Clase 14:Transacciones y Concurrencias en MySQL

Prof. Ania Cravero

Introducción

- Bases de Datos.

  Definición: Una base o banco de datos es un conjunto de datos que pertenecen al mismo contexto almacenados sistemáticamente para su posterior uso.

  En informática existen los sistemas gestores de bases de datos (SGBD), que permiten almacenar y posteriormente acceder a los datos de forma rápida y estructurada.

  Las aplicaciones más usuales son para la gestión de empresas e instituciones públicas. También son ampliamente utilizadas en entornos científicos con el objeto de almacenar la información experimental.

Transacciones en un SGBD.

- Una transacción en un sistema de gestión de bases de datos (SGBD), es un conjunto de órdenes que se ejecutan formando una unidad de trabajo, es decir, en forma indivisible o atómica.

- Un SGBD se dice transaccional si es capaz de mantener la integridad de los datos, haciendo que estas transacciones no puedan finalizar en un estado intermedio. Cuando por alguna causa el sistema debe cancelar la transacción, empieza a deshacer las órdenes ejecutadas hasta dejar la base de datos en su estado inicial (llamado punto de integridad), como si la orden de la transacción nunca se hubiese realizado

Transacciones en Bases de datos.

¿Cuál es la razón para que una base de datos maneje transacciones?

- Las transacciones se utilizan para garantizar la seguridad de los datos.
- Imaginemos que un cliente esté retirando fondos de una cuenta para ingresarlos a su vez en otra. Como los equipos pueden sufrir interrupciones de funcionamiento (por interrupción del suministro eléctrico, problemas de red, etc.), podría darse el caso de que se actualizara o agregara un registro determinado, pero no el otro.
- Para evitar estas situaciones, lo importante ante cualquier tipo de fallo es asegurar que, después de una actualización, la base de datos se queda en un estado consistente. Para conseguir esto se crean unidades de ejecución denominadas transacciones que pueden definirse como secuencias de operaciones que han de ejecutarse de forma atómica, es decir, o bien se realizan todas las operaciones que comprenden la transacción globalmente o bien no se realiza ninguna.

Concurrencia

- La concurrencia de datos se produce cuando mas de una transacción accede a los mismos grupos de datos en el mismo intervalo de tiempo; Cuando dos transacciones están interconectadas , lo que es muy común en un ambiente multiusuario, esto se conoce como *procesamiento concurrente*

- Los dos problemas básicos que pueden producirse por Concurrencia de Datos son:
  - Pérdida de operaciones: principalmente actualizaciones.
  - Inconsistencias: las reglas de integridad pueden llegar a no cumplirse por una mala gestión de la concurrencia.

Concurrencia

Control de concurrencia

- El control de transacciones concurrentes en una base de datos brinda un eficiente desempeño del Sistema de Base de Datos, puesto que permite controlar la ejecución de transacciones que operan en paralelo, accesando a información compartida y, por lo tanto, interfiriendo potencialmente unas con otras.
- El hecho de reservar un asiento en una avión mediante un sistema basado en aplicaciones web, cuando decenas de personas en el mundo pueden reservarlo también, nos da una idea de lo importante y crucial que es el control de concurrencia en un sistema de base de datos a mediana o gran escala.
- Otro ejemplo en el que podemos observar la incidencia del control de concurrencia es el siguiente: en una Base de Datos bancaria podría ocurrir que se paguen dos cheques en forma simultánea sobre una cuenta que no tiene saldo suficiente para cubrirlos en su totalidad, esto es posible evitarlo si se tiene un control de concurrencia.

BLOQUEO DE DATOS

Para garantizar la secuencialidad de una transacción, cuando una transacción accede a un grupo de datos, otra transacción **no** puede cambiar o modificar el elemento de dato. La forma para ejecutar lo dicho anteriormente, es que una transacción accede a un elemento de datos solo si esta posee un bloqueo sobre los datos.

Hay básicamente dos formas de bloquear un dato

- **Modo compartido**: si una transacción(t) obtiene un bloqueo compartido sobre un dato (d) , esta podrá solamente leer sobre d , y no escribir sobre d
- **Modo exclusivo**: si una transacción(t) obtiene un bloqueo exclusivo sobre d , esta podrá leer y escribir sobre d.

BLOQUEO DE DATOS

- Una transacción deberá solicitar el bloqueo mas apropiado para su concretación , dependiendo de la operación que se realizara sobre los datos.

- La transacción realizará la operación una vez se halla obtenido el bloqueo (del gestor de control de concurrencia) sobre los elementos de datos.

- Una transacción que se realice con un bloqueo compartido es compatible con otra (s) transacciones que posean un bloqueo compartido sobre el mismo elemento de datos; si se quisiera obtener un bloqueo exclusivo estando activado un bloqueo compartido , deberá esperar hasta que se liberen todos los bloqueos compartidos que están actuando en ese momento.

EJEMPLO

Ejemplo de transacción concurrente (Lectura inconsistente)

- En un banco una transacción 1 (t1) quiere trasladar 100 pesos de la cuenta A, a la cuenta B , la transacción 2 (t2) quiere visualizar la cantidad total de la cuentas A y B

- Si se realizara secuencialmente estas transacciones no se produciría ningún problema, pero al ejecutarlas concurrente mente habría problemas.

T1	T2	Gestor de control de concurrencia (DBM)
Bloqueo exclusivo (B) Leer (B) B = B – 100 Escribir (B) Desbloquear (B)		Concede Bloq. Exclusivo sobre B (T1)
	Bloqueo Compartido (A) Leer (A) Desbloquear (A) Bloquear compartido (B) Leer (B) Desbloquear (B) Visualizar (A+B)	Concede bloq. Compartido Sobre A (T2) Concede bloq. Compartido Sobre B (T2)
Bloqueo Exclusivo (A) Leer (A) A= A+100 Escribir (A) Desbloquear (A)		Concede Bloq. Exclusivo Sobre A (T1)

Ejemplo

- En el ejemplo anterior se produce una lectura inconsistente ya que la transacción 1, libera muy luego el bloqueo sobre A, por lo tanto la transacción t2 vio un estado inconsistente.

- Ahora si retrasara el bloqueo hasta el final de la transacción

- La transacción T1 tiene un bloqueo exclusivo sobre B,.en tanto T2 pide un bloqueo compartido sobre B , pero como no se le puede asignar ya que no son compatibles, T2 debe esperar hasta que T1 libere B;
- Mientras tanto T2 tiene un bloqueo compartido sobre A y T1 hace una petición de bloqueo exclusivo sobre A y como en el caso anterior no se le puede asignar ya que no son compatibles , por lo tanto debe esperar,
- Con lo cual se produce una situación donde ninguna de las dos transacciones puede seguir su ejecución.
- Lo descrito anteriormente se llama *interbloqueo,* para solucionar este problema el sistema debe abortar una de las 2 transacciones para liberarse de esta situación.

T1	T2
Bloqueo exclusivo (B) Leer (B) B=B-100 Escribir (B) Bloqueo exclusivo (A)	Bloqueo compartido (A) Leer (A) Bloqueo Compartido (B)

Asignación de Bloqueos

- Una Transacción puede obtener un bloqueo siempre y cuando no exista otra transacción con un bloqueo de modo conflictivo (no compatibles).

- Pero hay que tener en cuenta la siguiente situación:
  - Se esta ejecutando una transacción T1 en modo compartido sobre determinados elementos de datos y una transacción T2 solicita un bloqueo exclusivo sobre los mismo elemento de datos,
  - T2 debe esperar hasta que T1 libere el bloqueo,
  - Pero en ese instante la transacción T3, solicita un bloqueo compartido sobre los elementos de datos,
  - Como es en modo compartido esta es compatible con T1 , por lo tanto se concede el bloqueo,
  - T1 libera el bloqueo pero T2 debe seguir esperando a que libere el bloqueo T3
  - Otra transacción T4 solicita un bloqueo compartido sobre los mismos elementos de datos que T3 y como son compatibles se concede el bloqueo
  - Y T2 debe seguir esperando;
  - Es posible que hayan mas transacciones que soliciten un bloqueo compartido y cada una de ellas libere el bloqueo al poco tiempo de ser concedido de tal forma que T2 Nunca puede obtener un bloqueo exclusivo sobre los elemento de datos. En este caso se dice que T2 tiene una *inanición.*

Ejemplo

- Ejemplo de una transacción concurrente (Actualización Perdida)
- T1 ,Lee 10
- T2, Lee 10
- T1, Saca 5 (=5)
- T2, Saca 3 (=7) **?**

Uso de Bloqueos

- En el ejemplo se realiza una transacción concurrente donde provoca una inconsistencia en la base de datos

- La transacción 1 (T1) lee 10 unidades
- Seguidamente otra transacción (T2) lee 10 unidades
- Posteriormente T1 saca 5 unidades
- Y T2 saca 3 unidades
- Como consecuencia quedan 7 erróneas unidades registradas en la base de datos, siendo que lo verdadero sería que quedaran 2 unidades .

MySQL

MySQL es un sistema de gestión de base de datos, multihilo y multiusuario con más de seis millones de instalaciones. MySQL AB desarrolla MySQL como software libre en un esquema de licenciamiento dual. Por un lado lo ofrece bajo la GNU GPL, pero, empresas que quieran incorporarlo en productos privativos pueden comprar a la empresa una licencia que les permita ese uso. Está desarrollado en su mayor parte en ANSI C.

MySQL es muy utilizado en aplicaciones web. Su popularidad como aplicación web está muy ligada a PHP, que a menudo aparece en combinación con MySQL. MySQL es una base de datos muy rápida en la lectura cuando utiliza el motor no transaccional MyISAM, pero puede provocar problemas de integridad en entornos de alta concurrencia en la modificación. En aplicaciones web hay baja concurrencia en la modificación de datos y en cambio el entorno es intensivo en lectura de datos, lo que hace a MySQL ideal para este tipo de aplicaciones.

Transacciones con MySQL

- Desde **MySQL 4.0**, se incluye un motor de transacciones llamado **InnoDB** que ha sido desarrollado por MySQL Inc. El motor es opcional (aparece en las versiones llamadas Max y Pro). Así que para las aplicaciones que no lo necesitan (por ejemplo aplicaciones web), MySQL puede configurarse para no sufrir esta sobrecarga adicional.

- **InnoDB** es un nuevo formato de tabla que soporta transacciones y bloqueos a nivel de registro. InnoDB ofrece unos rendimientos superiores a MyISAM la anterior tecnología de tablas de MySQL, aunque si bien el mejor rendimiento de uno u otro formato dependerá de la aplicación específica. Su característica principal es que soporta transacciones de tipo ACID y bloqueo de registros e integridad referencial.

ACID test

- Se dice que toda Transacción debe cumplir con la prueba del ácido, denominada 'ACID' (*Atomicity, Consistency, Isolation, Durability*)

Atomicity

- La atomicidad de una transacción garantiza que todas sus acciones sean realizadas o ninguna sea ejecutada. En el caso de la transacción bancaria o se ejecuta toda la operación de la transferencia, o ninguna acción será realizada.

Consistency

- Muy similar a la Atomicidad, la consistencia garantiza que las reglas que hayan sido declaradas para una transacción sean cumplidas. Regresando a la transacción bancaria, supongamos que cada vez que se realice una transferencia interbancaria de X pesos, sea necesario notificar la operación a la sucursal que va a recibir el dinero. Si no es posible comunicarse y actualizar la información en la sucursal del cliente, toda la transacción será abortada.

Isolation

- Esto garantiza que las transacciones que se estén realizando en el sistema sean invisibles a todos los usuarios hasta que estas hayan sido declaradas finales. En la transacción bancaria se garantiza que los usuarios del sistema no observen estos cambios intermedios hasta que sea finalizada la última acción de actualización.

Durability

- La durabilidad de una transacción garantiza que al instante en el que se finaliza la transacción esta perdure a pesar de otras consecuencias. Esto es, si el disco duro falla, el sistema aún será capaz de recordar todas la transacciones que han sido realizadas en el sistema.

Transacciones con MySQL

- Las transacciones aportan una fiabilidad superior a las bases de datos. Si disponemos de una serie de consultas SQL que deben ejecutarse en conjunto, con el uso de transacciones podemos tener la certeza de que nunca nos quedaremos a medio camino de su ejecución.

- Para este fin, las tablas que soportan transacciones, como es el caso de InnoDB, son mucho más seguras y fáciles de recuperar si se produce algún fallo en el servidor, ya que las consultas se ejecutan o no en su totalidad. Por otra parte, las transacciones pueden hacer que las consultas tarden más tiempo en ejecutarse.

Transacciones con MySQL

- Para asegurarnos que tenemos soporte para el tipo de tablas InnoDB podemos ejecutar la siguiente sentencia:

- ```
  mysql> SHOW VARIABLES LIKE '%innodb%';
  +---------------------------------+------------+
  | Variable_name                   | Value      |
  +---------------------------------+------------+
  | have_innodb                     | YES        |
  | innodb_additional_mem_pool_size | 1048576    |
  | innodb_buffer_pool_size         | 8388608    |
  | innodb_data_file_path           | ibdata:30M |
  | innodb_data_home_dir            |            |
  | innodb_file_io_threads          | 4          |
  | innodb_force_recovery           | 0          |
  | innodb_thread_concurrency       | 8          |
  | innodb_flush_log_at_trx_commit  | 1          |
  | innodb_fast_shutdown            | ON         |
  | innodb_flush_method             |            |
  | innodb_lock_wait_timeout        | 50         |
  | innodb_log_arch_dir             | .\         |
  | innodb_log_archive              | OFF        |
  | innodb_log_buffer_size          | 1048576    |
  | innodb_log_file_size            | 5242880    |
  | innodb_log_files_in_group       | 2          |
  | innodb_log_group_home_dir       | .\         |
  | innodb_mirrored_log_groups      | 1          |
  | innodb_max_dirty_pages_pct      | 90         |
  +---------------------------------+------------+
  20 rows in set (0.00 sec)
  ```

- La variable más importante es por supuesto have_innodb que tiene el valor YES.
  ```

Transacciones con MySQL

Los pasos para usar transacciones en MySQL son:

- Iniciar una transacción con el uso de la sentencia BEGIN.
- Actualizar, insertar o eliminar registros en la base de datos.
- Si se quieren los cambios a la base de datos, completar la transacción con el uso de la sentencia COMMIT. Únicamente cuando se procesa un COMMIT los cambios hechos por las consultas serán permanentes.
- Si sucede algún problema, podemos hacer uso de la sentencia ROLLBACK para cancelar los cambios que han sido realizados por las consultas que han sido ejecutadas hasta el momento.

Creación de tablas innodb en MySQL

- Para crear una tabla InnoDB, procedemos con el código SQL estándar CREATE TABLE, pero debemos especificar que se trata de una tabla del tipo InnoDB (TYPE= InnoDB). Esto es aplicable a cualquier tipo de tabla, pero cuando no se especifica nada, MySQL supone que se trata de una tabla MyISAM.

```
mysql> CREATE TABLE innotest (campo INT NOT NULL PRIMARY KEY) TYPE = InnoDB;
Query OK, 0 rows affected (0.10 sec)

mysql> INSERT INTO innotest VALUES(1);
Query OK, 1 row affected (0.08 sec)

mysql> INSERT INTO innotest VALUES(2);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO innotest VALUES(3);
Query OK, 1 row affected (0.04 sec)

mysql> SELECT * FROM innotest;
+-------+
| campo |
+-------+
|     1 |
|     2 |
|     3 |
+-------+
3 rows in set (0.00 sec)
```

Transacciones con tablas innodb

```
mysql> BEGIN;
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO innotest VALUES(4);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM innotest;
+--------+
| campo  |
+--------+
|      1 |
|      2 |
|      3 |
|      4 |
+--------+
4 rows in set (0.00 sec)
```

Transacciones con tablas innodb

- Si en este momento ejecutamos un ROLLBACK, la transacción no será completada, y los cambios realizados sobre la tabla no tendrán efecto.

- ```
  mysql> ROLLBACK;
  Query OK, 0 rows affected (0.06 sec)

  mysql> SELECT * FROM innotest;
  +--------+
  | campo  |
  +--------+
  |      1 |
  |      2 |
  |      3 |
  +--------+
  3 rows in set (0.00 sec)
  ```
  ```

Transacciones con tablas innodb

- Ahora vamos a ver que sucede si perdemos la conexión al servidor antes de que la transacción sea completada.

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO innotest VALUES(4);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM innotest;
+-------+
| campo |
+-------+
|     1 |
|     2 |
|     3 |
|     4 |
+-------+
4 rows in set (0.00 sec)

mysql> EXIT;
Bye
```

Transacciones con tablas innodb

- Cuando obtengamos de nuevo la conexión, podemos verificar que el registro no se insertó, ya que la transacción no fue completada.

- ```
  Welcome to the MySQL monitor.  Commands end with ; or
  \g.
  Your MySQL connection id is 449 to server version:
  4.0.13

  Type 'help;' or '\h' for help. Type '\c' to clear the
  buffer.


  mysql> SELECT * FROM innotest;
  +--------+
  | campo |
  +--------+
  |      1 |
  |      2 |
  |      3 |
  +--------+
  3 rows in set (0.00 sec)
  ```
  ```

Transacciones con tablas innodb

- Ahora vamos a repetir la sentencia INSERT ejecutada anteriormente, pero haremos un COMMIT antes de perder la conexión al servidor al salir del monitor de MySQL.

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO innotest VALUES(4);
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.02 sec)

mysql> EXIT;
Bye
```

Una vez que hacemos un COMMIT, la transacción es completada, y todas las sentencias SQL que han sido ejecutadas previamente afectan de manera permanente a las tablas de la base de datos.

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 450 to server version: 4.0.13

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT * FROM innotest;
+--------+
| campo  |
+--------+
|      1 |
|      2 |
|      3 |
|      4 |
+--------+
4 rows in set (0.00 sec)
```

Ejemplo.

- Vamos a crear una sencilla tabla llamada ventas que sea del tipo InnoDB.

- ```
  mysql> CREATE TABLE ventas(
      -> id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
      -> producto VARCHAR(30) NOT NULL,
      -> cantidad TINYINT NOT NULL) TYPE = InnoDB;
  Query OK, 0 rows affected (0.96 sec) Insertamos un registro.
  ```

- ```
  mysql> INSERT INTO ventas VALUES(0,'Lápices',3);
  Query OK, 1 row affected (0.16 sec)

  mysql> SELECT * FROM ventas;
  +----+------------------+----------+
  | id | producto         | cantidad |
  +----+------------------+----------+
  |  1 | Lápices          | 3        |
  +----+------------------+----------+
  1 row in set (0.01 sec)
  ```

Ejemplo

- Ahora vamos a iniciar una transacción con la sentencia BEGIN.

- mysql> BEGIN;
  Query OK, 0 rows affected (0.00 sec) Actualizamos el registro.

- mysql> UPDATE ventas SET cantidad=4 WHERE id=1;
  Query OK, 1 row affected (0.07 sec)
  Líneas correspondientes: 1 Cambiadas: 1 Avisos: 0 Verificamos que los cambios han sucedido.

- mysql> SELECT * FROM ventas;

```
+----+------------------+----------+
| id | producto         | cantidad |
+----+------------------+----------+
|  1 | Lápices          | 4        |
+----+------------------+----------+
1 row in set (0.00 sec)
```

Ejemplo

- Si queremos deshacer los cambios, entonces ejecutamos un ROLLBACK.

- mysql> ROLLBACK;
  Query OK, 0 rows affected (0.06 sec) Verificamos que se deshicieron los cambios.

- mysql> SELECT * FROM ventas;

```
+----+--------------------+----------+
| id | producto           | cantidad |
+----+--------------------+----------+
|  1 | Lápices            | 3        |
+----+--------------------+----------+
1 row in set (0.00 sec)
```

Ejemplo

- Vamos a actualizar el registro usando otra transacción.

- ```
  mysql> BEGIN;
  Query OK, 0 rows affected (0.00 sec)

  mysql> UPDATE ventas SET cantidad=2 WHERE id=1;
  Query OK, 1 row affected (0.00 sec)
  Líneas correspondientes: 1  Cambiadas: 1  Avisos:
  0

  mysql> SELECT * FROM ventas;
  +----+--------------------+----------+
  | id | producto           | cantidad |
  +----+--------------------+----------+
  |  1 | Lápices            | 2        |
  +----+--------------------+----------+
  1 row in set (0.00 sec)
  ```
  ```

Ejemplo

- Vamos a confirmar que deseamos los cambios.

- mysql> COMMIT;
  Query OK, 0 rows affected (0.05 sec) En este momento los cambios son permanentes y definitivos.

- mysql> SELECT * FROM ventas;

```
+----+--------------------+----------+
| id | producto           | cantidad |
+----+--------------------+----------+
|  1 | Lapices            | 2        |
+----+--------------------+----------+
1 row in set (0.00 sec)
```