ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Бази даних та інформаційні системи

ЛАБОРАТОРНА РОБОТА №2

Транзакції в СКБД PostgreSQL

Виконав:

Ст. Прізвище Ім’я

Група _____

2022

Тема: Вивчення понять транзакції та управління конкурентним доступом в СКБД PostgreSQL.

Мета роботи: Ознайомлення з використанням транзакцій, їх розробкою та застосуванням, рівнями ізоляцій та механізмом управління конкурентним доступом в СКБД PostgreSQL.

Теоретичний матеріал.

Транзакції та їх властивості

Транзакції - це фундаментальне поняття в усіх СКБД. Суть транзакції в тому, що *вона об'єднує послідовність дій в одну операцію* - ***"все або нічого"***. Проміжні стани всередині послідовності невидимі іншим транзакціям і якщо щось завадить успішно завершити транзакцію, жоден з результатів цих дій не збережеться в базі даних.

Важливими властивостями транзакцій є:

- Атомарність - гарантує, що транзакція завершується за принципом «все або нічого».
- Узгодженість - гарантує, що зміни даних, записаних у базу даних, мають бути коректними та відповідати попередньо визначеним правилам.
- Ізоляція - визначає, як (коли) зміни даних виконані транзакцією будуть видимі для інших транзакцій.
- Довговічність - гарантує, що результати транзакції, які були зафіксовані, зберігатимуться в базі даних постійно.

Наприклад, розглянемо базу даних банку, в якій міститься інформація про рахунки клієнтів, а також загальні суми по відділеннях банку. Припустимо, що ми хочемо перевести 100 доларів з рахунку Аліси на рахунок Боба. Відповідні SQL-команди можна записати так:

```
UPDATE accounts SET balance = balance - 100.00
                WHERE name = 'Alice';

UPDATE branches SET balance = balance - 100.00
       WHERE name = (SELECT branch_name FROM accounts WHERE name = 'Alice');

UPDATE accounts SET balance = balance + 100.00
                WHERE name = 'Bob';

UPDATE branches SET balance = balance + 100.00
       WHERE name = (SELECT branch_name FROM accounts WHERE name = 'Bob');
```

Точний зміст команд тут не важливий, важливо лише те, що для виконання цієї досить простої операції було потрібно декілька окремих дій. При цьому з точки зору банку *необхідно, щоб всі ці дії виконалися разом, або не виконалися зовсім*. Якщо Боб отримає 100 доларів, але вони не будуть списані з рахунку Аліси, пояснити це помилкою системи, безумовно, не вдасться. І навпаки, Аліса навряд чи буде задоволена, якщо вона переведе гроші, а до Боба вони не дійдуть. Нам потрібна гарантія, що якщо щось завадить виконати операцію до кінця, жодна з дій не залишить сліду в базі даних. І ми отримуємо цю гарантію, об'єднуючи дії в одну *транзакцію* . Кажуть, що транзакція *атомарна,* якщо з точки зору інших транзакцій вона або виконується і фіксується повністю, або не фіксується зовсім.

Нам також потрібна гарантія *довговічності* змін, тобто, що після завершення і підтвердження транзакції системою баз даних, її результати справді зберігаються і не будуть втрачені, навіть якщо незабаром відбудеться аварія. Наприклад, якщо ми списали суму і видали її Бобу, ми повинні виключити можливість того, що сума на його рахунку відновиться, як тільки він вийде за

двері банку. Транзакційна база даних гарантує, що *всі зміни записуються* в постійне сховище (наприклад, на диск) *до того*, як транзакція буде вважатися завершеною.

Інша важлива характеристика транзакційних баз даних – *ізоляція*, тісно пов'язана з атомарністю змін: коли одночасно виконується безліч транзакцій, кожна з них не бачить незавершені зміни, зроблені іншими. Наприклад, якщо одна транзакція підраховує баланс по відділеннях, буде неправильно, якщо вона підрахує витрати в відділенні Аліси, але не врахує прихід в відділенні Боба, або навпаки. Тому властивість транзакцій "все або нічого" має визначати не тільки, як зміни зберігаються в базі даних, але і як їх видно в процесі роботи. *Зміни, зроблені відкритою транзакцією, невидимі для інших транзакцій, поки вона не буде завершена, а потім вони стають видні всі відразу*.

Управління транзакціями

У PostgreSQL транзакція визначається набором SQL-команд, оточеним командами **`BEGIN`** і **`COMMIT`**. Таким чином, наша банківська транзакція повинна була б виглядати так:

```
BEGIN;

UPDATE accounts SET balance = balance - 100.00
                WHERE name = 'Alice';
- ...

COMMIT;
```

Якщо в процесі виконання транзакції ми вирішимо, що не хочемо фіксувати її зміни (наприклад, тому що виявилося, що баланс Аліси став негативним), ми можемо виконати команду **`ROLLBACK`** замість `COMMIT`, і всі наші зміни будуть скасовані.

PostgreSQL насправді відпрацьовує кожен SQL-оператор як транзакцію. Якщо ви не вставите команду `BEGIN`, то кожен окремий оператор буде неявно оточений командами `BEGIN` і `COMMIT` (в разі успішного завершення). Групу операторів, оточених командами `BEGIN` і `COMMIT` іноді називають *блоком транзакції* .

Операторами в транзакції можна також керувати на більш детальному рівні, використовуючи *точки збереження* . Точки збереження дозволяють вибірково скасовувати деякі частини транзакції і фіксувати всі інші. Визначивши точку збереження за допомогою **`SAVEPOINT`**, при необхідності ви можете повернутися до неї за допомогою команди **`ROLLBACK TO`**.

Всі зміни в базі даних, що відбулися після точки збереження і до моменту відкату, скасовуються, але зміни, зроблені раніше, зберігаються.

Коли ви повертаєтеся до точки збереження, вона продовжує існувати, так що ви можете відкочуватися до неї кілька разів. З іншого боку, якщо ви впевнені, що вам не доведеться відкочуватися до певної точки збереження, її можна видалити, щоб система вивільнила ресурси.

При видаленні або відкаті до точки збереження всі точки збереження визначені після неї, автоматично знищуються.

Все це відбувається в блоці транзакції, так що в інших сеансах роботи з базою даних цього не видно. Виконані дії стають видимі для інших сеансів, тільки коли ви фіксуєте транзакцію, а скасовані дії не видно взагалі ніколи.

Повернувшись до банківської бази даних, припустимо, що ми списуємо 100 доларів з рахунку Аліси, додаємо їх на рахунок Боба, і раптом виявляється, що гроші потрібно було перевести Уоллі. В даному випадку ми можемо застосувати точки збереження:

```
BEGIN;

UPDATE accounts SET balance = balance - 100.00
                WHERE name = 'Alice';
SAVEPOINT my_savepoint;

UPDATE accounts SET balance = balance + 100.00
                WHERE name = 'Bob';
- помилкова дія ... ігнорувати її і використовувати рахунок Уоллі

ROLLBACK TO my_savepoint;

UPDATE accounts SET balance = balance + 100.00
                WHERE name = 'Wally';
COMMIT;
```

Цей приклад, звичайно, трохи надуманий, але він показує, як можна управляти виконанням команд в блоці транзакцій, використовуючи точки збереження. Більше того, '**`ROLLBACK TO`**' - *єдиний спосіб* повернути контроль над блоком транзакцій, які опинилися в перерваному стані через помилки системи, беручи до уваги можливості повністю скасувати її і почати знову.

У процедурах, що викликаються командою CALL, а також в анонімних блоках коду (в команді DO) можна завершувати транзакції, виконуючи COMMIT і ROLLBACK. Після завершення транзакції цими командами нова буде розпочато автоматично, тому окремої команди START TRANSACTION немає. (Зауважте, що команди BEGIN і END в PL / pgSQL мають інший сенс).

Наприклад:

```
CREATE PROCEDURE transaction_test1 ()
LANGUAGE plpgsql
AS $$
BEGIN
    FOR i IN 0..9 LOOP
        INSERT INTO test1 (a) VALUES (i);
        IF i% 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END;
$$;

CALL transaction_test1 ();
```

Управління транзакціями можливо тільки у викликах CALL або DO *в коді верхнього рівня* або *у вкладених CALL або DO без інших проміжних команд.*

Наприклад, в стеку викликів

`CALL proc1()` → **`CALL proc2()`** → **`CALL proc3()`**

друга і третя процедури можуть управляти транзакціями.

Але в стеці

`CALL proc1()` → **`SELECT func2()`** → **`CALL proc3()`**

остання процедура позбавлена цієї можливості через проміжного **`SELECT`**.

Циклам з курсором притаманні деякі особливості. Наприклад:

```
CREATE PROCEDURE transaction_test2 ()
LANGUAGE plpgsql
AS $$
DECLARE
    r RECORD;
BEGIN
    FOR r IN SELECT * FROM test2 ORDER BY x LOOP
        INSERT INTO test1 (a) VALUES (r.x);
        COMMIT;
    END LOOP;
END;
$$;

CALL transaction_test2 ();
```

Зазвичай курсори автоматично закриваються при фіксуванні транзакції. Однак курсор, створюваний всередині циклу подібним чином, автоматично перетворюється в утримуваний курсор першою командою `COMMIT` або `ROLLBACK`. Це означає, що курсор повністю обчислюється при виконанні першої команди `COMMIT` або `ROLLBACK`, а не для кожного чергового рядка. При цьому він автоматично видаляється після циклу, так що це відбувається практично непомітно для користувача.

Команди управління транзакціями не допускаються в циклах з курсором, якими керують запити, що не лише читають, а й модифікують дані (наприклад, `UPDATE ... RETURNING`).

Транзакція не може завершуватися всередині блоку з обробниками винятків.

Ресурси для ознайомлення з теоретичним матеріалом теми «Транзакції в СКБД PostgreSQL»:

1.PostgreSQL Documentation
2. PostgreSQLTutorial.com website. Section 10. Transactions
3. PostgreSQL Documentation. Concurrency Control

Перелік розділів та понять, з якими необхідно ознайомитись для виконання завдання лабораторної роботи (за документацією до PostgreSQL):

Управління конкурентним доступом
1. Вступ
2. Ізоляція транзакцій
    2.1. Рівень ізоляції Read Committed
    2.2. Рівень ізоляції Repeatable Read
    2.3. Рівень ізоляції Serializable
3. Явні блокування
    3.1. Блокування на рівні таблиці
    3.2. Блокування на рівні рядків
    3.3. Блокування на рівні сторінок
    3.4. Взаємоблокування
    3.5. Рекомендаційні блокування
4. Перевірки цілісності даних на рівні додатку
    4.1. Забезпечення узгодженості в Серіалізованих транзакціях
    4.2. Застосування явних блокувань для забезпечення узгодженості
5. Обмеження
6. Блокування та індекси

Хід роботи

1. Опрацювати теоретичний матеріал.
2. Розробити кілька транзакцій відповідно до власних потреб роботи зі створюваною базою даних. У транзакціях необхідно продемонструвати:
    - команди початку та фіксації транзакції;
    - використання точки збереження SAVEPOINT;
    - відкат виконання транзакції ROLLBACK TO;
    - блок обробки помилок EXCEPTION та вивід повідомлень RAISE;
3. На прикладі паралельного виконання двох транзакцій однакового рівня ізоляції **Repeatable Read** пояснити їх роботу з точки зору видимості змін під час виконання та після фіксації.
4. Оформити звіт про виконання лабораторної роботи, який має містити:
    - титульну сторінку;
    - тему, мету та завдання лабораторної роботи;
    - короткий опис роботи створених транзакцій;
    - навести скріни екрану з кодом транзакцій, видимості змін даних під час їх виконання та після фіксації та отриманими результатами.
5. Завантажити в канал “БД. Лабораторна робота” своєї команди в Teams.