# Seguridad e Integridad de datos: veremos

## Transacciones

* Propiedades de las transacciones.
* Estados de las transacciones.

## Transacciones monousuarias

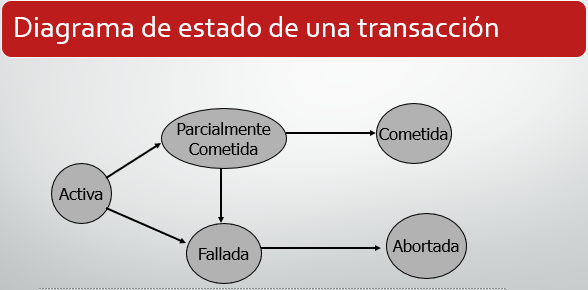
* Atomicidad
* Protocolos

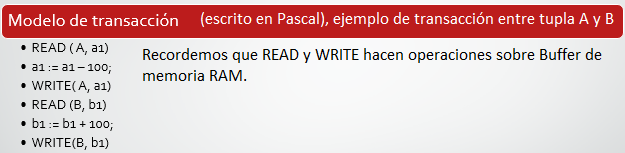
## Transacciones centralizadas

* Aislamiento
* Consistencia
* Durabilidad

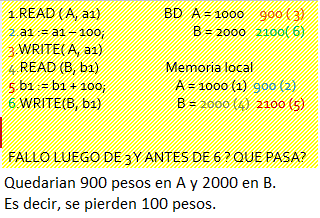
Integridad de datos: la perdemos cuando a partir del mismo uso cotidiano terminamos rompiendo la integridad de la BD

# Transacciones

* Es una colección de operaciones que forman una única unidad lógica de trabajo, esa transacción tiene un efecto sobre la BD.
* Propiedades ACID que deben cumplir las transacciones
  + **A**tomicidad: todas las operaciones de la transacción se ejecutan, o no lo hacen ninguna de ellas (por si ocurren errores).
  + **C**onsistencia: la ejecución aislada de la transacción conserva la consistencia de la base de datos (siempre con la idea de que la transacción esté bien escrita).
  + Aislamiento (**i**solation): cada transacción ignora el resto de las transacciones que se ejecutan de forma concurrente en el sistema, cada transacción actúa como única.
  + **D**urabilidad: una transacción terminada con éxito realiza cambios permanentes en la BD (dejando la BD en un estado consistente), incluso si hay fallos en el sistema.
* Estados de una transacción
  + Activa: estado inicial, estado normal durante la ejecución y hasta que se ejecuta la última instrucción.
  + Parcialmente cometida: después de ejecutarse la ultima instrucción.
  + Fallada: luego de descubrir que no puede seguir la ejecución normal.
  + Abortada: después de haber retrocedido la transacción y restablecido la BD al estado anterior al comienzo de la transacción.
  + Cometida: tras completarse con éxito.
* 
  + ¿Por qué puede fallar en estado de parcialmente cometida?
    - Puede ocurrir por una escritura sobre la BD (Update).
    - ¿Por qué?, simple, las escrituras se hacen primero sobre Buffer, recordemos que es memoria RAM (volátil).
    - Puede ocurrir que se pierda dicha información, por lo que se puede llevar a un error que implique el paso al estado de transacción fallida.

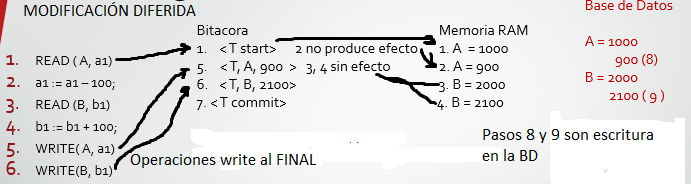
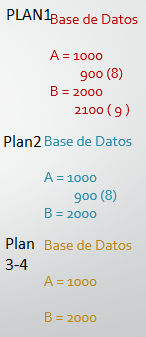


* Las transacciones pueden generar problemas, sí, lo sabemos, y en base al tipo del sistema que tengamos, vamos a tener que actuar de una forma u otra.
  + Sistemas monousuario: en el sistema monousuario siempre hay una transacción ocurriendo en cada momento, al ser monousuario no se permiten varias transacciones en el mismo tiempo, por lo que de las propiedades ACID se eliminan la de consistencia, aislamiento y durabilidad, sólo debemos asegurar que se cumpla la propiedad de atomicidad.
  + Sistemas concurrentes.
  + Sistemas distribuidos.
  + Distribuidos (no se ve en ésta materia) y concurrentes no nos interesan por ahora.

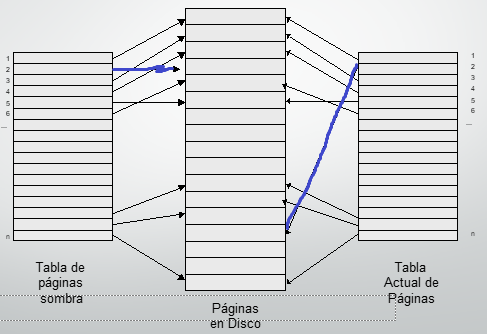


* ¿Qué hacer luego de dicho fallo?
  + Reejecutar la transacción fallada 🡪 no sirve.
  + Dejar el estado de la BD como está 🡪 no sirve.
  + SE PERDIERON 100 PESOS LOCO, ESTÁ MAL ESO.
* ¿Cuál fue nuestro problema?
  + Modificar la base de datos sin seguridad de que las transacciones se puedan cometer o no.
  + Solución a eso?
    - Indicar las modificaciones.
    - Es decir, ya saber de antemano que voy a hacer antes de hacerlo.
  + ¿Qué soluciones podríamos implementar para cumplir lo anterior?
    - Solución usando registro histórico.
    - Solución usando doble paginación.
    - Las veremos a continuación.

**Registro histórico**

* Bitácora (registro cronológico) o por su nombre en inglés “Log”.
  + Contenido de la bitácora **en orden** (es escrita antes de que se realice la transacción, estará en disco rígido y especifican las operaciones a futuro que se realizarán en la BD, es un archivo más)
    - <T iniciada> **T es transacción**
    - <T, E, Va, Vn>
      * Identificador de la transacción (T).
      * Identificador del elemento de datos (E, registra que valor se va a cambiar).
      * Valor anterior (Va, de ese atributo).
      * Valor nuevo (Vn, de ese atributo).
    - <T Commit> **Se agrega al final cuando la transacción fue exitosa.**
    - <T Abort> **Se agrega al final cuando la transacción fue fallida.**
* Las operaciones sobre la BD deben almacenarse luego de guardar en disco el contenido de la bitácora (la bitácora sigue siendo un archivo más).
* Dos técnicas de bitácora para ello:
* **Modificación diferida de la BD**
  + Las operaciones write se aplazan hasta que la transacción esté parcialmente cometida (hasta el final), en ese momento se actualiza la bitácora y la BD.
  + ****
  + **Plan 1: 1,2,3,4,5,6,7,8,9,FALLO**
    - Como el fallo ocurre después de 9, entonces la BD tiene datos correctos, no se hace nada.
  + **Plan 2: 1,2,3,4,5,6,8, FALLO**
    - Como el fallo ocurre después del 8 y antes del 9 la BD queda con datos erróneos, ALGO se debe hacer.
  + **Plan 3: 1,2,3,4,5,6,7, FALLO**
    - El fallo ocurre antes de la escritura sobre la BD, pero SÍ escribí sobre la bitácora, se debería hacer algo?.
  + En los planes 1,2,3 la bitácora queda en exactamente el mismo estado, con su start y su commit, por lo que nos sirve de indicativo.
  + En cualquier bitácora cuando tengo un start y commit, y ocurre un fallo, debo obligatoriamente reejecutar la transacción ya que de esa forma aseguro que la BD queda bién.
  + Se aplica un REDO
  + **Plan otro fallo antes del 7**
    - La base de datos está bien ya que no se realizaron cambios en la bitácora ni en la BD.
    - Por lo que no debo seguir ejecutando dicha transacción
  + ****
  + En modificación diferida (no necesitamos valor viejo):
    - Ante un fallo y luego de recuperarse
      * REDO(Ti) para toda transacción i que tenga un start y un commit en la bitácora.
      * Si no tiene commit entonces se ignora, dado que no llegó a hacer nada con la BD
* **Modificación inmediata de la BD**
  + La actualización de la BD se realiza mientras la transacción está activa y se va ejecutando.
  + Se necesita el valor viejo, pues los cambios se fueron efectuando (diferencia con modificación diferida).
  + Ante un fallo, y luego de recuperarse:
    - REDO(Ti), para todo Ti que tenga un start y un commit en la bitácora.
    - UNDO(Ti), para todo Ti que tenga un start y NO un commit.
  + Es un protocolo más complejo que modificación diferida, por que tiene otra operación adicional.
    - Tiene mejor distribuida la carga de trabajo (ya que diferida usa el disco rígido mucho al final de todo).
* Transacción:
  + Condición de idempotencia: yo puedo rehacer o deshacer una transacción de bitácora 1 o 1000 veces que el resultado siempre es el mismo.
* Buffers de bitácora:
  + Grabar en disco c/registro de bitácora insume gran costo de tiempo 🡪 se utilizan buffer, como proceder…
    - Transacción está parcialmente cometida después de grabar en memoria secundaria el commit en la bitácora.
    - Un commit en la bitácora en memoria secundaria, implica que todos los registros anteriores de esa transacción ya están en memoria secundaria.
    - Siempre grabamos primero la bitácora y luego la BD.
      * WRITE(Bitacora)
      * WRITE(BD)
      * OUTPUT(Bitacora)
      * OUTPUT(BD)
* Puntos de verificación
  + Ante un fallo, ¿Qué hacer?
    - REDO, UNDO: según el caso.
  + Revisar la bitácora:
    - Desde el comienzo?: probablemente gran porcentaje del trabajo esté correcto y terminado, por eso lleva MUCHO tiempo la revisión de bitácora.
  + Checkpoints (solo para sistemas monousuario)
    - Se agregan periódicamente indicando desde allí hacia atrás todo OK (ósea, no se debe revisar nada en ese lugar).
    - Periodicidad en la cual se debe poner un checkpoint?, no hay respuesta
      * Checkpoints muy cerca entre sí: debo asegurar que el buffer debe estar bajado a disco, puede agregar mayor carga al sistema.
      * Checkpoints más lejos entre sí: en un caso de fallo debo revisar mucho.
      * Se debe buscar un punto intermedio.

**Doble paginación**

* **Paginación en la sombra:**
  + Ventaja: menos accesos a disco, en caso de fallo la recuperación es muy rápida.
  + Desventaja: complicada de aplicar en sistemas concurrentes o distribuidos.
  + N paginas equivalente a páginas del SO.
    - Tabla de páginas actual.
    - Tabla de páginas sombra (nos ayuda ante fallos).
    - Ambas tienen la misma información al comenzar
  + ****
  + Vemos que la pagina 2 de la sombra apunta a la pagina 4 del disco, bueno, esto es el estado ANTERIOR a que ocurra una transacción.
  + El resultado de dicha transacción lo vemos reflejado en la pagina 2 de la tabla de paginas actual, vemos que el puntero escribió el resultado en otro sector, por lo que…
    - Si hay error: entonces puedo volver al estado anterior de la BD usando la tabla de páginas sombra.
    - Si no hay error: reescribo la tabla de paginas sombra.
* Ejecución de la operación **escribir**
  + Ejecutar **entrada**(x) si la página i-ésima no está todavía en memoria RAM.
  + Si es la primer escritura sobre la página i-ésima, modificar la tabla actual de páginas así:
    - Encontrar una página en el disco no usada.
    - Indicar que a partir de ese momento está ocupada.
    - Modificar la tabla actual de página indicando que la i-ésima entrada ahora apunta a la nueva página.
* En caso de fallo y luego de la recuperación
  + Copia la tabla de páginas sombra en memoria principal.
  + Abort automáticos, se tienen la dirección de la página anterior sin las modificaciones.
* Ventajas
  + Elimina la sobrecarga de escrituras del log.
  + Recuperación rápida (no existe REDO o UNDO).
* Desventajas
  + Sobrecarga en el compromiso: la técnica de paginación es por CADA transacción.
  + Fragmentación de datos: cambian las ubicaciones de los datos de forma continua.
  + Garbage collector: ante un fallo queda una página que no es más referenciada.