Conceptos de Recuperación de Información en Bases de Datos

[EN - 21]

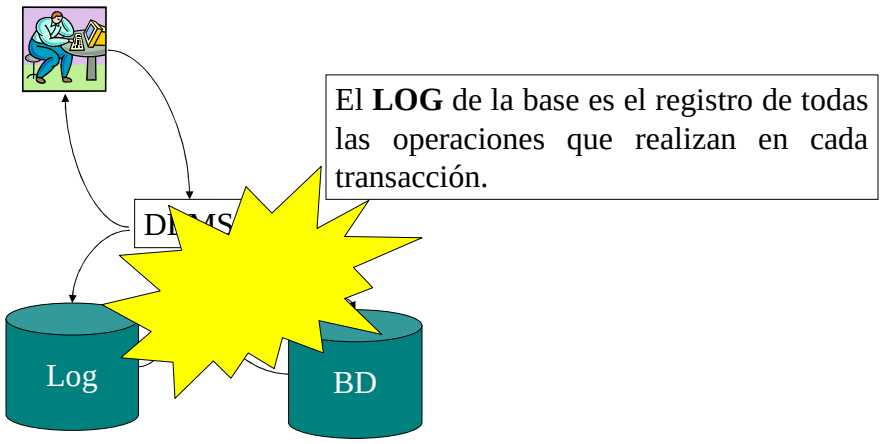
De que hay que Recuperarse?

- En un sistema, se pueden dar fallas que pongan en riesgo la integridad y la existencia misma de la base y por lo tanto de los datos.
  - Fallas en la CPU: Ej. falla en la memoria.
  - Error en la transacción o del sistema: Ej. División por 0
  - Excepciones sin programar. Ej. no están los datos necesarios para realizar la transacción.
  - Por el control de concurrencia. Ej. Para evitar deadlock.
  - Falla en el disco.
  - Catástrofes. Ej. Incendios, robo, etc.
- En cualquier caso, el DBMS debe poder recuperar un estado consistente y conocido de la base.

Conceptos de Recuperación de Información: Temario

- Conceptos Basicos
  - Escenario de Trabajo.
  - El Log
  - Algoritmos de Recuperación
  - Checkpoint
  - Rollback
- Actualización Diferida
- Actualización Inmediata
- Shadow Pages

Escenario de Trabajo



El Log del DBMS

- Es un registro de la actividad sobre la base, típicamente con entradas como las siguientes:
  - **[start_transaction,T]**: Comenzó la transacción T
  - **[write_item,T,X,v_ant,v_actual]**: La transacción T cambió el item X del valor v_ant al valor v_act.
  - **[read_item,T,X]**: La transacción T leyó el valor del item X
  - **[commit,T]**: La transacción T confirma que todos los efectos deben ser permanentes en la base.
  - **[abort, T]**: La transacción T abortó, por lo que ningún cambio realizado por T debe ser permanente en la base

Construcción y Uso del Log.

- Cada vez que una transacción va a realizar una operación en la base, se agrega un registro en el log.
- Típicamente esta actualización se hace en un buffer en memoria, por lo que cuando una transacción T confirma:
  - Primero se baja a disco todo el log de T que ya no esté guardado.
  - Luego se actualiza la base con los efectos de la transacción.
- Nunca se debe perder a la vez la base y el log por lo que:
  - Deben estar en discos distintos
  - Deben realizarse respaldos independientes de uno y otro.

Algoritmos de Recuperación

- Si hubo un desastre (se pierde el disco, etc.) entonces:
  - Se debe recuperar el último respaldo conocido de la base.
  - Se debe recuperar el LOG hasta donde se pueda
  - Se deben rehacer (redo) todas las operaciones indicadas en el log desde el momento en que se hizo el respaldo a la base.
- Si la falla fue menor (corte de energía, falla en la red, etc):
  - puede ser que haya que deshacer (undo) cambios ya realizados.
  - puede ser que haya que rehacer cambios que no se hayan confirmado.

Algoritmos de Recuperación: Actualización Diferida e Inmediata

- Si la falla no fue catastrófica, hay dos técnicas básicas:
  - **Actualización Diferida**: Cada transacción trabaja en un área local de disco o memoria y recién se baja al disco después que la transacción alcanza el commit.
    - Si hay un abort o una falla, no es necesario deshacer ninguna operación (NO-Undo/Redo).
  - **Actualización Inmediata**: La base es actualizada antes de que la transacción alcance el commit.
    - Si hay un abort o falla, se deben deshacer las operaciones de la transacción.
- Siempre se graba primero el Log para garantizar la recuperación.

BeFore IMage y AFter IMage

- Para que el mecanismo de recuperación sea viable, es necesario considerar al menos dos valores para cada item:
  - **Before Image (BFIM)**: El valor del ítem antes de ser actualizado por una transacción.
  - **After Image (AFIM)**: El valor del ítem después de ser actualizado por una transacción.
- De esta forma, los registros del Log pueden estar clasificados en:
  - De Undo: Contienen la operación y el BFIM
  - De Redo: Contienen la operación y el AFIM
  - Combinados: Contienen la operación y los dos. Se usan en estrategias UNDO/REDO.

Write-Ahead Loggin

- Es una estrategia en la cual el mecanismo de recuperación garantiza que el BFIM está en el Log y el Log está en el disco **ANTES** que el AFIM actualice el BFIM en la base en el disco.
- Un protocolo **WAL** podría ser el siguiente:
  - Un AFIM de un ítem no puede actualizar el BFIM de ese ítem hasta que todos los registros del Log de tipo Undo para esa transacción haya sido escritos en el disco.
  - Nunca se puede completar el commit de una transacción hasta que todos los registros de Log (Undo o Redo) hayan sido escritos en el disco.
- Para Implementar esto, se deben llevar listas de transacciones activas, confirmadas y abortadas.

Checkpoint

- **[checkpoint]**: Registro del Log que indica que todos los buffers modificados de la base fueron actualizados al disco.
- Ninguna transacción T tal que [commit,T] aparece en el Log antes que [checkpoint] necesita Redo.
- El checkpoint consiste de los siguientes pasos:
  - Suspender la ejecución de todas las transacciones.
  - Grabar todos los buffers modificados en el disco
  - Registrar el [checkpoint] en el log y grabar el Log en disco.
  - Permitir la continuación de las transacciones.

RollBack

- Se debe realizar cuando una transacción aborta o no termina por una falla.
- Es la recuperación de todos los BFIM's de los ítems que modificó es transacción y de todas las transacciones que leyeron de la que abortó. (Abortos en Cascada)
- Lo Abortos (o Rollbacks) en Cascada pueden consumir mucho tiempo por lo que deben evitarse garantizando historias EAC o estrictas.

Algoritmos de Recuperación

- Cualquier algoritmo de recuperación debería implementar las siguientes operaciones o procedimientos:
  - **Recuperación**: Indica cual es la estrategia general de recuperación
  - **Undo**: Indica cómo se debe hacer el undo de una operación.
  - **Redo**: Indica como se debe hacer el redo de una operación.
- Es fundamental que la operación de **Redo** sea idempotente.
- En general, se asume que se están generando historias estrictas y serializables.

Recuperación Basada en Actualización Diferida

- Idea Básica: Demorar la escritura de la base en el disco hasta que la transacción alcance el commit.
- Para esto, durante la ejecución la actualización se realiza en el Log y en los buffers o en un área local a la transacción.
- El protocolo tiene dos reglas básicas:
  - Los cambios realizados por una transacción T nunca son grabados en el disco hasta que la T alcanza el commit.
  - Una transacción T nunca puede alcanzar el commit hasta que grabó todas sus operaciones de actualización en el Log y el log fue grabado en el disco.

Un Algoritmo Basado en Actualización Diferida

```
RDU(CT,AT){
/* Recovery using Deferred Update.
   CT:transacciones confirmadas desde el ultimo checkpoint
   AT: transacciones activas */
para cada T en CT
  para cada W = (write_item de T) en Log
        Redo_W(W)
  fin
fin
para cada T en AT
        Start(T) /* se lanza la transacción nuevamente */
}
```

```
Redo_W(Op){
/* Rehace la operación Op, asumiendo que es la entrada de un write T en el log. */
   [write,T,X,v_actual] = Op;
   w_f(X,v_actual); /* graba en el disco el item X con v_actual como valor */
}
```

No necesita Undo: las T en AT no modificaron nunca la base.

Ejemplo de Actualización Diferida

- Log:

[start_transaction,T_1]	[start_transaction,T_2]
[write_item,T_1,D,20]	[write_item,T_2,B,12]
[commit,T_1]	[start_transaction,T_3]
[checkpoint]	[write_item,T_3,A,30]
[start_transaction,T_4]	[write_item,T_2,D,25]
[write_item,T_4,B,15]	
[write_item,T_4,A,20]	
[commit,T_4]	

CT={T_4}

AT={T_2, T_3}

T_1: Ignorar porque terminó antes del CHKP

T_2, **T_3**: Ignorar en Redo, relanzar al final.

T_4: Aplicar Redo_w.

Recuperación Basada en Actualización Inmediata

- Idea Básica: grabar en el disco sin esperar al commit.
- Siempre se trabaja con la estrategia WAL (grabar el log antes que los datos).
- Dos familias de algoritmos:
  - Undo/No-Redo: hay que garantizar que todas las modificaciones efectivamente fueron grabadas en la base.
  - Undo/Redo: No hay que garantizar nada en particular.
- Undo/Redo es más complejo.
- Se asume que se generan historias estrictas y serializables.

Un Algoritmo Basado en Actualización Inmediata (Undo/Redo)

```
RIU(CT,AT){
/* Recovery using Inmediate Update.
   CT: Lista de transacciones confirmadas desde el ultimo checkpoint
   AT: Lista de transacciones activas */
Para cada T en AT
  para cada W = (write_item de T) en Reverse(Log)
     Undo_W(W)
  fin
fin
 para cada T en CT
   para cada W = (write_item de T) en el Log
        redo_W(W)
   fin
fin
```

```
Para cada T en AT
        Start(T) /* se lanza la transacción nuevamente */
}
```

```
Undo_W(Op){
/* Deshace la operación Op, asumiendo que es la entrada de un write T en el log. */
[write,T,X,v_ant,v_actual] = Op;
w_f(X,v_ant); /* graba en el disco el item X con v_ant como valor */
}
```

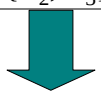
Redo_w es el visto anteriormente.

Ejemplo de Actualización Inmediata

- Log:

[start_transaction,T_1] [start_transaction,T_2]
[write_item,T_1,D,20] [write_item,T_2,B,12]
[commit,T_1] [start_transaction,T_3]
[checkpoint] [write_item,T_3,A,30]
[start_transaction,T_4] [write_item,T_2,D,25]
[write_item,T_4,B,15]
[write_item,T_4,A,20]
[commit,T_4]

CT={T_4}
AT={T_2, T_3}



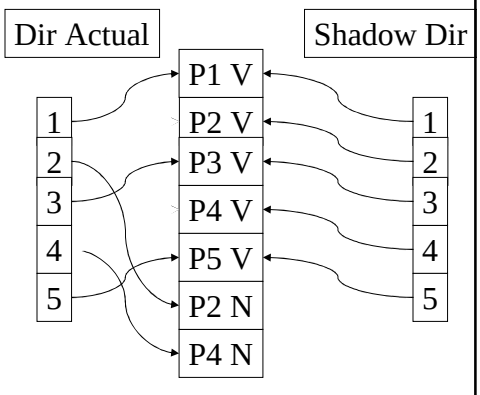
T_1: Ignorar porque terminó antes del CHKP

T_2, T_3: Hacer Undo_w sobre Log inverso, relanzar al final.

T_4: Aplicar Redo_w.

Recuperación Basada en Shadow Paging

- Idea Básica: La base de datos es un conjunto de paginas con un directorio con una entrada para cada página.
- Cada transacción que modifica algo mantiene dos copias una que no modifica nunca y otra que cada vez que modifica una página, crea una pagina nueva.



Recuperación Basada en Shadow Paging

- Ventajas
  - La recuperación (o confirmación) se limita a elegir con que versión de directorio se actualiza la base: En rollback se actualiza con el shadow, en commit se actualiza con el actual.
  - Si se trabaja en un ambiente monotarea, no se necesita Log. Si se necesita si se trabaja en un entorno multitarea.
- Desventajas
  - La páginas cambian de lugar, con lo que se complican las estrategias manipulación del disco para mantener cerca las páginas relacionadas.
  - Hay que tener una estrategia Garbage Collecting.
  - Hay que implementar en forma atómica la actualización final de actualización del directorio.