Unidad 2 – Transacciones

Transacciones en MySQL

Hay situaciones en las que necesitamos hacer más de una consulta al mismo tiempo, y todas tienen que ser correctas para que los datos sean consistentes y tengan sentido, de otro modo tendríamos información “colgada” y desvinculada o una base de datos desbalanceada. En el mejor de los casos esta información provocaría un desperdicio de espacio en la base de datos, pero lo más probable es que además lleve a errores a la hora de mostrarse y de hacer cálculos con ella.

Parte de este problema se resuelve teniendo una buena estructura de base de datos, con claves foráneas relacionando los campos correspondientes.

Pero para mayor seguridad, es bueno usar lo que en jerga de bases de datos se conoce como transacciones.

Las transacciones sirven para asegurar la consistencia de la información, asegurando que un conjunto de sentencias se ejecuten correctamente, o no se ejecuten.

En un principio MySQL no las soportaba, esto cambió a partir de la versión 4, y con el uso de tablas InnoDB. Nótese que el motor por defecto para las tablas en MySQL hasta las versiones 5.x era MyISAM, que no soporta transacciones.

Un ejemplo:

Supongamos que un sitio web bancario tiene 2 usuarios, ambos trabajando sobre la misma cuenta.

El usuario 1 pide incrementar su saldo en 10, mientras que el usuario 2 pide disminuirlo (a través de un formulario, por ejemplo)

El programador del sistema no puede decidir el orden en el que se ejecutarán las consultas, así que bien podría suceder lo siguiente:

```
bal1 := ... SELECT balance FROM cuentas WHERE cuenta=X -- usuario 1
bal2 := ... SELECT balance FROM cuentas WHERE cuenta=X -- usuario 2
```

En este punto, existen dos copias de la aplicación que contienen una variable $balance cada una. Supongamos que ambas necesitan actualizar el valor en la base de datos:

```
UPDATE cuentas SET balance=(bal1+10) WHERE cuenta=X -- usuario 1
UPDATE cuentas SET balance=(bal2-10) WHERE cuenta=X -- usuario 2
```

El resultado es que ambas copias del programa ejecutaron sus consultas con la información de balance que tenían, por lo que el resultado final es como si la consulta del usuario 1 no se hubiera ejecutado nunca, ya que el usuario 2 actualiza el registro con información vieja. Al final, en vez de quedar con el mismo saldo, la cuenta termina perdiendo 10.

Lo que se necesita para este conjunto de consultas, es lo que se denomina ACID, un acrónimo inglés que quiere decir Atomicidad, Consistencia, Aislamiento y Durabilidad. Las transacciones son un conjunto de consultas que se ejecutan como si fuesen una. Y por esto, permiten asegurar la consistencia de los datos, ya que si en mitad del proceso una consulta falla, todos los cambios producidos por consultas anteriores pueden ser revertidos.

Como usar transacciones

Usar transacciones es muy simple: antes de ejecutar la primer consulta, se ejecuta una que solamente contiene BEGIN. Luego se ejecutan las consultas que deban ejecutarse. Si éstas resultan exitosas, se termina la transacción con COMMIT, lo cual provoca que los cambios hechos por las consultas anteriores sean permanentes. Si las consultas fallan en algún paso, se puede volver al estado anterior al comienzo de la transacción ejecutando ROLLBACK

Aunque los datos no sean realmente escritos en las tablas involucradas hasta ejecutar el COMMIT, las consultas devuelven lo mismo que si lo fueran, es decir, para saber si una consulta falló basta con ver el valor de retorno de mysql_query y para ver el número de filas afectadas sigue valiendo usar mysql_num_rows

Mientras la transacción está ejecutándose, los datos (en el caso de InnoDB las filas y en el caso de MyISAM las tablas) afectados quedan bloqueados, nadie puede acceder a ellos. Cualquier consulta que tenga que ver con los mismos datos será demorada hasta que la transacción termine. Esto implica que usar transacciones es un poco más lento que no usarlas, pero a la vez implica que los datos involucrados no pueden ser modificados por otra copia de la aplicación, y por lo tanto se evita la situación planteada al principio como ejemplo.

IMPORTANTE: Los insert ejecutados dentro de una BEGIN sin COMMIT solo pueden ser visualizados en la misma conexión y nunca desde otra conexión.

autocommit

MySQL tiene una variable de entorno llamada autocommit, que por defecto tiene el valor 1. Configurado de esta manera no se pueden usar transacciones, porque MySQL automáticamente hace un COMMIT luego de cada consulta.

Para usar transacciones entonces, hay que poner autocommit a 0 (desactivarlo).

Nota: si autocommit se pone a cualquier número N > 1, MySQL hace un COMMIT automático luego de N consultas.

Para cambiar el valor de autocommit, simplemente se usa: **SET autocommit = 0;**

ACID

En bases de datos se denomina ACID a un conjunto de características necesarias para que una serie de instrucciones puedan ser consideradas como una transacción. Así pues, si un sistema de gestión de bases de datos es ACID compliant quiere decir que el mismo cuenta con las funcionalidades necesarias para que sus transacciones tengan las características ACID.

En concreto ACID es un acrónimo de Atomicity, Consistency, Isolation and Durability: Atomicidad, Consistencia, Aislamiento y Durabilidad en español.

- **Atomicidad:**
  Es la propiedad que asegura que la operación se ha realizado o no, y por lo tanto ante un fallo del sistema no puede quedar a medias.
- **Consistencia:**
  Integridad. Es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto se ejecutan aquellas operaciones que no van a romper las reglas y directrices de integridad de la base de datos. La propiedad de consistencia sostiene que cualquier transacción llevará a la base de datos desde un estado válido a otro también válido.
- **Aislamiento:**
  Es la propiedad que asegura que una operación no puede afectar a otras. Esto asegura que la realización de dos transacciones sobre la misma información sean independientes y no generen ningún tipo de error.
- **Durabilidad:**
  Es la propiedad que asegura que una vez realizada la operación, ésta persistirá y no se podrá deshacer aunque falle el sistema.

Cumpliendo estos 4 requisitos un sistema gestor de bases de datos puede ser considerado ACID Compliant.

Puesta en práctica

Poner las características ACID en ejecución no es tan sencillo. El proceso de una transacción requiere a menudo un número de cambios pequeños al ser realizado, incluyendo la puesta al día de los índices que son utilizados en el sistema para acelerar búsquedas. Esta secuencia de operaciones puede fallar por un número de razones; por ejemplo, el sistema puede no tener ningún sitio disponible en sus accionamientos de disco, o puede haber sobrepasado su tiempo de CPU asignado.

ACID sugiere que la base de datos pueda realizar todas estas operaciones inmediatamente. De hecho esto es difícil de conseguir. Hay dos clases de técnicas populares: escribir a un registro antes de continuar y la paginación de la sombra. En ambos casos, los bloqueos se deben implantar antes que la información sea actualizada, y dependiendo de la técnica puesta en práctica, todos los datos se tienen que haber leído. En escribir a un registro antes de continuar, la atomicidad es garantizada asegurándose que toda la información esté escrita a un registro antes que se escriba a la base de datos. Eso permite que la base de datos vuelva a un estado anterior en caso de un desplome. En sombrear, las actualizaciones se aplican a una copia de la base de datos, y se activa

la nueva copia cuando la transacción sea confiable. La copia refiere a partes sin cambios de la vieja versión de la base de datos, en vez de ser un duplicado entero.

Esto significa que debe realizarse un bloqueo en cualquier momento antes de procesar datos en una base de datos, incluso en operaciones leídas. Mantener una gran cantidad de bloqueos da lugar a un aumento substancial indirecto de los procesos así como a una alteración de la concurrencia de ellos. Si el usuario A está procesando una transacción que ha leído una fila de los datos que el usuario B desea modificar, por ejemplo, el usuario B debe esperar hasta que el otro usuario acabe.

Una alternativa a la fijación es mantener copias separadas de cualquier dato que se modifique. Esto permite a usuarios leer datos sin adquirir ningún bloqueo. Usando de nuevo el ejemplo anterior, cuando la transacción del usuario consigue los datos que el usuario B ha modificado, la base de datos puede recuperar la versión exacta de los datos para que el usuario A comience su transacción. Esto asegura de que el usuario A consiga una vista constante de la base de datos aunque otros usuarios estén cambiando datos.

Es difícil garantizar características en un ambiente de red. Las conexiones de red pudieron fallar, o dos usuarios pudieron utilizar la misma parte de la base de datos al mismo tiempo.

El bifásico se aplica típicamente en transacciones distribuidas para asegurarse de que cada participante en la transacción conviene aceptar si se debe confiar en la transacción o no.

Se debe tener cuidado cuando trabajan transacciones en paralelo. La fijación bifásica se aplica típicamente para garantizar el aislamiento sin que esté completo.