BDII 5 – SISTEMA DE RECUPERACIÓN

- Una parte integral de un SGBD es el SISTEMA DE RECUPERACIÓN, responsable de restaurar la bd a un estado consistente después de un fallo.
- Debe proporcionar alta **disponibilidad**, es decir, debe minimizar el tiempo durante el cual la bd no se puede usar después de un fallo.

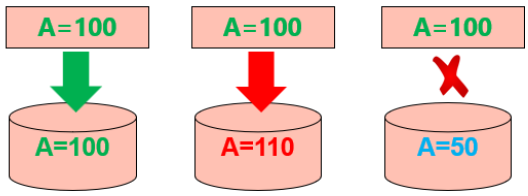
CLASIFICACIÓN DE LOS FALLOS

- FALLO EN LA TRANSACCIÓN:
  - ERROR LÓGICO → la transacción no puede continuar con su ejecución normal a causa de alguna **condición interna**:
    - Debido a un error **interno lógico** que sólo se puede corregir reescribiendo el programa de aplicación.
    - Debido a una **entrada incorrecta**.
    - Debido a que no se encontraron los **datos** deseados en la bd.
  - ERROR DEL SISTEMA → el **sistema** se encuentra en un **estado no deseado** (por ejemplo, de interbloqueo) y como consecuencia una transacción no puede continuar con su ejecución normal. Aun así, la transacción **puede volver a ejecutarse** más tarde.
- CAÍDA DEL SISTEMA:
  - FALLO DEL HARDWARE O SOFTWARE DE BASE (SO o SGBD) → se pierde el contenido de la memoria **volátil** y se **abortan** las transacciones.
    ↳ Según el SUPUESTO DE FALLO-PARADA el contenido de la memoria **no volátil** permanece intacto.
  - FALLO DE DISCO → uno o varios **bloques** del disco pierden su contenido como consecuencia de una **colisión** de la cabeza lectora o de un fallo durante una **transferencia**.
- Los ALGORITMOS DE RECUPERACIÓN se encargan de garantizar la **consistencia** de la bd y la **atomicidad** de las transacciones a pesar de los fallos.
  Constan de 2 partes:
  1. Acciones llevadas a cabo **durante el procesamiento normal** de transacciones para asegurar que existe información para la recuperación.
  2. Acciones llevadas a cabo **después de ocurrir el fallo** para restablecer el contenido de la bd a un estado que asegure las propiedades ACID.

ALMACENAMIENTO

IMPLEMENTACIÓN DEL ALMACENAMIENTO ESTABLE

- Las transferencias de bloques entre la **memoria y el disco** pueden acabar de 3 formas:
  - **Éxito** → la información transferida llega a su destino con seguridad.
  - **Fallo parcial** → se produce un fallo en medio de la transferencia, y al bloque de destino le llega información incorrecta.
  - **Fallo total** → el fallo se produce lo bastante pronto en la transferencia para que el bloque de destino permanezca intacto.

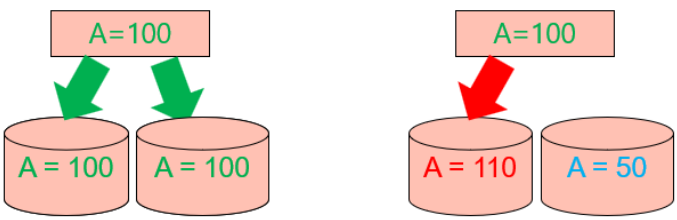


- Para implementar el almacenamiento estable se debe **replicar** la información necesaria en **varios** medios de almacenamiento **no volátil** con **modos de fallo independientes** y **actualizarla** controladamente.

- DISCOS CON IMAGEN → por ejemplo, los sistemas RAID, que en su versión más simple almacenan **2 copias** de cada **bloque** en distintos **discos**.
  × No previenen **desastres** como incendios o inundaciones.
- COPIAS REMOTAS → se guardan copias de las **cintas** en **lugares remotos** como protección frente a los desastres.
  × Los cambios que se realicen desde el **último traslado** de las cintas son **vulnerables** a perderse si se produce un desastre, pero es muy **costoso** trasladar las cintas asiduamente.
- SISTEMAS REMOTOS → se guardan copias tanto en el almacenamiento **local** como en **lugares remotos**, a los que acceden por medio de **redes de datos**.
  ✓ Una vez se completa una operación, los bloques copiados están protegidos ante **desastres**, pues se **envían** al sistema remoto al mismo **tiempo** y de la misma **forma** que se guardan en almacenamiento local.

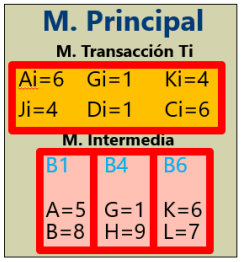
- Entonces, la operación de **salida** de un bloque se realiza así:
  1. Se escribe la información en el primer bloque físico.
  2. Se escribe la información en el segundo bloque físico.
  3. La salida sólo está completa después de que la segunda escritura finalice bien.
  ↳ Así se garantiza la **atomicidad** de las salidas a almacenamiento estable: o bien se completa correctamente (se actualizan todas las copias) o no produce ningún cambio.

- Si se produce un **fallo durante la salida**, es posible que las dos copias sean inconsistentes entre sí.
  Durante la recuperación se examina **cada par** de bloques físicos, comprobando sus **códigos de redundancia** para detectar errores.
  - Si ambos son iguales y no hay ningún error → no son necesarias más acciones.
  - Si un bloque tiene un error → se reemplaza su contenido por el del otro bloque.
  - Si el contenido es diferente pero no hay ningún error → el sistema sustituye el contenido del primer bloque por el del segundo.

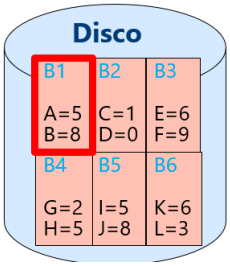


ACCESO A LOS DATOS

- Los bloques que conforman la bd pueden residir en la memoria **no volátil**, en cuyo caso se conocen como BLOQUES FÍSICOS, o temporalmente en **memoria principal**, en cuyo caso se conocen como BLOQUES DE MEMORIA INTERMEDIA DE DISCO.
- Las **transferencias** de datos entre el **disco** y la **memoria** principal se realizarán usando:
  - $entrada(B) \rightarrow$ transfiere el bloque físico B a la memoria intermedia.
  - $salida(B) \rightarrow$ transfiere el bloque de memoria intermedia B al disco.



- Cada transacción T_i posee un ÁREA DE TRABAJO PRIVADA (que se crea cuando comienza y se elimina cuando finaliza) en la que almacena copias de los elementos de datos a los que ha accedido y ha actualizado.
- Las **transferencias** de datos entre el **área** de trabajo de las transacciones y la **memoria** intermedia se realizarán usando:
  - $leer(X) \rightarrow$ asigna el valor del elemento de datos X a la variable local x_i.
    1. Si el bloque B_X en el que reside X no está en memoria principal, entonces se ejecuta $entrada(B_X)$.
    2. Asigna a x_i el valor de X del bloque de memoria principal.
  - $escribir(X) \rightarrow$ asigna el valor de la variable local x_i al elemento de datos X en memoria principal.
    1. Si el bloque B_X en el que reside X no está en memoria principal, entonces se ejecuta $entrada(B_X)$.
    2. Asigna el valor de x_i a X en la memoria intermedia.
  - La operación $salida(B_X)$ no tiene por qué tener efecto inmediatamente después de ejecutar $escribir(X)$. Se realizará cuando:
    - El gestor de la memoria principal necesita **espacio**.
    - El SGBD desea **reflejar** en el disco los **cambios** en B_x.
    - ↳ Si el sistema falla después de ejecutar $escribir(X)$ pero antes de ejecutar $salida(B_X)$, no se escribe nunca en el disco el nuevo valor de X y por tanto se **pierde**.



RECUPERACIÓN Y ATOMICIDAD

- El objetivo de la **atomicidad** es realizar todos los cambios inducidos por una transacción o no realizar ninguno.
  Si para una transacción se necesitan varias operaciones de salida, se puede producir un fallo después de haber concluido alguna de ellas, pero no todas.
- Para asegurar la atomicidad, primero se debe efectuar una operación de salida de información que **describa las modificaciones** en el almacenamiento **estable** sin modificar todavía la bd.

REGISTRO HISTÓRICO

- El REGISTRO HISTÓRICO es una secuencia de registros que almacenan todas las actividades de **actualización** de la bd.

Hay varios tipos de registros en los registros históricos:

- REGISTRO DE ACTUALIZACIÓN → identificador de la transacción, identificador del elemento de datos
- (normalmente su ubicación en el disco), valor anterior y valor nuevo: $< T_i, X_j, V_1, V_2 >$.
- $< T_i\ iniciada >,\ < T_i\ comprometida >,\ < T_i\ abortada >$.

leer(A);
A:= A – 50;
escribir(A);
leer(B);
B:=B+50;
escribir(B);

<T1 iniciada>
<T1, A, 200, 150>
<T1, B, 300, 350>
<T1 comprometida>

<T1 iniciada>
<T1, A, 200, 150>
<T1, B, 300, 350>
<T1 abortada>

- Es fundamental que el registro se **cree antes** de que la escritura **modifique la bd**.
- Para que los registros del rh sean útiles para recuperarse frente a fallos, deben residir en **almacenamiento estable**.

MODIFICACIÓN DE LA BASE DE DATOS

- Se dice que una transacción **modifica** la bd si realiza una actualización en la **memoria intermedia** de disco o en el propio **disco**.
  Las actualizaciones en su parte privada de la memoria principal no cuentan como modificación de la bd.
  - Técnica de MODIFICACIÓN INMEDIATA → la transacción modifica la bd mientras está activa.
  - Técnica de MODIFICACIÓN DIFERIDA → la transacción no modifica la bd hasta que está comprometida.
    - ↳ Implica que se necesita hacer una **copia local** de todos los elementos de datos actualizados.
- $deshacer \rightarrow$ utilizando el rh, se ajusta el elemento de datos especificado en el registro a su valor anterior.
- $rehacer \rightarrow$ utilizando el rh, se ajusta el elemento de datos especificado en el registro a su valor nuevo.

CONTROL DE CONCURRENCIA Y RECUPERACIÓN

- Las técnicas de **aislamiento de instantáneas** y de control de concurrencia **basado en validación** adquieren **bloqueos** X sobre el elemento en el momento de la **validación**, antes de modificarlos, y lo mantienen hasta que se comprometen → la técnica de **modificación diferida** casa con estos esquemas.
  - Algunas implementaciones del aislamiento de instantáneas usan **modificación inmediata**, pero proporcionan una **instantánea lógica** bajo demanda:
    - ↳ Cuando una transacción necesita leer un elemento que otra concurrente ha actualizado, se realiza una copia de dicho elemento ya actualizado y se deshacen en ellas las actualizaciones de las otras transacciones.
- El **bloqueo en 2 fases** casa con la **modificación inmediata**.
  - ↳ También se puede usar modificación diferida.
  - Si el sistema de control de concurrencia permite que un elemento que ha modificado una transacción sea modificado después por otra antes de que se comprometa la primera, entonces **deshacer** los efectos de la **primera** también desharía los de la **segunda**.
  - Para evitar esto, los algoritmos de recuperación requieren que, si una transacción modifica un elemento, **ninguna otra puede modificarlo** hasta que la primera se **comprometa** o **aborte**.
    - Una manera de conseguir esto es usar un **bloqueo en 2 fases estricto**.

TRANSACCIÓN COMPROMETIDA

- Se dice que una transacción está **comprometida** cuando su registro $< T\ comprometida >$ se ha guardado en **almacenamiento estable**.
  - ↳ Si ocurre un **fallo** del sistema **antes** de que se guarde en almacenamiento estable el registro de comprometida, la transacción **retrocederá**.
- En ese momento, **todos los registros** ya se han guardado en **almacenamiento estable**,
  por lo que hay suficiente información como para asegurar que si se produce un fallo del sistema se pueden **rehacer** las actualizaciones de la transacción.

USO DEL REGISTRO PARA DESHACER Y REHACER TRANSACCIONES

- $rehacer(T) \rightarrow$ establece el valor de todos los elementos actualizados por la transacción T a los valores nuevos.
  - Las actualizaciones deben mantener el **orden** en el que se realizaron originalmente.
- $deshacer(T) \rightarrow$ restaura el valor de todos los elementos actualizados por la transacción T a sus valores anteriores.
  - Las actualizaciones deben mantener el **orden** en el que se realizaron originalmente.
  - ▸ Cuando se completa la operación $deshacer$ para la transacción T, se escribe el registro $< T\ aborta >$.

- La mayoría de algoritmos **no** realizan estas operaciones **independientemente** para cada transacción, si no que **recorren el rh** aplicando la operación $rehacer$ o $deshacer$ sobre los registros que encuentren.
  - ✓ Se mantiene el **orden** de las actualizaciones.
  - ✓ Sólo se lee el **rh una vez**.

PUNTOS DE REVISIÓN

- ▸ Consultar el **rh entero** para determinar cuáles transacciones se **rehacen** y cuáles se **deshacen** es costoso:
  - × El proceso de búsqueda **consume tiempo**.
  - × La mayoría de transacciones que deben **rehacerse** ya tienen escritas sus **actualizaciones en la bd**, por lo que rehacerlas es una pérdida de tiempo.
- Los PUNTOS DE REVISIÓN permiten reducir esta sobrecarga. Se realizan de la siguiente manera:
  1. Escribir en **almacenamiento estable** todos los registros del **rh** que residan en ese momento en la memoria principal.
  2. Escribir en **disco** de todos los **bloques de memoria intermedia** que se hayan **modificado**.
  3. Escribir en **almacenamiento estable** de un registro $< revisión\ L >$ donde L es una lista de transacciones activas en el momento.
  - × Mientras se lleva a cabo **no** se permite que ninguna transacción realice **actualizaciones** en los bloques de **memoria intermedia** o en el **rh**.
    Por tanto, el **procesamiento** de transacciones se debe **detener** mientras se realiza el registro histórico.
    - ↳ El PR DIFUSO se permite que las transacciones sigan activas mientras se realiza el pr.

- Cuando se produce un fallo, el esquema de recuperación examina **hacia atrás** el rh para encontrar el último registro $< revisión\ L >$.
  Las operaciones $rehacer$ y $deshacer$ se aplican sólo a:
  - Transacciones de $\boldsymbol{L}$.
  - Transacciones que **iniciasen** su ejecución **después** de la escritura de $< \boldsymbol{revisión\ L} >$.
  - ↳ Para $deshacer$ las transacciones de L se necesitarán registros anteriores al pr, pero posteriores al primero de sus registros de inicio, por lo que el resto se pueden **eliminar** si se necesita ese espacio.

ALGORITMO DE RECUPERACIÓN

- Este algoritmo requiere que ninguna transacción pueda **modificar** datos de **transacciones no finalizadas** (comprometidas o abortadas).

RETROCESO DE TRANSACCIONES

Método de retroceso si falla la transacción T_i (NO el sistema entero):

1. Se explora el rh **desde atrás** y para cada registro de la forma $< T_i, X_j, V_1, V_2 >$ que se encuentre:
   1.1. Se aplica la operación $deshacer$ sobre el registro.
   1.2. Se escribe un registro especial SOLO-REHACER $< T_i, X_j, V_1 >$.
   - A veces a estos registros se les llama REGISTROS DE COMPENSACIÓN DE RH.
2. Cuando se encuentre $< T_i\ iniciada >$ se detiene la exploración y se escribe $< T_i\ abortada >$.

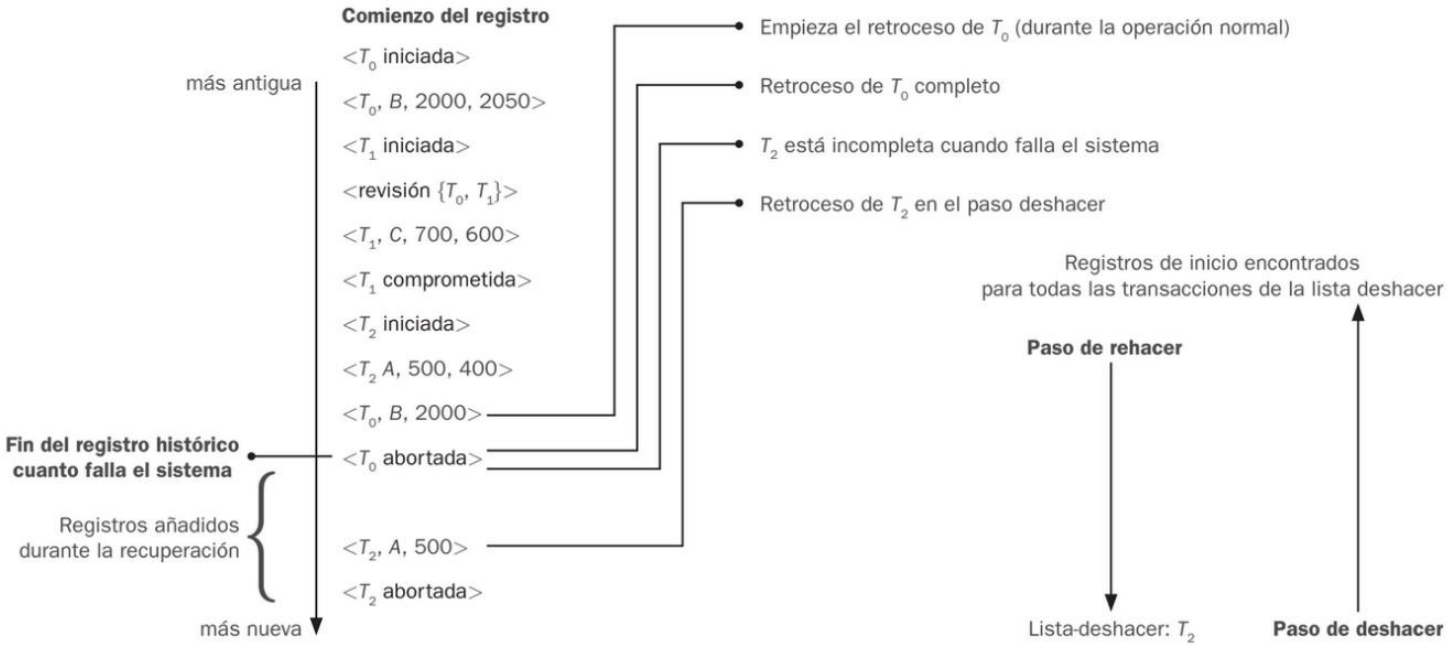
RECUPERACIÓN TRAS UN FALLO DEL SISTEMA

- Después de ocurrir un fallo del sistema, el sistema de recuperación consulta el rh para determinar cuáles transacciones se rehacen y cuáles se deshacen.
  - T debe **deshacerse** si el rh contiene $< T\ iniciada >$ pero no $< T\ comprometida >$/$< T\ abortada >$.
  - T debe **rehacerse** si el rh contiene los registros $< T\ iniciada >$ y $< T\ comprometida >$/$< T_i\ abortada >$.
    - ↳ **Rehacer** también las transacciones **abortadas** simplifica mucho el algoritmo de recuperación y permite que se ejecute más rápido.

Método de retroceso si falla el sistema entero:

1. En la FASE REHACER se explora el rh **hacia delante** a partir del **último pr**.
   1.1. La lista-deshacer se establece inicialmente a la lista L del pr.
   1.2. Cuando se encuentra un registro normal de la forma $< T_i, X_j, V_1, V_2 >$ o uno solo-rehacer de la forma $< T_i, X_j, V_1 >$ se le aplica $rehacer$.
   1.3. Cuando se encuentra un registro de la forma $< T_i\ iniciada >$ se añade T_i a la lista-deshacer.
   1.4. Cuando se encuentra un registro de la forma $< T_i\ abortada >$ o $< T_i\ comprometida >$ se elimina T_i de la lista-deshacer.
2. En la FASE DESHACER se explora el rh **hacia detrás**.
   2.1. Cuando se encuentra un registro de a una transacción de la lista-deshacer se deshace como se explicó en la sección anterior.
   2.2. Cuando se encuentra en un registro $< T_i\ iniciada >$ para una transacción T_i de la lista-deshacer, se escribe un registro $< T_i\ abortada >$ y se elimina T_i de la lista-deshacer.
   2.3. La fase termina cuando la lista-deshacer se vacía, es decir, se han encontrado los registros $< T_i\ \ iniciada >$ de todas las transacciones de la lista-deshacer.

- Este proceso se denomina REPETICIÓN DE LA HISTORIA porque las acciones se repiten en el **mismo orden** en el que se llevaron a cabo originalmente.



FALLO CON PÉRDIDA DE ALMACENAMIENTO NO VOLÁTIL

- A pesar de que es **raro** encontrarse con un fallo en el que se pierda información de almacenamiento no volátil, es necesario prepararse para afrontarlos.
- La idea básica es **volcar periódicamente** (por ejemplo, una vez al día) el contenido entero de la bd en **almacenamiento estable**.

- PROCESO DE VOLCADO ⟶ requiere que **ninguna transacción** esté activa durante el proceso (es similar al procedimiento de los pr).
  1. Escribir en **almacenamiento estable** todos los registros del **rh** que residan en ese momento en la memoria principal.
  2. Escribir en **disco** todos los bloques de la **memoria intermedia**.
  3. Copiar el contenido de la **bd** en **almacenamiento estable**.
  4. Escribir el registro del rh $< volcar >$ en **almacenamiento estable**.
  - × Debe copiarse en almacenamiento estable toda la bd, lo que conlleva una **considerable transferencia de datos**.
  - × Se **pierden ciclos de CPU** porque se detiene el procesamiento de transacciones.
    ↳ En los esquemas de VOLCADO DIFUSO se permite que las transacciones sigan activas mientras se realiza el volcado.
- PROCESO DE RECUPERACIÓN:
  1. Se usa el volcado **más reciente** para que la bd recupere **un estado consistente**.
  2. Se usa el **rh** para llevar al sgbd al **último estado consistente** en el que estuvo antes del fallo.
  - Si se produce un **fallo parcial** (de un bloque o de unos pocos) sólo hay que restaurar esos bloques y rehacer solo las acciones realizadas en ellos.
- Los volcados de la bd también se llaman VOLCADO DE ARCHIVOS, ya que se pueden **archivar** y usarse más tarde para examinar **estados anteriores de la bd**.
- La mayoría de sgbd usan también VOLCADO SQL, que escribe sentencias SQL DDL e $insertar$ en un archivo que se podrá ejecutar para **reconstruir** la bd.
  - ✓ Es útil al migrar datos a un **ejemplar diferente** de la bd o a una **versión diferente** del software del sgbd, ya que las ubicaciones físicas y la configuración pueden diferir.