«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра прикладної математики

Лабораторна робота №1

із кредитного модуля

«Бази Даних»

на тему:

«Перевірка рівнів ізоляції в Postgres**.»**

Виконав: Недзельський І.О.

Група: КМ-12 факультет ФПМ

Керівник: Жук І.С.

Київ 2024

Мета Роботи

Необхідно декількома способами реалізувати оновлення значення каунтера в СКБД PostgreSQL та оцінити час кожного із варіантів.

Структура таблиці:

user_counter

USER_ID	Counter	Version
1	0	0
...	...	

Нехай маємо таблицю **user_counter** наступної структури

- **USER_ID** - ключ з ідентифікатором користувача
- **Counter** - Integer - значення каунтера
- **Version** - Integer - допоміжне поле, буде використовуватись в одному з варіантів реалізації.

Основна частина

1. Варіант реалізації **Lost Update**:

Варіант реалізації "Lost Update" передбачає виконання оновлення значення каунтера без використання механізмів блокування або контролю версій. Основні особливості цього підходу: конфлікти при одночасних оновленнях, брак гарантії атомарності, потреба в додаткових механізмах захисту. Цей варіант реалізації показує ризики та недоліки в контексті одночасного доступу до даних та може призводити до несподіваних результатів при одночасних оновленнях з різних джерел.

Час: 18.77 с.

Результат:

USER_ID	Counter	Version
1	13039	0

2. Варіант реалізації **In-place update**:

Використовує базовий синхронізований блок, щоб гарантувати, що лише один потік може оновлювати лічильник за один раз, потенційно створюючи конкуренцію та зменшуючи паралельність. "In-place update" є простим методом, але він може призвести до проблем конфліктів та втрати даних в умовах великої конкуренції між потоками або клієнтами.

Час: 140.5 с.

Результат:

USER_ID	Counter	Version
1	100000	0

3. Варіант реалізації **Row-level locking**:

Варіант реалізації "Row-level locking" передбачає застосування блокування на рівні рядка для забезпечення консистентності при одночасних зчитуваннях та оновленнях. Основні особливості цього підходу: блокування на рівні рядка, забезпечення атомарності операцій, зменшення конфліктів для інших рядків, можливість блокування на довший термін. "Row-level locking" є ефективним методом для забезпечення консистентності даних в умовах великої конкуренції, але варто ретельно враховувати ризики блокування та вибирати оптимальний підхід в конкретному контексті.

Час: 251.64 с.

Результат:

USER_ID	Counter	Version
1	100000	0

4. Варіант реалізації **Optimistic concurrency control**:

Варіант реалізації "Optimistic Concurrency Control" використовує позначення версії для уникнення конфліктів при одночасних зчитуваннях та оновленнях. Основні особливості цього підходу: використання версій, оптимістичний підхід, перевірка версій під час оновлення, не вимагає блокування на тривалий термін, використання транзакцій. "Optimistic Concurrency Control" ефективний в умовах невеликої конкуренції та призначений для ситуацій, де конфлікти виникають не дуже часто.

Час: 194.21 с.

Результат:

USER_ID	Counter	Version
1	100000	100000

Висновки

У сценаріях, подібних до нашого, де база даних повинна обробляти тисячі одночасних запитів, стратегія «**In-place update**» стає найбільш прийнятним вибором, так як пропонує баланс між надійністю, яка проявляється в запобіганні втрат оновлень, і підтриманням достатньо швидшого виконання порівняно з іншими реалізаціями.

Додаток А

1) Програма що реалізує варіант **Lost Update:**

```
import psycopg2
import time
import concurrent.futures

username = 'nedzelsky'
password = '3030'
database = 'DB_LAB_1'

query_00 = '''
DROP TABLE IF EXISTS user_counter
'''

query_0 = '''
CREATE TABLE user_counter (
    user_id INT PRIMARY KEY,
    counter INT,
    version INT
)
'''

query_001 = '''
DELETE FROM user_counter
'''

query_1 = '''
INSERT INTO user_counter (user_id, counter, version) VALUES (1, 0, 0)
'''

conn = psycopg2.connect(user=username, password=password, dbname=database)

cursor = conn.cursor()

cursor.execute(query_00)

cursor.execute(query_0)

cursor.execute(query_1)

def lost_update(user_id):
    for _ in range(10_000):
        cursor.execute("SELECT counter FROM user_counter WHERE user_id = %s",
(user_id,))
        counter = cursor.fetchone()[0]
        counter += 1
        cursor.execute("UPDATE user_counter SET counter = %s WHERE user_id =
%s", (counter, user_id))
```

```
    conn.commit()

start_time = time.time()

with concurrent.futures.ThreadPoolExecutor(max_workers=10) as executor:
        executor.map(lost_update, [1 for _ in range(10)])

end_time = time.time()
total_time = end_time - start_time

print(f"Total time: {total_time} seconds")

cursor.close()
conn.close()
```

2) Програма що реалізує варіант **In-place update:**

```
import psycopg2
import time
import concurrent.futures

username = 'nedzelsky'
password = '3030'
database = 'DB_LAB_1'

query_00 = '''
DROP TABLE IF EXISTS user_counter
'''

query_0 = '''
CREATE TABLE user_counter (
    user_id INT PRIMARY KEY,
    counter INT,
    version INT
)
'''

query_1 = '''
INSERT INTO user_counter (user_id, counter, version) VALUES (1, 0, 0)
'''

conn = psycopg2.connect(user=username, password=password, dbname=database)

cursor = conn.cursor()

cursor.execute(query_00)

cursor.execute(query_0)

cursor.execute(query_1)
```

```
def in_place_update(user_id):
    for _ in range(10_000):
        cursor.execute("UPDATE user_counter SET counter = counter + 1 WHERE
user_id = %s", (user_id,))
    conn.commit()

start_time = time.time()

with concurrent.futures.ThreadPoolExecutor(max_workers=10) as executor:
        executor.map(in_place_update, [1 for _ in range(10)])

end_time = time.time()
total_time = end_time - start_time

print(f"Total time: {total_time} seconds")

cursor.close()
conn.close()
```

3) Програма що реалізує варіант **Row-level locking:**

```
import psycopg2
import concurrent.futures
import time

username = 'nedzelsky'
password = '3030'
database = 'DB_LAB_1'

query_00 = '''
DROP TABLE IF EXISTS user_counter
'''

query_0 = '''
CREATE TABLE user_counter (
    user_id INT PRIMARY KEY,
    counter INT,
    version INT
)
'''

query_1 = '''
INSERT INTO user_counter (user_id, counter, version) VALUES (1, 0, 0)
'''

conn = psycopg2.connect(user=username, password=password, dbname=database)

with conn:
    cursor = conn.cursor()
```

```
    cursor.execute(query_00)

    cursor.execute(query_0)

    cursor.execute(query_1)

def row_level_locking_update(user_thread_id):
    conn = psycopg2.connect(user=username, password=password,
dbname=database)

    with conn:
        cursor = conn.cursor()

        for _ in range(10_000):
            cursor.execute("SELECT counter FROM user_counter WHERE user_id =
%s FOR UPDATE", (user_thread_id[0],))
            counter = cursor.fetchone()[0]
            counter += 1
            cursor.execute("UPDATE user_counter SET counter = %s WHERE
user_id = %s", (counter, user_thread_id[0]))

        conn.commit()

start_time = time.time()

with concurrent.futures.ThreadPoolExecutor(max_workers = 10) as executor:
        executor.map(row_level_locking_update, [(1, i) for i in range(10)])

end_time = time.time()
total_time = end_time - start_time

print(f"Total time: {total_time} seconds")
```

4) Програма що реалізує варіант **Optimistic concurrency control:**

```
import psycopg2
import concurrent.futures
import time

username = 'nedzelsky'
password = '3030'
database = 'DB_LAB_1'

query_00 = '''
DROP TABLE IF EXISTS user_counter
'''

query_0 = '''
CREATE TABLE user_counter (
```

```
    user_id INT PRIMARY KEY,
    counter INT,
    version INT
)
'''

query_1 = '''
INSERT INTO user_counter (user_id, counter, version) VALUES (1, 0, 0)
'''

conn = psycopg2.connect(user=username, password=password, dbname=database)

with conn:
    cursor = conn.cursor()

    cursor.execute(query_00)

    cursor.execute(query_0)

    cursor.execute(query_1)

def optimistic_concurrency_control_update(user_thread_id):
    conn = psycopg2.connect(user=username, password=password,
dbname=database)

    with conn:
        cursor = conn.cursor()

        for i in range(10_000):
            while True:
                cursor.execute("SELECT counter, version FROM user_counter
WHERE user_id = %s", (user_thread_id[0],))
                current_values = cursor.fetchone()
                counter, version = current_values[0], current_values[1]

                counter += 1
                cursor.execute("UPDATE user_counter SET counter = %s, version
= %s WHERE user_id = %s AND version = %s",
                               (counter, version + 1, user_thread_id[0],
version))

                conn.commit()
                count = cursor.rowcount
                if count > 0:
                    break

start_time = time.time()

with concurrent.futures.ThreadPoolExecutor(max_workers = 10) as executor:
        executor.map(optimistic_concurrency_control_update, [(1, i) for i in
range(10)])
```

```
end_time = time.time()
total_time = end_time - start_time

print(f"Total time: {total_time} seconds")
```