Problemas de Concurrencia

Problema "Actualización Perdida"

Inconsistencia de datos que se produce en la ejecución concurrente de transacciones, donde el orden de las lecturas y escrituras en las distintas transacciones producen la inconsistencia de datos.

Si se necesita actualizar el stock de un producto, el procedimiento P0 suma las compras para el producto, luego incrementa el stock del producto y escribe el stock, y el procedimiento P1 suma las ventas del producto, luego decrementa el stock del producto y escribe el stock, si dichas operaciones se realizan en el orden siguiente se produce la inconsistencia del stock.

Orden	P0 (SumarStock)	P1 (RestarStock)	
1	Leer(Compras)		
2	Leer (Stock)		
3	newStock = cantidadAux + totalcompras		
4		Leer(Ventas)	
5		Leer(Stock)	La Lectura del Stock en P1 se hace antes de que P0 tenga la oportunidad de actualizar el Stock.
6		newStock = cantidadAux - totalventas	
7	Escribir(Stock)		El orden de ejecución de las escrituras es indistinto siempre producen inconsistencia del Stock. La lectura del Stock de P1, debería realizarse luego de la escritura del Stock de P0.
8		Escribir(Stock)	

Otra Posibilidad de inconsistencia de datos.

P0 (SumarStock)	P1 (RestarStock)	
	Leer(Ventas)	
	Leer(Stock)	
	newStock = cantidadAux - totalventas	
Leer(Compras)		
Leer (Stock)		La lectura del Stock en P0 se hace antes de que P1 tenga la oportunidad de actualizar el Stock.
newStock = cantidadAux + totalcompras		
	Escribir(Stock)	El orden de ejecución de las escrituras es indistinto siempre producen inconsistencia del Stock. La lectura del Stock de P0, debería realizarse luego de la escritura del Stock de P1.
Escribir(Stock)		

Prueba del ejemplo “Actualización Perdida”

Los siguientes ejemplos fueron realizados en un motor de BD MySql

Tenemos una BD llamada “ProblemasDeConcurrencia”

```
create database ProblmasDeConcurrencia;
```

y las tablas llamadas stock, compra y venta.

```
create table stock
(
   producto int not null primary key,
   cantidad int
);

create table compra
(
   producto int not null,
   cantidad int
);

create table venta
(
   producto int not null,
   cantidad int
);

/* Insertar datos de prueba en las tres tablas */
```

También tenemos dos procedimientos , el primero de ellos suma la cantidad comprada de un producto, busca el stock del producto, calcula el nuevo stock y lo guarda en una variable, luego modifica el stock y termina.

```
delimiter //
CREATE PROCEDURE SumarStock(IN producto int, OUT newStock int)
      LANGUAGE SQL
      NOT DETERMINISTIC
      CONTAINS SQL
      SQL SECURITY DEFINER
      COMMENT ''
BEGIN
      declare totalcompras int;
      declare cantidadAux int;
      SELECT sum(cantidad) INTO totalcompras FROM compra
            WHERE compra.producto = producto;
      SELECT cantidad INTO cantidadAux FROM stock
            WHERE stock.producto = producto;
      set newStock = cantidadAux + totalcompras ;
      select sleep(10) ;
      update stock set cantidad = newStock WHERE stock.producto = producto ;
END //
delimiter ;
```

El procedimiento para sumar al stock del producto 1, se llama a ejecución de la siguiente forma:

```
call SumarStock(1,@newStock);
```

Cuando el procedimiento termina, para visualizar el contenido de la variable newStock escribimos

```
select @newStock;
```

El segundo procedimiento suma la cantidad vendida de un producto, busca el stock del producto, calcula el nuevo stock y lo guarda en una variable, luego modifica el stock y termina.

```
delimiter //
CREATE PROCEDURE RestarStock(IN producto int, OUT newStock int)
     LANGUAGE SQL
     NOT DETERMINISTIC
     CONTAINS SQL
     SQL SECURITY DEFINER
     COMMENT ''
BEGIN
     declare totalventas int;
     declare cantidadAux int;
     SELECT sum(cantidad) INTO totalventas FROM venta
         WHERE venta.producto = producto;
     SELECT cantidad INTO cantidadAux FROM stock
         WHERE stock.producto = producto;
     set newStock = cantidadAux - totalventas ;
     select sleep(10) ;
     update stock set cantidad = newStock WHERE stock.producto = producto ;
END //
delimiter ;
```

El procedimiento para restar al stock del producto 1, se llama a ejecución de la siguiente forma:

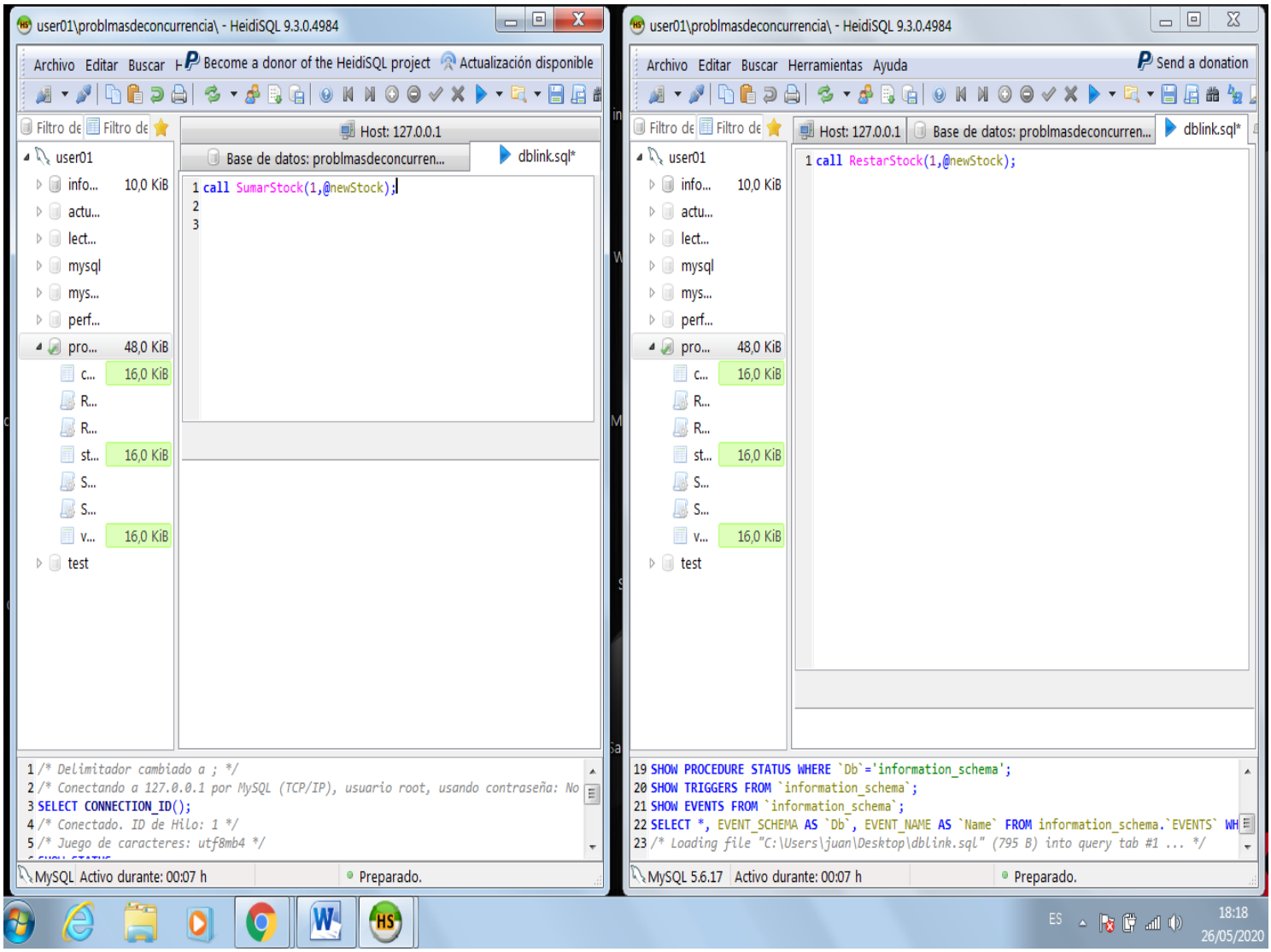
```
call RestarStock(1,@newStock);
```

Para visualizar el contenido de la variable newStock escribimos

```
select @newStock;
```

Para probar el ejemplo se deben abrir dos sesiones del usuario de MySql conectados a la misma base de datos ProblemasDeConcurrencia, en una sesión se ejecuta el procedimiento `SumarStock` y en la otra sesión se ejecuta el procedimiento `RestarStock`.

Como muestra la siguiente imagen.



Se puede lanzar a ejecución primero, a cualquiera de los dos procedimientos, el orden de comienzo en este caso es indistinto, pero siempre el segundo procedimiento que se lance a ejecución deberá hacerse antes de que expiren los 10 segundos del sleep que tienen en su código ambos procedimientos, como el sleep está ubicado en la misma secuencia en ambos procedimientos, puede comenzar la actualización cualquiera de ellos.

Problema “Lectura Sucia”

Inconsistencia de datos que se produce en la ejecución concurrente de transacciones, cuando una transacción aborta luego de realizar una actualización con una escritura, y otra transacción ya realizo la lectura de la actualización antes que la otra transacción abortara, de esta manera el dato leído por la transacción deja de ser válido, así se produce la actualización inconsistente.

Si se necesita actualizar el stock de un producto, la transacción T0 suma las compras para el producto, luego la suma al stock del producto y escribe la actualización del stock, pero luego por algún motivo, la transacción T0 aborta, y si la transacción T1 leyó el stock actualizado antes que la transacción T0 abortara y actualiza el stock con la suma de las ventas, entonces dichas operaciones producen la inconsistencia del stock, ya que el valor leído por la transacción T1 deja se ser válido.

T0 (SumarStock)	T1 (RestarStock)	
Leer(Compras)		
Leer (Stock)		
newStock = cantidadAux + totalcompras		
Escribir(Stock)		
	Leer(Ventas)	
	Leer(Stock)	
	newStock = cantidadAux - totalventas	T1 va a realizar sus cálculos con un valor de stock que no es válido porque T0 aborta
	Escribir(Stock)	
Aborta		

Para probar el problema de la lectura sucia, escribimos dos procedimientos.

El primer procedimiento se llama SumaStockDirty, y será el procedimiento que aborta, nosotros vamos a simular la falla del procedimiento, escribiendo y forzando un rollback.

```
delimiter //
CREATE PROCEDURE SumarStockDirty(IN producto int, OUT newStock int)
LANGUAGE SQL
NOT DETERMINISTIC
CONTAINS SQL
SQL SECURITY DEFINER
COMMENT ''
BEGIN
start transaction;
set @totalcompras = 0 ;
set @cantidadAux  = 0 ;
SELECT sum(cantidad) INTO @totalcompras FROM compra
WHERE compra.producto = producto;
if (@totalcompras) <> NULL
then begin
SELECT cantidad INTO @cantidadAux FROM stock
WHERE stock.producto = producto;
set newStock = @cantidadAux + @totalcompras ;
update stock set cantidad = newStock WHERE stock.producto = producto;
select sleep(10) ;
```

```
update compra set cantidad = 0 where compra.producto = producto ;
end;
end if;
rollback;
END //
delimiter ;
```

La forma de llamar a ejecución al procedimiento `SumarStockDirty` es la siguiente:

```
call SumarStockDirty(1,@newStock);
```

El segundo procedimiento es el que realiza la lectura sucia y se llama `RestarStockDirty`

```
delimiter //
CREATE PROCEDURE RestarStockDirty(IN producto int, OUT newStock int)
LANGUAGE SQL
NOT DETERMINISTIC
CONTAINS SQL
SQL SECURITY DEFINER
COMMENT ''
BEGIN
start transaction;
set @totalventas = 0 ;
set @cantidadAux = 0 ;
select sleep(3) ;
SELECT sum(cantidad) INTO @totalventas FROM venta
WHERE venta.producto = producto;
if ( @totalventas <> NULL )
then begin
SELECT cantidad INTO @cantidadAux FROM stock
WHERE stock.producto = producto;
set newStock = @cantidadAux - @totalventas ;
update stock set cantidad = newStock WHERE stock.producto = producto ;
end;
end if;
commit;
END //
delimiter ;
```

La forma de llamar a ejecución al procedimiento `RestarStockDirty` es la siguiente:

```
call RestarStockDirty(1,@newStock);
```

Prueba del ejemplo “Lectura Sucia”

En este caso el orden de comienzo de las transacciones es importante para mostrar en el ejemplo la lectura sucia, primero hay que poner en ejecución al procedimiento `SumarStockDirty` y dentro de los 10 segundos a partir del comienzo hay que poner en ejecución al procedimiento `RestarStockDirty.`
Para lograr la lectura sucia usamos un sleep(10) en `SumarStockDirty` y un sleep(3) en `SumarStockDirty`, de esta manera si primero se ejecuta SumarStockDirty y antes que expiren los 10 segundos se lanza a ejecución `RestarStockDirty`, entonces `RestarStockDirty` realiza una lectura sucia del stock y el stock queda inconsistente.

Y al igual que en el ejemplo anterior del problema actualización perdida, hay que abrir dos sesiones del mismo usuario conectados a la misma base de datos ProblemasDeConcurrencia.