Transacciones

Antonio Espín Herranz

Transacciones

- Una transacción es un conjunto de instrucciones DML (Insert, Update, Delete) que se ejecutan entre dos comandos begin, Commit o Rollback.

- La transacción nos asegura que un conjunto de instrucciones SQL se van a ejecutar como **una sola instrucción** o no se van a ejecutar.
  - Son operaciones de todo o nada.

- La transacción termina con una instrucción:
  - **Commit**: Confirmando los cambios.
  - **Rollback**: Deshaciendo los cambios.

Transacciones

- Una vez que cerramos una transacción (con Commit o Rollback) la siguiente instrucción DML iniciará otra transacción nueva.

- PostgreSQL mantiene la **coherencia con los datos** y los usuarios. Un usuario no verá los cambios confirmados hasta que el otro usuario no haga el commit.

- Las instrucciones DDL no forman parte de transacciones. La ejecución de un comando Create, Alter o Drop confirmará la transacción que esté iniciada.

Transacciones

- Las transacciones son un concepto fundamental de todos los sistemas de bases de datos.

- El punto esencial de una transacción es que agrupa varios pasos en una única **operación de todo o nada**.

- Los estados intermedios entre los pasos no son visibles para otras transacciones simultáneas y, si se produce algún error que impide que se complete la transacción, ninguno de los pasos afecta en absoluto a la base de datos.

Propiedades ACID

- **Atomicidad**
  - El primer término de ACID en base de datos se refiere a la atomicidad de las transacciones, es decir, que el sistema permite que se lleven a cabo las operaciones atómicas. ***Esta propiedad indica que, para que una transacción se dé por «completada», deben haberse realizado todas sus partes o ninguna de ellas.***

- **Consistencia**:
  - El concepto de consistencia en el modelo ACID en base de datos está relacionado con la propiedad de atomicidad y hace referencia a la capacidad que tiene un sistema para iniciar solo operaciones que puede concluir. Esto implica que solo se pueden ejecutar pasos de la transacción que no incumplan con las reglas o directrices de integridad definidas, incluyendo los triggers, cascades y constraints, así como sus combinaciones.

Propiedades ACID

- **Aislamiento**
  - La propiedad de aislamiento del modelo ACID en bases de datos se refiere a la manera y el momento en el que los cambios resultantes de una operación se harán visible para las demás operaciones concurrentes. Es decir, ***la realización de una operación no debería afectar a las otras, debido a que cada una de las transacciones debe ser ejecutada en aislamiento total***, sin importar si se llevan a cabo de manera simultánea.

- **Durabilidad**
  - La durabilidad de ACID en bases de datos hace referencia a la propiedad que ***garantiza que, una vez se haya llevado a cabo una determinada operación (aquellas transacciones que tuvieron un commit), estas tengan la capacidad de persistir y no puedan ser deshechas*** incluso si el sistema falla o se presentan eventos como errores o caídas o pérdida de alimentación eléctrica, entre otros

Ejemplo

- 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');

Transacciones

- En PostgreSQL , una transacción se configura rodeando los comandos SQL de la transacción con comandos BEGIN y COMMIT.

- Para poder utilizar estos comandos mencionados (BEGIN, COMMIT y ROLLBACK) debemos de desactivar el AUTOCOMMIT. Ésta, opción es a nivel de cliente y por defecto está activada. De forma que toda sentencia ejecutada queda confirmada y registrada en la base de datos.

  \set autocommit off → desactivar

  \set autocommit on → activar

BEGIN;

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

-- etc etc

COMMIT;

Savepoint

- Puntos intermedios para confirmar o revocar parte de una transacción.
- Podemos volver a un punto guardado con la instrucción **rollback to**.

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00 WHERE name = 'Alice';
SAVEPOINT my_savepoint;
UPDATE accounts SET balance = balance + 100.00 WHERE name = 'Bob';
ROLLBACK TO my_savepoint;
UPDATE accounts SET balance = balance + 100.00  WHERE name = 'Wally';
COMMIT;
```

Transacciones en procedimientos almacenados

- Dentro de un procedimiento almacenado que hemos llamado con **CALL**.

- Cuando finalizamos una transacción con commit o rollback automáticamente se inicia otra sin tener una instrucción que sea start transaction.

Ejemplo

```
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();
```

Ejemplo 2

```
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();
```

Transacciones

- Normalmente, los cursores se cierran automáticamente al confirmar la transacción.
- Sin embargo, un cursor creado como parte de un bucle se convierte automáticamente en un cursor que se puede sostener mediante el primer COMMIT o ROLLBACK.
- Eso significa que el cursor se evalúa completamente en la primera fila COMMIT o ROLLBACK en lugar de fila por fila.
- El cursor aún se elimina automáticamente después del bucle, por lo que es prácticamente invisible para el usuario.

Enlaces

- https://www.postgresql.org/docs/current/plpgsql-transactions.html