Resumen Transacciones

=====================

Introducción

Una de las principales tareas del diseñador de una BD es preservar la integridad de los datos contenidos en la misma. Una transacción es un conjunto de instrucciones que actúa como una unidad lógica de trabajo que tendrá éxito si se ejecuta completamente, o, en su defecto, nada de ella deberá reflejarse en la BD. Para garantizar que una transacción mantenga la consistencia de la BD es necesario que cumpla con cuatro propiedades (las llamadas ACID):

**Atomicidad**: Garantiza que todas las instrucciones de una transacción se ejecutan sobre la BD, o ninguna de ellas se lleva a cabo. Es responsabilidad del SGDB implementar los algoritmos necesarios para que se controle la atomicidad de la transacción.

**Consistencia**: La ejecución completa de una transacción lleva a la BD de un estado consistente a otro estado de consistencia. Para ello la transacción debe estar escrita correctamente y es responsabilidad total de quien la programe.

**Aislamiento**: Cada transacción actúa como única en el sistema. Esto significa que la ejecución de una transacción no debe afectar la ejecución simultánea de otra transacción

**Durabilidad**: Una vez finalizada una transacción los efectos producidos en una BD son permanentes, sin posibilidad de retrotraerlos.

Las transacciones son **idempotentes**, ésta condición asegura que aunque se ejecute la transacción N veces se genera siempre el mismo resultado sobre la BD.

Estados de las transacciones

Una transacción puede estar en alguno de los siguientes cinco estados:

**Activo**: Éste el estado que la transacción posee desde que comienza su ejecución hasta completar la última instrucción, o hasta que se produzca un fallo.

**Parcialmente finalizado o parcialmente cometido**: Se alcanza en el momento posterior a que la transacción ejecuta su última instrucción. En este momento el valor que debería quedar almacenado en la BD está almacenado en el buffer de la memoria principal, el siguiente paso consiste en bajar a disco. Si se produjera un error en éste último paso se debe abortar la transacción.

**Finalizado o cometido**: Se obtiene cuando la transacción finalizó su ejecución, y sus acciones fueron almacenadas correctamente en memoria secundaria.

**Fallado**: A éste estado se arriba cuando la transacción no puede continuar con su ejecución normal.

**Abortado**: Éste estado garantiza que una transacción fallada no ha producido ningún cambio en la BD, manteniendo la integridad de la información allí contenida.

Acciones antes la aborción de una transacción

**Reiniciar la transacción**: Ésta acción se puede llevar a cabo solo cuando el error se produjo por el entorno y no por la transacción. Entonces se puede generar una nueva transacción, la transacción anterior que falló ya terminó, su estado es abortado y por la propiedad de durabilidad no puede reiniciarse.

**Cancelar definitivamente la transacción**: Ésta acción se lleva a cabo cuándo se detecta un error interno en la lógica de la transacción. Éste error solamente puede ser salvado si la transacción es redefinida.

Fallos

Los fallos pueden ser clasificados en dos tipos:

**Fallos que no producen pérdida de información**: Si una transacción que solamente está consultando la BD falla y debe abortarse, no produce pérdida de información.

**Fallos que producen pérdida de información**: Involucran transacciones que incluyen las cláusulas SQL: INSERT, DELETE, UPDATE. Son transacciones que afectan el contenido de la BD y debe asegurarse que la integridad de los datos se mantenga.

Entornos de las transacciones: Las transacciones pueden ser analizadas desde tres perspectivas básicas.

**Entornos monousuarios:**

Éstos entornos admiten sólo un usuario operando con la BD y el mismo solamente puede realizar una actividad por vez, ésto garantiza que la propiedad de **aislamiento** se cumpla, mientras que se debe prestar atención a la atomicidad de la transacción, frente a un fallo re-ejecutar o no una transacción fallida no asegura la consistencia de la BD. El motivo de la inconsistencia es que la transacción efectúa cambios sin la seguridad de finalizar correctamente, entonces es necesario retrotraer el estado de la BD al instante anterior al comienzo de la ejecución de la transacción. Para ello existen dos métodos de recuperación, el de **bitácora** y el de **doble paginación**, cualquier de éstos métodos de recuperación necesita de dos algoritmos básicos:

1. Algoritmos que se ejecutan durante el procesamiento normal de las transacciones, que realizan acciones que permiten recuperar el estado de consistencia de la BD ante un fallo.
2. Algoritmos que se activan luego de detectarse un error en el procesamiento de las transacciones, que permiten recuperar el estado de consistencia de la BD.

**Bitácora (Log)**

El método de bitácora registra todas las acciones llevadas a cabo sobre la BD en un archivo (auxiliar y externo a la BD) histórico de movimientos. Éstos movimientos son plasmados en el archivo **antes** que los cambios lo sean sobre la BD misma. Con éste registro de información es posible garantizar que el estado de consistencia de la BD es alcanzable en todo momento, aún ante un fallo.

La estructura del archivo bitácora es simple. Cada transacción debe indicar su comienzo, su finalización y cada una de las acciones llevadas a cabo sobre la BD que modifiquen su contenido (operaciones de lectura), especificando el valor viejo y el valor nuevo. El gestor del SGDB es responsable de controlar la ejecución de las transacciones, de administrar el archivo de bitácora y de utilizar los algoritmos de recuperación cuándo se produzca un fallo. Para el método de bitácora, una transacción que registra un comienza y no tiene un registro de fin es una transacción que debe abortarse. El proceso de bitácora debe garantizar que primero se graben los buffers correspondientes al archivo de bitácora en memoria secundaria para luego grabar los de la BD. Para todas las transacciones que alcanzan el estado de finalizada, el trabajo registrado en bitácora es innecesario. La bitácora presenta dos alternativas para implementar el método de recuperación:

**Modificación diferida de una base de datos**:

Este método demora todas las escrituras en disco de las actualizaciones de la BD, hasta que la transacción alcance el estado de finalizada en bitácora. Bajo éste esquema existe una condición bajo la cual la BD puede perder la integridad y es cuándo la bitácora registra que la transacción ha finalizado pero los cambios sobre la BD no han bajado a disco, entonces como una transacción finalizada en bitácora puede haber realizado cambios parcialmente es necesario que ante la recuperación de un fallo se vuelva a ejecutar la última transacción ejecutada para dejar BD consistente. Por último, ésta política genera una gran carga al finalizar la transacción, que es cuando se produce el grabado en disco de la BD.

**Modificación inmediata de una base de datos**: Con éste método se efectúan los cambios sobre la BD a medida que la transacción avanza en su ejecución, la sobrecarga de trabajo tienda a desaparecer distribuyéndose en el tiempo. Siempre primero se registra un cambio en la bitácora y luego se graba en la BD. Con éste método es necesario contar con una operación que **deshaga** los cambios hechos por las transacciones que han fallado, dejando el estado de la BD consistente. Entonces, si se produce un error se debe **rehacer** toda la transacción de la bitácora que haya finalizado, y se debe **deshacer** toda la transacción iniciada que no haya finalizado.

Putos de verificación

Con el fin de evitar la revisión de toda la bitácora ante un fallo, el gestor de BD agrega en forma periódica, al registro histórico, una entrada checkpoint. Éstos puntos de verificación se agregan cuando se tiene la seguridad de que la BD esté en estado consistente. La finalidad es indicar que, ante un fallo, sólo debe revisarse la bitácora desde el punto de verificación en adelante, dado que las transacciones anteriores finalizaron correctamente sobre la BD. En entornos monousuarios siempre es posible colocar el punto de verificación en un instante del tiempo sin transacciones activas.

Doble paginación

Este método plantea dividir al BD en nodos virtuales (páginas) que contienen determinados datos. Se generan dos tablas en disco y cada una de las tablas direcciona a los nodos (páginas) generados. En caso de producirse un fallo, luego de la recuperación, s desecha la página actual y se toma como válida la página a la sombra. Ésto presenta ventajas sobre le método de bitácora, primero, se elimina la sobrecarga producida por las escrituras al registro histórico, y la recuperación ante un error es mucho más rápida. Sin embargo. el método también tiene algunas desventajas importantes. Primeramente, exige dividir la BD en nodos o páginas, y esta tarea puede ser compleja. En segunda lugar, la sobrecarga del método aparece en entornos concurrentes.

En caso de fallos en una transacción una vez recuperada la situación de fallo, el nodo nuevo no se utiliza; sin embargo, sigue ocupando lugar en el disco, para solucionar ésto es necesario contar con algoritmos de recolectado de basura.

**Entornos concurrentes**

En un entorno concurrente se debe asegurar, además, la propiedad de aislamiento aislamiento de una transacción, un entorno concurrente que no garantice ésto puede generar situaciones de pérdida de integridad de los datos aún sin producirse errores. Una solución simple sería secuencializar las transacciones pero ésto impediría la atención en simultaneo de transacciones.

Conceptos de transacciones serializables

Es necesario establecer un criterio de ejecución para que la concurrencia no afecte la integridad de la BD, el orden de ejecución en serie de transacciones se denomina planificación. En general un entorno concurrente con N transacciones en ejecución puede generar N! trazas o planificaciones en serie diferentes. Una planificación tiene las siguientes características:

Involucra todas las instrucciones de la transacción.

Conservan el orden de ejecución de éstas transacciones.

Cuándo la ejecución de una transacción no afecta el desempeño de otra transacción sobre la BD, ambas pueden ejecutarse concurrentemente sin afectar la propiedad de aislamiento, el mismo se garantiza porque operan sobre datos diferentes.

La inconsistencia temporal puede ser causa de inconsistencia en planificaciones concurrentes. La inconsistencia temporal se presenta durante la ejecución de una transacción que modifica la BD, hasta que termine su ejecución.

Una planificación concurrente, para que sea válida, debe equivaler a una planificación en serie.

Sólo las instrucciones READ y WRITE son importantes y deben considerarse para cualquier tipo de análisis. Esto se debe a que el resto de las operaciones se pueden llevar a cabo sobre la computadora del usuario, en su memoria local, sin afectar el contenido de la BD.

Como conclusión, dos instrucciones son conflictivas si operan sobre el mismo dato, y al menos una de ellas es una operación de escritura.

Dos planificaciones P1 y P2 son equivalentes en cuanto a conflictos cuando P2 se logra a partir del intercambio de instrucciones conflictivas con P1.

Una planificación P2 es serializable en cuanto a conflictos si existe otra planificación P1 equivalente en cuanto a conflictos con P2, y además P1 es una planificación en serie.

En otras palabras. Si P1 es una planificación en serie entonces su ejecución garantiza aislamiento (si no hay fallos). Si P1 y P2 son planificaciones equivalentes en cuanto a conflictos. significa que P2 obtiene un resultado similar a P1. Por lo tanto, P2 mantiene la consistencia de la BD. P2 será entonces una planificación concurrente que garantiza la integridad de la BD.

Una planificación concurrente debe ser equivalente a una planificación en serie para que se mantenga la integridad de la BD.

Pruebas de seriabilidad

Existen métodos para demostrar la seriabilidad de planificaciones concurrentes. Estos algoritmos están diseñados para detectar conflictos en la ejecución de dos transacciones, los mismos generan un grafo, que de presentar al menos un ciclo determina que la planificación no es serializable en cuanto a conflictos.

Implementación del aislamiento. Control de concurrencia

Existen dos variantes para lograr implementar el aislamiento

Protocolo de bloqueo

El método de bloqueo se basa en que una transacción debe solicitar el dato con el cual desea operar, y tenerlo en su poder hasta que no lo necesite más. Existen dos tipos de bloqueo a realizar sobre un elemento de datos:

Bloqueo compartido: Debe ser realizado por una transacción para leer un dato que no modificará. Mientras dure este bloqueo se impide que otra transacción pueda escribir el mismo dato. Un bloqueo compartido puede coexistir sobre otro bloqueo compartido sobre el mismo dato, es decir, varias transacciones pueden pedirlo y obtenerlo.

Bloqueo exclusivo: Se genera cuando una transacción necesita escribir un dato. Dicho bloqueo garantiza que otra transacción no pueda utilizar ese dato (ya sea una lectura o escritura). Un bloque de éste tipo es único, require que ninguna otra transacción esté operando sobre el dato en ese momento.

Si el dato está siendo utilizado en ese momento la transacción tiene dos opciones, esperar por el dato o fallar y abortar, para luego comenzar una nueva transacción.

Aún utilizando protocolos de bloqueo no se garantiza aislamiento de una transacción. Una transacción debe pedir todo lo que necesita al principio, utilizarlo y luego liberarlo. De ésta forma se pueden mejorar las condiciones de aislamiento, llevando la ejecución de transacciones que pueden generar conflicto (dado que operan sobre el mismo dato) a una ejecución en serie. Solo las transacciones que operen sobre datos diferentes podrán ejecutarse concurrentemente con total aislamiento. Trasladar los bloqueos al comienzo de la transacción no soluciona el problema en un 100%,y además puede planter situaciones de deadlock, con dos procesos obteniendo el recurso que requiere el otro y esperando por el recurso que el otro tiene.

Cómo conclusión, una transacción que traslada sus bloques al comienzo pueden generar deadlock. Por el contrario, si los bloqueos se ejecutan cuando son necesarios, las posibilidades de deadlock disminuyen, pero el riego de perdida de consistencia aumenta. Es preferible generar situaciones de deadlock, dado que éstas no conducen a la pérdida de consistencia de los datos.

El protocolo de bloqueo de dos fases garantiza aislamiento en la ejecución de transacciones, utilizando el principio de las dos fases: crecimiento y decrecimiento.

Para que una transacción sea correcta, el orden de pedidos de bloqueos debe ser como sigue:

1. Face crecimiento: se solicitan bloqueos similares o se crece de compartido a exclusivo.
2. Fase decrecimiento: Exclusivo, compartido, liberar datos.

Protocolo basado en hora de entrada

Es una variante del protocolo de bloqueo donde la ejecución exitosa de una transacción se establece de antemano, en el momento en que la transacción fue generada. Cada transacción recibe una **Hora de Entrada (HDE)** en el momento en que inicia su ejecución. La HDE es una marca única asignada a una transacción. Ésta marca única puede ser el valor de un contador o la hora del reloj interno del servidor de la BD.

El método continua asignado a cada elemento de dato de la BD dos marcas temporales: la hora de la última lectura HL(D) y la hora de la última escritura HE(D). Estas marcas corresponden a la HDE de la última transacción que leyó o escribió el dato. Para decidir si la transacción puede o no ejecutar su operación de lectura/escritura sobre la BD, el protocolo toma la decisión basado en el siguiente cuadro:

* Una operación de lectura. Suponga que T1 desea leer el dato D.
  + Para poder leer el dato se debe cumple que HDE(T1) > HE(D)
  + HDE(T1) < HE(D) el dato D es demasiado nuevo para que T1 pueda leerlo.

Para poder leer un dato de la BD, el protocolo basado en HDE no necesita chequear la HL(D). Esto se debe a que las lecturas pueden ser compartidas. Varias transacciones pueden leer el mismo dato sin generar conflicto entre ellas. En caso de que la operación tenga éxito debe establecer la HL(D) como el máximo entre HDE(T1) y HL(D)

* Una operación de escritura. Suponga que T1 desea escribir el dato D.
  + Si HDE(T1) < HL(D), la operación falla. No es posible que T1 escriba el dato D si fue leído por una transacción posterior.
  + Si HDE(T1) < HE(D), la operación falla. no es posible que T1 escriba el dato, dicho dato fue escrito por una transacción posterior.
  + En cualquier otro caso, T1 puede ejecutar la operación de escritura y establece HE(D) con el valor de HDE(T1).

Cualquier transacción que no puede seguir su ejecución falla y aborta su trabajo. Posteriormente, se puede generar una nueva transacción con una nueva HDE, e intentar ejecutar la operación deseada.

Granularidad

El concepto de granularidad en el acceso a los datos está vinculado con el tipo de bloqueo que se puede realizar sobre la BD. La granularidad gruesa permite solamente bloquear toda la BD, y a medida que la granularidad disminuye, es posible bloquear a nivel tabla, registro, o hasta campos que la componen. En general se permite diversos niveles de bloques.

Las transacciones en entornos concurrentes necesitas bloqueos a nivel de registros. Los bloqueos compartidos y exclusivos de datos apuntan a bloquear el uso o la escritura de registros de la BD.

Otras operaciones concurrentes

Las operaciones de inserción y borrado sobre la BD cumplen los mismos requisitos que las operaciones de lectura/escritura.

Bitácora en entornos concurrentes

En el caso de entornos concurrentes, se agrega un nuevo tipo de fallo, una transacción que no puede continuar su ejecución por problemas de bloqueos o accesos a la BD. En ese caso, como cualquiera situación de fallo, la transacción debe abortar, dejando la BD en el estado de consistencia anterior al comienzo de su ejecución. Si bien el método es similar es necesario notar algunas consideraciones adicionales:

* Existe un único buffer de datos tanto para la BD como para la bitácora.
* Como se presentó anteriormente, cada transacción tiene su propia área donde administra la información localmente.
* El fallo de una transacción significa deshacer el trabajo llevado a cabo por ella.

Retroceso en cascada de transacciones

El retroceso de una transacción puede llevar a que otras transacciones también fallen. Esto sucede cuando una transacción utiliza datos que fueron modificados por otra transacción que no ha finalizado, para evitar que eso produzca inconsistencia se debe tener una nueva condición: Una transacción Tj no puede finalizar su ejecución, si una transacción Ti anterior utiliza datos que Tj necesita, y Ti no está en estado finalizada. De ésta forma si Ti falla, Tj también deberá fallar.

Éste tipo de acciòn puede llevar a retrotraer varias transacciones.

Puntos de verificación de registro histórico

El único cambio que requiere el método de registro histórico con respecto a entornos monousuarios está vinculado con los puntos de verificación. En un entorno monousuario, un punto de verificación se agrega periódicamente luego de que finalice una transacción y antes de que comience otra.

El registro histórico en un esquema concurrente no puede garantizar la existencia de un momento temporal, donde ninguna transacción se encuentre activa. Por éste motivo la sentencia de checkpoint agrega un parámetro L. Este parámetro contiene la lista de transacciones que, en el momento de colocar el checkpoint, se encuentran activas. Ante un fallo, el algoritmo de recuperación actúa normalmente, a excepción de las transacciones que están en la lista del último punto de verificación. Para esas transacciones se debe revisar la bitácora aún más atrás del punto de verificación, dado que el comienzo de cada una de ellas quedó definido previo a ese punto.