6.9 Gestión de Transacciones

Una transacción es la herramienta de los SGBD relacionales que aseguran la ATOMICIDAD EN LAS OPERACIONES.

La atomicidad facilitará que podamos hacer operaciones consecutivas sobre los datos de forma que se hagan todas o no se haga ninguna. Es decir, trataremos varias operaciones como si fuera una sola.

Una transacción es una unidad lógica de trabajo formada por un conjunto de instrucciones (selecciones, inserciones, actualizaciones o eliminaciones). El objetivo del uso de transacciones será que si no hay errores durante una transacción, todas las modificaciones llevadas a cabo en la misma se convierten en parte permanente de la base de datos. Si se producen errores, no se realiza ninguna modificación en la base de datos, esto es, el estado de la BD debe revertirse al estado anterior al comienzo de la transacción.

Estas son también conocidas como “transacciones ACID”. Siendo ACID las siglas de las características que possen las transacciones.

Una unidad lógica de trabajo (o transacción) debe exhibir cuatro propiedades, conocidas como propiedades ACID (atomicidad, coherencia, aislamiento y durabilidad), para ser calificada como transacción:

Atomicidad

Una transacción debe ser una unidad atómica de trabajo, tanto si se realizan todas sus modificaciones en los datos, como si no se realiza ninguna de ellas. Es decir, se trata de un conjunto de operaciones que son tratadas como si fuera una sola.

Coherencia

Cuando finaliza, una transacción debe dejar todos los datos en un estado coherente. En una base de datos relacional, se debe garantizar que todas las estructuras de datos, deben ser coherentes al final de la transacción. (Por ejemplo debemos asegurar que no va a quedar una factura sin productos).

Aislamiento

Las modificaciones realizadas por transacciones simultáneas se deben aislar de las modificaciones llevadas a cabo por otras transacciones simultáneas. Una transacción ve los datos en el estado en que estaban antes de que otra transacción simultánea los modificara o después de que la segunda transacción se haya concluido, pero no ve un estado intermedio. Esto se conoce como seriabilidad debido a que su resultado es la capacidad de volver a cargar los datos iniciales y reproducir una serie de transacciones para finalizar con los datos en el mismo estado en que estaban después de realizar las transacciones originales. (Mientras no termine una transacción los datos que se están modificando en ella no son visibles desde fuera, por ejemplo desde otra sesión).

Durabilidad

Una vez concluida una transacción, sus efectos son permanentes en el sistema. Las modificaciones persisten aún en el caso de producirse un error del sistema.

Los efectos son permanentes tanto si el resultado es el de la modificación de los datos o el de volver al estado anterior a la transacción.

Podríamos ver una transacción como un proceso en fases:

1. Antes de que se inicie la transacción, la base de datos está en un estado coherente.

2. La aplicación indica el inicio de la transacción. Esto puede realizarse explícitamente con la instrucción BEGIN o START TRANSACTION.

3. La aplicación comienza a modificar los datos. Estas modificaciones no son permanentes de momento y no son visibles desde fuera. Dentro de la propia transacción y temporalmente la base de datos puede estar en un estado intermedio incoherente (mientras no se inserta el primer producto de la factura, tenemos una factura sin productos).

4. En este momento pueden suceder dos cosas:

A) Cuando la aplicación alcanza un punto donde todas las modificaciones se han completado correctamente y la base de datos es de nuevo coherente, la aplicación confirma la transacción. Esto hace que todas las modificaciones sean una parte permanente de la base de datos.

B) Si la aplicación encuentra un error que impide completar la transacción, deshace todas las modificaciones de datos. Esto devuelve a la base de datos al punto de coherencia en que se encontraba antes de que se iniciase la transacción.

Sintaxis de las sentencias asociadas a Transacciones en MySQL

```
BEGIN | START TRANSACTION   -- comienza una transacción

COMMIT    -- finaliza una transacción positivamente (las sentencias que se han ejecutado dentro de
-- la transacción se harán definitivamente)

ROLLBACK -- finaliza una transacción negativamente (las sentencias que se han ejecutado dentro
-- de la transacción se desharán definitivamente)
```

NOTA.- En general, las sentencias DDL (creación/modificación de la estructura de la BD) no se pueden deshacer, por tanto, no podremos utilizar transacciones con sentencias DDL .

Modo de trabajo respecto a las transacciones en MySQL Server

Aunque todas, o prácticamente todas, las herramientas de gestión de bases de datos implementan transacciones ACID, cada SGBD tiene un modo diferente de trabajar por defecto, por lo que una de las tareas del administrador es conocer y formarse en el modo transaccional del SGBD con el que trabaja. Incluso tendremos que aprender, según la herramienta sobre la que corran nuestras bases de datos, diferentes tipos de transacciones, cada uno de ellos se activará/desactivará mediante el uso de variables de sistema

Debemos recordar que en el caso de MySQL, para trabajar con transacciones, el motor de almacenamiento con el que trabajemos debe ser **InnoDB.**

El modo en el que trabaja MySQL por defecto es "**autocommit activado**", ésto es, la variable "autocommit" tiene por defecto el valor TRUE (1).

Podremos deshabilitar este modo de trabajo deshabilitando esta variable.

"autocommit" es una variable de sesión, lo que significa que si la deshabilitamos, volverá a estar habilitada cuando cerremos la sesión y volvamos a conectarnos o para cualquier otra sesión que habilitemos.

```
Set autocommit = 0; ==> deshabilitará el modo autocommit para la sesión actual.
```

Cuando tenemos el como autocommit activo, cada instrucción SQL que ejecutemos implica una transacción implícita.

Desactivar el modo autocommit implica que con la primera instrucción que ejecutemos en una conexión, comenzará una transacción que no se terminará hasta que hagamos un commit o rollback de forma explícita.

Si olvidamos este último paso y cerramos la conexión, cuando volvamos a comprobar nuestros datos, todo lo que hagamos hecho se habrá perdido.

Veamos con un ejemplo el uso de transacciones:

Por ejemplo, cada una de las siguientes instrucciones implican que se abra una transacción antes de su ejecución que acabará justo después de su ejecución:

Supongamos el siguiente bloque de código en el que intentamos insertar un empleado nuevo y para ello primero buscamos el último código de empleado:

```
select @maxemple = max(codem) from empleados;

insert into empleados
        (codem, nomem, ….)
values
        (@maxemple+1, 'pepe', ….);
```

Inicio transacción implicita 1

Fin transacción implicita 1

Inicio transacción implicita 2

Fin transacción implicita 2

Con este código, nadie nos garantiza que esto vaya a ejecutarse sin errores, por ejemplo, si justo entre la instrucción 1 y la 2 alguien, desde otra conexión, se nos adelantara e insertara un empleado, la segunda instrucción nos podría dar error, ya que el código del empleado “@maxemple + 1” podría ser el que ha insertado el otro usuario desde otra conexión (El error sería por clave duplicaca).

Para solucionar esto, tenemos dos posibilidades:

A) Mantener el modo autocommit activo y abrir una transacción explicita con BEGIN TRANSACTION

B) Deshabilitar el modo autocommit y hacer un commit justo al final.

Veamos cada opción:

A) En este ejemplo estaremos utilizando 3 transacciones, lo que supone recursos del sistema.

BEGIN/START TRANSACTION ← Inicio transacción explicita

- - usaremos begin dentro de rutinas y start fuera

```
    select @maxemple = max(codem) from empleados;
```

← Inicio transacción implicita 1

← Fin transacción implicita 1

← Inicio transacción implicita 2

```
    insert into empleados
        (codem, nomem, ….)
    values
        (@maxemple+1, 'pepe', ….);
```

← Fin transacción implicita 2

COMMIT; ← Fin transacción explicita

B) En este caso solo tendremos una transacción involucrada

```
set autocommit = 0;
    select @maxemple = max(codem) from empleados;
```

← Inicio transacción implicita 1

```
    insert into empleados
        (codem, nomem, ….)
    values
        (@maxemple+1, 'pepe', ….);
```

COMMIT; ← Fin transacción implicita 1

/* Si mantenemos el modo autocommit desactivado, con la siguiente instrucción, comenzará una nueva transacción que tendremos que terminar con un commit/rollback EXPLICITAMENTE */

/* Podriamos volver al modo autocommit activo con:

set autocommit = 1; */