PROTECCIÓN, TRANSACCIONES, CONCURRENCIA Y MÁS

PROTECCIÓN A LOS DATOS

La protección de los datos es un aspecto muy vinculado con el concepto mismo de base de datos.

La protección de los datos debe realizarse contra fallos físicos, fallos lógicos y fallos humanos. Estas situaciones alteran los datos, con lo que la base de datos ya no puede servir a los fines para los que fue creado.

En este contexto aparecen los temas de: recuperación, concurrencia, seguridad e integridad de la base de datos.

Los problemas de recuperación y concurrencia están muy relacionados con lo que se conoce como procesamiento de transacciones

El SGBD proporciona mecanismos para prevenir fallos (subsistemas de control), para detectarlos una vez que se han producido (subsistema de detección) y para corregirlos

1.-RECUPERACION

El objetivo del concepto de recuperación es el de proteger la BD contra fallas lógicas y físicas que destruyan los datos en todo o en parte. Independiente de la naturaleza de las fallas estás pueden afectar a dos aspectos del almacenamiento de la Base de Datos, como son:

- Fallas que provocan la pérdida de memoria volátil
- Fallas que provocan la pérdida del contenido de memoria secundaria.

Para asegurar que la BD siempre este en un estado consistente, se crean unidades de ejecución llamadas transacciones, que pueden definirse como una secuencia de operaciones que han de ejecutarse en forma atómica, es decir, se realizan todas las operaciones que comprende la transacción o no se realiza ninguna.

Ej: una transacción bancaria que saca dinero de una cuenta y lo dispone en otra.

Las transacciones o terminan con éxito y son grabadas en la base o bien fracasan y debe ser restaurado el estado anterior de la BD.

El componente del sistema encargado de lograr la atomicidad se conoce como administrador de transacciones y las operaciones COMMIT (comprometer) y ROLLBACK (retroceder) son la clave de su funcionamiento.

La operación COMMIT señala el término exitoso de la transacción: le dice al administrador de transacciones que se ha finalizado con éxito una unidad lógica de trabajo, que la base de datos está o debería estar de nuevo en un estado consistente y que se pueden comprometer, o hacer permanentes todas las modificaciones efectuadas por ese unidad de trabajo.

La operación ROLLBACK, en cambio, señala el término no exitoso de la transacción: le dice al administrador de transacciones que algo salió mal, que la base de datos podría estar en un estado inconsistente y que todas las modificaciones efectuadas hasta el momento por la unidad lógica de trabajo deben retroceder o anularse.

Las características de una transacción son:

- *Atomicidad*, en el sentido que hemos especificado anteriormente: se ejecutan todas las sentencias o ninguna.
- *Preservación de la consistencia*: la ejecución de una transacción deja la BD en un estado consistente.
- *Aislamiento*, ya que una transacción no muestra los cambios que produce hasta que finaliza.
- *Persistencia*, ya que una vez que finaliza la transacción con éxito, sus efectos perduran en la BD.
- *Seriabilidad*, en el sentido de que el efecto de ejecutar transacciones concurrentemente debe ser el mismo que se produciría al ejecutarlas por separado en un orden secuencial según van entrando en el sistema.

Para conseguir anular y recuperar transacciones, el método más usado consiste en utilizar un archivo de diario o log en el que va guardando toda la información necesaria para deshacer (en caso de fracasar) o rehacer (en caso de recuperar) las transacciones. Este archivo consta de:

- identificador de la transacción
- hora de modificación
- identificador del registro afectado
- tipo de acción
- valor anterior del registro
- nuevo valor del registro
- información adicional.

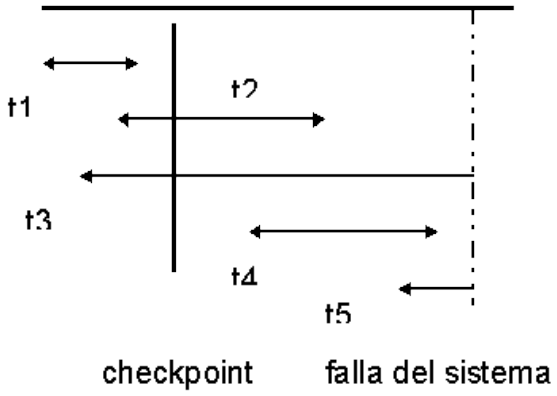
Otra alternativa es manejar 2 archivos de *log*, uno con la imagen anterior a las modificaciones y otro con la imagen posterior a las modificaciones. El archivo log es usualmente una pila que una vez llena va eliminado registros según van entrando nuevos.

Un concepto relacionado con los archivos de log es el CHECKPOINT, que permite manejar en forma eficiente el contenido de los archivos log, ya que permiten no tener que recorrer todo el archivo de log, ante fallas.

El establecimiento de puntos de revisión implica:

- grabar físicamente el contenido de los buffers de datos a la base de datos física
- grabar físicamente un registro de punto de revisión especial dentro del archivo de log o bitácora

Los puntos marcados como checkpoint, permiten la recuperación de la base de datos en caliente, es decir, después de la caída del sistema se obtiene la dirección del registro de recuperación más reciente y se recorre el archivo de log desde el punto marcado como checkpoint.



La transacción t1 no se ve afectada por la falla del sistema, ni por el proceso de recuperación, por haberse completado antes del último punto de recuperación. Las transacciones t2 y t4, a pesar de haber terminado no han sido grabadas en la base de datos, ya que éstas serían cometidas en un checkpoint. Las transacciones t3 y t5 deberán rehacerse ya que no han concluido.

El procedimiento que deberá realizar el sistema al reiniciarse consiste en:

1. Comenzar con dos listas de transacciones, la lista ANULAR y la lista REPETIR. Igualar la lista ANULAR a la lista de todas las transacciones incluidas en el registro de punto de revisión. Dejar vacía la lista REPETIR.
2. Examinar la bitácora hacia adelante a partir del registro de punto de revisión.

3. Si se encuentra una entrada de bitácora de "iniciar transacción" para la transacción T, añadir T a la lista ANULAR.
4. Si se encuentra una entrada una entrada de bitácora "comprometer" para la transacción T, pasar esa transacción de la lista ANULAR a la lista REPETIR.
5. Cuando se llegue al final de la bitácora, las listas ANULAR y REPETIR identificarán, respectivamente, las transacciones de los tipos T3 y T5 y las de los tipos T2 y T4.

Posteriormente el sistema revisará la bitácora hacia atrás, anulando todas las transacciones de la lista ANULAR. A continuación la revisará hacia adelante, realizando de nuevo todas las transacciones en la lista REPETIR. Por último, una vez terminada todas las actividades de recuperación, el sistema estará listo para aceptar nuevos trabajos.

La recuperación en frío, consiste en disponer de un backup o respaldo de la BD, que permitirá junto con los archivos de log, que se han ido produciendo desde el ultimo backup, reconstruir la BD, para dejarla consistente.

El error fatal, se produce cuando se pierde el archivo de log. En este caso resulta imposible recuperar la base. La solución pasa por disponer los archivos de log en respaldos.

El DBA debe definir responsabilidades, procedimientos, situaciones y plazos en los que se deben realizar las copias de seguridad y el respaldo del archivo de log, especificando a los operadores los procedimientos de recuperación ante caídas. El principio básico en el que se basa la recuperación es la redundancia.

1.1.-Compromiso en dos fases

Este concepto es particularmente importante cuando se tienen múltiples SGBD en el contexto de los sistemas de bases de datos distribuidas.

Supongamos una transacción que actualiza una base de datos Oracle y una base de datos ADABAS. Si la transacción se completa con éxito, todas las modificaciones, tanto de los datos Oracle como de los datos Adabas, deberán comprometerse. Si fracasa todas deberán anularse.

Aquí el COMMIT (o ROLLBACK) afecta a todo el sistema. Estas instrucciones son manejadas por un componente llamado coordinador, y puede realizarse correctamente porque utiliza el protocolo de compromiso en dos fases.

1. El coordinador ordena a los administradores de recursos (Oracle , Adabas) que se preparen para las dos posibilidades con respecto a la transacción. Esto significa que cada administrador guarda físicamente (almacena) las entradas de la bitácora que corresponden a los recursos locales empleados por la transacción. Si la grabación tiene éxito, el manejador de recursos contestará "todo bien" al coordinador. En caso contrario, contestará "no está bien".

2. Cuando el coordinador reciba la respuesta de todos los administradores de recursos, grabará una entrada en su propia bitácora física. Si todas las respuestas fueron "todo bien", la decisión será comprometer. Si cualquiera de las respuestas fue "no está bien", la decisión será retroceder. En cualquier caso, el coordinador le informará de su decisión a todos los administradores de recursos y cada administrador de recursos deberá comprometer o anular la transacción localmente.

Cada administrador deberá hacer lo que el coordinador le ordena en la fase 2. Así es el protocolo.

2.- Concurrencia

En sistemas multiusuario, es necesario un mecanismo para controlar la concurrencia. Se pueden producir inconsistencias importantes derivadas del acceso concurrente, como por ejemplo, el problema de la operación perdida.

En un sistema de biblioteca, existe un campo que almacena el nro de copias disponibles para préstamo. Este campo debe incrementarse en uno cada vez que se devuelve un ejemplar del libro y disminuirse en uno cada vez que se presta un ejemplar.

Si existen varias bibliotecarias, una de ellas inicia la transacción t1, leyendo la variable nro ejemplares (n), cuyo contenido se guarda en la variable n1. Tiempo después, otra bibliotecaria podría leer la misma variable incrementándola en una unidad, transacción t2. Después, la transacción t1 añade una unidad a esa variable y la actualiza, el resultado es erróneo, ya que la variable N debería haber aumentado en 2 unidades, y solo ha aumentado en una. La transacción t2 se ha perdido.

2.1.- Técnicas de control de concurrencia

- Pesimistas: bloqueo y marcas de tiempo
- Optimistas

Técnicas de bloqueo: es una variable asociada a cada elemento de datos que describe el estado de dicho elemento respecto a las posibles operaciones (recuperación o actualización) que se pueden realizar sobre ellos en cada momento.

Las transacciones pueden llevar a cabo bloqueos, impidiendo a otros usuarios la recuperación o actualización de los elementos bloqueados, para evitar inconsistencias en el acceso concurrente.

Los SGBD tienen bloqueos(por tupla, por tabla) para asegurar la consistencia. Los usuarios también pueden bloquear explícitamente los objetos, impidiendo el acceso por parte de otros usuarios.

Tipos de bloqueo

- Exclusivos: cuando una transacción mantiene un bloqueo de este tipo, ninguna otra transacción puede acceder a el objeto bloqueado, ni bloquearlo, hasta que sea liberado por la transacción que lo había retenido. Se utiliza cuando se quiere actualizar datos.
- Bloqueo compartido: cuando una transacción bloquea en este modo, permite que otras transacciones retengan también el objeto en bloque compartido, pero no exclusivo. Este tipo se utiliza cuando no se requiere actualizar datos, pero se desea impedir cualquier modificación mientras los datos son consultados.

El algoritmo que se utiliza se llama bloqueo de dos fases (two phase locking).

El problema de las técnicas de bloqueo es que puede producirse un interbloqueo (deadlock), dos o mas transacciones están esperando cada una de ellas que la otra libere algún objeto antes de seguir

Se puede solucionar:

Prevenir el deadlock: obliga a que las transacciones bloqueen todos los elementos que necesitan por adelantado. EN caso de no poder conseguir todos esos elementos no bloquea ninguno y se queda en espera hasta volver a intentarlo.

Detectar el deadlock: Se controla de forma periódica si se ha producido un deadlock. Se construye un grafo en espera, cada nodo es una transacción en ejecución y un arco de una transacción Ti a Tj, en caso que Ti esté esperando un elemento que ocupa Tj. Si existe un ciclo en el grafo tenemos un deadlock. La solución es escoger transacciones víctimas y deshacerlas, hasta que desaparezca el deadlock. Cada SGBD tiene políticas diferentes para escoger víctimas.

Este tema influye notoriamente en el rendimiento de los sistemas. Los SGBD pueden bloquear:

- un campo de un registro (un atributo de una tabla)
- un registro (una tupla)
- un archivo (una tabla)
- la BD total

Esto se llama *granularidad* del bloqueo.

Granularidad muy gruesa implica gestionar menor nro de bloqueos, pero retrasa la ejecución de muchas transacciones (los objetos no se van liberando). Una granularidad muy fina, permite mayor concurrencia, pero aparecen más situaciones de deadlock que han de ser resueltas.

Técnicas de marca de tiempo (timestamping)

Las marcas de tiempo son identificadores únicos que se asignan a las transacciones, que se consideran como el tiempo de inicio de una transacción. Con esta técnica no existen bloqueos. Ordena las transacciones. Se retrasan.

Técnicas optimistas

Las transacciones acceden libremente a los elementos, y antes de finalizar se determina si ha habido interferencias.

Este tipo de técnicas considera que las transacciones tienen 3 fases:

- *Lectura*: las transacciones realizan operaciones sobre copias privadas de los objetos (accesibles solo por la transacción)
- *Validación* : en la que se comprueba si el conjunto de objetos modificados por una transacción se solapa con el conjunto de objetos modificados por alguna otra que haya hecho la validación durante la fase de lectura de dicha transacción
- *Grabación*:, en el caso de no detectar interferencias se graban las modificaciones, convirtiendo las versiones privadas de los objetos en versiones actuales.

3.- Integridad

La integridad tiene como función proteger la BD contra operaciones que introduzcan inconsistencias en los datos. Se habla de integridad en el sentido de corrección, validez o precisión de los datos.

El subsistema de integridad de un SGBD debe por tanto detectar y corregir, en la medida de lo posible, las operaciones incorrectas. En la práctica es el punto débil de los SGBD comerciales, ya que casi toda la verificación de integridad se realiza mediante código de procedimientos escritos por los usuarios.

Habrá operaciones cuya falta de corrección no sea detectable, por ejemplo, introducir un fecha de nacimiento 25/12/1945 cuando en realidad era 25/12/1954.

3.1.-Operaciones semánticamente inconsistentes

Son las que transgreden las restricciones que ha definido el administrador al diseñar la base de datos, tales como:

- Restricciones sobre dominios por ejemplo, el dominio edad esté comprendido entre 18 y 65 años.
- Restricciones sobre los atributos, por ejemplo, la edad de los empleados ingenieros debe ser mayor de 21 años.

Estas restricciones pueden ser estáticas, como las anteriores o dinámicas por ejemplo, el sueldo de un empleado no puede disminuir.

Otra forma de clasificar las restricciones es en:

- simples: si se aplican a una ocurrencia de un atributo con independencia de los demás, por ejemplo, el sueldo de un empleado tiene que ser mayor que 60000.
- compuestas: si implican más de una ocurrencia, como es el caso de las restricciones de comparación, por ejemplo, el sueldo de un empleado debe ser menor que el de su jefe. O bien , las llamadas de globalidad, por ejemplo, el sueldo medio de los empleados de un determinado depto debe ser menor de 250000.

Los SGBD tienen que ofrecer en su lenguaje de definición, facilidades que permitan describir las restricciones con una sintaxis adecuada.

Por ejemplo, CREATE DOMAIN, CREATE ASSERTION, CREATE INTEGRITY RULE

En general, una regla de integridad está compuesta por tres componentes:

- La restricción propiamente tal, que establece la condición que deben cumplir los datos
- La respuesta a la transgresión, que especifica las acciones a tomar, como rechazar las operaciones, informar al usuario, corregir el error con acciones complementarias, etc.
- Condición de disparo, que especifica cuándo debe desencadenarse la acción especificada en la restricción de integridad: antes, después o durante cierto evento.

Los triggers, son casos especiales de reglas de integridad. Un trigger es un procedimiento que se activa o dispara al ocurrir un evento, que tienen muchas utilidades.

Las reglas de integridad deben almacenarse en el diccionario de datos, como parte integrantes de los datos(control centralizado de la semántica), de modo que no han de incluirse en los programas. Esto trae algunas ventajas:

- Las reglas de integridad son más sencillas de entender y de cambiar, facilitando su mantenimiento.
- Se detectan mejor las inconsistencias
- Se protege mejor la integridad, ya que ningún usuario podrá escribir un programa que las transgreda, llevando a la BD a estados inconsistentes.

El subsistema de integridad del SGBD debe realizar las siguientes funciones:

- Comprobar la coherencia de las reglas que se definen
- Controlar las distintas transacciones y detectar las transgresiones de integridad
- Cuando se produce una transgresión, ejecutar las acciones pertinentes.

4.-Seguridad

El objetivo es proteger la BD contra accesos no autorizados. Se llama también privacidad.

Incluye aspectos de:

- Aspectos legales, sociales y éticos
- Políticas de la empresa, niveles de información publica y privada
- Controles de tipo físico, acceso a las instalaciones
- Identificación de usuarios: voz, retina del ojo, etc.
- Controles de sistema operativo

En relación al SGBD, debe mantener información de los usuarios, su tipo y los accesos y operaciones permitidas a éstos.

Tipos de usuarios:

- DBA, están permitidas todas las operaciones, conceder privilegios y establecer usuarios
- Usuario con derecho a crear, borrar y modificar objetos y que además puede conceder privilegios a -otros usuarios sobre los objetos que ha creado.
- Usuario con derecho a consultar, o actualizar, y sin derecho a crear o borrar objetos.

Privilegios sobre los objetos, añadir nuevos campos, indexar, alterar la estructura de los objetos, etc. Los SGBD tienen opciones que permiten manejar la seguridad, tal como GRANT, REVOKE, etc. También tienen un archivo de auditoría en donde se registran las operaciones que realizan los usuarios.

Otro mecanismo de seguridad que ofrecen los SGBD en entregar información a los usuarios a través de vistas (CREATE VIEW)

TRANSACCIONES EN MYSQL

Antes de todos, hay que aclarar que para utilizar transacciones en MySQL (versión 5.0.x), debemos utilizar el motor de almacenamiento **InnoDB**.

Una transacción tiene dos finales posibles, **COMMIT** y **ROLLBACK**. Por defecto, MySQL trae activado el modo **autocommit**, es decir, realizada una transacción (por ejemplo un **INSERT**, **UPDATE** o **DELETE**) el mismo es confirmado apenas es ejecutado. Para desactivar el autocommit, se puede desactivar el autocomit ejecutando el comando: SET AUTOCOMMIT=0;

Una vez deshabilitado el autocommit, tendremos que utilizar obligatoriamente el COMMIT para confirmar o ROLLBACK para deshacer la transacción.

Si se quiere deshabilitar el autocommit para una serie de comandos, lo ideal es utilizar START TRANSACTION (sin necesidad de setear el AUTOCOMMIT en 0).

Al ejecutar una transacción, el motor de base de datos nos garantizará la **atomicidad, consistencia, aislamiento y durabilidad (ACID)** de la transacción (o conjunto de comandos) que se utilice.

Veremos un ejemplo completo, extraído del articulo fuente de esta publicación, donde utilizaremos START TRANSACTION (no es necesario AUTOCOMMIT en 0)

```
CREATE TABLE `departamentos` (
`CODIGO` INTEGER(11) NOT NULL DEFAULT '0',
`NOMBRE` VARCHAR(100),
`PRESUPUESTO` INTEGER(11) DEFAULT NULL,
PRIMARY KEY (`CODIGO`)
)ENGINE=InnoDB
```

Ahora, insertaremos registros de la tabla departamentos_externos a departamentos mediante una transacción:

```
START TRANSACTION;
SELECT @A := presupuesto
FROM departamentos_externos
```

```
WHERE codigo =11;
INSERT INTO departamentos( codigo, nombre, presupuesto )
VALUES ( 11, 'Department test', @A );
COMMIT;
```

En el ejemplo anterior se guardo el presupuesto del departamento externo 11 en la variable @A y luego fue asignado al presupuesto en la tabla departamentos.

Otro ejemplo:

```
START TRANSACTION;
SELECT @A := presupuesto, @B := codigo, @C := nombre
FROM departamentos_externos
WHERE codigo=33;
INSERT INTO departamentos( codigodep, nombredep, presupuesto ) VALUES (@B , @C ,
@A );
COMMIT ;
```

Otro ejemplo más:

```
START TRANSACTION;
SELECT @A:=PRESUPUESTO FROM departamentos_externos WHERE codigo=11;
UPDATE departamentos SET PRESUPUESTO = PRESUPUESTO + @A WHERE codigo=33;
COMMIT;
```

Al realizar una transacción SQL hay que tener en cuenta que apenas se realice un INSERT, UPDATE o DELETE se genera un bloqueo sobre la tabla y que otros clientes no pueden acceder para escribir esta tabla. Otros clientes podrán realizar SELECTs sobre la tabla, pero no podrán ver los datos del primer cliente hasta que los mismos sean confirmados. Prometo pronto, tratar con más detalle el tema de bloqueos, específicamente el **procesamiento concurrente de transacciones**.