Bases de Datos

Unidad 7:

Edición avanzada de los datos. Concurrencia

Concurrencia

CONCURRENCIA: Acceso a los mismos datos desde dos o más sesiones de forma simultánea.

PROBLEMA DE LA CONCURRENCIA: En algunas sesiones se están actualizando datos dentro de transacciones y estos datos están siendo usados en otras sesiones. LOS DATOS MODIFICADOS EN UNA TRANSACCIÓN NO ESTÁN CONFIRMADOS.

Transacción A	Transacción B
Leer R ($250)	
Actualizar R= R-20 ($230)	
	Leer R ($230)
rollback	
	Actualizar R=R-30 ($200)
	commit

Ejemplo de situación que se puede producir y que debe tener una solución:

Supongamos que en una aplicación de reserva de pasajes para un vuelo existe un procedimiento que:

1. busca un asiento libre
2. lo marca como ocupado
3. asigna el asiento a la persona que realizó la petición

Es totalmente posible que al mismo tiempo dos personas ejecuten el procedimiento simultáneamente, busquen el mismo asiento libre y dejen la BD en un estado “indeseable o inconsistente”.

Ambos pasajeros quedan con el mismo asiento asignado, la BD queda en un estado indeseable.

Las modificaciones realizadas por una operación deben aislarse de las modificaciones llevadas a cabo por otras operaciones que se ejecuten al mismo tiempo. DEBE HABER ALGÚN NIVEL DE AISLAMIENTO

Problemas de la concurrencia cuando no hay aislamiento entre transacciones:

- **LOST UPDATE O ACTUALIZACIÓN PERDIDA**
- **DIRTY READ O LECTURA SUCIA**
- **PHANTOM READ O LECTURA FANTASMA**
- **NON REPEATABLE READ O LECTURA NO REPETIBLE**

- # LOST UPDATE O ACTUALIZACIÓN PERDIDA

Transacción A	Transacción B	Total_votos en A,B
SELECT total_votos FROM canciones WHERE numcancion=1;		5, ?
	SELECT total_votos FROM canciones WHERE numcancion=1;	5,5
UPDATE canciones SET total_votos=total_votos+1 WHERE numcancion=1;		6, 5
	UPDATE canciones SET total_votos=total_votos+1 WHERE numcancion=1;	6, 6 Al hacer UPDATE se sigue leyendo 5 votos ya que la transacción A no ha sido confirmada
	COMMIT	6, 6
COMMIT		6. 6

Se pierde una actualización de un dato en una transacción porque otra cambia el valor del mismo dato. Se podría producir cuando una transacción NO LEE los cambios hechos en los datos por otra transacción.

- **DIRTY READ O LECTURA SUCIA**

Transacción A	Transacción B	Precio en A y en B
Leer precio de coche 1234BCD		50€, ?
Incrementar precio de coche 1234BCD en un 10%		55€, ?
	Leer precio de coche 1234BCD	55€, 55€
	Obtener importe de un contrato de 2 días	55€, 55€ El importe es 110€
	Establecer en una columna el importe del contrato	55€, 55€ El importe es 110€
ROLLBACK		50€ Si se leyera en B también sería 50 €
	COMMIT	

Ocurre cuando **una transacción cambia un dato pero luego cancela el cambio y** antes de la cancelación **otra transacción ha leído el dato**. En B se ha leído un dato que estaba actualizándose pero dicha actualización ha sido posteriormente revertida.

- **PHANTOM READ O LECTURA FANTASMA**

Transacción A	Transacción B	
START TRANSACTION		
	START TRANSACTION	
	INSERT INTO votos (usuario,fecha, cancion) VALUES ('02luis', curdate(), 1)	1 voto más para canción 1
SELECT count(*) FROM votos WHERE cancion=1		
UPDATE canciones SET totalvotos = (SELECT count(*) FROM votos WHERE cancion=1)		Se supone que cuenta también el voto insertado en B
	ROLLBACK;	Se anula el voto pero se ha contado ya. Fue un voto fantasma

Una transacción lee un dato que deja de existir porque lo había introducido otra que no se confirma. En el ejemplo se ha leído una fila que no existía al iniciarse la transacción y, como se hace un rollback esa fila deja de existir de nuevo.

- **NON REPEATABLE READ O LECTURA NO REPETIBLE**

Transacción A	Transacción B	
START TRANSACTION		
SELECT total_votos FROM canciones WHERE numcancion=1		
	START TRANSACTION	
	UPDATE canciones SET total_votos = total_votos+1 WHERE numcancion=1	Dato se ha modificado
	COMMIT	
SELECT numcancion,total_votos FROM canciones;		Se lee algo distinto para la canción 1
…….		

Sucede cuando **dos o más operaciones de lectura en la misma transacción leen valores diferentes de un dato** porque ha sido modificado por otra. En el ejemplo se lee total_votos dos veces y la segunda ha sido modificado.

Lecturas Coherentes

De forma predeterminada en **MySQL**, sobre las tablas de tipo **innodb** se realizan **lecturas coherentes.**

Esto significa que al realizar una **SELECT** se devuelven los datos existentes en las tablas hasta la última transacción que se haya completado.

Así, si al hacer una consulta, en otra o en otras sesiones se tienen en curso transacciones que han modificado los datos de la tabla o de las tablas objeto de la consulta**, los datos modificados no se reflejarán en el resultado, con una excepción:**

- **Si la SELECT se incluye en la transacción que ha modificado los datos, SI QUE SE MUESTRAN las modificaciones en el resultado.**

Además, si en una transacción se repite la lectura de un dato que ha sido modificado en otra transacción, en la segunda lectura no se ven los cambios, se repite lo leído en la primera lectura.

Lecturas Coherentes

Ejemplo: *Tenemos dos sesiones clientes MySQL abiertas a las que desde ahora nos referiremos a ellas por sesión 1 y sesión 2. En ambas sesiones se tiene en uso la base de datos* ***concursomusica.*** *De esta base de datos, utilizaremos las tablas* ***canciones y votos.***

Vamos primero a obtener el contenido de la tabla votos y el contenido de la tabla canciones

SELECT * FROM votos;

usuario	fecha	cancion
AnaArias01	2017-03-15	66
03ez	2017-02-07	67
03lemon	2017-02-07	67
03pz	2017-02-07	67
03squeezy	2017-02-07	67
NULL	NULL	NULL

SELECT * FROM canciones;

numCancion	grupo	duracion	titulo	total_votos
1	1	00:03:56	Cancion de cuna	4
2	1	00:02:57	Cientos y cientos	4
3	1	00:02:28	Un sueño	3
4	1	00:03:32	Jovenes	3
5	2	00:02:44	No gires	5
6	2	00:03:42	Calor	0

Lecturas Coherentes

Ahora en sesión 1:

- *Iniciamos una transacción.*
- *Registramos un nuevo voto dado por **02Ana** a la canción número 1.*
- *Incrementamos el total de votos de la canción número 1,*

```
start transaction;
insert into votos (usuario,fecha,cancion) values ('02Ana',curdate(),1);
update canciones set total_votos=total_votos+1 where numCancion=1;
```

En sesión 1 veremos los cambios hechos, aunque no hayamos confirmado la transacción

usuario	fecha	cancion
03pz	2017-03-15	25
24Mary	2017-03-15	30
AnaArias01	2017-03-15	66
02Ana	2017-03-16	2
02Ana	2017-04-03	1

numCancion	grupo	duracion	titulo	total_votos
1	1	00:03:56	Cancion de cuna	5
2	1	00:02:57	Cientos y cientos	4
3	1	00:02:28	Un sueño	3
4	1	00:03:32	Jovenes	3
5	2	00:02:44	No gires	5

Lecturas Coherentes

Ahora, sin haber confirmado aún la transacción, el usuario de la sesión 2 consulta cuantos votos tiene la canción número 1 y los datos de los votos dados a esa canción.

select * from canciones where numcancion=1;

numCancion	grupo	duracion	titulo	total_votos
1	1	00:03:56	Cancion de cuna	4

No se ve el cambio hecho en la sesión 1!

select * from votos where cancion=1;

usuario	fecha	cancion
11Eustaquio	2017-01-13	1
17Cantua	2017-01-02	1
26Fer	2017-01-01	1
27Daniel	2017-01-08	1

No se ve el voto dado por 02Ana!

Esto que ocurre por defecto, es lo más lógico dado que la transacción no se ha confirmado. Si ahora confirmamos con COMMIT la transacción en sesión 1, veremos en sesión 2 los cambios que se realizaron en esa transacción.

Lecturas Coherentes

Ejemplo 2: *Ahora vamos a ver lo que ocurre cuando en la sesión 1 iniciamos una transacción y modificamos una tabla y, sin confirmar la transacción, intentamos modificar la misma tabla.*

En sesión 1 iniciamos una transacción, incrementamos el total_votos de la canción 2 y obtenemos cuantos votos tiene la canción 2:

```
start transaction;
update canciones set total_votos=total_votos+1 where numCancion=2;
SELECT * FROM canciones WHERE numCancion=2;
```

numCancion	grupo	duracion	titulo	total_votos
2	1	00:02:57	Cientos y cientos	5

Lecturas Coherentes

Ahora en sesión 2 intentamos incrementar en 1 el total de los votos de la canción número 24.

update canciones set total_votos=total_votos+1 where numCancion=24;

Y no se produce ningún problema, se modifican los votos de esa canción,

Sin embargo, si en la sesión 2 intentamos incrementar en 1 el total de votos de la canción 2:

update canciones set total_votos=total_votos+1 where numCancion=2;

La instrucción queda en estado running, en espera de ejecución. No se ejecutará hasta que se haya confirmado la transacción en la sesión 1 con COMMIT o se haya anulado con ROLLBACK. Esto ha sido así debido a que se está intentando modificar un dato que está modificado y no confirmado en otra transacción. La otra transacción ha bloqueado el acceso a los datos que ha ido modificando.

Bloqueos de lectura en modo compartido

En una consulta con **bloqueo de lectura en modo compartido**, no se obtiene un resultado si en una transacción de otra sesión se tienen modificados datos usados en la consulta. Se obtiene el resultado cuando haya finalizado la transacción.

Ejemplo: *En la sesión 1 iniciamos una transacción y añadimos un nuevo registro en la tabla votos para indicar que* ***20Luis*** *ha votado la canción 3.*

```
insert into votos (usuario,fecha,cancion) values ('20Luis',curdate(),3);
```

Ahora en sesión 2 realizamos una consulta con ***bloqueo de lectura en modo compartido*** *para contar cuántos votos tiene la canción 3 en la tabla votos:*

```
select count(*) from votos where cancion=3 lock in share mode;
```

No obtendrá un resultado hasta que se complete la transacción. Cuando finaliza la transacción, se completará la consulta.

Bloqueos de tablas

Usando tablas de tipo InnoDB se producen, como hemos visto, bloqueos a nivel de **registro** o a nivel de fila.

MySQL dispone de otro tipo de bloqueo, **el bloqueo de tablas**, disponible en todos los tipos de tabla incluyendo MyISAM y, por supuesto, también en las InnoDB. Existen dos tipos de bloqueo:

- **De lectura:** la sesión que tiene bloqueada la tabla puede hacer lecturas pero no escrituras. Varias sesiones pueden ejercer este bloqueo sobre una misma tabla a la vez. Otras sesiones que no lo tienen bloqueado pueden leer también.

- **De escritura:** la sesión que tiene bloqueada la tabla puede leer y escribir. El resto de sesiones no pueden leer ni escribir.

La sintaxis para bloquear una tabla es:

LOCK TABLE nombretabla READ|WRITE

Ejemplo 1 de Bloqueo de tablas:

En la sesión 1 se hace un bloqueo de lectura en la tabla votos y se añade un voto a esa tabla:

```
LOCK TABLE votos READ;
insert into votos (usuario,fecha,cancion) values ('27Daniel',curdate(),5);
```

No se puede insertar ni siquiera en la sesión en que se ha hecho el bloqueo. Se produce el error:

Error Code: 1099. Table 'votos' was locked with a READ lock and can't be updated

Pero sí que podemos hacer, tanto en la sesión 1 como en otras sesiones simultáneas:

```
select cancion,count(*) from votos group by cancion;
```

Ejemplo 1 de Bloqueo de tablas (continuación):

Si en la sesión 2, intentamos realizar una actualización sobre la tabla votos, por ejemplo:

update votos set cancion=7 where cancion=3;

La instrucción queda en espera de ejecución. No se finaliza su ejecución hasta que se desbloquee la tabla.

Para poder volver a hacer actualizaciones en la tabla votos, habrá que desbloquearla:

UNLOCK TABLES;

Cuando se crea un bloqueo para acceder a una tabla, dentro de la zona de bloqueo no podremos acceder a otras tablas hasta que no se finalice el bloqueo:
LOCK TABLE votos READ;
*SELECT * FROM votos;*
*SELECT * FROM canciones;* ***Esta línea nos dará un error***

Ejemplo 2 de Bloqueo de tablas:

En la sesión 1 hace un bloqueo de escritura en tabla votos y se añade un voto a esa tabla:

```
LOCK TABLE votos WRITE;
insert into votos (usuario,fecha,cancion) values ('16Victor',curdate(),5);
```

Se permite realizar la actualización dentro de la sesión en que se ha producido el bloqueo.

Obtenemos los datos de todos los votos:

```
select * from votos;
```

Se permite realizar la consulta dentro de la sesión en que se ha producido el bloqueo.

Ejemplo 2 de Bloqueo de tablas (continuación):

En la sesión 1 se hizo un bloqueo de escritura en tabla votos.

LOCK TABLE votos WRITE;

Si realizamos en sesión 2 la consulta:

SELECT * FROM votos;

La instrucción queda en espera de ejecución. No se finaliza su ejecución hasta que se desbloquee la tabla. Lo mismo ocurriría con una instrucción de actualización.

Niveles de aislamiento

Dependiendo del nivel de concurrencia que se desee es posible solicitar al SGBD 4 niveles de aislamiento.

•Un nivel de aislamiento define cómo los cambios hechos por una transacción son visibles a otras transacciones.

Los **niveles de aislamiento** que nos ofrece MySQL son los siguientes:

- **SERIALIZABLE**
- **REPEATABLE READ (Lectura Repetida) En MysQl las tablas InnoBD tienen por defecto este nivel.**
- **READ COMMITED (Lectura acometida)**
- **READ UNCOMMITTED(Lectura no acometida)**

Niveles de aislamiento

SERIALIZABLE: Cada transacción ocurre de forma totalmente aisladas de otras transacciones. Nunca hay concurrencia.

REPEATEABLE READ (Lectura repetible). Consiste en que si un registro leído con una SELECT en una transacción, se modifica y confirma en otra transacción, una nueva lectura en la primera transacción repite el resultado que se generó inicialmente.

READ COMMITED (Lectura acometida): Los datos leídos por una transacción pueden ser modificados por otras transacciones. Una transacción no leerá datos que no han sido confirmados por el **commit** de otra transacción.

READ UNCOMMITED (Lectura no acometida): Las sentencias SELECT son efectuadas sin realizar bloqueos, por tanto, todos los cambios hechos por una transacción pueden verlos las otras transacciones.

¿Cómo cambiar el nivel de aislamiento?

```
SET TRANSACTION ISOLATION LEVEL
{READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE}
```