Transacciones y Plan de consulta

Transacciones

Las transacciones son secuencias de instrucciones ordenadas, las cuales pueden ser indicadas de forma manual o pueden ser aplicadas automáticamente. Estas realizan cambios en las bases de datos, a la hora de aplicar comandos de manipulación de columnas y registros. Estas transacciones tienen las siguientes propiedades:

- **Atomicidad:** Todas las operaciones realizadas en la transacción deben ser completadas. En el caso que ocurra un fallo, está la transacción es abortada y devuelve todo al estado previo a la transacción.
- **Consistencia:** La base de datos cambiará solamente cuando la transacción se haya realizado.
- **Aislamiento:** Las transacciones pueden ocurrir independientes unas de otras.
- **Durability:** El resultado de la transacción persiste a pesar de que el sistema falle.

Es posible tener control de las transacciones. Para eso, existen los siguientes comandos:

- **COMMIT:** Guarda los cambios de la transacción.
- **ROLLBACK:** Retrocede los cambios realizados.
- **SAVEPOINT:** Guarda el punto de partida al cual volver a la hora de aplicar **ROLLBACK.**
- **SET TRANSACTION:** Le asigna nombre a la transacción.

Estos comandos sólo pueden ser usados con las operaciones **INSERT**, **UPDATE** y **DELETE**, ya que aquellos que manipulan toda la tabla hacen este proceso automáticamente. La sintaxis de estos comandos es la siguiente:

```
COMMIT;
```

```
SAVEPOINT nombre_savepoint;
```

```
ROLLBACK [TO nombre_savepoint];
```

Lo que esta entre corchetes es de carácter opcional, por lo que podemos decirle a ROLLBACK a que punto volver, o este volverá al último punto por defecto.

```
SET TRANSACTION [READ ONLY|WRITE][NAME nombre_transaccion];
```

En este caso, podemos usar **READ ONLY** para solamente leer la base de datos, **READ WRITE** para leer y escribir sobre ella, y poder nombrar la transacción con el comando **NAME**.

Veamos el siguiente ejemplo:

```
SET TRANSACTION READ WRITE NAME primera_transaccion;
INSERT INTO nombre_tabla (columna1, columna2, columna3) VALUES (valor1,
valor2, valor3);
SAVEPOINT registro_ingresado;
```

En este caso creamos una transacción en la que se puede leer y escribir sobre la base de datos que tiene de nombre 'primera_transaccion', luego insertamos un valor e hicimos **SAVEPOINT** para poder volver a este punto en cualquier caso.

Veamos el siguiente ejemplo:

```
UPDATE nombre_tabla SET columna1=valor_nuevo WHERE condicion;
ROLLBACK TO registro_ingresado;
COMMIT;
```

Ahora, realizamos un **UPDATE**, sin embargo, nos devolvimos al punto guardado en SAVEPOINT, para así finalmente guardarlo haciendo **COMMIT**.

Plan de consulta

Un plan de consulta es una muestra de cuanto demoraría realizar una consulta y los pasos ordenados que esta sigue para poder realizarla sin la necesidad de hacer la consulta como tal. Cuando utilizamos comandos tales como **SELECT**, **UPDATE** o **DELETE**, el motor de la base de datos debe elegir cuál es el mejor realizar una consulta, sin embargo, no podemos saber a simple viste que es lo que este elige. Para poder saber eso, utilizamos el comando **EXPLAIN**.

Este lo que haces es mostrar un árbol con distintos niveles, el cual esta ordenado según las acciones que esta toma. La sintaxis es la siguiente:

```
EXPLAIN [ANALYZE] operacion_a_ejecutar;
```

El comando opcional **ANALYZE** permite que no sólo pregunte por el plan, sino que también realice la operación indicada. Si utilizamos **EXPLAIN ANALYZE** pero no queremos que se aplica la operación debemos hacer lo siguiente:

```
BEGIN;
EXPLAIN ANALYZE operacion_a_ejecutar;
ROLLBACK;
```

Veamos el siguiente ejemplo que hace una consulta con **WHERE**:

```
EXPLAIN ANALYZE SELECT * FROM Directorio_telefonico WHERE
apellido='Quezada';
```

Nos da como respuesta:

```
Seq Scan on directorio_telefonico  (cost=0.00..11.38 rows=1 width=690)
(actual time=2.905..2.908 rows=1 loops=1)
   Filter: ((apellido)::text = 'Quezada'::text)
   Rows Removed by Filter: 9
 Planning Time: 0.189 ms
 Execution Time: 2.954 ms
```

Donde nos indica que se hizo un escaneo por secuencia, donde el primer paréntesis indica lo que espera obtener en cuanto al costo de iniciación (antes de los 2 puntos) y el costo de ejecución (después de los puntos), la cantidad de columnas seleccionadas y el ancho de estas, y en el segundo paréntesis indica lo que realmente se obtuvo, más la cantidad de ciclos realizados. También indica el filtro que se utilizó, cuantas columnas removió, el tiempo del plan y el tiempo de ejecución de este.

Hacer uso del plan de consulta es útil para saber qué es lo que está pasando durante la ejecución de nuestra consulta y detectar si es que existe algún problema que esté haciendo tomar más tiempo de lo que es necesario.

Ahora, veamos este ejemplo utilizando **JOIN**:

```
EXPLAIN ANALYZE SELECT a.nick, dT.direccion
FROM Directorio_telefonico dT
JOIN Agenda a ON dT.numeroTelefonico=a.numeroTelefonico;
```

Nos da como respuesta:

```
Hash Join  (cost=12.47..30.34 rows=620 width=584) (actual
time=0.250..0.259 rows=10 loops=1)
   Hash Cond: ((a.numerotelefonico)::text = (dt.numerotelefonico)::text)
   ->  Seq Scan on agenda a  (cost=0.00..16.20 rows=620 width=102)
(actual time=0.056..0.058 rows=10 loops=1)
   ->  Hash  (cost=11.10..11.10 rows=110 width=550) (actual
time=0.106..0.106 rows=10 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Seq Scan on directorio_telefonico dt  (cost=0.00..11.10
rows=110 width=550) (actual time=0.067..0.074 rows=10 loops=1)
 Planning Time: 39.777 ms
 Execution Time: 0.337 ms
```

En este caso, podemos ver que el árbol es más grande, dado la complejidad de la operación, sin embargo, es el mismo formato anterior, donde lo único que cambia es la acción ejecutada en el plan. Cada símbolo `->` representa un nivel del árbol, donde las acciones se ven en orden desde arriba hacia abajo.