BASES DE DATOS
TEMA 5
RECUPERACIÓN DE FALLAS

5.1 Clasificación de fallas

El sistema debe estar preparado para recuperarse no sólo de fallas puramente locales, como la aparición de una condición de desborde dentro de una transacción, sino también de fallas globales, como podría ser la interrupción del suministro eléctrico al CPU

Las **fallas locales** son las que afectan sólo a la transacción en donde ocurrió. Por el contrario las **fallas globales**, afectan a varias y casi siempre a todas las transacciones que se estaban efectuando en el momento de la falla, por lo cual tienen implicaciones importantes en el sistema. Estas fallas pueden ser:

Fallas del sistema

Afectan a todas las transacciones que se estaban ejecutando pero no afectan a la base de datos, se conocen también como ***caídas suaves*** *(crash)*. El problema aquí es que se pierda el contenido de memoria principal, en particular, las áreas de almacenamiento temporal o buffers.

Si esto ocurre, no se conocerá el estado preciso de la transacción que se estaba ejecutando en el momento de la falla, esta transacción jamás se podrá completar con éxito por lo que será preciso anularla cuando se reinicie el sistema.

Además, puede ocurrir que sea necesario volver a ejecutar algunas transacciones que sí se realizaron con éxito antes de la falla pero cuyas modificaciones no lograron efectuarse sobre la base de datos porque no lograron ser transferidas de los buffers de la base de datos a la base de Datos física (en disco).

Fallas en los medios de almacenamiento

Una falla de los medios de almacenamiento es un percance en el cual se destruye físicamente alguna porción de la DB. La recuperación de una falla semejante implica en esencia cargar de nuevo la DB a partir de una copia de respaldo y utilizar después la bitácora para realizar de nuevo todas las transacciones terminadas desde que se hizo esa copia de respaldo. No hay necesidad de anular las transacciones inconclusas en el momento de la falla, porque por definición todas las modificaciones de esas transacciones ya se anularon de todas maneras.

La parte de restauración de la utilería servirá entonces para recrear la DB después de una falla de los medios de almacenamiento a partir de una copia de respaldo especificada.

Las fallas de los medios de almacenamiento se llaman **caídas duras**. La Recuperación de una falla semejante implica, en esencia, cargar de nuevo la base de datos a partir de una copia de respaldo (database backup) y después utilizar la bitácora, o system log, para realizar de nuevo todas las transacciones terminadas desde que se hizo esa copia para respaldo.

Fallas por catástrofes

Por terremotos, incendios, inundaciones, etc. Su tratamiento es similar al de fallas de los medios. La principal técnica para manejar este tipo de fallas es la del **database backus.** Este es un respaldo periódico que se hace de la DB. Después de una caída de esta índole el sistema se restaura recargando la DB con la copia del último respaldo y recreando la DB mediante la bitácora o system log.

Errores del sistema

Como realizar operaciones que causen un **overflow** de un entero o la división por cero, así mismo puede ocurrir que se pasen valores erróneos a algún parámetro o que se detecte un error en la lógica de un programa, o que sencillamente no se encuentren los datos del programa. Además, en algunos ambientes de desarrollo el usuario puede explícitamente interrumpir una transacción durante su ejecución

Aplicación del control de concurrencia

Que ocurre por ejemplo cuando una transacción viola las reglas de serialización o cae en abrazo mortal o interbloqueo.

5.2 Modelo de transacciones

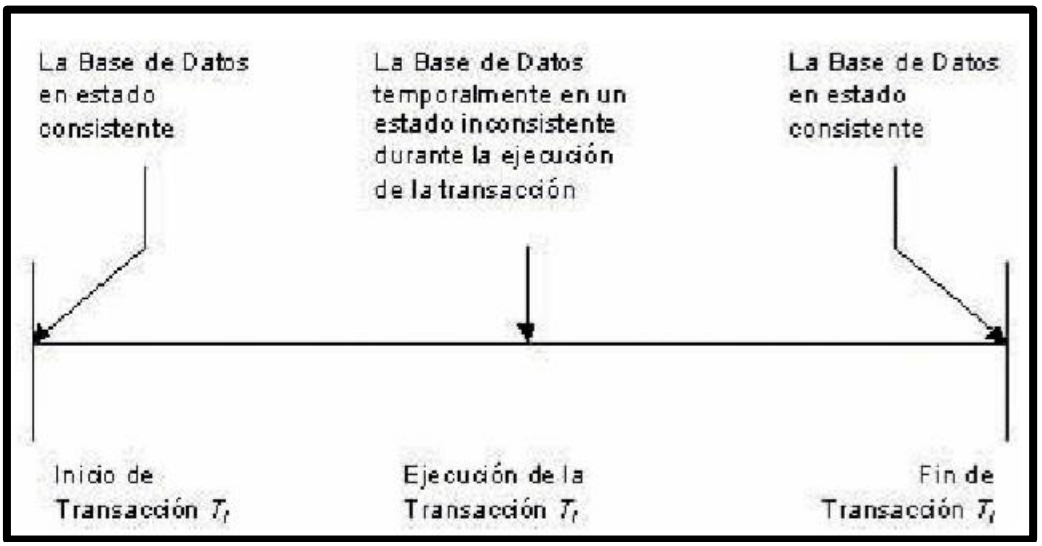
Un **algoritmo de control de concurrencia** asegura que las transacciones se ejecuten atómicamente controlando la intercalación de transacciones concurrentes, para dar la ilusión de que las transacciones se ejecutan serialmente, una después de la otra, sin ninguna intercalación. Las ejecuciones intercaladas cuyos efectos son los mismos que las ejecuciones seriales son denominadas **serializables** y son correctos ya que soportan la ilusión de la atomicidad de las transacciones.

Informalmente, una **transacción** es la ejecución de ciertas instrucciones que accesan a una DB compartida. El **objetivo del control de concurrencia y recuperación** es asegurar que dichas transacciones se ejecuten atómicamente, es decir:

Cada transacción accede a información compartida sin interferir con otras transacciones, y si una transacción termina normalmente, todos sus efectos son permanentes, en caso contrario no tiene afecto alguno.

Una base de datos está en un estado consistente si obedece todas las restricciones de integridad definidas sobre ella.

Los cambios de estado ocurren debido a actualizaciones, inserciones y supresiones de información. Para asegurar que la base de datos nunca entre en un estado de inconsistencia. Sin embargo, durante la ejecución de una transacción, la base de datos puede estar temporalmente en un estado inconsistente. El punto importante aquí es asegurar que la base de datos regresa a un estado consistente al fin de la ejecución de una transacción.



Elaboro: LIA. Blanca García Sánchez

Propiedades fundamentales de una transacción

1. **Atomicidad:** Una transacción se trata como una unidad de operación. Todas las acciones de la transacción se realizan o ninguna de ellas se lleva a cabo. La atomicidad requiere que si una transacción se interrumpe por una falla, sus resultados parciales sean anulados.
2. **Consistencia**: Es simplemente la correctitud. En otras palabras, una transacción es un programa correcto que lleva a la base de datos de un estado consistente a otro con la misma característica. Debido a esto, las transacciones no violan las restricciones de integridad de una base de datos.
3. **Aislamiento**: Una transacción en ejecución no puede revelar sus resultados a otras transacciones concurrentes antes de finalizar. Si varias transacciones se ejecutan concurrentemente, los resultados deben ser los mismos que si ellas se hubieran ejecutado de manera secuencial.
4. **Permanencia**: Es la propiedad de las transacciones que asegura que una vez que una transacción finaliza exitosamente, sus resultados son permanentes y no pueden ser borrados de la base de datos por alguna falla posterior.

Los sistemas manejadores de DB aseguran que los resultados de una transacción sobrevivirán a fallas del sistema. Motiva el aspecto de ***recuperación de base de datos***, el cual trata sobre cómo recuperar la DB a un estado consistente donde todas las acciones que han finalizado con éxito queden reflejadas en la base.

En esencia, lo que se persigue con el procesamiento de transacciones es, por una parte obtener una transparencia adecuada de las acciones concurrentes a una DB y por otra, manejar adecuadamente las fallas que se puedan presentar en una DB.

El procesamiento de transacciones representa una enorme y significativa porción del mercado de los sistemas informáticos y es, probablemente, la aplicación simple más amplia de las computadoras, se ha convertido en el elemento que facilita el comercio electrónico. Es una de las tareas más importantes dentro de un sistema de base de datos, pero a la vez, es una de las más difíciles de manejar debido a diversos aspectos, tales como:

- ✓ **Confiabilidad** Puesto que los sistemas de DB en línea no pueden fallar.
- ✓ **Disponibilidad** Debido a que los sistemas de DB en línea deben estar actualizados correctamente todo el tiempo.
- ✓ **Tiempos de Respuesta** En sistemas de este tipo, el tiempo de respuesta de las transacciones no debe ser mayor a doce segundos.
- ✓ **Throughput** Los sistemas de DB en línea requieren procesar miles de transacciones por segundo.
- ✓ **Atomicidad** En el procesamiento de transacciones no se aceptan resultados parciales.
- ✓ **Permanencia** No se permite la eliminación en la DB de los efectos de una transacción que ha culminado con éxito.

Control de concurrencia en Bases de Datos

Brinda un eficiente desempeño del DBMS, puesto que permite controlar la ejecución de transacciones que operan en paralelo, accesando a información compartida y, por lo tanto, interfiriendo potencialmente unas con otras.

La Concurrencia en las DB es de suprema importancia en los sistemas de información, ya que evita errores en el momento de ejecutar las diferentes transacciones.

Procesamiento de las transacciones

Está íntimamente ligada al procesamiento de las transacciones. La atomicidad se controla con la llegada al **COMMIT**. Si una transacción no sufrió ningún problema y se pudo ejecutar completa, entonces el DBMS debe "comprometerse" a hacer permanentes los

cambios que la transacción hizo sobre la base de datos ya que ésta debió quedar en un estado consistente. La atomicidad también ocurre en caso de que la transacción sufra algún problema que le impida ejecutarse. En este caso, la operación **ROLLBACK** señala el término no exitoso de la transacción, indicando al DBMS que "algo" debió salir mal, advierte que la DB puede estar en un estado inconsistente y que todas las modificaciones, que la transacción haya hecho hasta el momento, deben "retroceder" o anularse.

Rollback de las transacciones

Si una transacción falla, por alguna razón, después de la actualización de la base de datos es necesario "anularla" (**RollBack o UNDO**). Cualquier valor del data ítem que haya sido cambiado por la transacción debe volver a su valor previo.

Si una transacción R ha leído un valor de un data ítem Y que ha sido modificado por S, entonces R también se debe anular y así sucesivamente. Este fenómeno se conoce como **rollback en cascada**, puede ser muy costoso y es por ello que los mecanismos de recovery han catalogado a este fenómeno como indeseable o nunca requerido.

Conceptos de Recovery

La recuperación de las fallas de transacciones significa que la DB se debe restaurar desde algún estado considerado correcto del pasado lo más cercano posible al momento de la falla. El sistema debe mantener la información de todo lo que afecta a la DB. A esto se le llama **Bitácora o system log** las dos principales técnicas de recovery debido a fallas no catastróficas son:

- **Deferred Update** (o actualización diferida), también conocida como NO UNDO/REDO: actualiza la base de datos sólo después de que la transacción ha llegado a su COMMIT. *Antes* de la llegada al COMMIT, todas las modificaciones son guardadas en un área de trabajo de la transacción local. *Durante* el COMMIT, las actualizaciones son guardadas en el system log y luego se guardan en la DB. Si la transacción falla antes de llegar al COMMIT no se guarda ningún cambio en la DB, de manera que no es necesario hacer ningún UNDO. Si la transacción llegó al COMMIT grabó en el system log pero no grabó en la DB, es necesario hacer un REDO, es decir, grabar los efectos de la transacción que se encuentran en el log sobre la base de datos.
- **Inmediate Update** (actualización inmediata) conocida también como UNDO/REDO: esta técnica propone que algunas operaciones actualicen la base de datos antes de que la transacción llegue al COMMIT. Estas actualizaciones son guardadas en el system log a nivel de disco antes de ser aplicadas a la DB. Si una transacción falla después de grabar algunos cambios en la DB y antes de llegar al COMMIT, se debe aplicar un UNDO para anular los efectos sobre la base de datos, es decir, la transacción debe aplicar un ROLLBACK y hacer un REDO para grabar los valores anteriores a la falla. Una variación de este algoritmo UNDO/REDO consiste en permitir que todos los cambios de hace la transacción antes de llegar al COMMIT afecten a la DB y si ocurre una falla sólo se requiere un UNDO de manera que esta técnica también se le conoce como **UNDO/NO REDO**.

Técnicas basadas en actualización diferida

Es postergar o diferir cualquier actualización a la DB hasta que la transacción complete su ejecución exitosamente y llegue a su COMMIT. Durante la ejecución de la transacción las actualizaciones son grabadas en el log y en el área de trabajo de la transacción. Cuando la transacción llega a su COMMIT, el log es forzado a escribirse en disco y luego las actualizaciones se reflejan en la DB. Si la transacción falla antes de llegar a su COMMIT no es necesario aplicar el UNDO a ninguna operación pues la transacción

no ha afectado a la base de datos de ninguna manera. El protocolo típico de la actualización diferida se define como sigue:

1. Una transacción no puede hacer cambios a la base de datos hasta que llegue a su COMMIT.
2. Una transacción no llega a su COMMIT hasta que todas las operaciones de actualización hayan sido registradas en el log y el log haya sido forzado a escribirse en disco.

Si la transacción falla después de llegar a su COMMIT pero antes de que los cambios sean reflejados en la base de datos, basta con aplicar el algoritmo REDO según las operaciones y los valores de los data ítems que aparecen en el log.

Recovery utilizando actualización diferida en ambientes monousuarios

El algoritmo de recovery es relativamente sencillo. El algoritmo conocido como **RDU_S** (recovery using referred update in a single-user environment) usa un procedimiento REDO para rehacer ciertas operaciones de escritura (write-ítem) sobre data ítems de la DB. **PROCEDURE RDU_S** usa dos listas de transacciones: las transacciones que han llegado a su COMMIT desde el último checkpoint y las transacciones activas. Aplica el algoritmo REDO a todas las operaciones que han escrito (write- ítem) sobre los ítems, de las transacciones que han llegado a su COMMIT según el orden en el que ellas aparecen en el log. Luego reinicia todas las transacciones activas. El REDO es definido de la siguiente manera: **REDO (WRITE_OP)** rehacer una operación de escritura sobre un data ítem **WRITE_OP** consiste en examinar sus entradas al log y colocar como valor del data ítem X el valor que aparece en new_value, considerado el valor del data ítem después de la actualización. La operación **REDO** se aplica para buscar lo idéntico. Esto es que, si ella se ejecuta una y otra vez, el resultado debe ser equivalente a que si se ejecutara una sola vez.

Recovery utilizando actualización diferida en ambientes multiusuarios

Para ambientes multiusuarios con control de concurrencia, el recovery puede hacerse más complejo dependiendo del protocolo de control de concurrencia usado. En general, a mayor grado de concurrencia que se desea llegar más difícil se vuelve el proceso de recovery. Para combinar el método de actualización diferida para recovery con la técnica para el control de la concurrencia, es necesario mantener todos los cerrojos sobre los ítems activos hasta que la transacción llegue a su COMMIT. Después de ello, los cerrojos se remueven.

PROCEDURE RDU_M: Usa dos listas de transacciones mantenidas por el sistema: las transacciones que llegaron a su COMMIT (T) desde el último checkpoint y las transacciones activas (T´). Aplica el algoritmo REDO a todas las operaciones que han escrito (write-ítem) sobre los ítems, de las transacciones que han llegado a su COMMIT según el orden en el que ellas aparecen en el log. Las transacciones que están activas y que no han llegado a su COMMIT deben ser canceladas y sometidas nuevamente a ejecución.

Operaciones que no afectan a la base de datos

En general, no todas las operaciones de las transacciones afectan a la base de datos: generación e impresión de mensajes o reportes son ejemplos de ello. Si una transacción falla antes de completarse, el reporte no debe ser tomado en cuenta, esto debe ocurrir sólo después de que la transacción haya llegado a su COMMIT. Esto suele controlarse dejando este tipo de acciones en una cola de **batch Jobs** que son ejecutados sólo si la transacción es exitosa si no lo es se cancelan.

TÉCNICAS BASADAS EN ACTUALIZACIÓN INMEDIATA

UNDO/REDO RECOVERY Utilizando actualización inmediata en ambientes monousuario

PROCEDURE RIU_S usa dos listas de transacciones: las transacciones que han llegado a su COMMIT desde el último checkpoint y las transacciones activas (por lo menos, una transacción cae en esta categoría ges el ambiente es monousuario).
Aplica el **algoritmo UNDO** a Todas las operaciones que han escrito (write-ítem) sobre los ítems, de las transacciones activas que aparecen en el log.
Aplica el **algoritmo REDO** a todas las transacciones que han llegado a su COMMIT según el orden en el que ellas aparecen en el log.

UNDO (WRITE_OP) deshacer una operación de escritura sobre un data ítem WRITE_OP consiste en examinar sus entradas al log [write_ítem,T,X,old_value,new_value] y colocar como valor del data ítem X el valor que aparece en old_value, considerado el BFIM (valor del data ítem antes de la actualización, Before Image).
Deshacer un número de operaciones de escritura de una o más transacciones desde el log implica proceder en orden inverso a como estas operaciones fueron grabadas en el log.

UNDO/REDO RECOVERY Utilizando actualización inmediata en ambientes multiusuarios

PROCEDURE RIU_M usa dos listas de transacciones: las transacciones que han llegado a su COMMIT desde el último checkpoint y las transacciones. Activa el algoritmo UNDO a todas las operaciones que han escrito (write-ítem) sobre los ítems, de las transacciones activas que aparecen en el log. Las operaciones deben ser deshechas en orden inverso a como aparecen en el log. Aplica el algoritmo REDO a todas las transacciones que han llegado a su COMMIT según el orden en el que ellas aparecen en el log.

RECOVERY en transacciones multi-DB

Existen casos de transacciones que trabajan con varias DB, estas son llamadas transacciones multi-DB. Incluso los datos pueden estar almacenados y manejados por DBMS, no sólo relacionales, sino también de redes o jerárquicos. Cada DBMS envuelto en una transacción multi-DB tendrá su propia técnica de recovery y su propio controlador de transacciones separado de los otros DBMS.

Para mantener la atomicidad de una transacción multi-DB es necesario tener un mecanismo de recovery de dos niveles y un manejador global de recovery, llamado a veces, el **coordinador** en adición a los manejadores locales. El coordinador sigue un protocolo llamado protocolo del commit en dos fases. Definido como:

- **Fase 1**: Cuando todas las DB participantes en la transacción han concluido su parte señalan esto al coordinador. Entonces el coordinador envía un mensaje de "prepararse para el commit" a todos los participantes (DBMS´s). Cada participante que recibe el mensaje se ve forzado a escribir su log en el disco y enviar un "listo para el commit" o un "OK" al coordinador. Si la escritura en disco falla o un participante falla, este envía un "no puedo hacer commit" o un "no OK" al coordinador. Si el coordinado no recibe réplica de ningún tipo después de un intervalo de tiempo, asume un "no OK" como respuesta.
- **Fase 2:** Si todos los participantes responden OK la transacción es exitosa y el coordinador envía el commit de la transacción a todos los participantes. Si algún participante respondió "no OK" el coordinador manda un rollback o un UNDO a todos los participantes de la transacción, esto se hace haciendo el UNDO según como lo estipule el log.