TEMA 7
CONFIABILIDAD EN BASES DE DATOS DISTRIBUIDAS

7.1. DBMS CONFIABLE

- Aquel que puede procesar peticiones a pesar de existir fallas en algunos componentes sin violar consistencia
- Tipos de fallas:
  - Fallas de transacciones
  - Fallas de sistema (energía, etc).
  - Fallas de medios
  - Fallas de comunicación.
- Un DBMS confiable tiene la capacidad de recuperar el estado consistente de la BD posterior a la ocurrencia de una falla (siempre y cuando el DBMS esté correctamente configurado).
- La confiabilidad está fuertemente relacionada al problema de mantener la ***atomicidad*** y ***durabilidad*** de una Txn

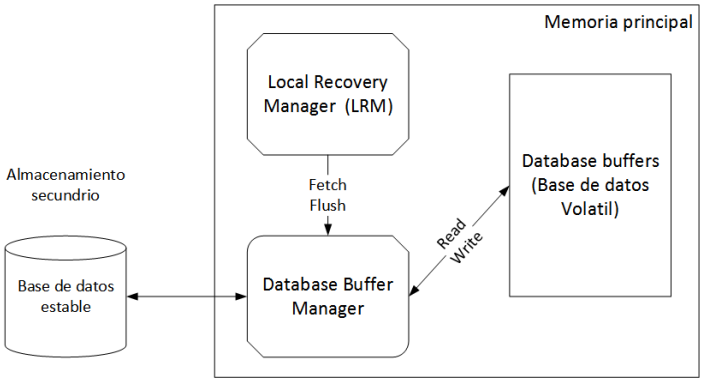
7.2. PROTOCOLOS LOCALES DE RECUPERACIÓN

- Existe un componente encargado de realizar esta tarea llamado ***Sistema de recuperación***.
- Este componente asegura la correcta implementación de las propiedades de atomicidad y durabilidad de una transacción.
- A nivel general el algoritmo de recuperación se divide en 2 partes:
  - Acciones preventivas tomadas durante la ejecución normal de una transacción. El objetivo principal de realizar estas acciones es el contar con la información suficiente para realizar una recuperación exitosa en caso de la ocurrencia de una falla.
  - Acciones tomadas después de la ocurrencia de una falla. El objetivo principal es la recuperación del estado de la base de datos que permita reestablecer la atomicidad, consistencia y durabilidad de las transacciones existentes al momento de ocurrir una falla.

7.2.1. Administrador de recuperación local (LRM: Local Recovery Manager)

- En una BDD este componente se encarga de mantener la atomicidad y durabilidad de las transacciones que se ejecutan en cada sitio.

7.2.1.1. Arquitectura básica



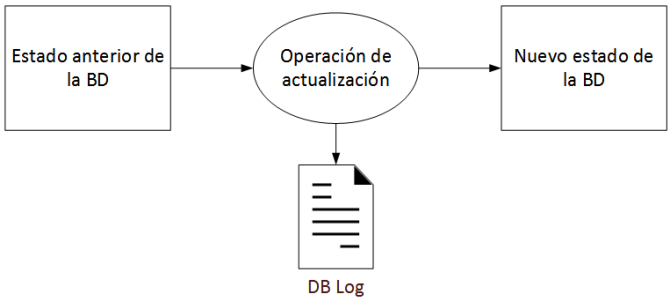
- ***Base de datos Volátil***. Área de memoria formada por un conjunto de buffers donde se guardan todas las operaciones de lectura y escritura que realizan las transacciones. En condiciones normales, todo dato debe pasar por esta área de memoria antes de aplicar una operación de lectura o escritura.
- ***Base de datos estable***. Representa al almacenamiento permanente. En teoría, los datos en esta base de datos nunca deben perderse. En la práctica esto se puede lograr empleando una combinación de recursos de hardware y software: Grupos de discos, respaldos con cintas, replicación, etc.
- ***Database Buffer Manager***. Una de sus principales tareas es mantener los datos accedidos recientemente en los buffers de memoria de la BD. En esta arquitectura se le llama "Base de datos Volátil".
- Típicamente la estructura física de la BD volátil es similar a la de la BD estable. Cada buffer de memoria se divide en un conjunto de páginas del mismo tamaño de las páginas de disco de la BD estable.
- ***Local Recovery Manager (LRM)***. Su funcionamiento principal se describe en los siguientes puntos:
    - Cuando una transacción requiere acceder un dato, el LRM lanza una operación ***Fetch*** hacia el DB buffer manager indicando la página que se requiere leer.
    - El DB buffer manager a su vez, revisa si las páginas solicitadas se encuentran en la BD volátil.
    - Si los bloques solicitados existen, se leen y se regresa el resultado par que pueda ser accedido por la Transacción.
    - En caso de no existir, DB Buffer manager solicita la carga de datos de la base estable y la carga e un buffer vacío dentro de la BD volátil.
    - En caso que ya no existan buffers vacíos:
        - el DB buffer manager selecciona algunos buffers con datos empleando ciertos criterios (por ejemplo, aquellos que no se han modificado recientemente: Last Recently Used: LRU), etc.
        - Solicita una operación de escritura de dichos buffers en la BD estable.
        - Al ser escritos o sincronizados con la BD estable, el DB buffer manager sobrescribe los buffer seleccionados con los nuevos datos.
    - Las operaciones de Flush que invoca el LRM se emplean para indicarle al DB buffer manager una operación de escritura de buffer hacia la BD estable. En secciones posteriores se revisará esta operación.

7.2.2. Información de recuperación.

- Cuando una falla de sistema ocurre, la base volátil se pierde.
- Para evitar la pérdida de los cambios aplicados en ella, el BMS debe mantener cierta información acerca del estado de la BD volátil hasta el momento de la ocurrencia de la falla con la finalidad de poder recuperar el estado justo antes de la ocurrencia de la falla. A esta información se le conoce como ***información de recuperación.***
- La información de recuperación que requiere el DBMS depende de la técnica que se emplea para aplicar los cambios a los datos. Existen 2 técnicas:
    - ***In place Update***: La actualización de los datos se realiza directamente en la BD. Esto provoca que el valor anterior se pierda.
    - ***Out of place***: Los cambios no se modifican en la BD estable, los cambios se mantienen de manera separada y de forma periódica, los cambios se integran a la BD estable.
- En la práctica la técnica In place Update es la que más se utiliza, principalmente por desempeño.

7.2.3. Información de recuperación empleando la técnica In place Update

- Para facilitar la recuperación, los cambios que se pierden en la BD estable debido a la técnica In place update deben ser respaldados en algún lugar ya que pueden requerirse para recuperar el estado consistente de una BD. Esta información es típicamente guardada en el ***llamado Log de la base de datos*** (DB Log)
- Lo anterior implica que: cada cambio que realiza una transacción, adicional a aplicar el cambio en la BD volátil, este se debe respaldar en el ***DB Log***.
- El DB Log contiene la información necesaria para poder aplicar una operación de recuperación.

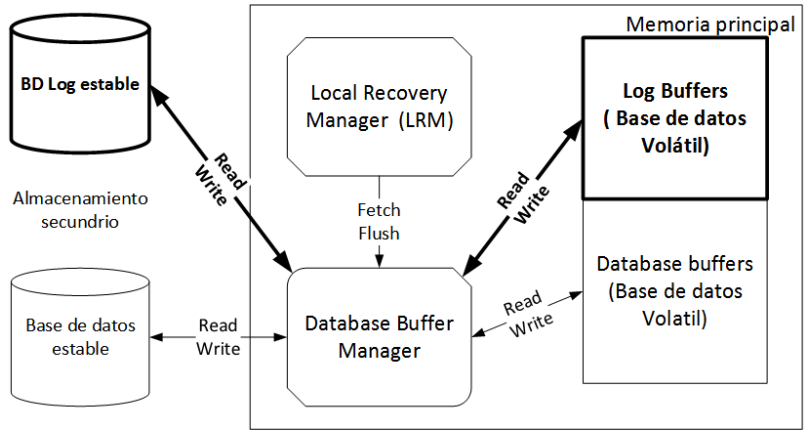


7.2.3.1. Características de un DB Log

- Representa la estructura más común para almacenar cambios aplicados a la BD.
- Está formado por una secuencia de registros comúnmente llamados ***Log records***.
- Cada log record representa eventos significativos que ocurren en una transacción asociados con la actualización de datos.
- Existen diversos tipos de registros a nivel general debe contar con la siguiente información como mínimo.
    - $< \boldsymbol{T_i, start} >$ Aparece cuando la transacción Ti inicia.
    - $< \boldsymbol{T_i, x_j, v_1, v_2} >$ Representa una operación de escritura de un dato X. v_1 representa el valor de X antes de la actualización y v_2 representa el valor de X nuevo.

- $< T_i, commit >$ Indica que se ha invocado el término de la transacción, en este caso con una confirmación de los cambios.
- $< T_i, rollback >$ Indica que se ha invocado el término de la transacción, en este caso con una operación de abort o rollback.
- $< start - checkpoint >$ Indica el inicio de la actualización de la BD volátil con la BD estable.
- $< end - checkpoint >$ Indica el fin del proceso de la actualización de la BD Volátil con la BD estable.

- El DB Log también se mantiene en memoria empleando los llamados ***Log Buffers***.
- De forma similar, Log buffers se actualizan con un almacenamiento estable llamado ***BD Log estable*** *o* ***Log estable***.
- De lo anterior, la arquitectura original incorpora el mecanismo de manejo de BD Logs como se muestra en la siguiente imagen:



- Observar los nuevos elementos que se incorporan resaltados en la imagen anterior.
- La escritura de los Log Buffers hacia el DB Log estable puede realizarse en 2 formas:
  - ***Escritura síncrona***: Cada entrada al DB Log obliga una escritura al Log estable. Como consecuencia, pueden existir ciertos retardos durante la ejecución de una transacción ya que la transacción se suspende hasta que se complete cada escritura. Por otro lado, si ocurre una falla justo después de una escritura en el Log estable, la recuperación es relativamente sencilla.
  - ***Escritura asíncrona***: Los Buffers Logs son actualizados en el Log estable a intervalos constantes de tiempo, o cuando los buffers llegan a cierta capacidad, entre otros criterios. Oracle utiliza esta técnica.

Independientemente al mecanismo de escritura, hay 2 escenarios más que se deben considerar con respecto a las fallas:

Escenario 1:

- Suponer que se ha realizado una actualización de los DB buffers a la BD estable antes de que los Log buffers sean actualizados en el Log estable. Si ocurre una falla antes de que los Logs buffers sean actualizados en el Log estable, los cambios estarán en la BD estable, pero Los DB Logs no reflejarán dichos cambios !

- La situación anterior es grave, ya que sería imposible recuperar el estado consistente de la BD. Recordar que el DB Log es fundamental para determinar las acciones a aplicar y poder así reconstruir el estado consistente de la BD (UNDO o REDO)

Escenario 2:

- Que sucede si una transacción hace `commit`. Los cambios que realizó están en la BD volátil y el correspondiente Log en los Logs buffers. Es decir, no hay nada escrito en las bases estables. Si ocurre una falla en este instante, nuevamente, se pierden tanto los datos de ambos buffers y se estaría violando la propiedad de durabilidad de las transacciones posterior a la confirmación de los cambios!.

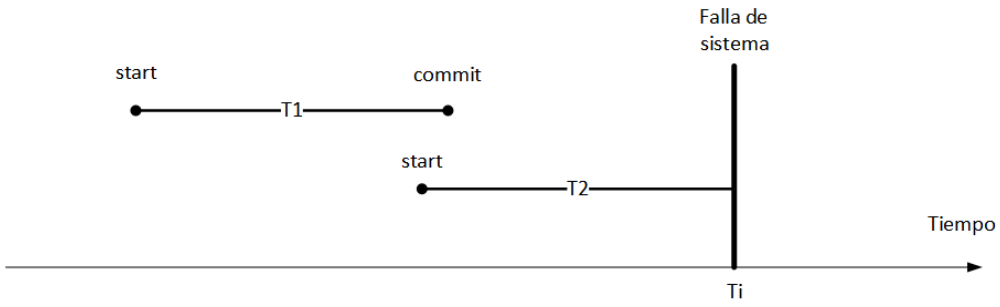
7.2.3.2. Protocolo Write – Ahead logging (WAL)

- Este protocolo permite resolver los 2 escenarios anteriores y es implementado por los BDMS. El protocolo establece las siguientes reglas:

1. Antes de que la BD estable sea actualizada (con posibles cambios que no han sido confirmados), todos los ***Logs buffers*** deben ser actualizados en el Log estable. Esto facilita la operación UNDO de recuperación (se explica más adelante).
2. Cuando una transacción hace `commit`, todos los cambios realizados y existentes en los Log buffers deben actualizarse con el Log estable y deben ocurrir antes de una actualización del DB buffer a la BD estable. Esto facilita la operación REDO de recuperación (se explica más adelante).

7.2.3.3. Estado de una transacción al ocurrir una falla

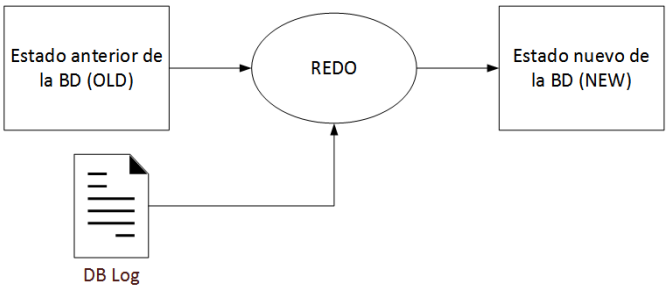
- Considerar la siguiente imagen que ilustra 2 escenarios que pueden presentarse al momento de producir una falla. Cada escenario se explica en las siguientes secciones.



7.2.3.4. Protocolo REDO

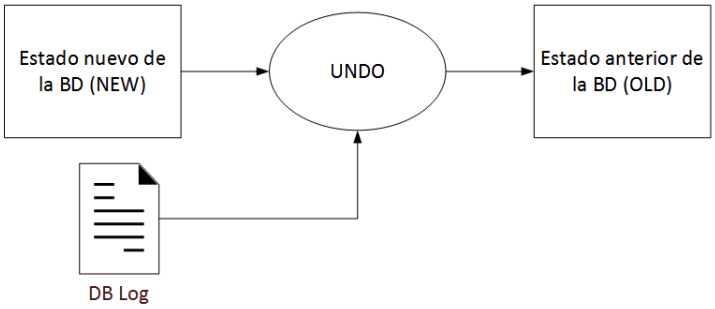
- Observar la transacción T1 en la figura. La transacción terminó de forma exitosa empleando la instrucción `commit`.
- Tiempo después ocurre una falla de sistema en el instante Ti. La propiedad de ***durabilidad*** de las transacciones establece que, si una transacción confirma sus cambios, se debe garantizar que dichos cambios serán ***permanentes*** en la BD.

- Sin embargo, existe la posibilidad de que los cambios aplicados a la BD volátil aún no se hayan aplicado a la BD estable al momento de ocurrir una falla.
- De lo anterior, es importante que una vez que se reestablezca el servicio derivado de la falla, el DBMS debe "***Rehacer***" los cambios que realizó la transacción T1. Para aplicar esta operación, típicamente llamada operación REDO, se emplea el DB Log y el protocolo REDO:



7.2.3.5. Protocolo UNDO

- Observar ahora la transacción T2 en la figura anterior. No existe marca que indique el término de la transacción al momento de ocurrir una falla.
- Por otro lado, la propiedad de ***atomicidad*** de las transacciones establece que un conjunto de operaciones se trata como una sola. Por lo tanto, al no concluir, significa que dicha operación no debe causar efecto alguno en la base de datos, simplemente no se realizó.
- Otra manera de entenderlo es, el DBMS no puede garantizar durabilidad de transacciones que no confirmaron sus cambios.
- Hasta este punto pareciera que al regresar de la falla no se tendría que hacer recuperación alguna ya que T2 no confirmó.
- Lo anterior es ***incorrecto***. ¡Puede existir un escenario en el que los cambios que realizó T2 fueron reflejados en la BD estable a pesar de no haber confirmado! ¿Cómo puede ser esto posible?
    - La actualización de datos de la BD Volátil a la estable es independiente al estado final de cada transacción.
    - Un escenario es el siguiente: Si se requiere liberar buffers de la BD volátil, el DB buffer manager lanzará una actualización hacia la BD estable sin importar el estado actual de las transacciones asociadas, la prioridad en este caso es la liberación de buffers. Esto provoca que ***pueden existir cambios en la BD estable que aún no han sido confirmados en su correspondiente transacción***.
- La situación anterior implica que cambios no confirmados encontrados antes de una falla deben "deshacerse" de la BD estable debido a que T2 no pudo confirmar sus cambios.
- Para recuperar el estado consistente de la BD se emplea el DB Log y el protocolo UNDO.



7.2.4. Flujo de ejecución de una Transacción con respecto al LRM

- En esta sección se muestra todo el flujo de ejecución que ocurre al interior del DBMS cuando se inicia una transacción dirigiendo el enfoque a las acciones que realiza el LRM (Administrador de recuperación local) empleando todos los conceptos vistos hasta ahora.
- Este flujo solo considera la ejecución normal de una transacción en la que no existen fallas. Las acciones que se realizan para recuperar una BD al ocurrir una falla se describen en la siguiente sección.
- El LRM cuenta con los siguientes comandos a nivel general (varían dependiendo el DBMS):
  - `begin_transaction`
  - `read`
  - `fetch`
  - `write`
  - `commit`
  - `abort`
  - `recover`

7.2.4.1. Comando begin_transaction

- Cuando inicia una transacción, entre otras cosas, se agrega $< T_i, start >$ al log buffer

7.2.4.2. Comando read

- Se emplea este comando cuando se requiere acceder a un dato.
- LRM intenta leerlo de las páginas del DB buffer asociadas a la transacción en turno.
- En caso de no existir, se emplea el comando **`fetch`** para indicarle al DB buffer manager que realice la lectura de la BD estable, cargar el dato en la BD volátil.
- Una vez obtenido el dato, LRM regresa el dato al scheduler (recordar el componente visto en el tema anterior).

7.2.4.3. Comando write

- De manera similar, si el dato a modificar no existe en los buffers de la transacción en turno, se hace uso de comando fetch.

- Se aplica en cambio en el BD volátil, se agrega la siguiente entrada en el Log buffer: $< \boldsymbol{T_i, x_j, v_1, v_2} >$
- Se reporta al Scheduler que la actualización se realizó con éxito.

7.2.4.4. Comando Abort

- Empleado para indicar que una transacción ha terminado y requiere restaurar el estado original (rollback).
- Se agrega la siguiente entrada al DB Log: $< \boldsymbol{T_i}, \boldsymbol{rollback} >$
- Para este momento, el DB buffer pudo haber escrito los cambios de esta transacción a la BD estable.
- LRM necesita deshacer (UNDO) los cambios aplicados en la base volátil. Para ello, se realiza una lectura al el Log Buffer para obtener los valores nuevos y anteriores de la transacción, en especial lee los valores viejos.
- LRM actualiza el valor nuevo por el valor viejo para recuperar el estado original de los datos modificados.
- LRM informa al Scheduler el éxito de la operación `rollback`.
- A esta operación se le conoce también como "***transaction undo***" o "***partial undo***".
- Retomando el caso en el que los datos modificados por la transacción que hizo `rollback` se hayan copiado al BD estable, aparentemente podría representar un problema. Esto no es así debido a que la operación `rollback` ya se aplicó en la BD Volátil que es la que interactúa con los usuarios. Tiempo después, cuando se decida realizar una nueva actualización hacia la BD estable, ¡los valores incorrectos que existen en la BD estable serán sobrescritos con los valores correctos de la BD volátil!

7.2.4.5. Comando commit.

- Empleado para indicar el fin de una transacción y requiere confirmar cambios.
- Se agrega la siguiente entrada al DB Log: $< \boldsymbol{T_i}, \boldsymbol{commit} >$
- Se invoca la actualización del Log estable con el contenido del DB Log.
- Se notifica el resultado de `commit` exitoso al Scheduler.

7.2.5. Proceso de recuperación de una BD.

Posterior a la ocurrencia de una falla, el DBMS detectará a través de diversos mecanismos la necesidad de aplicar un proceso de recuperación debido a la ocurrencia de 2 situaciones principales:

- Transacciones pudieron haber quedado inconclusas (sin confirmar), sus cambios fueron actualizados en la BD estable.
- Transacciones que confirmaron sus cambios no fueron actualizadas en la BD estable
- La recuperación se implementa a través del comando `recover` el cual se explica a continuación.

7.2.5.1. Comando recover

- Empleado después de la ocurrencia de una falla para recuperar el estado consistente de la BD.

- En el caso de Oracle, la BD no se abre hasta que se aplica el proceso de recuperación. Un usuario no puede interactuar con la BD si esta no se encuentra abierta.
- Para realizar la recuperación se realiza la lectura del Log estable.
- Se recorre el archivo de inicio a fin y se realizan las siguientes acciones para cada una de las transacciones incluidas el en log dependiendo cada uno de los siguientes escenarios:

Escenario 1:

$< T_i, start >$

…

…

$< T_i, commit >$

- Se trata de una transacción que confirmó sus cambios, pero estos no han sido reflejados en la BD estable, se requiere aplicar una operación **REDO**.
- El LRM solicita al DB buffer manager cargar las páginas correspondientes a la BD volátil.
- Una vez cargados los datos, se aplican los cambios contenidos en el log hacia la base Volátil.
    - Lo más lógico es pensar que las correcciones se realizan en la BD estable. Esto no se realiza por cuestiones de desempeño. Las correcciones se realizan sobre la BD volátil, para posteriormente ser sincronizadas con la BD estable durante el proceso normal de sincronización.
    - Esta técnica permite reducir el tiempo requerido para aplicar un proceso de recuperación que en algunos casos puede ser crítico ya que los usuarios no pueden acceder a la instancia mientras el proceso de recuperación esté ejecutándose.

Escenario 2:

$< T_i, start >$

…

…

- Se trata de una transacción que inició, pero no tiene un registro de fin.
- Toda transacción que no fue confirmada debe deshacerse.
- En este escenario, pueden existir cambios en la BD estable que deben deshacerse puesto que la transacción no se confirmó. Se requiere aplicar una operación **UNDO**.
- El LRM solicita al DB Buffer manager cargar las páginas correspondientes a la BD volátil. Los datos cargados podrían contener los datos actualizados por la transacción que no se confirmó.
- Empleando el DB Log, se lee el valor original (dato viejo) y se actualiza en la BD volátil.
- A esta operación se le conoce como ***global UNDO***. Todas las transacciones que quedan en este estado se les debe aplicar esta operación antes de poder permitir nuevos cambios.

Escenario 3:

$< T_i, start >$

…

…

$< T_i, rollback >$

- Se trata de una transacción que inició, indicó su final con una operación `rollback`.
- El comportamiento es similar al escenario 2. La transacción solicitó deshacer sus cambios. Los cambios se restauraron en el DB buffer, pero podrían existir datos actualizados en la BD estable. EL LRM aplicará una operación UNDO.

7.2.5.2. Recuperación del estado consistente de en Oracle.

El siguiente proceso ilustra la forma en la que Oracle realiza su proceso de recuperación.

- La BD no puede abrirse si el proceso de recuperación no ha sido aplicado.
- El proceso de recuperación es completamente automático, no se puede controlar su ejecución.
- Se dispara justo después de ejecutar el comando `startup` dependiendo del resultado de la validación que detecta un estado inconsistente de la BD.
- De forma similar, Oracle usa el contenido del Log estable para realizar la recuperación. En Oracle se le conoce como ***Online redo Log***.
- ***Las correcciones se aplican en la BD volátil***. En Oracle a la BD volátil se le conoce como DB ***Buffer cache***.
- Una vez que las correcciones se han aplicado, la BD es abierta lista para recibir peticiones.
- Notar que, en este punto, la BD aún se encuentra en un estado inconsistente ya que los cambios solo se aplicaron al DB Buffer cache. Como se mencionó anteriormente, esto no representa problema alguno ya que el usuario no interactúa con la BD, interactúa con el DB Buffer cache el cual ya contiene los datos correctos. La sincronización con los ***Data files*** se realiza tiempo después.
- A esta fase de recuperación se le conoce como ***roll forward***.
- Durante este proceso, cada Online redo Log es leído, los bloques correspondientes son cargados al DB buffer cache y los cambios son aplicados.
- El proceso SMON es el encargado de detectar la necesidad de una recuperación y de abrir la BD.
- Primero se lee el archivo de control (control file) mientras la BD está en estado NO MOUNT.
- Al pasar al estado MOUNT SMON verifica los headers de los Datafiles y de los Online Redo Logs. Compara si estos están fuera de sincronía con respecto al valor del campo SCN (System Change Number). Cada cambio que se realiza en la BD implica un nuevo valor de SCN.

7.2.6. Checkpoints.

- En la mayoría de las implementaciones de los LRM, la ejecución del proceso de recuperación requiere la lectura completa del BD Log.
- Esta condición podría causar un impacto en desempeño debido a que el LRM tiene que encontrar todas las transacciones que necesitan la aplicación de una operación UNDO o REDO.
- Una forma de reducir la cantidad de información a procesar es mediante el uso de los registros $< \boldsymbol{start - checkpoint} >$ y $< \boldsymbol{end - checkpoint} >$
- La aparición de ambos registros en el DB Log indica que ***todos*** los registros contenidos en el BD Log previos a un `checkpoint` fueron ya actualizados en la BD estable.

- Lo anterior implica que la detección de registros en el Log que requieren aplicar una operación UNDO o REDO puede considerar únicamente a operaciones que se realizaron después de la última ocurrencia de un $< \boldsymbol{end - checkpoint} >$.
- Para una operación REDO, el LRM inicia la aplicación de cambios a partir de la ocurrencia $< \boldsymbol{end - checkpoint} >$ y hasta el final del archivo.
- Para una operación UNDO, el LRM inicia la aplicación de cambios a partir del final del archivo hasta encontrar $< \boldsymbol{end - checkpoint} >$.
- Pudiera existir el caso en el que en únicamente aparece el registro $< \boldsymbol{start - checkpoint} >$ Esto significa que el proceso de actualización de la BD volátil hacia la BD estable no pudo ser concluido por una falla. En esta situación el proceso se considera como incompleto.

7.2.6.1. Checkpoints en Oracle.

- Existen diversos tipos:
  - *Incremental checkpoint*
  - *Partial checkpoint*: Ocurren de manera automática durante la operación normal de la BD cuando estos son requeridos. Por ejemplo, cuando ya no existen buffers disponibles para cargar más datos en el db buffer cache. El proceso DBWn se encarga de escribir unos cuantos buffers a disco para liberar buffers.
  - *Full checkpoint*: ***Todos*** los buffers del DB Buffer cache son actualizados en los data files. Por el costo que implica dicha actualización, ocurren únicamente a petición explicita del DBA a través de la instrucción `alter system checkpoint`, o cuando se hace un `shutdown` ordenado de la instancia (NORMAL, IMMEDIATE, TRANSACTIONAL).

7.2.7. Ejemplos – Recuperación del estado consistente de una BD.

Empleando los conceptos vistos hasta el momento, a continuación, se presentan ejemplos que ilustran de forma teórica y básica el proceso de recuperación.

- Suponer que los siguientes DB Logs que se obtuvieron justo después de la ocurrencia de una falla. Para cada caso:
  A. Determinar para cada transacción la operación correctiva a realizar: UNDO, REDO, o NINGUNA.
  B. Determinar el valor que tendrán las variables en la BD estable al momento de ocurrir la falla.
  C. Determinar los valores que tendrán las variables en la BD volátil después de haber aplicado el proceso de recuperación.

Archivo 1	Archivo 2	Archivo 3	Archivo 4	Archivo 5
<t1: start> <t1: x, 1,2> <t1: y,10,20> <t2: start> <t1: commit> <t2: a,50,51> <t2: b,500,501>	<t1: start> <t1: x, 1,2> <t1: y,10,20> <t1: rollback> <t2: start> <start-checkpoint> <end-ckeckpoint> <t2: a,50,51> <t2: b,500,501> <t2: commit>	<t1: start> <t1: x, 1,2> <t1: y, 10,20> <t1: x, 2,4> <t1: y, 20,40> <t1: z, 100,200> <start-checkpoint> <end-ckeckpoint>	<start-checkpoint> <end-ckeckpoint> <t1: start> <t1: x, 1,2> <t1: y, 10,20> <t1: x, 2,4> <t1: y, 20,40> <t1: z, 100,200> <t1: commit>	<t1: start> <t1: x, 1,2> <t1: rollback> <start-checkpoint> <t2: start> <t2: x, 11,2>

Archivo 1:

- T1 requiere REDO
- Datos en la BD estable al ocurrir la falla:
  - x = 1
  - y = 10
- Datos en DB volátil después de la recuperación:
  - x = 2
  - y = 20
- *Explicación:* T1 confirmó cambios, pero no fueron actualizados en la BD estable (no hay checkpoint) por lo tanto sus cabios se deben rehacer en la BD volátil. El valor 2 y 20 corresponden a los últimos valores que fueron actualizados en el DB Log. Por lo tanto, son los valores que deben aparecer en la BD volátil.
- T2 requiere UNDO
- Datos en la BD estable al ocurrir la falla:
  - a = 50
  - b = 500
- Datos en BD volátil después de la recuperación:
  - a = 50
  - b = 500
- *Explicación:* T2 no confirmó sus cambios por lo que la BD volátil debe contener la versión original, se requiere deshacer los cambios aplicados en el DB Log (UNDO). Notar que, al aplicar el proceso de recuperación, los datos son exactamente los mismos, tanto en BD volátil como en DB estable. Esto se debe a que la transacción nunca se confirmó y tampoco ocurrió un checkpoint. En estos casos, el proceso de recuperación podría haberse omitido.

Archivo 2:

- T1 no requiere acción de recuperación.
- Explicación: Existe un checkpoint posterior a su término, por lo tanto, la BD estable fue actualizada correctamente.
- T2 requiere REDO.
- Datos en la BD estable al ocurrir la falla:
  - a = 50
  - b = 100
- Datos en BD volátil después de la recuperación
  - a = 51
  - b = 501
- Explicación: La confirmación de T2 está hasta posterior al checkpoint y existen cambios posteriores por lo que se requiere aplicar los cambios a la BD volátil ya que estos no fueron actualizados en la BD estable.

Archivo 3:

- T1 requiere UNDO
- Datos en la BD estable al ocurrir la falla:
  - x =4
  - y =40
  - z =200
- Datos en la BD volátil:
  - **z = 100**
  - y = 20
  - x = 2
  - **y = 10**
  - **x = 1**
- Explicación: Ocurruó un checkpoin y T1 no confirmó sus datos. Al existir este checkpoint, la BD estable contiene datos de una transacción que debe hacer rollback ya que nunca fue confirmada. Es decir, en la BD estable se tienen los siguientes valores: x = 4, y = 40, z = 200 lo cual es incorrecto.
- Por lo tanto, la BD volátil debe ser actualizada con los valores originales: x = 1, y =10, z = 100
- Observar la aplicación de los cambios de abajo hacia arriba. Esto es importante ya que los valores de x , y se actualizaron 2 veces, por lo que se aplican 2 actualizaciones en la BD volátil, primero x = 2, y después x = 1 que corresponde con el valor original.

Archivo 4:

- T1 requiere REDO
- Datos en la BD estable al momento de ocurrir la falla.
  - x = 1
  - y = 10
  - z = 100
- Datos en la BD volátil:
  - x = 2
  - y =20
  - **x =4**
  - **y = 40**
  - **z = 200**

- Explicación: No hay checkpoint posterior al commit de T1, por lo tanto, se requiere REDO. Los cambios de T1 no están en la BD estable.
- Observar el orden de actualización de la BD volátil de arriba hacia abajo. Esto permite obtener los valores correctos finales x = 4, y = 40 y z =200.

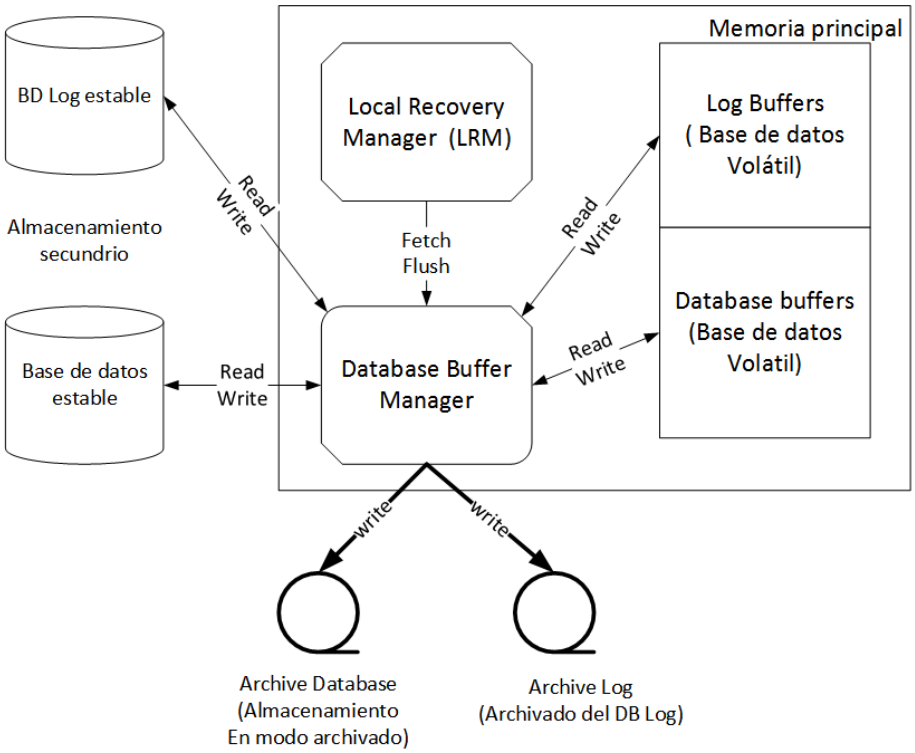
Archivo 5:

- T1 requiere UNDO
- Datos en la estable al ocurrir la falla:
  - x = 1

- Datos en BD Volátil:
    - x = 1
- T2 requiere UNDO
- Datos en BD Volátil:
    - x = 11
- Explicación:
- A pesar de existir un `start-checkpoint` al final de T1, se aplica UNDO. Esto se debe a que no aparece un registro `end-checkpoint`. Lo anterior significa que ocurrió una falla y la actualización de la BD volátil a la estable no fue exitosa. Por lo tanto, se considera como si no se hubiera realizado (propiedad de atomicidad).
- Notar que los valores de la BD estable y la Volátil son los mismos. En este caso ocurrió una situación similar al archivo 1. La transacción nunca confirmó y tampoco hubo checkpoint. En este escnario el proceso de recuperación para esta transacción se pudo haber omitido.

7.2.8. Manejo de fallas en medios.

- Existe un escenario más donde a pesar de contar con este mecanismo de recuperación pudiera existir una situación en la que sea imposible recuperar el estado consistente de la BD.
- ¿Qué sucede si los medios de almacenamiento fallan?
- Tanto la BD estable como el Log estable se almacenan en medios de almacenamiento secundario y también pueden fallar.
- La solución a estos problemas es la creación de redundancia de almacenamiento.
- Típicamente se crean grupos de almacenamiento secundario formados por diferentes medios: discos, e inclusive cintas.
- La redundancia de almacenamiento garantiza la posibilidad de recuperar una copia de alguno de los miembros del grupo en caso de una falla en otro miembro.
- Cada miembro del grupo debería ser administrado por un controlador independiente y minimizar así la posibilidad de fallas genéricas del grupo.
- En Oracle, existen diversas herramientas, configuraciones y opciones para implementar este requerimiento, tanto para los Data files, como para los DB Logs así como para otros archivos importantes.
- Con esta nueva consideración, la versión final de la arquitectura de recuperación quedará de la siguiente manera:



7.3. PROTOCOLOS DE CONFIABILIDAD EN BDD

- En su conjunto permiten implementar los mismos requerimientos de confiabilidad revisados para bases de datos centralizadas, pero ahora considerando a un conjunto de BD distribuidas.
- Existen protocolos para implementar los comandos del LRM:
  - Protocolos para implementar el comando **`commit`**.
    - ¿Cómo ejecutar el comando para una transacción distribuida?
  - Protocolos para implementar el comando **`rollback`**
    - ¿Cómo deben comportarse los sitios de una BDD cuando uno de ellos presenta una falla?
    - La ocurrencia de una falla en un sitio no debería hacer esperar a los otros sitios a que el sitio dañado se reestablezca.
  - Protocolos para recuperación
    - ¿De qué manera cada sitio debe ejecutar un proceso de recuperación en caso de una falla?

7.3.1. Conceptos fundamentales comunes a los protocolos:

- Coordinador: Se refiere al proceso que inicia una transacción en alguno de los sitios.
- Participante: Los procesos que se encuentran en otros sitios que ejecutan instrucciones de la transacción.

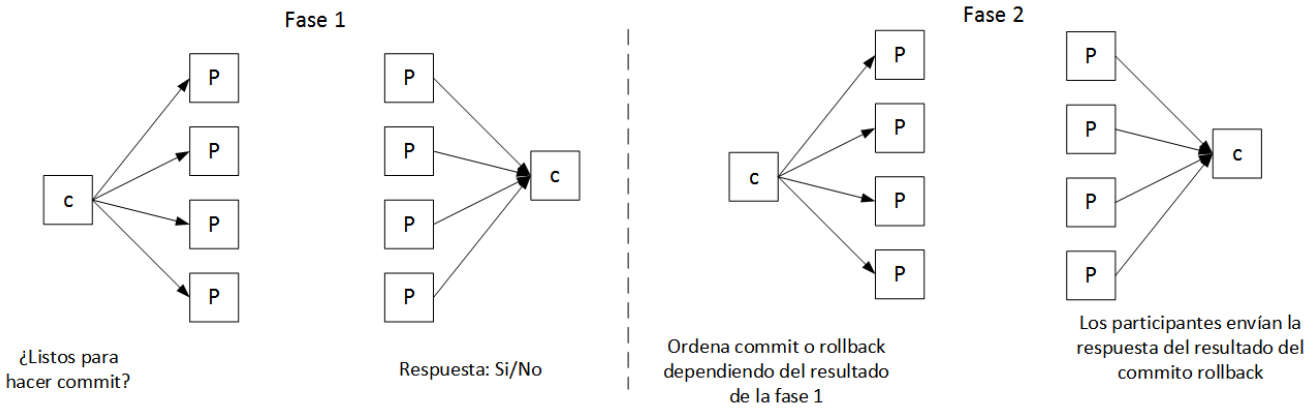
7.3.2. Protocolo commit de 2 fases centralizado (2PC: Centralized Two Phase Commit Protocol)

- Representa a uno de los protocolos más sencillos empleado para garantizar la correcta confirmación de una transacción y garantizar la propiedad de atomicidad de las transacciones.
- Como su nombre lo indica el protocolo se divide en 2 fases:
    - Fase 1: El coordinador obtiene la lista de todos los participantes que están involucrados en una transacción y realiza la siguiente pregunta a cada uno: ¿Están listos para confirmar los cambios en sus BD locales?
    - Fase 2: Si la respuesta anterior es afirmativa, el coordinador ordena a todos los participantes confirmar sus cambios en sus BD locales.

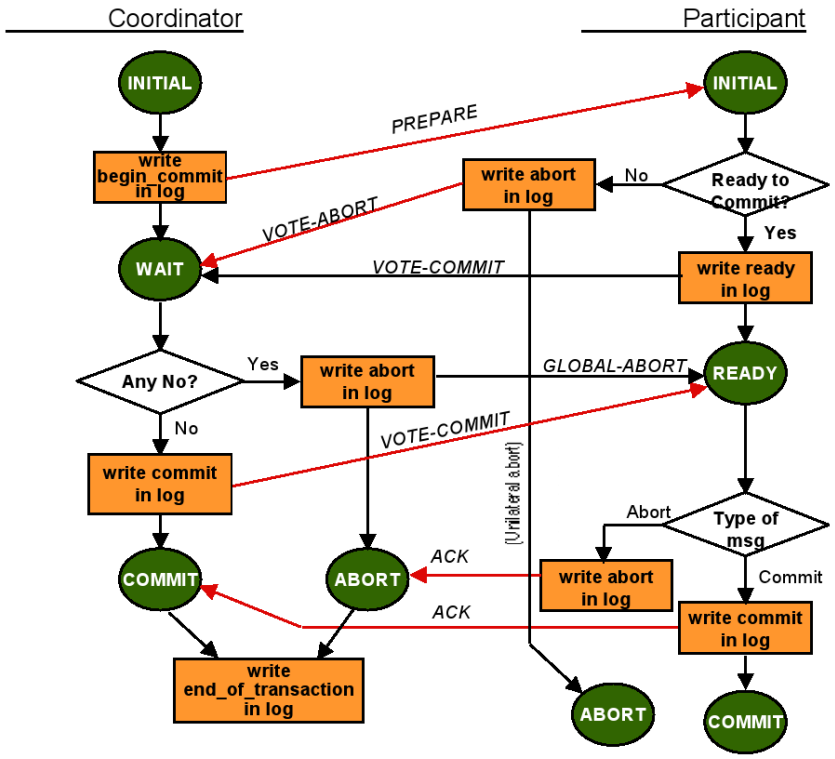
7.3.2.1. Regla global para commit

- El coordinador ordenará un rollback de la transacción distribuida si y solo si al menos un participante aborta la transacción.
- El coordinador ordenará un commit de la transacción distribuida si y solo si todos los participantes confirman sus cambios.
- No confundir el término centralizado con BD centralizadas. Este se refiere a que solo existe 1 coordinador en un sitio llamado sitio central y N participantes en los demás sitios.

Este protocolo se ilustra en la siguiente imagen:



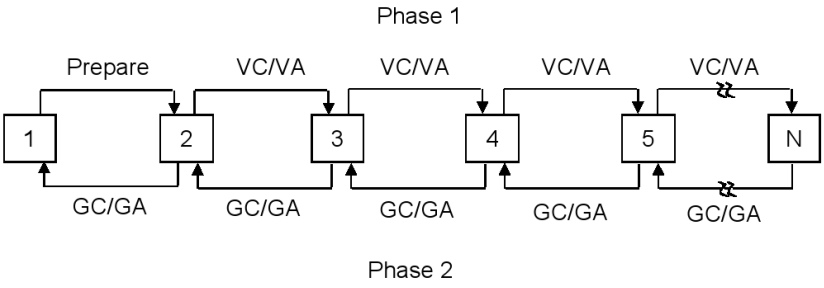
El flujo de comunicación entre estos componentes se ilustra en la siguiente imagen:



- Observar las entradas especiales en el DB Log:
  - Begin_commit
  - Ready
  - Abort
  - End_ transaction
- Como se puede observar el flujo es similar al de una BD centralizada pero ahora considerando N participantes y un coordinador.

7.3.3. Protocolo commit de 2 fases lineal.

- La diferencia con el anterior es que existe un orden de comunicación: Comunicación lineal
- Minimiza el tráfico de mensajes, pero la respuesta es más lenta ya que la comunicación al ser lineal, se serializa.



VC: Vote-Commit, VA: Vote-Abort, GC: Global-commit, GA: Global-abort

7.3.4. Protocolo commit de 2 fases distribuido.

- En este protocolo se incrementa la comunicación entre sitios
- La fase 2 ya no es necesaria ya que cada participante envía sus respuestas a todos los demás participantes y al coordinador.
- Debido a que cada participante tiene las respuestas de los demás, este puede decidir por sí solo que acción ejecutar. De aquí que la fase 2 ya no es necesaria.