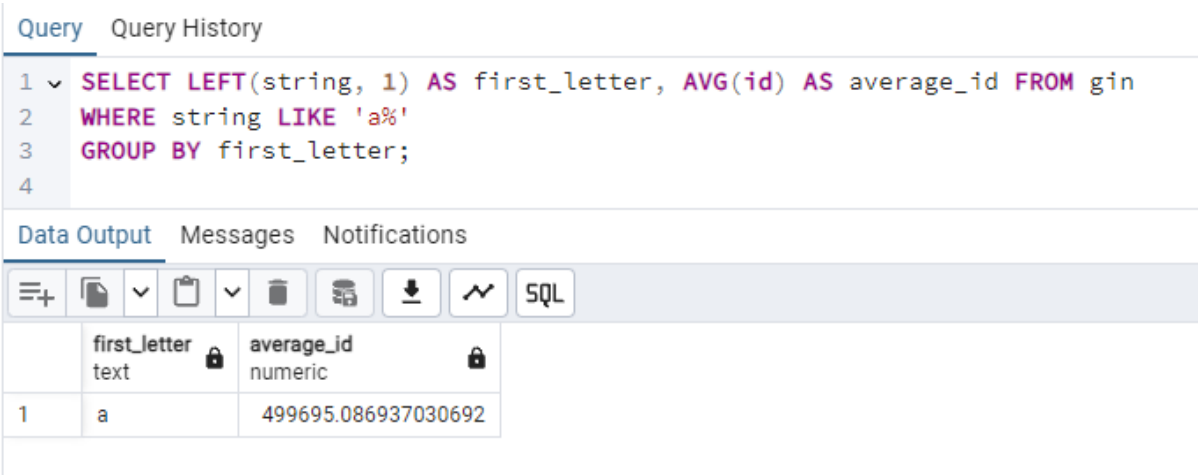
Без індекса BTree



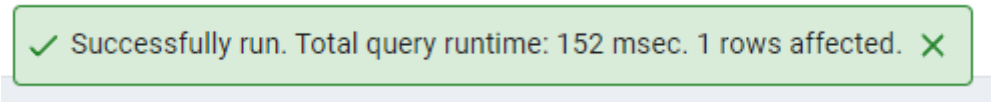
З індексом BTree



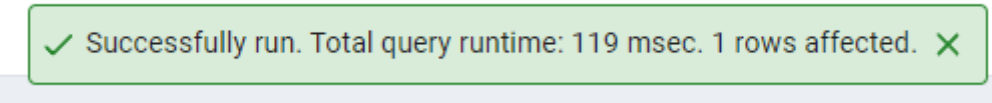
Запит 4



Без індекса BTree



З індексом BTree



Як бачимо, лише запит 3 виконався швидше без використання індексу gin.

Причини, чому індекс може не використовуватись у такому запиті:

1. Агрегація даних:
   GIN-індекси зазвичай оптимізують пошук і фільтрацію, але не завжди корисні для запитів, що виконують агрегацію. Під час групування (GROUP BY) і підрахунку (COUNT), база даних обробляє всі записи для створення груп, а індекс не обов'язково прискорює цей процес.
2. Велика кількість унікальних значень:

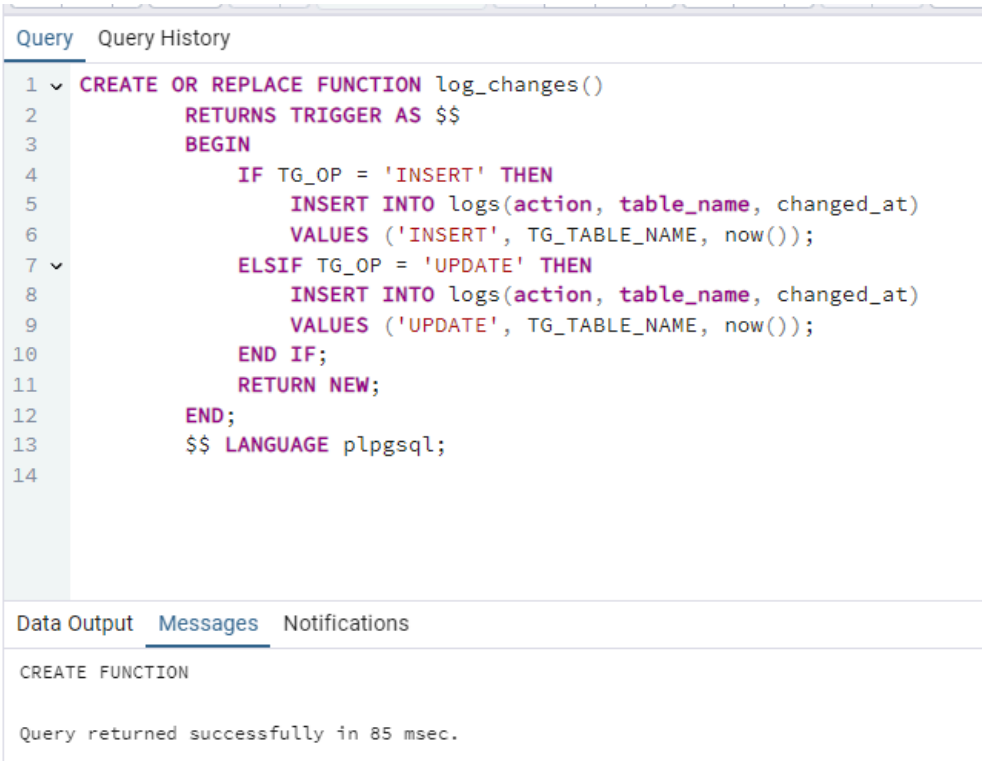
Якщо стовпець string має багато унікальних значень, оптимізатор бази даних може оцінити, що повне сканування таблиці (Sequential Scan) буде ефективнішим, ніж використання індексу. Це пояснюється додатковими витратами на вилучення даних із таблиці після пошуку індексу.

3. Поведінка оптимізатора:
   Оптимізатор PostgreSQL приймає рішення про використання індексу на основі статистики таблиці (розподіл даних, кількість рядків, кардинальність стовпця тощо). Якщо він прогнозує, що використання індексу буде дорожчим, ніж пряме сканування таблиці, індекс не застосовується.

Розробка тригерів

Код тригера:

```
CREATE OR REPLACE FUNCTION log_changes()
        RETURNS TRIGGER AS $$
        BEGIN
            IF TG_OP = 'INSERT' THEN
                INSERT INTO logs(action, table_name, changed_at)
                VALUES ('INSERT', TG_TABLE_NAME, now());
            ELSIF TG_OP = 'UPDATE' THEN
                INSERT INTO logs(action, table_name, changed_at)
                VALUES ('UPDATE', TG_TABLE_NAME, now());
            END IF;
            RETURN NEW;
        END;
        $$ LANGUAGE plpgsql;
```



Прикріплення тригеру до таблиці users:

```
CREATE TRIGGER after_users_change
        AFTER INSERT OR UPDATE ON users
        FOR EACH ROW
        EXECUTE FUNCTION log_changes();
```

Query Query History

```
1
2  CREATE TRIGGER after_users_change
3          AFTER INSERT OR UPDATE ON users
4          FOR EACH ROW
5          EXECUTE FUNCTION log_changes();
6
```

Data Output Messages Notifications

```
CREATE TRIGGER

Query returned successfully in 59 msec.
```

Створення таблиці логів:

Query Query History

```
1  CREATE TABLE logs (
2      log_id SERIAL PRIMARY KEY,
3      action VARCHAR(10) NOT NULL,
4      table_name VARCHAR(50) NOT NULL,
5      changed_at TIMESTAMP NOT NULL
6  );
7
```

Data Output Messages Notifications

```
CREATE TABLE

Query returned successfully in 57 msec.
```

Тестування тригеру

Вставка запису про користувача

Query Query History

```
1  INSERT INTO users (user_id, name, email, role)
2  VALUES (1, 'John Doe', 'john.doe@example.com', 'tenant');
```

Data Output Messages Notifications

```
INSERT 0 1

Query returned successfully in 65 msec.
```

Отримано відповідне повідомлення про вставку від тригера.

Оновлення запису про користувача

Отримано відповідне повідомлення про оновлення від тригера.

Використання рівнів ізоляції

1. READ COMMITTED - кожна команда в транзакції бачить лише дані, зафіксовані до початку команди; вона не бачить змін від паралельних незафіксованих транзакцій. Однак результати запиту можуть змінюватися під час транзакції.

- Вікно 1:

```
BEGIN;

UPDATE users SET email= 'kag@gmail.com' WHERE user_id = 1;

-- Не фіксуємо зміни
```

- Вікно 2:

```
BEGIN;

SELECT email FROM users  WHERE user_id = 1;

-- Вивід буде не ‘kag@gmail.com’, а початкова пошта

COMMIT;
```

- Вікно 1:

```
 COMMIT;
```

- Вікно 2:

```
SELECT email FROM users  WHERE user_id = 1;

-- Тепер вивід буде 'kag@gmail.com'

COMMIT;
```

Зміна, зроблена у Вікні 1, не бачиться запитом у Вікні 2, доки вона не буде зафіксована.

2. REPEATABLE READ - гарантує, що будь-які дані, прочитані під час транзакції, не будуть змінені іншими транзакціями до завершення першої транзакції.

- Вікно 1:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

BEGIN;

SELECT email FROM users WHERE user_id = 2;

-- Не фіксуємо
```

- Вікно 2:

```
BEGIN;

UPDATE users  SET email = 'loro@email.com' WHERE user_id
= 2;

COMMIT;
```

- Вікно 1:

```
SELECT email FROM users  WHERE user_id = 2;

-- Вивід буде початковим прізвищем.

COMMIT;
```

Незважаючи на оновлення у Вікні 2, другий запит SELECT у Вікні 1 все ще показує початкові дані, демонструючи захист від неповторюваних читань на рівні REPEATABLE READ.

3. SERIALIZABLE - найстрогіший рівень, який повністю ізолює транзакцію від будь-якої іншої паралельної транзакції

- Вікно 1:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN;

DELETE FROM users WHERE user_id= 3;
```

-- Не фіксуємо

- Вікно 2:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN;

INSERT INTO users (user_id, name, email, role)

VALUES (3, 'loro', 'lmoe@email.com', 'tennant');
```

-- Ця операція буде заблокована або не вдасться через незафіксоване видалення у Вікні 1

```
COMMIT;
```

- Вікно 1:

```
COMMIT;
```

INSERT у Вікні 2 буде заблокованим або відхилено через конфлікт з не зафіксованим DELETE у Вікні 1. Це показує строгу ізоляцію, яку забезпечує рівень SERIALIZABLE, уникаючи "брудних" читань, неповторюваних читань та "фантомних" читань.

Отже,

1) READ COMMITTED підходить для застосунків, де потрібно збалансувати узгодженість із конкурентоспроможністю та продуктивністю.
2) REPEATABLE READ підходить для застосунків, яким потрібні узгоджені дані протягом всієї транзакції, але які можуть впоратися з певним ступенем конкуренції.
3) SERIALIZABLE ідеально підходить для застосунків, яким потрібна повна ізоляція та узгодженість, зазвичай за рахунок зниження конкурентоспроможності.

Код програми

main.py

```
from controller import Controller


if __name__ == "__main__":
    controller = Controller()
    controller.run()
```

model.py

```
from sqlalchemy import create_engine, Column, Integer, String, ForeignKey,
Float, Date, Text
from sqlalchemy.orm import declarative_base, relationship, sessionmaker
from sqlalchemy.sql import text

Base = declarative_base()

class User(Base):
    __tablename__ = 'users'

    user_id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String(50), nullable=False)
    email = Column(String(50), nullable=False, unique=True)
    role = Column(String(50), nullable=False)

    rentals = relationship("Rental", back_populates="owner")
    reviews = relationship("Review", back_populates="author")

class Rental(Base):
    __tablename__ = 'rental'

    rental_id = Column(Integer, primary_key=True, autoincrement=True)
    title = Column(String(50), nullable=False)
    description = Column(Text, nullable=False)
    price = Column(Float, nullable=False)
    user_id = Column(Integer, ForeignKey('users.user_id'), nullable=False)

    owner = relationship("User", back_populates="rentals")
```

```
    reservations = relationship("Reservation", back_populates="rental")
    reviews = relationship("Review", back_populates="rental")

class Reservation(Base):
    __tablename__ = 'reservation'

    reservation_id = Column(Integer, primary_key=True, autoincrement=True)
    rental_id = Column(Integer, ForeignKey('rental.rental_id'),
nullable=False)
    start_date = Column(Date, nullable=False)
    end_date = Column(Date, nullable=False)

    rental = relationship("Rental", back_populates="reservations")
    transaction = relationship("Transaction", uselist=False,
back_populates="reservation")

class Review(Base):
    __tablename__ = 'reviews'

    review_id = Column(Integer, primary_key=True, autoincrement=True)
    user_id = Column(Integer, ForeignKey('users.user_id'), nullable=False)
    rental_id = Column(Integer, ForeignKey('rental.rental_id'),
nullable=False)
    rating = Column(Integer, nullable=False)
    comment = Column(String(300))

    author = relationship("User", back_populates="reviews")
    rental = relationship("Rental", back_populates="reviews")

class Transaction(Base):
    __tablename__ = 'transactions'

    transaction_id = Column(Integer, primary_key=True, autoincrement=True)
    user_id = Column(Integer, ForeignKey('users.user_id'), nullable=False)
    reservation_id = Column(Integer,
ForeignKey('reservation.reservation_id'), nullable=False)

    reservation = relationship("Reservation", back_populates="transaction")
```

```
DATABASE_URL =
"postgresql+psycopg2://postgres:38743874@localhost:5432/booking_online"
engine = create_engine(DATABASE_URL)
Base.metadata.create_all(engine)

Session = sessionmaker(bind=engine)
session = Session()

class Model:
    def __init__(self):
        self.session = session

    def get_all_tables(self):
        try:
            result = self.session.execute(text("SELECT table_name FROM
information_schema.tables WHERE table_schema = 'public'"))
            return [row[0] for row in result]
        except Exception as e:
            print(f"Error: {e}")
            return []

    def get_all_columns(self, table_name):
        try:
            result = self.session.execute(text("SELECT column_name FROM
information_schema.columns WHERE table_name = :table"), {"table":
table_name})
            return [row[0] for row in result]
        except Exception as e:
            print(f"Error: {e}")
            return []

    def add_data(self, obj):
        try:
            self.session.add(obj)
            self.session.commit()
            return 1
        except Exception as e:
            self.session.rollback()
            print(f"Error: {e}")
            return 0
```

```
    def update_data(self, obj):
        try:
            self.session.commit()
            return 1
        except Exception as e:
            self.session.rollback()
            print(f"Error: {e}")
            return 0

    def delete_data(self, obj):
        try:
            self.session.delete(obj)
            self.session.commit()
            return 1
        except Exception as e:
            self.session.rollback()
            print(f"Error: {e}")
            return 0

    def search_data(self, model_class, **filters):
        try:
            query = self.session.query(model_class).filter_by(**filters)
            return query.all()
        except Exception as e:
            print(f"Error: {e}")
            return []

    def generate_data(self, model_class, count):
        try:
            for i in range(count):
                if model_class == User:
                    obj = User(name=f"User{i}", email=f"user{i}@example.com",
role="tenant" if i % 2 == 0 else "landlord")
                elif model_class == Rental:
                    obj = Rental(title=f"Rental {i}",
description=f"Description {i}", price=100 + i, user_id=1)
                else:
                    continue

                self.session.add(obj)
```

```
            self.session.commit()
        except Exception as e:
            self.session.rollback()
            print(f"Error: {e}")

    def execute_raw_query(self, query, params=None):
        try:
            result = self.session.execute(text(query), params or {})
            return result.fetchall()
        except Exception as e:
            print(f"Error: {e}")
            return []
```

view.py

```
import time


class View:
    def show_menu(self):
        while True:
            print("Menu:")
            print("1. Display table names")
            print("2. Display column names of a table")
            print("3. Add data to a table")
            print("4. Update data in a table")
            print("5. Delete data from a table")
            print("6. Generate data for a table")
            print("7. Search data")
            print("8. Exit")

            choice = input("Make a choice: ")

            if choice in ('1', '2', '3', '4', '5', '6', '7', '8'):
                return choice
            else:
                print("Please enter a valid option number (1 to 8)")
                time.sleep(2)

    def search_menu(self):
```

```
    while True:
        print("\nSelect query type:")
        print("1. Information about users and their rentals")
        print("2. Information about rentals and reservations")
        print("3. Information about rating and rental")
        print("4. Back to main menu")

        choice = input("Enter your choice (1-4): ")

        if choice == '4':
            return None, None

        if choice in ('1', '2', '3'):
            filter_conditions = self.search_data_input(choice)
            return choice, filter_conditions
        else:
            print("Please enter a valid option number (1 to 4)")

def show_message(self, message):
    print(message)
    time.sleep(2)

def ask_continue(self):
    agree = input("Continue making changes? (y/n) ")
    return agree

def show_tables(self, tables):
    print("Table names:")
    for table in tables:
        print(table)
    time.sleep(2)

def ask_table(self):
    table_name = input("Enter the table name: ")
    return table_name

def show_columns(self, columns):
    print("Column names:")
    for column in columns:
        print(column)
    time.sleep(2)
```

```
    def insert(self):
        while True:
            try:
                table = input("Enter the table name: ")
                columns = input("Enter column names (space-separated):
").split()
                val = input("Enter corresponding values (space-separated):
").split()

                if len(columns) != len(val):
                    raise ValueError("The number of columns must match the
number of values.")

                return table, columns, val
            except ValueError as e:
                print(f"Error: {e}")

    def update(self):
        while True:
            try:
                table = input("Enter the table name: ")
                column = input("Enter the name of the column to update: ")
                id = int(input("Enter the ID of the row to update: "))
                new_value = input("Enter the new value: ")
                return table, column, id, new_value
            except ValueError as e:
                print(f"Error: {e}")

    def delete(self):
        while True:
            try:
                table = input("Enter the table name: ")
                id = int(input("Enter the ID of the row to delete: "))
                return table, id
            except ValueError as e:
                print(f"Error: {e}")

    def generate_data_input(self):
        while True:
            try:
```

```
                table_name = input("Enter the table name: ")
                num_rows = int(input("Enter the number of rows to generate:
"))
                return table_name, num_rows
            except ValueError as e:
                print(f"Error: {e}")

    def search_data_input(self, choice):
        while True:
            try:

                filter_conditions = {}

                print("\nEnter search parameters:")

                price_min = input("Minimum price: ")
                price_max = input("Maximum price: ")
                if price_min:
                    filter_conditions['price_min'] = int(price_min)
                if price_max:
                    filter_conditions['price_max'] = int(price_max)

                if choice == '1' or choice == '2':
                    title = input("Title (LIKE pattern): ")
                    if title:
                        filter_conditions['title'] = title


                if choice == '1':
                    name = input("Name (LIKE pattern): ")
                    email = input("Email (LIKE pattern): ")

                    if name:
                        filter_conditions['name'] = name
                    if email:
                        filter_conditions['email'] = email

                if choice == '3':
                    rating_min = input("Minimum rating: ")
                    rating_max = input("Maximum rating: ")
                    if rating_min:
```

```
                        filter_conditions['rating_min'] = int(rating_min)
                    if rating_max:
                        filter_conditions['rating_max'] = int(rating_max)

                filter_conditions['group_by'] = (
                    ['t1.user_id', 't2.rental_id'] if choice == '1' else
                    ['t1.rental_id', 't2.reservation_id'] if choice == '2'
else
                    ['t1.rental_id', 't2.review_id']
                )

                return filter_conditions

            except ValueError as e:
                print(f"Error: {e}")
```

controller.py

```
import sys

from model import Model
from view import View


class Controller:
    def __init__(self):
        self.view = View()
        try:
            self.model = Model()
            self.view.show_message("Connected to the database")
        except Exception as e:
            self.view.show_message(f"An error occurred during initialization:
{e}")
            sys.exit(1)

    def run(self):
        while True:
            choice = self.view.show_menu()
            if choice == '1':
                self.view_tables()
```

```
            elif choice == '2':
                self.view_columns()
            elif choice == '3':
                self.add_data()
            elif choice == '4':
                self.update_data()
            elif choice == '5':
                self.delete_data()
            elif choice == '6':
                self.generate_data()
            elif choice == '7':
                self.search_data()
            elif choice == '8':
                break

    def view_tables(self):
        tables = self.model.get_all_tables()
        self.view.show_tables(tables)

    def view_columns(self):
        table_name = self.view.ask_table()
        columns = self.model.get_all_columns(table_name)
        self.view.show_columns(columns)

    def add_data(self):
        while True:
            table, columns, val = self.view.insert()
            error = self.model.add_data(table, columns, val)
            if int(error) == 1:
                self.view.show_message("Data added successfully!")
                agree = self.view.ask_continue()
                if agree == 'n':
                    break
            elif int(error) == 2:
                self.view.show_message("Unique identifier already exists!")
                agree = self.view.ask_continue()
                if agree == 'n':
                    break
            else:
                self.view.show_message("Invalid foreign key")
                agree = self.view.ask_continue()
```

```
                if agree == 'n':
                    break

    def update_data(self):
        while True:
            table, column, id, new_value = self.view.update()
            error = self.model.update_data(table, column, id, new_value)
            if int(error) == 1:
                self.view.show_message("Data updated successfully!")
                agree = self.view.ask_continue()
                if agree == 'n':
                    break
            elif int(error) == 2:
                self.view.show_message(f"Unique identifier {new_value}
already exists!")
                agree = self.view.ask_continue()
                if agree == 'n':
                    break
            else:
                self.view.show_message(f"Invalid foreign key {new_value} in
column {column}")
                agree = self.view.ask_continue()
                if agree == 'n':
                    break

    def delete_data(self):
        while True:
            table, id = self.view.delete()
            error = self.model.delete_data(table, id)
            if int(error) == 1:
                self.view.show_message("Row deleted successfully!")
                agree = self.view.ask_continue()
                if agree == 'n':
                    break
            else:
                self.view.show_message("Cannot delete row due to related data
existing")
                agree = self.view.ask_continue()
                if agree == 'n':
                    break
```

```
    def generate_data(self):
        table_name, num_rows = self.view.generate_data_input()
        self.model.generate_data(table_name, num_rows)
        self.view.show_message(f"Data for table {table_name} has been
generated successfully")

    def search_data(self):
        choice, filters = self.view.search_menu()
        if choice == '1':
            table1 = 'users'
            table2 = 'rental'
        elif choice == '2':
            table1 = 'rental'
            table2 = 'reservation'
        else:
            table1 = 'rental'
            table2 = 'reviews'

        result = self.model.search_data(table1=table1, table2=table2,
query_type=choice, filter_conditions=filters)

        if result:
            print("\nSearch results:")
            for row in result:
                print(row)
        else:
            print("\nNo data matching the search criteria.")
```