**Реалізація каунтера з використанням PostgreSQL**

**Мета**

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 | 1 | 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.

Для реалізації практичної та перевірки особливостей роботи транзакцій достатньо мати один рядок.

На додаткові бали можна ще перевірити різницю в часі виконання операцій, якщо в таблиці буде, нехай, 100 000 рядків, апдейт виконуватиметься то першого, то рендомного (довільного не однакового) рядка. Яка буде різниця в часі виконання всіх запитів та у кількості лайків.

Ця перевірка стосується пошуку за кластерним індексом.

**Варіанти реалізації**

1. **Lost-update (реалізація що втрачатиме значення)**

З 10 потоків чи окремих клієнтів одночасно запустити

*for (i in 1..10\_000) {*

*counter = cursor.execute(“SELECT counter FROM user\_counter WHERE user\_id = 1”).fetchone()*

*counter = counter + 1*

*cursor.execute(("update user\_counter set counter = %s where user\_id = %s", (counter, 1))*

*conn.commit()*

*}*

Необхідно виміряти час виконання

1. **In-place update**

З 10 потоків чи окремих клієнтів одночасно запустити

*for (i in 1..10\_000) {*

*cursor.execute(("update user\_counter set counter = counter + 1 where user\_id = %s", (1))*

*conn.commit()*

*}*

Необхідно виміряти час виконання

1. **Row-level locking**

Row level locking with *SELECT ... FOR UPDATE*:

<https://www.postgresql.org/docs/current/explicit-locking.html#LOCKING-ROWS>

<https://www.postgresql.org/docs/current/applevel-consistency.html#NON-SERIALIZABLE-CONSISTENCY>

З 10 потоків чи окремих клієнтів одночасно запустити

*for (i in 1..10\_000) {*

*counter = cursor.execute(“SELECT counter FROM user\_counter WHERE user\_id = 1 FOR UPDATE”).fetchone()*

*counter = counter + 1*

*cursor.execute(("update user\_counter set counter = %s where user\_id = %s", (counter, 1))*

*conn.commit()*

*}*

Необхідно створювати окреме підключення та курсор для кожного потоку, щоб *“select … for update”* мав ефект.

Необхідно виміряти час виконання

1. **Optimistic concurrency control**

З 10 потоків чи окремих клієнтів одночасно запустити

*for (i in 1..10\_000) {*

*while (True) {*

*(counter, version) = cursor.execute(“SELECT counter, version FROM user\_counter WHERE user\_id = 1”).fetchone()*

*counter = counter + 1*

*cursor.execute(("update user\_counter set counter = %s, version = %s where user\_id = %s" and version = %s, (counter, version + 1, 1, version))*

*conn.commit()*

*count = cursor.rowcount*

*if (count > 0) break*

*}*

*}*

Необхідно виміряти час виконання

**Вимоги до реалізації**

* Мова реалізації будь-яка
* Не використовувати ORM-фреймворки (Hibernate, SQLAlchemy, …)
* Не забувати про необхідність окремої транзакції на кожен запис

**Звіт**

У результаті виконання роботи має бути надано код скрипта та отримані результати тестування для різних сценаріїв.

В усіх варіантах, крім першого, кінцеве значення каунтера має бути 100К

**Посилання**

* Взаємодія з БД для Python <https://www.psycopg.org/docs/usage.html>