Manejo de conectores (4ª parte)

1. Transacciones. Introducción

Uno de los objetivos de un SGBD es que múltiples usuarios puedan acceder a la base de datos de forma simultánea sin que el trabajo de uno interfiera en el trabajo del otro.

Un problema que puede aparecer cuando dos usuarios acceden a la misma base de datos es que quieran acceder a los mismos datos y que al menos uno quiera actualizar esos datos. Si esto no se controla de forma idónea podrían aparecer problemas de inconsistencia de la base de datos, por eso los SGBD suelen incorporar sistemas de bloqueos de registros para limitar este tipo de errores.

Por ejemplo, supongamos que tenemos una base de datos para llevar el control de reservas de plazas de avión.

Cuando un usuario pide reservar una plaza en un determinado vuelo, el sistema tiene que comprobar que queden plazas libres, si quedan plazas reservará la que quiera el usuario generando un nuevo billete y marcando la plaza como ocupada.

Aquí tenemos un proceso que consta de una lectura y dos operaciones de actualización de la base de datos:

- Leer plazas libres
- Crear una nueva fila en la tabla de billetes
- Actualizar la plaza reservada en el vuelo, poniéndola como ocupada

Estas operaciones se tienen que ejecutar o todas o ninguna, si después de crear el billete y antes de actualizar la plaza, otro usuario consulta las plazas libres podría reservar la misma plaza o si por la razón que sea no se actualiza la plaza por ejemplo porque el programa no finaliza correctamente, la base de datos quedaría en un estado inconsistente ya que la plaza constaría como libre cuando realmente habría un billete emitido para esta plaza, o dos billetes para la misma plaza.

Las razones por las que la plaza no se pueda actualizar pueden ser varias, unas ocasionadas por problemas de concurrencia (por ej. el propio sistema aborta nuestro proceso por haberse producido un bloqueo mortal entre nuestro proceso y otro), y otras por cuestiones ajenas a la concurrencia (por ejemplo que ocurra un error en el programa que haga que termine su ejecución a mitad, que se caiga el sistema, etc.).

Para evitar este problema de inconsistencia disponemos de un mecanismo que permite indicar que un grupo de operaciones se tienen que ejecutar o todas o ninguna, además de determinar un tipo de bloqueo de los registros, el mecanismo de transacciones.

Una transacción se puede definir como un conjunto de acciones que se tienen que realizar todas o ninguna para preservar la integridad de la base de datos.

En nuestro ejemplo se incluirían las tres operaciones en la misma transacción así el sistema sabría que las tiene que ejecutar las tres, si por lo que sea no se puede ejecutar alguna, se encarga de deshacer los cambios que se hubiesen producido para no ejecutar ninguna. Además se podría indicar que mientras el primer usuario no termina la transacción, otro usuario no puede consultar las plazas libres del mismo avión.

Podemos utilizar las transacciones también para realizar pruebas sobre una base de datos, pruebas que no queremos que queden efectivas. Podemos incluir las operaciones de actualización dentro de una transacción, incluir en esa misma transacción las operaciones para comprobar que la actualización funciona, y después deshacer la transacción con lo cual la base de datos no quedaría actualizada.

2. Propiedades ACID

Se dice que una transacción es fiable si cumple las propiedades ACID.

ACID es acrónimo de Atomicity, Consistency, Isolation, Durability , que representan las características de Atomicidad, Consistencia, Aislamiento y Durabilidad.

Atomicidad. Una transacción es considerada como una única operación sobre los datos, si se compone de varias operaciones se deberán ejecutar todas o ninguna.

Consistencia. Debe preservar la integridad de la base de datos.

Aislamiento. Las transacciones no deben interferir entre sí.

Durabilidad. Una vez que se confirma una transacción, se garantiza que sus efectos persistan incluso en el caso de fallos posteriores del sistema.

3. Niveles de aislamiento

Asociado a una transacción está su nivel de aislamiento que define el grado en que se debe aislar la transacción de las modificaciones de recursos o datos realizadas por otras transacciones.

El acceso simultáneo a la misma información puede generar efectos secundarios no deseables:

- **Lecturas de datos sucios**, lectura de datos modificados por otra transacción pero no confirmados.

- **Lecturas no repetibles**, "relectura" de filas que han sido modificadas o eliminadas por otra transacción. Filas que no se pueden volver a leer.

- **Lecturas fantasma**, la transacción vuelve a ejecutar una consulta que devuelve un conjunto de filas que responden a una determinada condición y encuentra que otra transacción completada ha insertado nuevas filas que responden también a la condición. Filas que antes no estaban y ahora están.

Los niveles de aislamiento de las transacciones controlan:

- Si se realizan bloqueos cuando se leen los datos y qué tipos de bloqueos se solicitan.

- Duración de los bloqueos de lectura.

- Si una operación de lectura que hace referencia a filas modificadas por otra transacción:

- Se bloquea hasta que se libera el bloqueo exclusivo de la fila.

- Recupera la versión confirmada de la fila que existía en el momento en el que se inició la instrucción o la transacción.

- Lee la modificación de los datos no confirmada.

Los niveles de aislamiento según el estándar ANSI SQL-92 están basados en qué efectos secundarios permiten:

Permite Nivel aislamiento\	Datos sucios	Lecturas no repetibles	Lecturas fantasma
Read Uncommitted	SI	SI	SI
Read Committed	NO	SI	SI
Repeatable Read	NO	NO	SI
Serializable	NO	NO	NO

Como se puede apreciar, los niveles aparecen en la tabla ordenados de menos restrictivo (Read Uncommitted) a más restrictivo (Serializable).

También podemos deducir qué tipos de bloqueos se imponen en cada nivel:

Nivel de aislamiento	Descripción
ReadUncommitted	Las instrucciones pueden leer filas que han sido modificadas por otras transacciones y que todavía no se han confirmado. Las transacciones que se ejecutan en el nivel READ UNCOMMITTED no emiten bloqueos compartidos para impedir que otras transacciones modifiquen los datos leídos por la transacción actual. Las transacciones READ UNCOMMITTED tampoco se bloquean mediante bloqueos exclusivos que impedirían que la transacción actual leyese las filas modificadas pero no confirmadas por otras transacciones. Cuando se establece esta opción, es posible leer las modificaciones no confirmadas, lecturas de datos sucios. Los valores de los datos se pueden cambiar, y las filas pueden aparecer o desaparecer en el conjunto de datos antes de que finalice la transacción. Se trata del nivel de aislamiento menos restrictivo.
ReadCommitted	Especifica que las instrucciones no pueden leer datos que hayan sido modificados y no confirmados, por otras transacciones. Esto evita las lecturas de datos sucios. Otras transacciones pueden cambiar datos entre cada una de las instrucciones de la transacción actual, dando como resultado lecturas no repetibles o datos fantasma.
RepeatableRead	Especifica que las instrucciones no pueden leer datos que han sido modificados pero aún no confirmados por otras transacciones y que ninguna otra transacción puede modificar o eliminar los datos leídos por la transacción actual hasta que ésta finalice. Se aplican bloqueos compartidos a todos los datos leídos por cada instrucción de la transacción, y se mantienen hasta que la transacción finaliza. De esta forma, se evita que otras transacciones modifiquen las filas que han sido leídas por la transacción actual. Otras transacciones pueden insertar filas nuevas que coincidan con las condiciones de búsqueda de las instrucciones emitidas por la transacción actual. Si la transacción actual vuelve a ejecutar la instrucción, recuperará las filas nuevas, dando como resultado lecturas fantasma. Debido a que los bloqueos compartidos se mantienen hasta el final de la transacción en lugar de liberarse al final de cada instrucción, la simultaneidad es inferior que en el nivel de aislamiento predeterminado READCOMMITTED. Se recomienda utilizar esta opción solamente cuando sea necesario.
Serializable	Las instrucciones no pueden leer datos que hayan sido modificados, pero aún no confirmados, por otras transacciones. No aparecerán pues datos sucios. Ninguna otra transacción puede modificar o eliminar los datos leídos por la transacción actual hasta que la transacción actual finalice. Se evita pues lecturas no repetibles. Otras transacciones no pueden insertar filas nuevas con valores de clave que pudieran estar incluidos en el intervalo de claves leído por las instrucciones de la transacción actual hasta que ésta finalice. Se evita pues lecturas fantasmas. Se colocan bloqueos de intervalo en el intervalo de valores de clave que coincidan con las condiciones de búsqueda de cada instrucción ejecutada en la transacción. De esta manera, se impide que otras transacciones actualicen o inserten filas que satisfagan los requisitos de alguna de las instrucciones ejecutadas por la transacción actual. Esto significa que, si alguna de las instrucciones de una transacción se ejecuta por segunda vez, leerá el mismo conjunto de filas. Los bloqueos de intervalo se mantienen hasta que la transacción finaliza. Este es el nivel de aislamiento más restrictivo, porque bloquea intervalos de claves completos y mantiene esos bloqueos hasta que la transacción finaliza. Al ser menor la simultaneidad, solo se debe utilizar esta opción cuando sea estrictamente necesario.

El nivel de aislamiento menor, ReadUncommitted, permite que muchas transacciones operen a la vez pero no proporciona protección contra daños sufridos por los datos debido a transacciones concurrentes.

El nivel de aislamiento mayor, Serializable, proporciona un alto grado de protección contra inconsistencias, pero requiere

que se complete la transacción antes de que ninguna otra transacción pueda operar en los datos.

4. Operativa general con transacciones

Podemos encontrar transacciones tanto a nivel de SQL como en otros lenguajes pero en cualquier caso el modo de operar

será el mismo:

- Determinar el nivel de aislamiento de la transacción.
- Iniciar la transacción.
- Incluir las operaciones que van incluidas en la transacción y que se tienen que ejecutar como un todo.
- Confirmar la transacción, finalizar la transacción indicando que los cambios queden permanentes, se suele indicar mediante la orden COMMIT.
- O bien anular la transacción, finalizar la transacción indicando que se tiene que deshacer todos los cambios realizados hasta ahora dentro de la transacción (se revierten todas las modificaciones), se suele indicar mediante la orden ROLLBACK.
- Si durante la transacción, el proceso que la lanzó se interrumpe, se efectúa un rollback implícito.

Según el lenguaje la sintaxis será diferente pero la idea la misma.

5. Transacciones en JDBC

Ref: http://docs.oracle.com/javase/tutorial/jdbc/basics/transactions.html

- Al crear una conexión se está en modo auto-commit que significa que cada instrucción SQL se trata como una transacción, al llegar al punto y coma, se ejecuta y confirma la transacción.
- Para permitir agrupar varias instrucciones en una transacción se debe desactivar ese modo con el método setAutoCommit de la conexión:

conexión.setAutoCommit(false);

- Para finalizar la transacción utilizamos los métodos commit (confirmar, dar por buenos los cambios) y rollback (deshacer los cambios) de la conexión:

conexion.commit; y conexion.rollback;

- Utilizamos **conexión.setAutoCommit(true);** para volver al estado por defecto que es un commit por cada instrucción.
- Teniendo en cuenta que las transacciones pueden bloquear otros procesos que usen los mismos datos, es obvio que el uso de transacciones con múltiples instrucciones solo se recomienda cuando sea necesario.
- Los niveles de aislamiento permitidos dependen del sistema gestor de la base de datos. Se pueden consultar mediante DatabaseMetaData.supportsTransactionIsolationLevel.
- El nivel de aislamiento utilizado en la transacción se puede obtener mediante el método **getTransactionIsolation**

de la conexión y se establece mediante el método **setTransactionIsolation** también de la conexión.

- También se pueden definir *savepoints* (puntos de recuperación) es una forma de implementar transacciones anidadas. Definiendo un punto de recuperación dentro de la transacción podemos ejecutar un rollback a partir de ese punto sin afectar a cualquier trabajo realizado en la transacción antes del punto. Varios savepoints pueden existir dentro de una transacción individual. Los Savepoints son útiles para implementar recuperación de errores complejos en aplicaciones de base de datos. Si ocurre un error en medio de una transacción de múltiples sentencias, la aplicación puede ser capaz de recuperarse del error, deshaciendo las actualizaciones a partir de un savepoint, sin necesidad de abortar la transacción completa.

- Para definir un savepoint: **Savepoint punto1 = conexion.setSavepoint();**

- Para eliminar un savepoint: **conexion.releaseSavepoint(punto1);** Cuando una transacción es confirmada (commit) todos los savepoints incluidos en ella quedan eliminados. Un rollback elimina cualquier punto incluido en las instrucciones incluidas dentro del rollback.

- Para deshacer las actualizaciones a partir de un savepoint: **conexion.rollback(punto1);**

- No todos los DBMS soportan savepoints.

Para consultar con más detalle los problemas que puede plantear la concurrencia, el acceso simultáneo de varios procesos a la misma información, te recomiendo esta lectura del profesor Jorge Pérez Rojas de la Universidad de Talca (Chile):

http://campuscurico.utalca.cl/~jperez/bd/documentos/transacciones.pdf