**Міністерство освіти і науки України   
Національний технічний університет України  
 “Київський політехнічний інститут ім. Ігоря Сікорського”**

**Факультет прикладної математики   
Кафедра спеціалізованих комп’ютерних систем**

**Лабораторна робота № 3**   
з дисципліни «Бази даних і засоби управління»   
«Засоби оптимізації роботи СУБД PostgreSQL »

Виконав:  
студент групи   
КВ-74 Мережко І.П.

Перевірив: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2019

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.   
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. 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 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, що Дисципліни «Бази даних та засоби управління» та «Бази даних. Частина 1». Осінь 2019 року містять фільтрацію, агрегатні функції, групування та сортування (у необхідних комбінаціях).

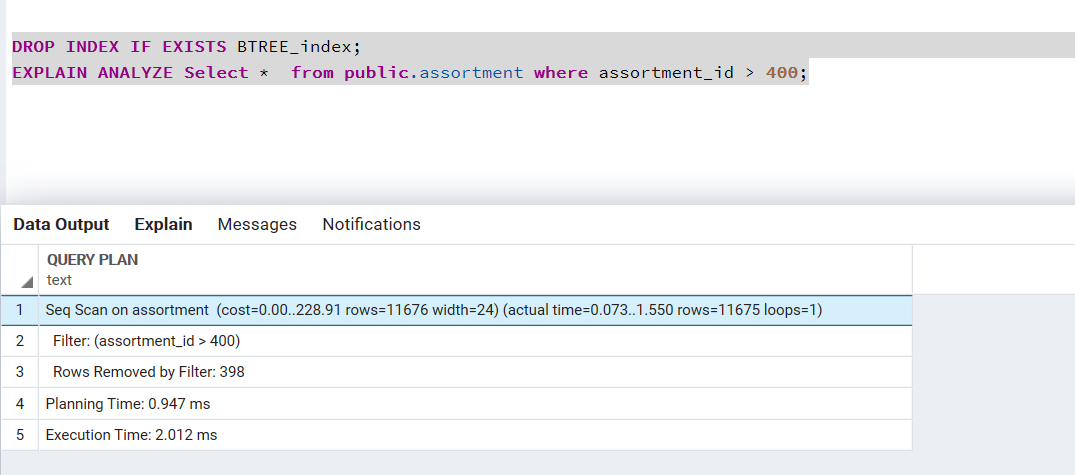
Вимоги до пункту завдання №3 Створити тригер бази даних PostgreSQL відповідно до варіанта. Тригерна функція має включати обробку запису, що модифікується (вставляється або вилучається), умовні оператори, курсорні цикли та обробку виключних ситуацій. Виконати відлагодження тригера при різних вхідних даних, 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

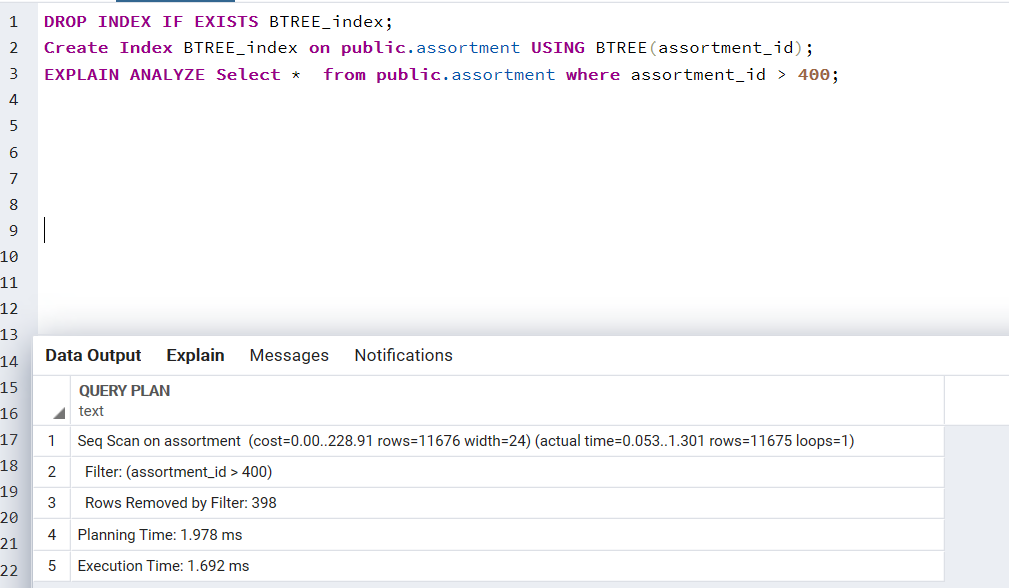
Варіант 12  
 indexes: Btree, GIN   
trigger: after insert, after update

1.

|  |  |
| --- | --- |
| def insert\_callback(self, current\_table, values, columns\_name): | |
|  | table\_object = self.tables\_map[current\_table] |
|  | try: |
|  | db\_object = table\_object.create() |
|  | for i in range(0, len(values)): |
|  | setattr(db\_object, columns\_name[i], values[i]) |
|  | self.session.add(db\_object) |
|  | self.session.commit() |
|  | except Exception as e: |
|  | self.session.rollback() |
|  | print(e) |
|  | return '', '' |
|  |  |
|  | def select\_callback(self, current\_table, fields\_value): |
|  | table\_object = self.tables\_map[current\_table] |
|  | query = self.session.query(table\_object) |
|  | if len(fields\_value) != 0: |
|  | for field\_name, fields\_value in fields\_value: |
|  | query = query.filter(get\_column\_object(table\_object, field\_name)==fields\_value) |
|  | for element in query.all(): |
|  | print(element.get\_item()) |
|  | return "", "" |
|  |  |
|  | def update\_callback(self, current\_table, primary\_key, new\_fields): |
|  | try: |
|  | table\_object = self.tables\_map[current\_table] |
|  | query = self.session.query(table\_object) |
|  | query = query.filter(get\_column\_object(table\_object, primary\_key[0]) == primary\_key[1]) |
|  | d\_ = dict() |
|  | for column, newValue in new\_fields: |
|  | d\_[column] = newValue |
|  | print(d\_) |
|  | query.update(d\_) |
|  |  |
|  | except Exception as e: |
|  | self.session.rollback() |
|  | print(e) |
|  | return "", "" |
|  |  |
|  | def delete\_callback(self, current\_table, fields): |
|  | table\_object = self.tables\_map[current\_table] |
|  | query = self.session.query(table\_object) |
|  | for field\_name, fields\_value in fields: |
|  | query = query.filter(get\_column\_object(table\_object, field\_name) == fields\_value) |
|  | try: |
|  | for obj in query.all(): |
|  | self.session.delete(obj) |
|  | self.session.commit() |
|  | except Exception as e: |
|  | self.session.rollback() |
|  | print(e) |
|  |  |
|  | return '', '' |

2.Вибірка з таблиці без використання індексів BTREE.

Вибірка з таблиці з використанням індексів BTREE.



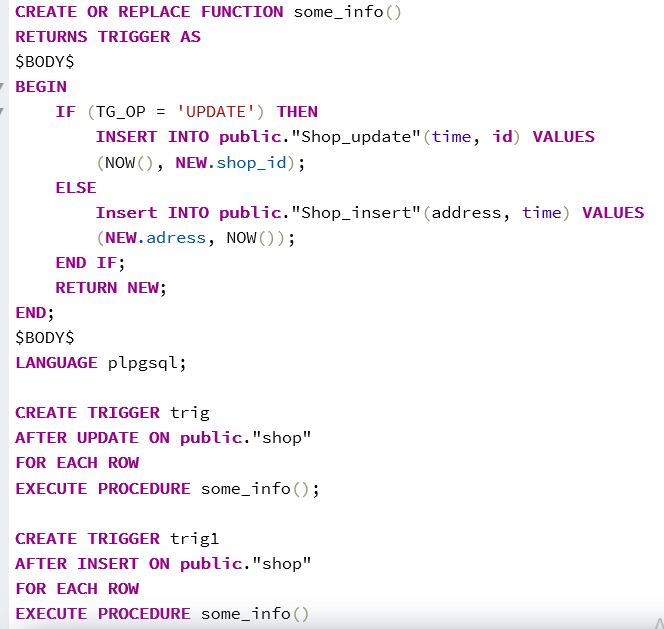
Вибірка з таблиці без використання індексів GIN на текстове поле.

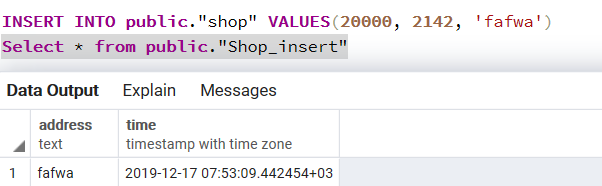


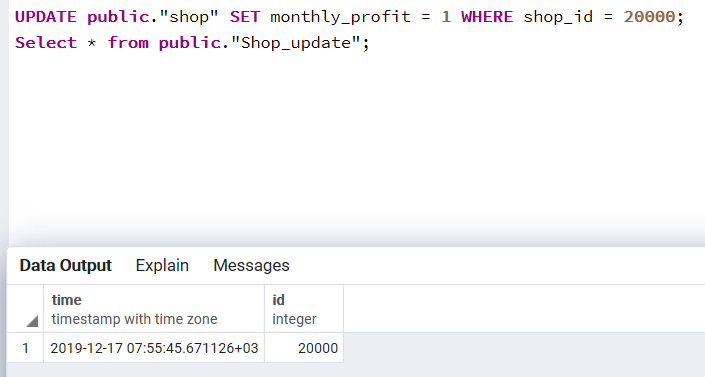
Вибірка з таблиці з використання індексів GIN на текстове поле.



3.

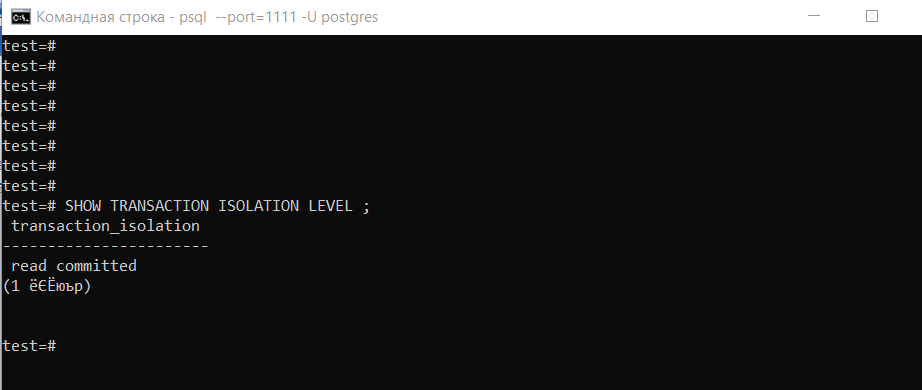


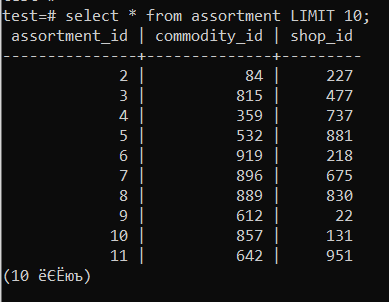


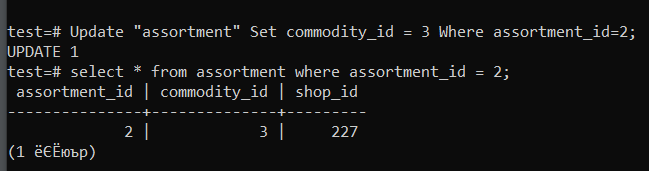


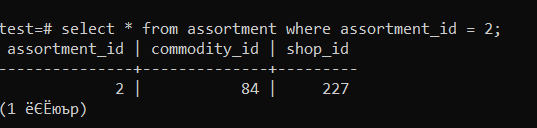
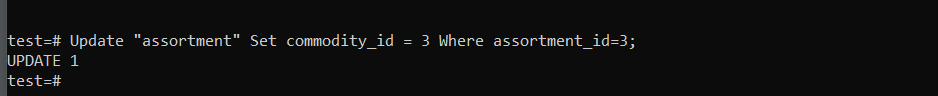
4.

Tr1

 Tr2



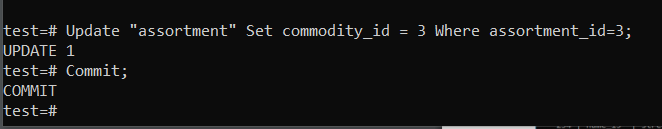
TR1

TR2

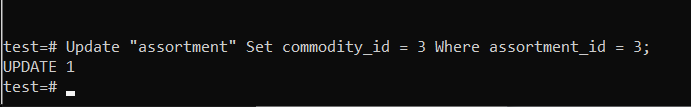
TR1

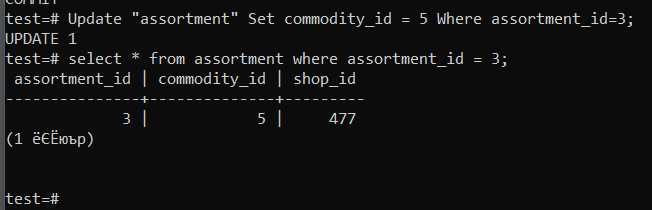
TR2

TR1

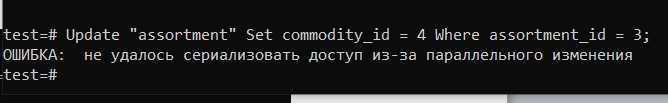


TR2

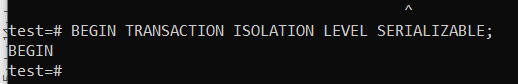


TR1

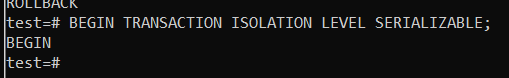
TR2

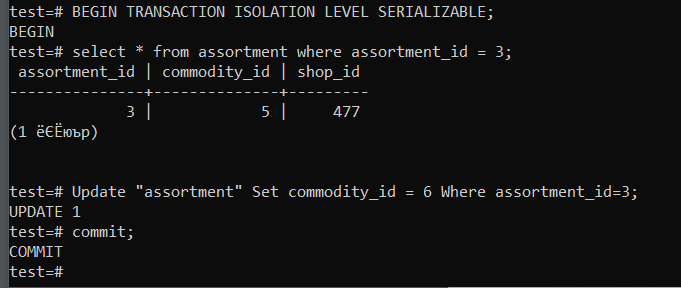


SERIALIZABLE

TR1

TR2



TR1

TR2

