RECUPERACIÓN DE CAÍDAS DEL SISTEMA

Siempre que se introduce una transacción T en el SGBD para ejecutarla, éste debe asegurarse de...

a) que todas las operaciones de T se completen con éxito y su efecto quede confirmado permanentemente en la base de datos, o

b) que la transacción no tenga efecto alguno sobre la BD ni sobre cualquier otra transacción.

El SGBD no debe permitir que se apliquen a la BD algunas operaciones de T y otras no. Esto puede suceder si T falla después de realizar alguna de sus operaciones pero antes de ejecutar otras[1].

La **recuperación** en un sistema de base de datos consiste en (volver a) dejar la información de la base de datos en un estado correcto, después de un fallo del sistema (caída) que ha llevado la base de datos a un estado inconsistente, o por lo menos "sospechoso" de serlo.

Los sistemas de base de datos pequeños no suelen proporcionar soporte para la recuperación. Sí lo hacen los grandes sistemas de base de datos (sistemas multiusuario...).

Los problemas de **recuperación** frente a fallos y de **control de la concurrencia** en un sistema de base de datos están muy relacionados con los conceptos del **procesamiento de transacciones**, ya vistos en temas anteriores.

CLASIFICACIÓN DE LOS FALLOS DE UN SISTEMA DE BASE DE DATOS

Existen diversas causas por las que el sistema de bases de datos puede fallar. Algunos tipos de fallos son los siguientes:

1. Errores locales previstos por la aplicación (rollback explícito o programado)
   Durante la ejecución de una transacción pueden presentarse condiciones que requieran la cancelación de la misma (por ejemplo, un saldo insuficiente en una cuenta bancaria implica la cancelación de una transacción de reintegro sobre dicha cuenta).

2. Errores locales no previstos (overflow, división por cero...)
   Error de programación o al provocar una interrupción (ctrl-C en entornos Unix...).

Un fallo local sólo afecta a la transacción que se está ejecutando donde ha ocurrido el fallo. Normalmente supone la pérdida de los datos en memoria principal y en los *buffers* de entrada/salida.

3. Fallos por imposición del control de concurrencia
   El Subsistema de Control de Concurrencia puede decidir abortar una transacción T (para reiniciarla posteriormente) bien porque viola la seriabilidad o porque varias transacciones están en un estado de bloqueo mortal y se ha seleccionado T como víctima.

4. Fallos del sistema (caídas suaves)
   Consisten en un mal funcionamiento del hardware o errores software (del SGBD o del SO) que afectan a todas las transacciones en progreso de ejecución en el sistema de BD. No dañan físicamente el disco (el contenido de la BD permanece intacto y no se corrompe), pero se pierden los datos en la memoria principal y en los *buffers* de entrada/salida.

[1] Esto está muy relacionado con las propiedades de atomicidad y durabilidad de las transacciones.

5. Fallos de disco (caídas duras)

Son fallos en los dispositivos de almacenamiento (por ejemplo, si ocurre una rotura o un aterrizaje de alguna cabeza lectora-escritora en el disco, si funciona mal la lectura o la escritura, o si ocurre un fallo durante una transferencia de datos). Afectan a todas las transacciones en curso y suponen la pérdida de información en disco (es decir, algunos bloques del disco pueden perder sus datos).

6. Fallos catastróficos o físicos

Algunos ejemplos son: corte del suministro eléctrico, robo del disco, incendio, sabotaje, sobreescritura en discos o cintas por error, etc.

Los fallos de tipo 5 y 6 suceden con menos frecuencia y su recuperación es más complicada.

El **módulo** componente del SGBD, encargado de que el sistema de base de datos sea seguro frente a posibles fallos, es el Subsistema **Gestor de Recuperación**, cuya función es velar por, entre otras cosas, que...

- las transacciones no se pierdan (es decir, que se ejecuten),
- las transacciones no se realicen parcialmente (deben ejecutarse en su totalidad),
- no se ejecute una operación más de una vez o, si se hace, que el resultado sea equivalente al obtenido si se hubiera realizado una única vez.

La recuperación de fallos en las transacciones suele equivaler a la restauración de la BD a alguno de sus estados anteriores, de forma que sea posible reconstruir un estado correcto de la BD cercano al momento del fallo, a partir de dicho estado anterior.

La cuestión es que, cuando ocurre un fallo... ¿Cómo podemos recuperar un estado consistente de la base de datos? ¿Cómo deshacer los cambios realizados por las transacciones que quedaron a medio y han dejado la BD en un estado inconsistente?

La recuperación se basa en unos principios bastante sencillos y que pueden resumirse en una sola palabra: **redundancia** (redundancia de información, por supuesto al nivel físico y por tanto de forma transparente al usuario, pues no es visible al nivel lógico).

Dicho de otro modo, la forma de garantizar que el estado de una base de datos es recuperable, es asegurar que cualquier trozo de información que ésta contiene, pueda ser reconstruido a partir de alguna otra información almacenada, de forma redundante, en algún lugar del sistema.

Para hacer posible la recuperación del sistema tras fallos o caídas del mismo, en primer lugar es necesario saber (anotar) cuándo se inicia, se confirma o se aborta cada transacción, y también qué modificaciones sobre qué elementos de la BD realiza cada una de ellas.

Y en segundo lugar, después de ocurrir un fallo, será necesario realizar una serie de acciones para restablecer el contenido de la BD a un estado que asegure la consistencia de la misma, la atomicidad de las transacciones y la durabilidad.

El sistema se mantiene al tanto de las operaciones que realizan las transacciones[2] guardando esta información en un fichero especial (fuera del área de la BD donde se almacenan los datos) llamado bitácora del sistema de BD, también conocido como fichero log, diario o registro histórico. Mediante el fichero de bitácora, por tanto, se sigue la pista a toda operación de transacción que afecta a los valores de los elementos de información de la BD.

[2] Ya se estudiaron las distintas operaciones que podían llevar a cabo las transacciones, así como los estados en los que éstas pueden encontrarse.

LA BITÁCORA DEL SISTEMA

La **bitácora** es un fichero en el que se almacena detalles sobre las operaciones efectuadas como parte de las transacciones. En particular, respecto de las actualizaciones de los elementos de la BD, se puede guardar el valor que tenía antes de la modificación y el que contiene después.

La bitácora se mantiene en disco, así que no le afecta ningún tipo de fallo, salvo (¡claro está!) los de tipo 5 y 6. Además, se suele realizar periódicamente una copia de seguridad (en cinta, por ejemplo) del fichero bitácora, para protegerse contra fallos catastróficos.

Cada registro almacenado en la bitácora se llama **entrada**, y será de uno de los siguientes tipos:

- < INICIAR, T >
  Indica que la transacción T ha comenzado[3].
- < ESCRIBIR, T, X, valor_anterior, valor_nuevo >
  Indica que la transacción T ha cambiado el valor del elemento X de la BD[4]. X contenía el valor_anterior antes de la escritura y contendrá el valor_nuevo después de la escritura.
- < LEER, T, X >
  Indica que T leyó el valor del elemento X de la base de datos
- < COMMIT, T >
  Indica que T finalizó con éxito y su efecto puede ser confirmado en la base de datos en disco (es decir, todos los cambios que ha realizado pueden ser consolidados o asentados en la BD –quedar permanentes– sin posibilidad de ser deshechos posteriormente).
- < ROLLBACK, T >
  Indica que la transacción T ha sido anulada (abortada, cancelada), de forma que ninguna de sus operaciones tendrá efecto sobre la base de datos en disco (como si T nunca se hubiera ejecutado: la transacción será revertida –todas sus operaciones serán deshechas–).

Suponemos que las transacciones no se pueden anidar (¿sabría explicar por qué?).

También suponemos que todos los cambios permanentes de la base de datos ocurren dentro de transacciones, así que recuperarse de un fallo de cierta transacción consiste en rehacer[5] o deshacer[6] individualmente sus operaciones recuperables, a partir del contenido del fichero bitácora.

Por operación recuperable entendemos aquella susceptible de ser deshecha o rehecha. Las operaciones de manipulación de datos (INSERT, UPDATE, DELETE, SELECT...) por ejemplo, son operaciones recuperables. Toda operación recuperable implica la anotación de una entrada en bitácora.

Tras una caída del sistema, se restablece un estado consistente de la base de datos, examinando la bitácora y usando una técnica de recuperación determinada (veremos varias más adelante).

Los protocolos de recuperación que evitan la reversión en cascada[7], no necesitan anotar en bitácora la ejecución de operaciones LEER. Pero otros protocolos sí las necesitan para la recuperación.

Además, algunos protocolos requieren que en las entradas de tipo ESCRIBIR siempre incluyan campos para valor_nuevo y para valor_anterior, mientras que otros protocolos necesitan entradas ESCRIBIR más simples que sólo incluyan el valor_nuevo, o bien, como los protocolos estrictos, que sólo incluyan el valor_anterior.

---

[3] T se refiere a un identificador de transacción único que el sistema genera de forma automática y que identifica a cada transacción.
[4] X es un identificador único del elemento de datos que se escribe. Normalmente suele coincidir con la ubicación del elemento de datos en el disco.
[5] Para rehacer el efecto de una operación ESCRIBIR de T se utiliza la entrada correspondiente en bitácora: se cambia el valor del elemento alterado en dicha operación, por su valor_nuevo.
[6] Es posible deshacer el efecto de una operación ESCRIBIR de T usando la entrada en bitácora correspondiente: se restablece el valor del elemento alterado a su valor_anterior.
[7] Véase tema de Control de la Concurrencia en Sistemas de Base de Datos.

ACCESO A LOS DATOS ALMACENADOS EN LA BASE DE DATOS

En este momento, hemos de recordar algunos conceptos acerca del almacenamiento y el acceso a los datos almacenados en un sistema de base de datos, para comprender cómo pueden ser garantizadas las propiedades de atomicidad y durabilidad de las transacciones.

Cada transacción T posee un **área de trabajo privada** (un espacio de memoria principal y local a dicha transacción) en la que guarda una copia de todo elemento de información accedido y actualizado por T. Dicho área se crea cuando T comienza y se elimina cuando T se confirma o se aborta.

Por otro lado el sistema mantiene un ***buffer* de BD**, que consiste en uno o varios bloques de memoria principal. También se le denomina **memoria intermedia**, y es común a todas las transacciones. En este área de memoria residen temporalmente los bloques de la base de datos, cuando las transacciones necesitan los datos almacenados en ellos. Esta jerarquía de almacenamiento en dos niveles que suele corresponder con el concepto de *memoria virtual*.

Cuando T realiza una operación de **lectura**, si el elemento buscado no está ya en el *buffer* de BD, se trae desde la BD en disco hasta dicho *buffer*. Una vez allí, se copia el valor del dato en el espacio local de la transacción (en una variable local de la transacción, normalmente).

Cuando T realiza una operación de **escritura**, si el elemento que se va a modificar no está ya en la memoria intermedia, se trae desde la BD en disco hasta dicho *buffer*. Una vez allí, se actualiza el valor nuevo del dato en la memoria intermedia, asignándole el valor que tiene la variable local.
Por tanto, las modificaciones realizadas por cada transacción se llevan a cabo en su espacio local y quedan almacenadas en el *buffer* de la BD.

Nótese que ambas operaciones pueden necesitar la transferencia de un bloque desde la BD en disco hasta el *buffer* de BD en memoria, pero ninguna de ellas requiere específicamente que se transfiera un bloque desde memoria principal al disco.

El sistema llevará eventualmente a la BD en disco un bloque de la memoria intermedia o bien porque el gestor de la memoria intermedia necesita ese espacio para otros propósitos, o bien porque el sistema de BD desea reflejar en disco los cambios sufridos por elementos guardados en dicho bloque.

Por tanto, una operación de escritura por parte de una transacción, no tiene por qué provocar la inmediata escritura (del bloque modificado) desde la memoria intermedia hasta el disco, porque ese bloque puede contener otros datos que estén siendo accedidos aún. La escritura real en disco será realizada después por parte del sistema.

Todos estos conceptos serán utilizados y ampliados a lo largo de este tema.

PUNTO DE CONFIRMACIÓN DE UNA TRANSACCIÓN T

- Cuando T termina de ejecutar una operación de COMMIT, significa que:
  - Todas sus **operaciones** de acceso a la base de datos han sido **ejecutadas con éxito**
  - El efecto de tales operaciones se ha **anotado en la bitácora**, incluida una última entrada <COMMIT, T>.

  Así que T **ha llegado a su punto de confirmación** (el final de una unidad lógica de trabajo)[8].

  Y a partir de este momento se puede suponer que:
  - ✓ T está **confirmada**,
  - ✓ Los **cambios** que ha realizado ya han sido **consolidados**. Esto significa que se ha de suponer que se han llevado a la BD, y por tanto ya son permanentes; hasta ese momento los cambios eran provisionales y podían ser anulados, pero una vez confirmados nunca pueden ser deshechos.
  - ✓ Los cursores abiertos han sido cerrados, y los bloqueos[9] han sido liberados.

- Cuando T ejecuta una operación de ROLLBACK, significa que T ha sido cancelada, de forma que el sistema debe anular todas sus operaciones (de actualización de datos), para llevar la base de datos al estado en el que estaba justo antes de que se iniciara T.

Nota1: una operación COMMIT o ROLLBACK finaliza una transacción, no un programa. La ejecución de un programa suele consistir en una secuencia de transacciones ejecutadas una tras otra.

Nota2: el punto de confirmación es un concepto lógico; no queda reflejado físicamente como tal en ningún lugar.

EL PROCESO DE RECUPERACIÓN DEL FALLO DE UNA TRANSACCIÓN

Supongamos que se están ejecutando varias transacciones de forma concurrente. Dichas transacciones habrán realizado modificaciones sobre elementos de la BD, algunas de las cuales pueden haberse llevado a disco.
Cuando el sistema se cae, el fallo puede ocurrir en un momento en el que algunas de las transacciones estén en curso (ejecutándose) y otras transacciones pueden haber terminado ya.

Está claro que **una vez que el sistema es reiniciado**, **deberán ser anuladas**[10] **aquellas transacciones** T **que han quedado a medio**, puesto que no alcanzaron su punto de confirmación (son aquellas que no llegaron a anotar su <COMMIT, T> en la bitácora).

**Nótese que para que sea posible deshacer cambios realizados sobre elementos de BD, es necesario que en el fichero de bitácora existan las entradas correspondientes a dichos cambios.

¿Y qué ocurre con las transacciones que habían finalizado cuando ocurrió el fallo?
Podríamos pensar algo como "estas transacciones sí han llegado a su punto de confirmación, luego podemos darlas por terminadas con éxito, y 'olvidarnos' de ellas...".
Sin embargo, no es seguro que estas transacciones estén confirmadas realmente, pues es posible que algún cambio sobre un elemento de la BD todavía no se haya llevado a la BD en disco.
Esto ocurre si el fallo tiene lugar mientras se están grabando en disco los elementos de la BD modificados: algunos cambios pueden quedar grabados en la BD en disco, pero otros no.

[8] Corresponde con el estado CONFIRMADA de T (véase tema de Conceptos de Procesamiento de Transacciones)
[9] Véase tema de Control de la Concurrencia en Sistemas de Base de Datos.
[10] revertidas, rolledback

Por esto, **una vez reiniciado el sistema, es necesario** rehacer **las modificaciones de las transacciones que terminaron con éxito**, para asegurar que quedan realmente consolidadas en disco (aquellas que sí llegaron a escribir <COMMIT, T> en la bitácora).

***Nótese que para que sea posible rehacer cambios, es necesario que en el fichero de bitácora, existan las entradas correspondientes a los mismos.

Por un lado, hemos de tener en cuenta que **la bitácora es un fichero en disco**. Como ya sabemos, actualizar un fichero en disco implica:

1. copiar el bloque adecuado del fichero en disco a un *buffer* de la memoria principal,
2. actualizar el bloque en el *buffer* (insertar una nueva entrada, por ejemplo) y
3. copiar el bloque desde el *buffer* al disco.

Esto supondría una escritura en disco cada vez que se inserta una nueva entrada en la bitácora.

Para conseguir una única escritura por bloque, en la práctica...
- se mantiene un bloque completo de **bitácora en memoria principal** (*buffer* de bitácora), y
- cuando se llena de entradas, el bloque completo se escribe en disco (fichero bitácora)

Con esto se evita escribir varias veces en disco un mismo bloque de fichero, pero provoca otros **problemas**, que ilustramos a continuación mediante ejemplos.
Supongamos un par de transacciones en ejecución, T1 y T2. Ambas realizan actualizaciones sobre elementos de la BD. Estos cambios se anotan sólo en el *buffer* de bitácora en memoria principal:

<ESCRIBIR,T2,X,7,15>,
<ESCRIBIR,T1,Y,2,5>, etc.

Supongamos también que algunos de estos cambios se consolidan en la BD en disco[11], por ejemplo estos dos anteriores: en la BD se tiene por tanto X=15 e Y=5.

T1 realiza dos cambios más sobre los elementos Z y V, y después finaliza con éxito, luego anota, todavía sólo en el *buffer* de bitácora, las entradas:

<ESCRIBIR,T1,Z,6,3>
<ESCRIBIR,T1,V,8,1> y
<COMMIT, T1>.

El COMMIT de T1 indica al sistema que puede confirmar en disco los cambios (hechos por T1) aún no llevados a la BD, es decir los que modifican los valores Z de 6 a 3 y V de 8 a 1.

Supongamos ahora que el sistema comienza a escribir en la BD en disco los elementos Z y V modificados por T1, pero, recordemos, estos cambios todavía no se han anotado en la bitácora en disco (sino que las entradas correspondientes siguen en el *buffer* de bitácora, que no está lleno aún).

Si el fallo ocurre justo durante la consolidación de los cambios de la BD en disco, entonces es posible que algunos cambios de T1 sí se graben en disco (en BD queda Z=3), pero otros no (en BD sigue V=8). Recordemos que T2 no había alcanzado su punto de confirmación.

Cuando el sistema es reiniciado, el proceso de recuperación accede al fichero de bitácora en disco. Lo más normal es que el fallo haya provocado la pérdida del contenido de la memoria principal, así que no encontrará ninguna de esas entradas del *buffer* de bitácora que todavía no se habían volcado en disco cuando ocurrió el fallo.

[11] Posteriormente se verá que esto es posible en algunas técnicas de recuperación y en otras no.

En particular no encontrará la entrada <COMMIT, T1> ni <COMMIT, T2>, así que intentará anular T1 y T2.

Sin embargo T1 no debe ser anulada, pues se realizó completamente y con éxito. Lo único que ocurre es que no se completó la confirmación en disco de sus cambios. Tan sólo se debería rehacer cada una de sus operaciones.

Este error se hubiera evitado si, antes de comenzar la consolidación de cambios de BD en disco, se hubiera grabado en disco (y con éxito, claro) la porción de bitácora que sólo estaba en el *buffer*: el sistema hubiera detectado la confirmación de T1 (pues en la bitácora en disco estaría la entrada de COMMIT T1), y además sí sería posible rehacer las modificaciones de T1, porque estarían en bitácora (en disco) todas las entradas <ESCRIBIR, T1,...>[12].

Pero aún existe un problema adicional. Decíamos que el sistema intentará anular T1 y T2.
Dos de las entradas perdidas a causa del fallo son
<ESCRIBIR,T2,X,7,15> y
<ESCRIBIR,T1,Y,2,5>

así que no se tiene anotado (en fichero bitácora) ni el valor_anterior ni el valor_nuevo para X ni para Y.
Pero ambos cambios fueron llevados a la BD en disco, de forma que X e Y ya contienen sus nuevos valores 15 y 5. No es posible deshacer estas operaciones de forma correcta.
Otra entrada perdida es <ESCRIBIR,T1,Z,6,3>, que corresponde a otro cambio de T1 que sí ha dado tiempo a consolidar en disco antes del fallo. Pero tampoco en este caso se puede deshacer esta operación, y por la misma razón que antes.

Vemos que es imposible recuperar la base de datos a un estado de consistencia anterior.

Todo esto también se hubiera evitado si, antes de comenzar la grabación de cambios de BD en disco, se hubiera grabado en disco la porción de bitácora que sólo estaba en el *buffer*: sí sería posible deshacer las modificaciones de T1 y T2 , porque estarían en bitácora (en disco) todas las entradas <ESCRIBIR,...>[13] necesarias.

En realidad, lo que hemos hecho es razonar que, para evitar estos problemas, es necesario seguir un protocolo de bitácora adelantada., puesto que asegurará que la base de datos podrá ser recuperada de forma correcta.

[12] Por esto último se afirmó la frase marcada con *** anteriormente.
[13] Por esto se afirmó la frase etiquetada con ** anteriormente.

BITÁCORA ADELANTADA

Este protocolo indica que ...

1. No se puede grabar en disco los cambios realizados por una transacción T[14], hasta que se haya forzado la escritura en disco de toda entrada de bitácora para T[15], hasta el momento actual.

2. La operación confirmar de una transacción no se puede completar hasta que se haya forzado la escritura en disco de cualquier entrada de bitácora para esa transacción[16].

Es decir fuerza la escritura de la bitácora en disco antes de consolidar cualquier cambio realizado por T. Ya sea para confirmar en disco cambios antes de que T alcance su punto de confirmación, o para confirmarlos después de que T lo haya alcanzado[17].

Así que **nunca** va a ocurrir que 1º) se lleven a disco los cambios de elementos de BD y 2º) el sistema falle al ir a grabar en disco la parte de bitácora.

Pero **sí** puede ocurrir que 1º) se grabe físicamente la parte de la bitácora de una modificación y 2º) el sistema falle cuando va a aplicar el cambio a la BD en disco.

En este último caso, al reiniciar el sistema, el Gestor de Recuperación verá que la modificación pudo hacerse o no físicamente, pues está registrada en bitácora. Podrá deshacerla o rehacerla usando la entrada correspondiente en bitácora.

Con el uso del protocolo de bitácora adelantada, siempre estamos seguros de que podemos llevar la base de datos a un estado consistente, aunque ocurra un fallo mientras se graban en disco los cambios de elementos de BD, pues será posible rehacer y/o deshacer los cambios.

PUNTOS DE VALIDACIÓN DEL SISTEMA

Retomemos ahora el proceso de recuperación, tras una caída del sistema de BD.
Al rearrancar el sistema el Gestor de Recuperación accede al fichero de bitácora (en disco). En él están anotadas todas las entradas correspondientes a modificaciones sobre elementos de la base de datos, realizadas por las transacciones. Para aquellas transacciones que alcanzaron su punto de confirmación, también estarán anotadas las entradas COMMIT adecuadas.

Así que el Gestor de Recuperación del sistema de BD lleva a cabo las siguientes operaciones:

1. DESHACER (undo) las transacciones activas, es decir las que estaban en progreso cuando ocurrió el fallo
2. REHACER (redo) las transacciones que alcanzaron su punto de confirmación antes de la caída, pero (quizás) no tuvieron tiempo de transferir sus actualizaciones desde los *buffer*s de BD a la BD física.

¿Cómo sabe el sistema qué transacciones estaban "activas" y debe deshacer?
Y ¿cómo sabe el sistema cuáles debe rehacer?

[14] Es decir, el valor de un elemento de BD no se puede reescribir con su nuevo valor (resultado de una modificación)...
[15] En particular toda entrada ESCRIBIR que incluya el valor_anterior; además de toda entrada LEER, si es posible la reversión en cascada.
[16] En particular toda entrada ESCRIBIR que incluya tanto valor_anterior como valor_nuevo; además, si es posible la reversión en cascada, toda entrada LEER.
[17] Esto dependerá de la técnica de recuperación empleada por el SGBD. Se verá más adelante.

Pues el sistema debe examinar la bitácora desde el principio, y rastrearla hacia adelante para determinar que ...

a) T estaba activa si ha escrito una entrada <INICIAR,T>, pero no ha escrito ninguna entrada <COMMIT,T> ni <ROLLBACK, T>, y que
b) T alcanzó su punto de confirmación si ha escrito <COMMIT, T> en bitácora.

Es posible evitar tener que recorrer *toda* la bitácora, en busca de las transacciones activas (para anularlas (deshacer sus operaciones) o descartarlas, para reiniciarlas posteriormente).

También se puede evitar tener que rehacer *todas* las transacciones confirmadas.

Ambas cosas se consiguen usando puntos de validación (también llamados puntos de revisión o puntos de control).

Un punto de validación es otro tipo de entrada en la bitácora. El sistema establece (marca) un punto de revisión de forma automática y periódica. En particular, es el Gestor de Recuperación el que establece cuándo marcar un punto de validación. Por ejemplo cada m minutos, o cuando se han grabado en el *buffer* de bitácora un número t de entradas del tipo <COMMIT, T> desde el último punto de validación[18].

Marcar un punto de revisión implica lo siguiente:

1. Suspender temporalmente la ejecución de todas las transacciones en curso.
2. Escribir en disco todas las entradas de bitácora que residan en ese momento en el *buffer* de bitácora (en memoria).
3. Forzar la escritura en disco de todos los bloques del *buffer* de BD que se hayan modificado (es decir, se llevan a disco todas las actualizaciones realizadas[19]).
4. Escribir una entrada especial (registro de validación) en la bitácora en disco.

   El registro de validación contiene:
   - Lista de identificadores de las transacciones activas (en el instante del punto de validación)
   - Dirección dentro del fichero bitácora, de la primera y última entradas correspondientes a cada transacción activa (para rapidez de acceso si es necesario anular una transacción)
5. Escribir en un fichero especial de arranque del sistema, la dirección física del registro de validación dentro del fichero bitácora. El Gestor de Recuperación accede a este fichero especial de arranque cuando el sistema es reiniciado tras el fallo.
6. Reanudar la ejecución de transacciones.

Así pues, cuando ocurra un fallo sólo será necesario examinar la bitácora desde el último punto de revisión hacia adelante, y toda transacción cuya entrada <COMMIT, T> esté en la bitácora antes de dicho punto de validación no deberán rehacer sus operaciones ESCRIBIR, pues ahora estamos seguros de que fueron consolidadas por completo (ya que en el punto de revisión se forzó la escritura en disco de tales actualizaciones).

Nota: el que marcar un punto de revisión implique la escritura en disco, no quiere decir que no existan otros momentos en los que también se consoliden cambios en disco, aparte de en estos puntos de validación (¿Sabría indicar cuáles serían esos otros momentos?).

[18] Los valores m y t son parámetros del sistema de base de datos.
[19] NOTA IMPORTANTE: Como se verá más adelante, dependiendo de la técnica de recuperación que utilice el SGBD, las modificaciones llevadas a la BD desde el *buffer* de BD en un punto de validación serán **a)** sólo las realizadas por transacciones T ya confirmadas (*técnica de actualización diferida de la BD*) o **b)** todas, independientemente de si la T que las hizo está o no confirmada (*técnica de actualización inmediata de la BD*).

Por otro lado, recordemos ahora que si una transacción T1 aborta (y, por tanto, es anulada), cualquier otra T2 que haya leído elementos escritos por T1, también debería ser anulada. Esto es la **reversión en cascada** y puede ocurrir si el protocolo de recuperación garantiza planes recuperables, pero no que sean estrictos o sin cascada.

La reversión en cascada consume mucho tiempo, y por esto casi todos los mecanismos de recuperación se diseñan de forma que nunca sea necesaria la reversión en cascada.

Nótese que para revertir transacciones es necesario deshacer las operaciones ESCRIBIR, y que las entradas tipo LEER se anotan en la bitácora para determinar si es necesaria la reversión de transacciones. Por tanto, si el método de recuperación garantiza que nunca ocurrirá reversión en cascada, no será necesario anotar en bitácora entradas correspondientes a lecturas de elementos.

TÉCNICAS DE RECUPERACIÓN DE CAÍDAS DEL SISTEMA DE BD

A continuación describiremos algunas técnicas de recuperación. No son descripciones de cómo un sistema específico implementa la recuperación, sino que nuestra intención es describir conceptualmente varias estrategias distintas para la recuperación de caídas del sistema de BD.

Con frecuencia las técnicas de recuperación están "ligadas" con los mecanismos de control de la concurrencia. Es decir, algunas técnicas funcionan mejor con unos métodos de control de concurrencia que con otros. Con todo, analizaremos en primer lugar los conceptos de recuperación sin tener en cuenta los mecanismos de control de la concurrencia, es decir considerando un sistema monousuario, y después indicaremos las circunstancias en las que, si se emplea un protocolo de control de concurrencia determinado, conviene usar un mecanismo específico de recuperación.

Una estrategia de recuperación representativa podría resumirse informalmente de la siguiente manera:

- Tras un **fallo en el sistema (de tipo 5 o 6)** que ha producido daños en una parte considerable de la BD, el método de recuperación consistirá en restaurar una copia de seguridad anterior de la BD y reconstruir un estado más actualizado, rehaciendo las operaciones de las transacciones confirmadas anotadas en la bitácora hasta el momento de la caída.

  Es conveniente, por tanto realizar periódicamente una copia de seguridad (en cinta, por ejemplo) del contenido de la base de datos.

  Además se debe hacer, con mayor frecuencia, una copia de respaldo del fichero bitácora para no perder los efectos de las transacciones ejecutadas desde la última copia de seguridad (backup) de la BD.

- Si el **fallo es no catastrófico (de los tipos 1 a 4)** la estrategia consistirá en invertir los cambios que provocaron la inconsistencia, es decir deshacer algunas operaciones. También puede ser necesario rehacer algunas operaciones para restaurar un estado consistente de la BD.
  En este caso, sólo se necesita consultar las entradas anotadas en la bitácora.

En ambos casos, consideramos que el fichero bitácora en disco (log) no sufre ningún daño.

Dos tipos fundamentales de **técnicas de recuperación de fallos no catastróficos** son las técnicas basadas en la *actualización diferida* y en la *actualización inmediata*, que vamos a ver a continuación.

TÉCNICA DE ACTUALIZACIÓN DIFERIDA DE LA BASE DE DATOS

Si el sistema sigue esta técnica, una transacción T no actualiza la BD hasta después de que T completa con éxito su ejecución y llega a su punto de confirmación; es decir, se difiere la grabación de actualizaciones en la BD hasta la confirmación.

Antes de ser confirmada T, todas sus actualizaciones se realizan en el espacio de trabajo local de dicha transacción (en memoria) y son anotadas en el *buffer* de bitácora (también en memoria). Una vez que T llega a su punto de confirmación, primero se fuerza la escritura en disco de la bitácora, y después se escriben sus actualizaciones en la BD en disco[20].

- Si T falla antes de llegar a su punto de confirmación, no habrá modificado la BD en absoluto, por lo que no es necesario deshacer ninguna operación.
- Si T falla una vez alcanzado su punto de confirmación, habrá que rehacer T (a partir de las entradas de bitácora) porque no es seguro que todos sus cambios se hayan llevado a disco.

Este protocolo, por tanto, quedaría definido con los siguientes puntos:

a) Una transacción no puede modificar la BD **antes** de llegar a su punto de confirmación.
b) Una transacción no llega a su punto de confirmación **antes de anotar** todas sus operaciones de actualización en bitácora y de **forzar** la escritura de la bitácora en disco. Una vez se ha hecho esto, la actualización real tiene lugar, y la transacción pasa al estado CONFIRMADA.

Si el sistema sigue esta técnica, una vez reiniciado el mismo, el Gestor de Recuperación ejecutará el **algoritmo de recuperación NO DESHACER / REHACER**:

1. Crear dos listas de transacciones[21], llamadas ACTIVAS y CONFIRMADAS, ambas vacías.
2. Inicializar ACTIVAS a la lista de transacciones activas que aparecen en el registro de validación (correspondiente al último punto de revisión) escrito en la bitácora.
3. Examinar la bitácora a partir del último punto de revisión hacia adelante.
4. Si se encuentra una entrada <INICIAR,T>, se añade T a la lista ACTIVAS.
5. Si se encuentra una entrada <COMMIT,T>, mover T de la lista ACTIVAS a la lista CONFIRMADAS.
6. Al terminar de examinar la bitácora,
   - REHACER las operaciones ESCRIBIR de las transacciones de la lista CONFIRMADAS, en el mismo orden en que se escribieron en bitácora,
   - REINICIAR las transacciones de la lista ACTIVAS (transacciones en curso, no confirmadas)

El sistema está ahora preparado para aceptar nuevos trabajos (la BD es consistente).

Rehacer una operación ESCRIBIR realizada por T consiste en examinar su correspondiente entrada en bitácora <ESCRIBIR,T,X,valor_nuevo> y asignar valor_nuevo al elemento X de la BD.
Esta operación rehacer debe ser idempotente (es decir, ejecutarla varias veces debe equivaler a ejecutarla una sola vez[22]).
Nótese que en las entradas de bitácora tipo ESCRIBIR sólo es necesario almacenar los valores nuevos y no los anteriores, puesto que nunca se requiere deshacer operaciones pero sí rehacerlas.

---

[20] La información de bitácora asociada a esta transacción se utiliza para el ejecución de las escrituras diferidas.
[21] Está claro que dichas listas no almacenarán transacciones en sí, sino los identificadores de dichas transacciones, representados aquí con la letra T.
[22] En realidad TODO el proceso de recuperación ha de ser idempotente, puesto que si el sistema falla durante el proceso de recuperación, el segundo intento de recuperación podría rehacer operaciones ESCRIBIR que ya hubieran sido restauradas durante el primer intento.

Y reiniciar una transacción significa reintroducirla en el sistema, como si fuera nueva. Puede hacerlo el sistema de forma automática, o el usuario manualmente.

Si el **sistema es monousuario**, en la lista ACTIVAS habrá como máximo una transacción.

Si el **sistema es multiusuario**, dependiendo de los protocolos de **control de la concurrencia** empleados, el proceso de recuperación puede ser más complejo. Cuando mayor sea el grado de concurrencia, más complicada será la tarea de recuperación:

Consideramos un sistema en el que la bitácora incluye puntos de validación y el control de la concurrencia utiliza bloqueo de dos fases y evita el bloqueo mortal haciendo que una transacción T obtenga, antes de comenzar a ejecutarse, todos los bloqueos de los elementos que necesita.

Para combinar el método de recuperación por actualizaciones diferidas con esta técnica de control de concurrencia, podemos mantener todos los bloqueos de los elementos hasta que T llegue a su punto de confirmación.
Después de ello, los bloqueos pueden liberarse.
Esto garantiza planes estrictos y seriables.

Una desventaja de este método es que limita la ejecución concurrente de las transacciones, pues los elementos permanecen bloqueados hasta que T llega a su punto de confirmación.

Pero su principal ventaja es que nunca es necesario deshacer operaciones, porque...

1) Una transacción no modifica la BD hasta que llega a su punto de confirmación, luego ninguna transacción se revierte por haber fallado durante su ejecución.

2) Una transacción T1 nunca lee un elemento modificado por otra transacción T2 no confirmada, porque los elementos permanecen bloqueados por T2 hasta que ésta alcanza su punto de confirmación, luego nunca hay reversión en cascada.

Nota: Un buen **ejercicio** para el alumno sería pensar en los cambios que sería necesario realizar en el algoritmo anterior (NO DESHACER/REHACER, y también en los que se verán a continuación), considerando que el sistema **no** utiliza puntos de validación o control.

TÉCNICA DE ACTUALIZACIÓN INMEDIATA DE LA BASE DE DATOS

Si el sistema utiliza esta técnica, es posible que algunas operaciones de una transacción T actualicen la BD en disco **antes** de que T llegue a su punto de confirmación (sin tener que esperar a que T finalice con éxito). Estas actualizaciones suelen ser denominadas *modificaciones no comprometidas.*
Recordemos que, para que sea posible la recuperación, estas actualizaciones deben ser anotadas en la bitácora en disco antes de ser aplicadas a la BD (protocolo de bitácora adelantada).

- Si T falla después de haber llevado algunos cambios a la BD, pero antes de llegar a su punto de confirmación, será necesario anular el efecto de esas operaciones sobre la BD: T debe ser abortada (rolledback) y hay que deshacer sus operaciones.
- Y, por el mismo motivo que en la técnica anterior, si T falla después de alcanzar su punto de confirmación será necesario rehacer sus operaciones.

Si el sistema sigue esta técnica, una vez reiniciado el mismo, el Gestor de Recuperación ejecutará el **algoritmo de recuperación DESHACER / REHACER**:

1. Crear dos listas de transacciones, llamadas ACTIVAS y CONFIRMADAS, ambas vacías.
2. Inicializar ACTIVAS a la lista de transacciones activas que aparecen en el registro de validación (correspondiente al último punto de revisión) escrito en la bitácora.
3. Examinar la bitácora a partir del último punto de revisión hacia adelante.
4. Si se encuentra una entrada <INICIAR,T>, se añade T a la lista ACTIVAS.
5. Si se encuentra una entrada <COMMIT,T>, mover T de la lista ACTIVAS a CONFIRMADAS.
6. Al terminar de examinar la bitácora,
   - DESHACER, con base en la bitácora, las operaciones ESCRIBIR de las transacciones en ACTIVAS. Deben deshacerse en orden inverso a aquel en que se anotaron en bitácora,
   - REHACER, con base en la bitácora, las operaciones ESCRIBIR de las transacciones en CONFIRMADAS, en el mismo orden en que se escribieron en bitácora.

El sistema está ahora preparado para aceptar nuevos trabajos (la BD es consistente).

Deshacer una operación ESCRIBIR realizada por T consiste en examinar su entrada en bitácora <ESCRIBIR,T,X,valor_anterior,valor_nuevo> y asignar valor_anterior al elemento X de la BD. Esta operación deshacer también debe ser idempotente.

----- -----

Una **variación** de esta técnica obliga a que, para que pueda decirse que T ha llegado a su punto de confirmación, todas las actualizaciones realizadas por T deben haberse grabado en la BD.

En este caso, si T falla una vez alcanzado su punto de confirmación, no se necesitará rehacer nada.

Si el sistema sigue esta técnica, una vez reiniciado el mismo, ejecutará el **algoritmo de recuperación DESHACER/NO REHACER**, cuyos pasos se dejan como ejercicio para el alumno.

------ -----

Si el **sistema es monousuario**, en la lista ACTIVAS habrá como máximo una transacción.

Si el **sistema es multiusuario** y se permite la **ejecución concurrente,** el proceso de recuperación también depende de los protocolos empleados para el control de la concurrencia:

Consideramos un sistema en el que la bitácora incluye puntos de validación y el control de la concurrencia usa bloqueo de dos fases estricto, el cual sólo permite que una transacción T lea o escriba un elemento si la última transacción que lo ha escrito ya se ha confirmado. Esto garantiza planes estrictos, pero pueden producirse bloqueos mortales, por lo que se necesita deshacer transacciones.

RECUPERACIÓN DE TRANSACCIONES DE MÚLTIPLES BASES DE DATOS

Hasta ahora hemos considerado que una transacción sólo tiene acceso a una base de datos. En algunos casos, una transacción puede necesitar acceso a varias bases de datos.

De hecho, en los sistemas llamados de múltiples bases de datos (o multibase de datos), una transacción puede acceder a las BD de varios SGBD diferentes e independientes. Cada SGBD podría tener su propio gestor de transacciones, su propia técnica de recuperación[23]...

Supongamos una transacción de multibase de datos T, que modifica datos en una base de datos IMS y también en otra base de datos DB2.

a) Si T acaba con éxito, debe realizar COMMIT de todas sus actualizaciones (tanto en el sistema IMS como en el sistema DB2)

b) Si T debe ser anulada, es necesario hacer ROLLBACK de todas sus actualizaciones (en ambos sistemas IMS y DB2)

Es decir, NO es posible que T realice un COMMIT en IMS y un ROLLBACK en DB2, ni tampoco puede suceder que T haga un ROLLBACK en IMS y un COMMIT en DB2, pues en estos casos la transacción NO sería atómica.

Es necesario, por tanto, que T acabe con la misma instrucción en ambos sistemas, es decir con dos COMMIT o dos ROLLBACK, uno en cada sistema.

Pero, aunque así sea ¿quién controla lo que ha ocurrido en el otro sistema?
Si ocurriera un fallo a la mitad de la transacción (es decir, cuando T ya ha hecho COMMIT en el sistema IMS, y está a la mitad de su ejecución en DB2) tendríamos la BD inconsistente, pues se anularía en DB2 ¡pero los cambios ejecutados en IMS ya serían permanentes!
La transacción se habría ejecutado parcialmente: violación de la atomicidad de T.

La solución es utilizar un mecanismo de recuperación de dos niveles, que necesita (además de los gestores de recuperación locales) un **coordinador** o Gestor de Recuperación global.
El coordinador seguiría un **protocolo de confirmación en dos fases**, para garantizar que los gestores de recuperación locales (de IMS, DB2...) hacen COMMIT o ROLLBACK de las actualizaciones (las realizadas en cada sistema) al unísono, aunque el sistema falle a mitad del procesamiento de la transacción en alguno de los sistemas participantes.

Supongamos que la transacción T finaliza con éxito, lo cual implica la ejecución de un COMMIT en todos los sistemas a los que accede. Cada sistema, por tanto, envía al coordinador un mensaje indicando que la parte de T en la que participa dicho sistema, ha acabado sin problemas.

[23] Esto es similar al caso de los Sistemas de base de datos Distribuidos, donde una transacción puede interactuar con (acceder a datos almacenados en) varias fuentes de datos diferentes, puesto que la base de datos está dividida en partes que residen en distintos sitios conectados en red.

PROTOCOLO DE CONFIRMACIÓN EN DOS FASES (two-phases commit)

Fase 1.

a. Cuando todos los SGBD participantes indican al coordinador que la parte de T en la que interviene cada uno ha acabado con éxito, el coordinador envía a cada participante el mensaje "prepárense para confirmar la transacción T".

b. Cada sistema participante fuerza la escritura en disco de todas las entradas de la bitácora (local). De este modo cada sistema consigue un registro permanente de las actualizaciones de T, así será capaz de consolidarlas o de anularlas, según sea necesario.

c. Si esta escritura se lleva a cabo sin problemas, cada sistema envía un mensaje de "ok" o "listo para confirmar T" al coordinador. Si falla la escritura forzada o la transacción local no puede confirmarse por alguna razón, el sistema participante envía una señal de "ko" o "imposible confirmar T".

   Si tras un tiempo de espera, el coordinador no recibe nada desde algún sistema, supone que su respuesta es "ko".

Fase 2.

a. Una vez que el coordinador ha recibido todos los mensajes, graba en su bitácora un registro de decisión (el cual contiene todos los mensajes recibidos) y después fuerza la escritura en disco de su bitácora.

b. Si todos los mensajes eran de tipo "ok", T tiene éxito y el coordinador envía una señal "confirmar T" a todos los sistemas participantes, para que cada uno escriba en su bitácora una entrada <COMMIT, T> y si es necesario consolide los cambios de T en su BD (es posible recuperarse de un fallo pues todos los efectos locales de T se han anotado en las bitácoras de las BD participantes).

   Si alguno de los mensajes era de tipo "ko", T ha fallado y el coordinador envía "anular T" a todos los sistemas participantes, para que cada uno deshaga las actualizaciones de T individualmente, usando su propia bitácora.

Ahora, cuando cae el sistema, el procedimiento de recuperación (al rearrancar) es el siguiente:

1. El coordinador accede al registro de decisión en su propia bitácora.
2. Si lo encuentra, el proceso sigue por donde se quedó (segunda fase), es decir, según los mensajes que el coordinador recibió, éste enviará "confirmar" o "anular" a todos los sistemas participantes.
3. Si no lo encuentra (el error tuvo lugar antes de la segunda fase), el coordinador envía "anular" a todos los sistemas participantes.