author: @nurialiano

Comandos transaccionales y de bloqueo de tablas

Las transacciones permiten agrupar un conjunto de operaciones de base de datos como una única unidad lógica, que se ejecuta de forma atómica (es decir, todas las operaciones se completan o ninguna lo hace), consistente (el resultado de la transacción deja la base de datos en un estado válido) y aislada (las operaciones dentro de una transacción son invisibles para otras transacciones).

Los comandos transaccionales típicos son COMMIT (para confirmar una transacción) y ROLLBACK (para deshacer una transacción). El uso adecuado de las transacciones puede mejorar la consistencia y la integridad de los datos de la base de datos y prevenir situaciones en las que se producen cambios parciales e inconsistentes.

Por otro lado, el bloqueo de tablas es una forma de controlar el acceso concurrente a los datos de una tabla. El bloqueo de tablas se utiliza para evitar situaciones en las que dos o más transacciones intentan modificar la misma fila al mismo tiempo. Cuando una transacción bloquea una tabla, impide que otras transacciones accedan o modifiquen los datos de la tabla hasta que se libere el bloqueo. Los bloqueos pueden ser explícitos (adquiridos mediante comandos LOCK TABLES) o implícitos (adquiridos automáticamente por la base de datos en función de la operación que se está ejecutando).

Sintaxis de START TRANSACTION, COMMIT y ROLLBACK

START TRANSACTION inicia una nueva transacción. Este comando indica a la base de datos que todas las operaciones que se ejecuten a partir de ese momento deben ser agrupadas como parte de la transacción actual.

EJEMPLO

Supongamos que tienes una tabla llamada "ventas" con dos columnas: "id" y "monto". Quieres actualizar el monto de la venta con el ID 1234 y agregar un registro a otra tabla llamada "registros" con información sobre la transacción.

```
START TRANSACTION;

UPDATE ventas
SET monto = 100
WHERE id = 1234;

INSERT INTO registros (fecha, monto, descripcion)
VALUES (NOW(), 100, 'Venta actualizada');

COMMIT;
```

En este ejemplo, se inicia una transacción con el comando START TRANSACTION. A continuación, se actualiza el monto de la venta con ID 1234 y se inserta un nuevo registro en la tabla "registros". Si ambas operaciones tienen éxito, se confirman con el comando COMMIT y se hacen permanentes en la base de datos. Si algo sale mal, se puede deshacer la transacción con el comando ROLLBACK para revertir todos los cambios realizados en la base de datos.

COMMIT se utiliza para confirmar una transacción y hacer que todos los cambios realizados durante la transacción se hagan permanentes en la base de datos. Cuando se ejecuta un COMMIT, se confirma que todas las operaciones de la transacción se han completado correctamente y que se pueden aplicar los cambios de forma permanente. Por defecto está habilitado.

EJEMPLO

Supongamos que tienes una tabla llamada "empleados" con las columnas "id", "nombre", "salario" y "departamento". Quieres actualizar el salario de un empleado y confirmar los cambios en la base de datos.

```
UPDATE empleados
SET salario = salario * 1.1
WHERE id = 123;

COMMIT;
```

En este ejemplo, se actualiza el salario del empleado con el ID 123 multiplicándolo por 1.1. Luego, se confirman los cambios en la base de datos con el comando COMMIT, lo que hace que los cambios sean permanentes y no reversibles. A partir de ese momento, el salario del empleado con el ID 123 será un 10% mayor que antes de la actualización.

Deshabilitar / habilitar autocommit

Por defecto, el valor de esta variable es "1" (encendido), lo que significa que cada sentencia SQL es confirmada automáticamente como una transacción independiente. Si lo estableces en "0" (apagado), las sentencias SQL se agruparán en transacciones que deben confirmarse manualmente mediante el comando COMMIT o revertirse mediante el comando ROLLBACK.

```
SET AUTOCOMMIT = 0;
```

ROLLBACK se utiliza para deshacer una transacción y volver a un estado anterior. Cuando se ejecuta un ROLLBACK, todas las operaciones realizadas desde el inicio de la transacción se deshacen y se vuelve a un estado anterior a la transacción. Esto significa que se revierten todos los cambios realizados durante la transacción y que la base de datos vuelve a estar en el estado en el que se encontraba antes de la transacción.

EJEMPLO

Supongamos que estás insertando datos en dos tablas, "clientes" y "pedidos", dentro de una transacción. Si la inserción en la tabla "pedidos" falla por algún motivo (por ejemplo, por una violación de clave foránea)

```
START TRANSACTION;

INSERT INTO clientes (id, nombre, email)
VALUES (1, 'Juan', 'juan@email.com');

INSERT INTO pedidos (id_cliente, fecha, cantidad)
VALUES (1, '2023-04-17', 5); -- Esta inserción fallará por una violación de
clave foránea

ROLLBACK;
```

Cuando esto suceda, el comando ROLLBACK deshará todas las operaciones realizadas dentro de la transacción, incluyendo la inserción en la tabla "clientes". Después de ejecutar el comando ROLLBACK, la base de datos volverá al estado en que estaba antes de que comenzara la transacción, sin haber realizado ningún cambio permanente.

Sentencias que no se pueden deshacer

Algunos comandos no se pueden deshacer. En general esto incluye comandos DDL, tales como los que crean y borrar bases de datos, los que crean o alteran tablas o rutinas almacenadas

Ejemplo

```
DROP TABLE mi_tabla;
```

Sentencias que causan una ejecución (commit) implícita

Las sentencias que causan una ejecución (commit) implícita son aquellas que, además de realizar una acción específica en la base de datos, también confirman cualquier transacción que se esté ejecutando actualmente. Esto significa que no es necesario ejecutar explícitamente un comando COMMIT para confirmar la transacción después de ejecutar una de estas sentencias.

Estos son todos los comandos que causan una transaccion implícita

ALTER TABLE	BEGIN	CREATE INDEX
CREATE TABLE	-	CREATE DATABASE
DROP DATABASE	DROP INDEX	DROP TABLE
LOAD MASTER DATA	LOCK TABLES	RENAME TABLE
SET AUTOCOMMIT	START TRANSACTION	TRUNCATE TABLE

Ejemplo

```
TRUNCATE TABLE mi_tabla;
```

Sintaxis de SAVEPOINT y ROLLBACK TO SAVEPOINT

Se utilizan en MySQL para crear puntos de guardado de transacciones llamados SAVEPOINT, lo que permite deshacer solo una parte de una transacción en lugar de deshacer toda la transacción.

```
SAVEPOINT savepoint_name;
```

Para revertir la transacción a un SAVEPOINT

```
ROLLBACK TO SAVEPOINT savepoint_name;
```

Esto restaura la transacción al SAVEPOINT indicado, revirtiendo cualquier cambio realizado a partir de ese punto en adelante.

> 📝**NOTA** es importante tener en cuenta que los SAVEPOINTS solo se pueden utilizar dentro de una transacción, y que no se pueden utilizar después de que se haya realizado un COMMIT en la transacción. Además, no se pueden utilizar fuera de la sesión de conexión actual.

Bloqueo de tablas

Es un mecanismo que se utiliza para controlar el acceso concurrente a las tablas de una base de datos. Cuando una transacción necesita realizar operaciones en una tabla, puede solicitar un bloqueo de esa tabla para garantizar que ningún otro proceso pueda modificar los datos de esa tabla mientras se realiza la transacción.

- Bloqueo de lectura: Permite que múltiples transacciones lean los datos de la tabla simultáneamente, pero impide que cualquier transacción realice modificaciones en la tabla.
- Bloqueo de escritura: Permite que una sola transacción realice modificaciones en una tabla, impidiendo que cualquier otra transacción lea o escriba en la misma tabla mientras el bloqueo está activo.

Sintaxis de LOCK TABLES

```
LOCK TABLES nombre_tabla WRITE;
```

Sintaxis de UNLOCK TABLES

Desbloqueo de todas la tablas

```
UNLOCK TABLES;
```

Desbloqueo de una tabla

```
UNLOCK TABLES nombre_tabla;
```

Sintaxis de SET TRANSACTION

Se utiliza para establecer el nivel de aislamiento de la transacción actual en una base de datos en particular. El nivel de aislamiento determina la forma en que una transacción interactúa con otras transacciones que pueden estar intentando acceder a los mismos datos en la misma base de datos.

> 📝 **NOTA** puedes usar SET TRANSACTION en cualquier momento dentro de una transacción, ya sea antes o después de iniciar la transacción con START TRANSACTION. La diferencia radica en cómo se aplican los cambios en los niveles de aislamiento.
>
> - antes de START TRANSACTION: los cambios en los niveles de aislamiento se aplicarán a todas las declaraciones dentro de la transacción que sigue a START TRANSACTION.
> - después de START TRANSACTION: los cambios en los niveles de aislamiento solo se aplicarán a las declaraciones que siguen a SET TRANSACTION.

```
SET TRANSACTION [ READ WRITE | READ ONLY ] [ , ISOLATION LEVEL {
SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED } ];
```

Las opciones **READ WRITE** y **READ ONLY** se utilizan para especificar si la transacción va a leer y escribir datos o solo leer datos. La opción **ISOLATION LEVEL** se utiliza para especificar el nivel de aislamiento de la transacción, que puede ser uno de los cuatro siguientes:

- **SERIALIZABLE**: El nivel de aislamiento más alto, que garantiza que ninguna otra transacción pueda modificar los datos que la transacción actual está leyendo o escribiendo hasta que la transacción actual se haya completado.
- **REPEATABLE READ**: Garantiza que todas las lecturas realizadas dentro de la transacción actual siempre vean los mismos datos, independientemente de si otros cambios están ocurriendo en la base de datos.
- **READ COMMITTED**: Garantiza que todas las lecturas realizadas dentro de la transacción actual vean solo los cambios confirmados realizados por otras transacciones.
- **READ UNCOMMITTED**: El nivel de aislamiento más bajo, que permite a la transacción leer datos que aún no han sido confirmados por otras transacciones.

Ejemplo

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

START TRANSACTION;

-- Aquí puedes ejecutar tus operaciones de base de datos
```

```
COMMIT;
```