Resumen para Promocion / Final DBD

Clase 1:

Definiciones generales de las base de datos:

Una base de datos es una colección de datos relacionados

No toda colección de datos va a generar una base de datos. Para una base de datos necesito una colección de datos relacionados.

Una BD es una colección coherente de datos con significados inherentes. Un conjunto aleatorio de datos no puede considerarse una BD.

Una BD se diseña,construye y completa de datos para un propósito especifico.

Una BD está sustentada físicamente en archivos en dispositivos de almacenamiento persistente de datos:

Donde se guarda una BD? En disco **De que manera?** a partir de archivos

La manipulación de una BD incluye funciones CRUD. (Alta, Lectura, Actualizar, Baja)

Que es un DMBS/SGBD:

Es una colección de programas que permiten a los usuarios crear y mantener la BD. Lo que no permite hacer el DBMS es diseñar una BD, porque no se encarga de eso.

Con el DBMS se crea y se mantiene, no se diseña.

Objetivos de un DBMS:

Evitar redundancia e inconsistencia de datos.(la redundancia es una mala palabra en el contexto de base de datos, el problema de la redundancia es la inconsistencia de datos.)

Permitir accesos a los datos en todo momento.

Evitar que situaciones no deseadas puedan ocurrir: a partir de la integridad de datos.

Actores involucrados con una BD:

DBA o ADB: Responsable de la BD. Autoriza accesos,coordina y vigila la utilización de recursos de hardware y software, responde ante problemas de violacion e seguridad o respuesta lenta del sistema.

Diseñador de DB: Dado un problema, se encarga de diseñar una BD.

Analista de Sistemas: Determina los requerimientos de los usuarios

Programadores: Se encarga de programar/construir la BD.

Usuarios(distintos tipos): Roles

Clase 2.1:

Cuando hacemos un modelado de una bd, debemos generar abstracciones, porque cada usuario o cada persona que interactué con el sistema, va a tener necesidades diferentes.

Ejemplos de distintas necesidades:

Para un alumno: Cuantas materias aprobó, a cuantas materias esta inscripto, cuantas materias tiene aprobadas

Para un profesor: la visión va a ser de su cátedra. Saber cuántos aprobaron, quienes van a rendir, quienes desaprobaron

Para un administrativo: No le importa un alumno en particular ni una cátedra en particular, maneja todo el contenido de la información.

Ante un mismo problema, hay distintas vistas, cada vista tiene diferente necesidad y diferente visión de los datos. Es importante que al momento de diseñar una BD, se puedan satisfacer las necesidades de cada actor/vista.

La BD puede ser una sola, que contenga toda la información, pero puede tener diferentes vistas.

Abstracciones:

Vision: Ve solo los datos de interés

Conceptual: Que datos se almacenan en la BD y que relaciones existen entre ellos.

Físico: Describe como se almacenan realmente los datos (archivos, hashing,arboles...etc)

Diseño:

Modelos basado en objetos, nos hace muy fácil interactuar con el cliente, para ver si realmente pudimos entender lo que el cliente quiera

Al momento de realizar un diseño:

La primer etapa es el modelado conceptual: Representación abstracta .Genérico, alejado de DBMS, alejado del producto particular. **Objetivos:** Representar la información de un problema con un alto nivel de abstracción, captar las necesidades del cliente, mejorar la interacción entre cliente/desarrollador.

La segunda etapa es el modelo Lógico: Representación en una computadora. Mas especifico, orientado a un tipo de DBMS, alegajo del producto particular

La tercer etapa es el modelo físico: Determinar estructuras de almacenamiento físico. Especifico, orientado a un producto

Características del diseño conceptual:

Formalidad: Cada elemento representado sea preciso y bien definido con una sola interpretación posible.

Nominalidad: Cada elemento tiene una unica representación posible.

Simplicidad: El modelo debe ser fácil de entender por el cliente y por el desarrollador.

Clase 2.2:

Identificadores: Es un atributo o conjunto de atributos que permite reconocer una entidad de manera univoca dentro del conjunto de entidades.

Pueden ser simples o compuestos, internos o externos.

En el modelado de datos, debemos definir UNA sola clave primaria/identificador

Una clave candidata es una clave que permite reconocer una entidad de manera univoca.

Una clave candidata PUEDE ser una clave primaria/identificador.

Un identificador DEBE ser una clave candidata y clave univoca

Clase 3.1:

Cuando un modelo es completo?
un modelo es completo cuando representa todas las características del dominio de aplicación (análisis de requerimientos). →revisar problemas, requerimientos, enunciados, y viendo que todo se cumple.

Decisiones sobre atributos derivados:

Atributos derivados: atenta contra la minimalidad.

- Ventajas de dejar? Nos permite conseguir un valor X muy rápido, es más eficiente que ESTÉ en la BD. El tema con los atributos derivados es que hay que mantenerlos ACTUALIZADOS.
- Ventajas de sacar? Si lo saco debo calcular el valor X, por lo que implica ineficiencia.
- Resumen → cambia mucho, se usa poco, ENTONCES SACO. Cambia poco, se usa mucho, LO DEJO. (decisión extrema, pero es decisión nuestra)

Decisiones sobre los ciclos:

Los ciclos PUEDEN generar información redundante. Si generan información redundante, atentan contra la minimalidad.

- Al fin y al cabo debemos tener una balanza para decidir si sacar la relación que provoca redundancia, o dejarla.

Ejemplo: yo quiero ir de Quilmes a Mar Del Plata, y quiero ir por ruta 2 pero está cortada, puedo ir por otra ruta y llegar a MDP?, la respuesta es SÍ, por lo tanto la ruta 2 es redundante.

- ➔ Si yo tengo más de un camino entre una entidad a otra, entonces hay redundancia que ATENTA CON LA MINIMALIDAD.
- ➔ El tema final es el siguiente, sí el camino adicional que atenta contra la minimalidad se trata de un camino que agiliza y que EFECTIVAMENTE es usado bastante entonces nos conviene tener dicha redundancia a mano.
- ➔ Puede ocurrir que exista un ciclo, pero que éste sea mínimo, es decir, no hay redundancia, y todas las relaciones son importantes: si saco alguna relación no se cumple la interpretación correcta del problema.

Clase 4.1:

Como elegir una clave primaria en el MODELO FÍSICO ?

- ➔ Podría preguntarme: ¿Cuál clave se va a buscar más?, y en base a la respuesta, elegir la respuesta como clave primaría (por el método de hash se va a conseguir más rápido el dato), el problema es que nos vamos a olvidar de hacer búsquedas secuenciales EFICIENTES usando dicha clave primaria.
- ➔ La clave primaria está propensa a errores (no está bueno).
- ➔ Desde fines del siglo pasado se propone que el atributo que sea clave primaría sea el atributo **AUTOINCREMENTAL.**

➔ Qué significa esto?:

- Significa definir en cada tabla un nuevo atributo que hasta ese momento no estaba presente en el modelo.
- Dicho atributo va a ser manipulado por el DBMS (el DBMS le pone un valor NUNCA REPETIDO) → va a ser ideal porque: está manejado por la DBMS, permite búsquedas eficientes y está más allá del usuario (el usuario no sabe qué existe).
- Por lo tanto, **LA CLAVE PRIMARÍA DEBE SER UN VALOR AUTOINCREMENTAL.**
- Los identificadores restantes pasan a ser claves candidatas.

Clave foránea e integridad referencial:

La integridad referencial asegura que las relaciones entre las tablas de una base de datos sean validas y que los datos sean consistentes. Establece que si una tabla hace referencia a otra (por ejemplo con una clave foranea), esa referencia debe ser valida.

Ejemplo: tabla cliente y tabla pedido, cada pedido debe estar asociado a un cliente existente. No puede haber un pedido sin un cliente relacionado.

La integridad referencial, se encarga de definir restricciones sobre tablas relacionadas.

4 Tipos de Integridad Referencial (IR):

- Restringir operaciones
- Realizar operaciones en cascada.
- Clave foránea en nulo
- No hacer ndada.

Tener en cuenta que solo se puede optar por un solo tipo de IR.

Explicaciones de cada tipo de integridad referencial:

Restringir operación (RESTRICT): No permite hacer cambios que rompan la relación entre tablas. Si se intenta borrar o modificar un registro en una tabla, y ese registro esta siendo utilizado en la tabla relacionada.

EJEMPLO: No se puede borrar un cliente si tiene pedidos asociados.

Realizar operación en cascada (CASCADE): Los cambios en la tabla principal, se propagan a la tabla relacionada.

EJEMPLO: Si se elimina un cliente, todos sus pedidos se eliminaran.

EJEMPLO: Si se cambia el id_cliente en la tabla principal, la tabla de pedidos también se actualiza con el nuevo valor.

Clave foránea en nulo (SET NULL): Si eliminar o modificas un registro en la tabla principal, los valores relacionados en la tabla dependiente se vuelven NULL.

No hacer nada (NO ACTION): No ocurre ningún cambio automático, pero deja que tú controles qué hacer.

- **Ejemplo**: Si borras un cliente que tiene pedidos, simplemente no pasa nada hasta que se decida cómo manejar esa situación (se puede borrar manualmente los pedidos o hacer algo más).

Para garantizar la integridad referencial, se usa la clave foránea. (también llamada CF o FK)

> CF → atributo/s de una tabla que en otra tabla es/son CP y que sirven para establecer una relación entre ambas tablas.

Restricciones:

Restricciones de dominio:

- Especifican que el valor de cada **atributo A** debe ser un valor atómico del dominio de A. Ósea qué si mi atributo A es un tipo integer, **me indica que sólo puedo ingresarle integers.**

Restricciones de clave:

- Si yo defino un atributo como clave primaria o candidata, éste **no puede tener valores repetidos.**

Restricciones sobre nulos:

- Evita que un atributo tome nulo en caso de no ingresarle valor.

Restricciones de integridad:

- Ningún valor de la clave primaria puede ser nulo.

Restricción de integridad referencial:

- Especificada entre dos tablas, sirve para mantener consistencia entre ambas tuplas.
- Establece que una tupla en una tabla que haga referencia a otra relación deberá referirse a una tupla existente en esa tabla.
- Clave foránea está representada por un atributo de una relación que en otra es clave primaria.

Clase 4.2:

Dependencias Funcionales (DF):

- Una DF es una restricción entre dos conjuntos de atributos de la BD
- La restricción nos indica que sí T1 y T2 son dos tuplas cualesquiera en la tabla r y que sí T1[X] = T2[X] entonces debe ocurrir que T1[Y] = T2[Y]
  - Esto significa que los valores del componente Y(atributos del subconjunto Y) de una tupla de r dependen de los valores del componente X(atributos del subconjunto X).
  - r → representa una tabla (que tiene sus varios atributos), X e Y representan un subconjunto de atributos de dicha tabla (ósea no todos los atributos de r).
  - R → representa los atributos de esa tabla r.

Ejemplos de DF:

Ejemplo1: NroDpto clave primaria, nombre es clave unívoca

➔ Departamento = (NroDpto, nombre, #empleados)

- Nrodpto → nombre? SÍ
- Nrodpto → #empleado? SÍ
- Nombre → #empleado??
  - Si nombre es clave unívoca, entonces sí cumple.
  - Si nombre NO es clave unívoca, entonces no cumple.

Ejemplo2: NroEmpl es clave primaria

➔ Empleado = (NroEmpl, nombre, DNI, Sexo)

- NroEmpl → nombre? SÍ.
- NroEmpl → dni? SÍ.
- NroEmpl → sexo? SÍ.
- DNI → NroEmpl??
  - Cuando sí? → cuando es clave unívoca (consideración propia).
  - Que otras dependencias pueden surgir con el DNI?
    - DNI → nombre.
    - DNI → NroEmpl.
    - DNI → Sexo.

Conclusión: tanto las claves primarias como las unívocas pueden determinar a los otros atributos.

Dependencia Funcional Completa:

>> *Definición:* Sí A y B son atributos de una tabla r, B depende funcionalmente de manera completa de A, sólo si B depende de A pero de ningún subconjunto de A, .

Preguntas:

- (nro_empl, nro proy) → nombre_empleado?
- Nro_empl → nombre_empleado
- ¿Cuál es la completa? Nombre_empleado cumple la definición ya que depende de nro_empl, siempre y cuando despedacemos (nro_empl, nro_proy).
-
- (Nro_empl,nro_proy) → nombre_proyecto
- Nro_pry → nombre_proyecto.
- Idem anterior

Dependencia Funcional Parcial:

>> *Definición:* A → B es una dependencia funcional parcial si existe algún atributo que puede eliminarse de A y la dependencia continúa verificándose.

- (nro_empl, nro proy) → nombre_empleado?
- Nro_empl → nombre_empleado
- La primera es una dependencia parcial, ya que podrías eliminar un atributo de A, y la dependencia seguiría siendo correcta.
-
- (Nro_empl,nro_proy) → nombre_proyecto
- Nro_pry → nombre_proyecto.
- Idem anterior

Dependencia Funcional Transitiva:

>>*Definición:* es una condición en la cuál A, B y C son atributos de una tabla tales que A → B y B → C, entonces C depende transitivamente de A mediante B (A → C). Dicha dependencia repite información (malo).

- Ejemplo:
  - Nro_empleado → nombre,posición, salario, nro_depto, nombre_depto (explícitamente, podría no estar y sería implicito que A → C).
  - Nro_depto → nombre_depto.
    - A = nro_empleado.
    - B = nro_depto.
    - C = nombre_depto.

Primera forma normal (1NF):

Una tabla que contiene atributos polivalentes no esta en 1NF.

Un modelo esta en 1NF si para toda tabla r del modelo, cada uno de sus atributos son monovalentes.

Ej persona = (dni, nombre, sexo, títulos* (polivalente))

- Solución a la polivalencia.
  - Persona = (dni, nombre, sexo)
  - Títulos = (id_titulo, descripción)

  Posee = (dni, id_titulo) → relación entre Persona y Títulos

Segunda forma normal (2NF)

Una tabla que tenga atributos que dependan parcialmente de otro no está en la 2NF.

Un modelo está en 2NF sí y sólo sí está en 1NF(incremental) y para toda tabla r del modelo no existen dependencias parciales.

Ej renta = (#cliente, #propiedad, nombrecliente, nombre propietario, monto renta, fecha inicio, duración)

- Ejemplos
  - Ejemplo1
  - Dependencias:
    - #Cliente, #Propiedad → nombreCliente, nombrePropietario, monto renta, fecha inicio, duración (DF)
    - #Cliente → nombreCliente (DP)
    - #Propiedad → nombrePropietario (DP)
    - Debo solucionar esas dos dependencias.
  - Solución:
    - Cliente = (#Cliente, nombre)
    - Propiedad = (#Propiedad, nombrePropietario)
    - Renta = (#Cliente,#Propiedad, monto renta, fecha inicio, duración)

Tercera forma normal (3NF)

Una tabla que tenga atributos que dependan transitivamente de otro no está en 3NF.

Un modelo está en 3NF sí y sólo sí está en 2NF(incremental) y para toda tabla r del modelo no existen dependencias transitivas.

Ejemplos:

- Ejemplo1: empleado = (DNI empleado, nombre empleado, #depto, nombre depto)
- Dependencias:
  - DNI empleado → nombre empleado, # depto, nombre depto (DF)
  - # depto → nombre depto (DT)
- Solución:
  - Empleados = (DNI empleado, nombre empleado, #Depto)
  - Deptos = (# depto , nombre depto)

Boyce Codd Forma Normal (BCNF):

Dependencia de Boyce Codd: es una dependencia poco común, puede generar grandes repeticiones de información

Un modelo está en BCNF sí y sólo sí está en 3NF (incremental) y para toda tabla r del modelo no existen dependencias de Boyce Codd.

Otra acepción de Boyce Codd-

- Una tabla está en BCNF si y sólo sí todo determinante (La X de X→Y) es una clave candidata o primaria.

➢ Ejemplos:
- Ejemplo entrevista = (#cliente, fecha entrevista, hora entrevista, empleado, lugar entrevista)
- Dependencias:
  - #cliente, fecha entrevista → hora entrevista, empleado, lugar entrevista. (CP)
  - Empleado, fecha entrevista, hora entrevista → #cliente (CC)
  - Lugar entrevista, fecha entrevista, hora entrevista → empleado, #cliente (CC)
  - Empleado, fecha entrevista → lugar entrevista.
- Analizando la cuarta DF:
  - El determinante no es CC o CP (solo puedo sacar el lugar de entrevista, no puedo saber ni el cliente ni la hora de entrevista, también un empleado en una misma fecha puede repetirse) → no está en BCNF.

- Como resolvemos el problema anterior
  - Entrevista= (#cliente, fechaentrevista, horaentrevista, empleado)
  - lugarreunión = (empleado, fechaentrevista, lugarenrevista)
- En la conversión realizada
  - #cliente, fechaentrevista → hora_entrevista, empleado (CP)
  - Empleado, fechaentrevista, horaentrevista → #cliente (CC)
  - Empleado, fecha entrevista → lugarentrevista (CP)
  - Pero se ha perdido una CC del problema
    - Lugarentrevista, fechaentrevista, horaentrevista → empleado, #cliente (CC)

- ¿Entonces? ¿Qué se hace?
  - La decisión de si es mejor detener el proceso en 3NF o llegar a BCNF depende de dos factores:
    - La cantidad de redundancias que resulten de la presencia de una DF de Boyce Codd.
    - De la posibilidad de perder una CC con la cuál se podrían realizar muchos más controles sobre los datos.
    - ¿Qué prefiero? Redundancia y mayor control sobre los datos, o no tener redundancia.
    - Se dice que Boyce Codd es más duro ya que tengo que sacrificar algún control que tenía por una clave.
    - Es una decisión enteramente del diseñador.

Cuarta Forma Normal (4NF):

- Un modelo está en 4NF sí y sólo sí está en BCNF y para toda tabla r del modelo sólo existen dependencias multivaluadas triviales.
  - ¿Cuáles son triviales? →
    - A→→B
    - A→→C
  - Y…
    - (A, B) →→C
    - (A, C) →→C
  - La dependencia multivaluada no trivial genera una mayor repetición de información.
- Del ejemplo anterior:
  - (Sucursal, empleado) →→propietario.
  - (Sucursal, propietario) →→ empleado.
- Pero yo también podría…
  - Sucursal →→Propietario.
  - Sucursal →→Empleado.

- Entonces, como el par marcado en verde puede ser reducido a un solo atributo que me obtiene la MISMA información (celeste). Decimos que el par es una dependencia multivaluada NO trivial.
- Solución:
    - T1 = (sucursal, empleado)
    - T2 = (sucursal, propietario)

Quinta Forma Normal (5NF):

- Un modelo está en 5NF si está en 4NF y no existen relaciones con dependencias de combinación.
- Una dependencia de combinación es una propiedad de la descomposición que garantiza que no se generen tuplas espurias al volver a combinar las relaciones mediante una operación del álgebra relacional.

Clase 5 y 6: Consultas AR y SQL

Lenguajes de consulta: Son utilizados para manipular una BD.

Dos tipos de lenguajes:

Procedurales: se define que hacer, y cómo hacerlo. Una secuencia de instrucciones y operaciones.

No procedurales: se solicita directamente la información requerida.

Las consultas representan el 80% de las operaciones registradas sobre una BD.

Álgebra relacional:

Lenguaje de consultas procedimental.

Operaciones de uno o dos relaciones (tablas) de entrada que generan una nueva relación (tabla) como resultado (es temporal).

Operaciones fundamentales:

- **Unitarias:** selección, proyección, renombre. (operan sobre una tabla)
- **Binarias**: producto cartesiano, unión, diferencia. (operan sobre dos tablas)
- Hay que resolver problemas de conjuntos.

Despues explica todas las operaciones de AR y SQL (asumo que para el teorico o final no son importantes porque es mas practica que teoría. Pero se pueden tener en cuenta algunas cuestiones sobre los productos cartesianos, naturales, etc.)

Clase 7:

Componentes que conforman el costo de la ejecución de una consulta.

Costo de acceso: Es el costo de acceso a almacenamiento secundario, acceder al bloque de datos correspondiente en el disco. Medido en milisegundos. Es costoso.

Costo de computo: Es el costo de realizar operaciones sobre RAM (filtrado de tuplas sobre memoria RAM). Medido en nanosegundos.

Costo de comunicación: Es el costo que implica enviar la consulta (resuelta sobre un servidor) y los resultados correspondientes (si es un sistema distribuido: una base de datos dispersa en varios servidores, se debe procesar la consulta en varios servidores).

Proceso de recepción de consulta:

- La consulta recibida se transforma en un formato interno usando el parser (reescritura de consulta que implica una consulta más fácil de leer para el DBMS).
- Sobre esa consulta reescrita se le aplica el proceso de optimización. El DBMS optimiza las consultas. De forma que la consulta sea expresada de la forma más eficiente posible.
- Una vez conseguida dicha forma eficiente, se eligen los índices para hacer el uso más optimo de las tuplas (esto implica reducir los accesos a discos).

Clase 8_1: Seguridad e integridad de datos y transacciones

Cuando hablamos de seguridad: Hablamos de uso indebido de la base de datos y eventualmente rotura de la base de datos a partir de la mala intención.

Integridad de datos: Tiene que ver con la rotura de la información a partir del uso cotidiano de la información y sin intención de que haya rotura.

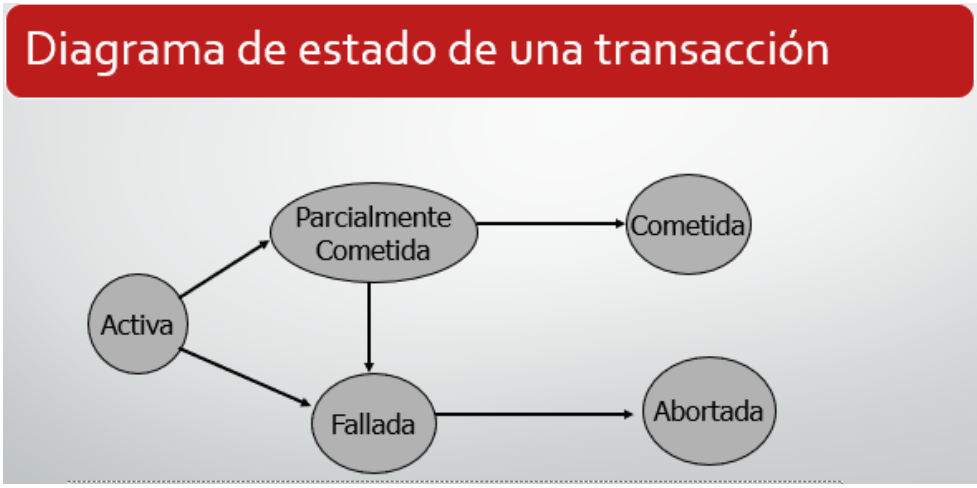
Transacciones

- Es una colección de operaciones que forman una única unidad lógica de trabajo, esa transacción tiene un efecto sobre la BD.
  La idea es que cuando se realice una transacción, el efecto de esa transacción se maneje como un TODO, osea todo el efecto de la transacción se produce, o ningún efecto de la transacción se produce, una o la otra, es decir, las transacciones no pueden ocurrir a medias.

  Si una transacción termina, no se puede llevar atrás, no se puede borrar lo que hizo una transacción.

- 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.
    Una transacción que falla,que no ha podido terminarse, tiene que dejar la base de datos en el mismo estado que estaba cuando empezó.
  - **Cometida:** Tras completarse con éxito.



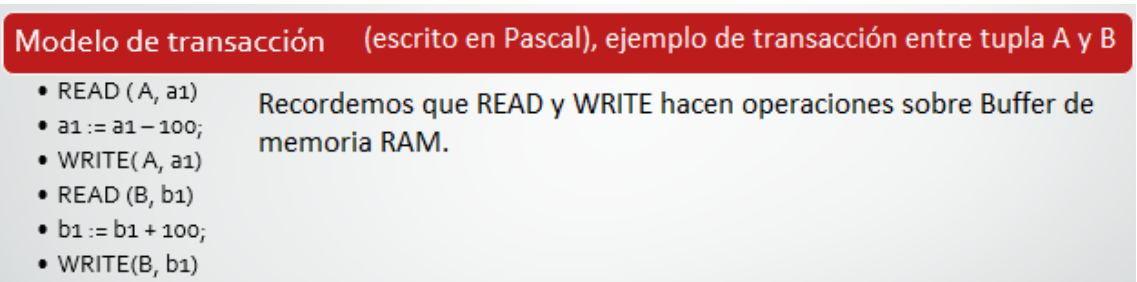
Si la transacción falla, debo asegurarme que todo lo que hizo la transacción se anule porque la transacción no pudo terminar, y en ese momento la transacción pasa al estado de abortada.

Una transacción va a llegar al estado de parcialmente cometida cuando finalizo la ejecución de su ultima instrucción.

¿Por qué puede fallar en estado de parcialmente cometida?

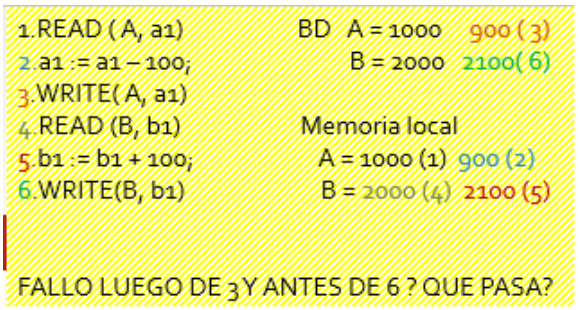
- Puede ocurrir por una escritura sobre la BD (Update).
- ¿Por qué?, simple, las escrituras se hacen primero sobre Buffer, recordemos que es memoria RAM (volátil).
- Puede ocurrir que se pierda dicha información, por lo que se puede llevar a un error que implique el paso al estado de transacción fallida.

El estado de transacción cometida: Asegura transacción terminada exitosamente.



Las transacciones pueden generar problemas, 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 **hay una transacción activa ocurriendo en un mismo 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.



Quedarian 900 pesos en A y 2000 en B.
Es decir, se pierden 100 pesos.

Que hacer luego de un fallo?

Re-ejecutar la transacción fallida => NO SIRVE
Dejar el estado de la BD como esta => NO SIRVE.

- **¿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.

Solucion Registro Histórico:

- Bitácora (registro cronológico) o por su nombre en inglés “Log”.
  Garantiza la atomicidad de una transacción.
  - 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.

Cualquier transacción que en bitácora tenga START y COMMIT, luego de un fallo, la DEBO REEJECUTAR, porque aseguro que la bd quede con valores correctos. En cualquier caso, cualquier transacción que tenga un START y un COMMIT la DEBERIAMOS a reejecutar.

Si el fallo se produce antes del COMMIT, la transacción no llego a terminar, y por ende la base de datos no sufrió ninguna modificación, no necesitamos reejecutar.

- **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, porque 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).

Conclusiones:

Modificacion diferida: Solo requiere una operación ,REDO.

Modificacion inmediata: Es mas compleja, pero tiene mejor distribuida la carga de trabajo. Requiere dos modificaciones, REDO y UNDO.

- 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 buffers.

Para poner un checkpoint, debo estar seguro de que el buffer se bajo a disco.

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:

Tecnica sencilla y efectiva, mas en entornos monousuarios.

- **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

Clase 8_2: Transacciones

Entornos concurrentes

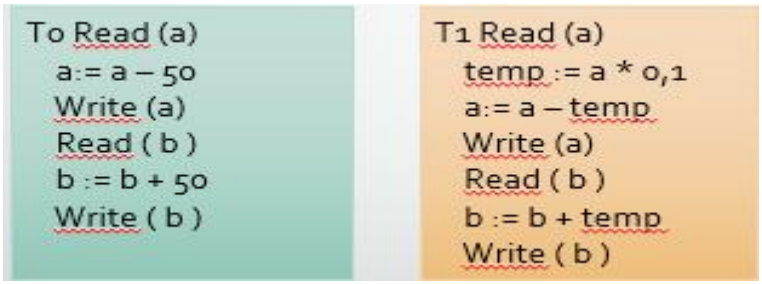
Entorno centralizado:

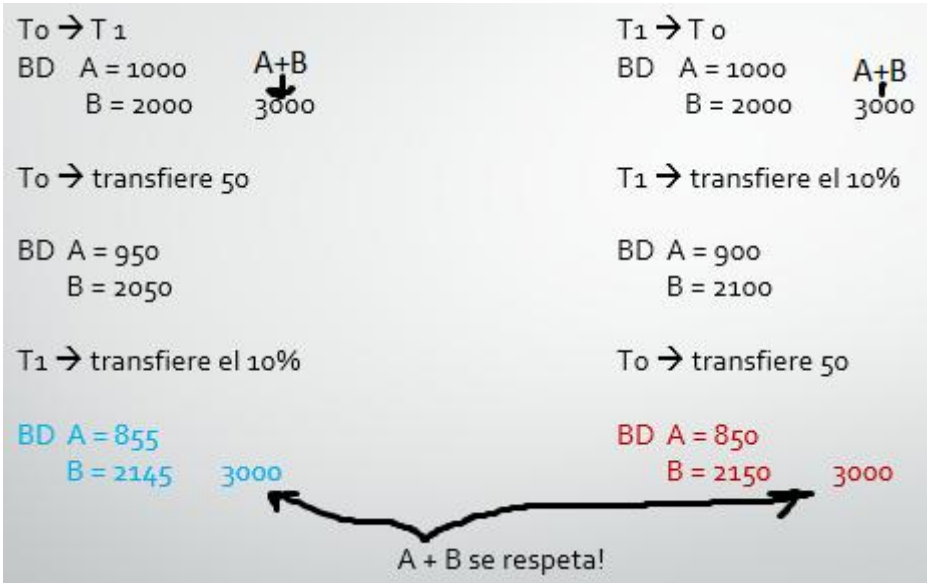
- Varias transacciones ejecutándose simultáneamente compartiendo recursos (el recurso es la BD).
- Deben evitarse los mismos problemas de consistencia de datos.
- Transacciones correctas (ocurriendo al mismo tiempo) en ambientes concurrentes pueden llevar a fallos y destruir la consistencia de los datos.

- Dos transacciones en un entorno monousuario hubieran dejado la BD consistente.

- En un entorno concurrente, ejecutándose a la vez, SIN FALLOS, pueden dejar la base de datos inconsistente.
- Si estoy en un entorno libre de fallos, la ejecución serie de dos transacciones, garantiza inconsistencia.
- La ejecución serie de transacciones, asegura la integridad de los datos.

Seriabilidad

- El concepto de seriabilidad es aquel que garantiza la consistencia de la BD después de que dos transacciones se ejecuten sobre los mismos datos.
- 

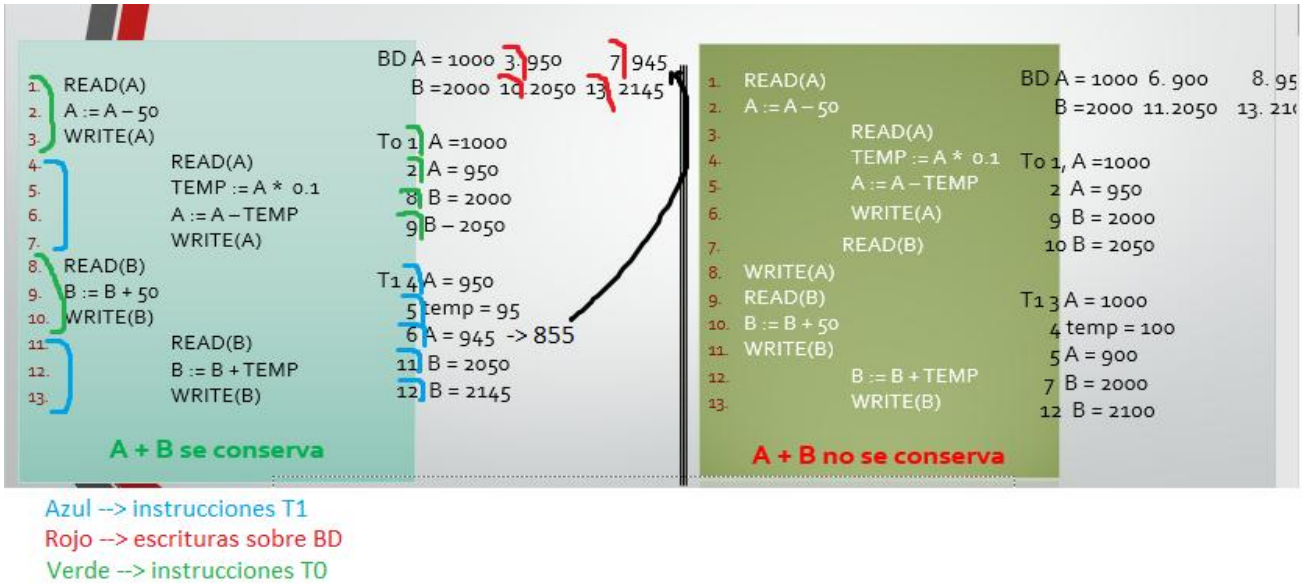
- Al resolver primero T0 o T1, o T1 y T0 se respeta que A+B es igual para ambos resultados.
- Ahora, T0 T1 <> T1 T0.
- ¿Qué es A+B?, la suma de ambos saldos debe ser igual a X valor sin importar el orden en que se ejecuten las transacciones
- 

- A pesar de que A y B sean distintos en ambos ordenes de ejecución, no me interesa eso, me interesa ver que A+B es igual para ambos casos.
- Si estoy en una ejecución sin fallos, al aplicar una ejecución secuencial de dos transacciones, la consistencia se mantiene.
- T0 y T1, o, T1 y T0 son dos planificaciones diferentes. Ver abajo.

Planificación: secuencia de ejecución de transacciones

- La planificación involucra todas las instrucciones de las transacciones.
- Conservan el orden de ejecución de las mismas.
- Un conjunto de m transacciones generan m! planificaciones en serie.
  - M es la cantidad de transacciones que tengo.
- La ejecución concurrente NO necesita una planificación en serie.

- Fácil, la concurrencia no tiene nada que ver con ejecutar las transacciones en un orden (1 x 1, como si fuera monousuario).
- Ejecutar en serie las transacciones es una locura ineficiente.
- Ejecutando concurrentemente se logran respuestas más rápidas.
- Hay ejecuciones concurrentes que sirven y otras que no sirven (dejan inconsistencia)



- A-temp debería ser 855. Solo así asegurás la consistencia de A+B = 3000. Se lo fumó Bertone al error xd.

Conclusiones

- El programa debe conservar la consistencia.
- La inconsistencia temporal (cuando una transacción está realizando algo y viene otra transacción a perjudicar el trabajo propio, ejemplo: T0 y T1 en el ejemplo anterior “A+B no se conserva”, específicamente entre instrucciones 2 y 6) puede ser causa de inconsistencia en planificaciones en paralelo.
- Una planificación concurrente debe equivaler a una planificación en serie, si demostramos que una planificación concurrente equivale a una planificación serie entonces demostramos que el resultado final va a ser el correcto y la consistencia no se va a ver perjudicada (ya que las planificaciones en serie son aquellas que no fallan).
- Solo las instrucciones READ y WRITE son importantes, hay que considerarlas.
  - Las operaciones ajenas a esas dos instrucciones son operaciones locales y no nos interesan para nuestro análisis.

Conflicto en planificaciones serializables

- Inst1, Inst2, dos instrucciones de T1 y T2.
  - Solo son read o write las que nos importan ya que actúan sobre la BD.
  - Si operan sobre datos DISTINTOS, NO hay conflicto.

- Si operan sobre el mismo dato, Inst1 e Inst2 estarán en conflicto si actúan sobre el mismo dato y AL MENOS UNA instrucción es un WRITE.
  - Inst1 = READ(Q) = Inst2 → no importa el orden de ejecución.
  - Inst1 = READ(Q), Inst2 = WRITE(Q) → depende el orden de ejecución (Inst1 leerá valores distintos en base al orden), provoca conflicto.
  - Inst1 = WRITE(Q), Inst2 = READ(Q) → depende el orden de ejecución (Inst2 leerá valores distintos en base al orden), provoca conflicto.
  - Inst1 = WRITE(Q) = Inst2 → depende el estado final de la BD, queda la ultima escritura, provoca conflicto.
  - Cuando hay conflicto el orden de ejecución será importantisimo.

Definiciones

- Una planificación S se transforma en una S' mediante intercambio de instrucciones no conflictivas (operan sobre datos diferentes o entre dos READ), entonces S y S' son equivalentes en cuanto a conflictos.
- Esto significa que si:
  - S' es consistente, S también lo será.
  - S' es inconsistente, S también será inconsistente.
- Si S' es serializable en conflictos si existe una planificación S tal que son equivalentes en cuanto a conflictos y S es una planificación serie.
  - Si una planificación S' concurrente es válida, entonces puedo demostrar que esa planificación concurrente equivale a una planificación serie.

Control de Concurrencia

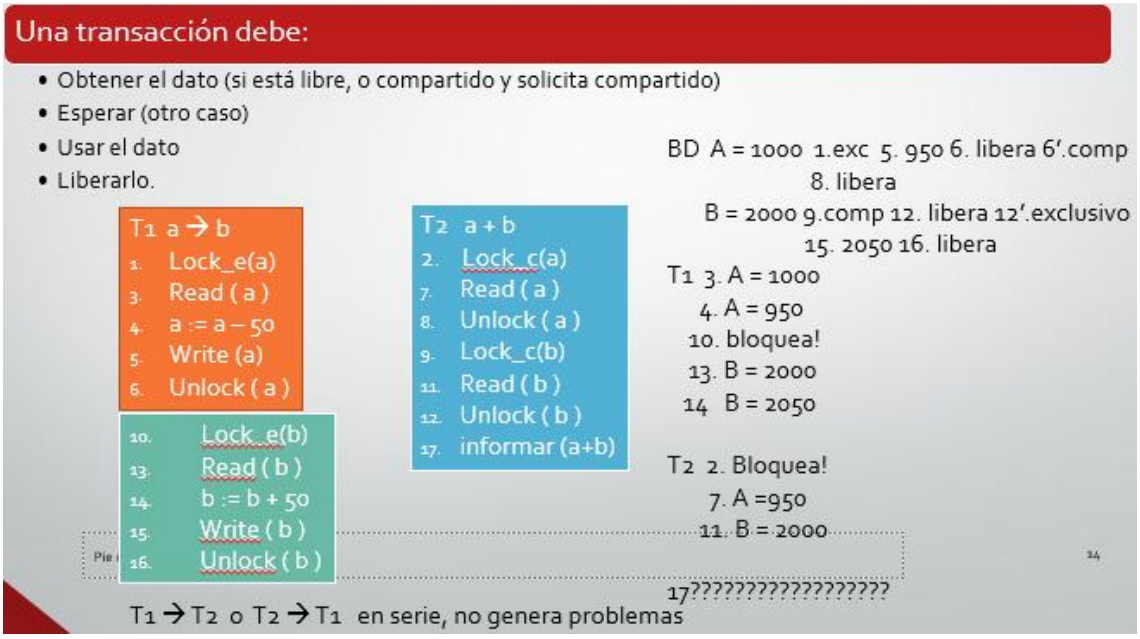
Métodos de control de la concurrencia:

- Mecanismo que controla que las planificaciones inválidas concurrentes NO se ejecuten.
- Aseguran que la ejecución simultánea de dos o más transacciones no conflictuen entre sí.
- Tenemos dos métodos.

Bloqueo: primer método de control

- Compartido Lock_c(dato) (solo para lectura): un dato bloqueado de forma compartida, varias transacciones pueden acceder al dato pero SOLO para leerlo.
- Exclusivo Lock_e(dato) (lectura/escritura): una transacción necesita escribir un dato, por lo que bloquea exclusivamente el dato, y el bloqueo exclusivo me

deja cambiar el dato y no permite que OTRA transacción lea el dato en pleno cambio (evita conflictos).

- Las transacciones piden lo que necesitan.
- Los bloqueos pueden ser compatibles y existir simultáneamente (compartidos).
- En una BD bloquear significa pedir el dato que necesito (bloquear), usarlo, y liberarlo (desbloquear).
- 

  - Las transacciones se bloquean y quedan en espera cuando piden bloquear un dato y éste ya está con un bloqueo (indiferente que sea compartido o exclusivo).
  - Por eso vemos que la instrucción 2 bloquea la transacción (T2) y la pone en espera, mismo con la instrucción 10 (T1).
  - Para el primer bloqueo que ocurre con T2, vemos que se ejecutan las instrucciones 3,4,5,6, donde la 6 libera el dato y ocurre una 6' que es básicamente el lock compartido que ocurre en la instrucción 2 (para seguir con la instrucción 7).
  - Para el segundo bloqueo que ocurre con T1, vemos que T2 hace un lock compartido de B, y en la instrucción 10 T1 intenta bloquear el dato pero es puesto en espera. Ocurren las instrucciones 11-12 y existe un 12' que aplica el bloqueo que quiso imponer la instrucción 10, para así seguir con la instrucción 13.
  - Informar (A+B) informa algo incorrecto, ya que informa 2950…
    - Se genera una inconsistencia de la información.
    - En resumen, el método del bloqueo aún puede generar inconsistencias.
    - Falta una vuelta de tuerca más.
- La solución es llevar los bloqueos de las transacciones al comienzo.

Deadlock

- Situación en la que una transacción espera un recurso de otra transacción y viceversa.
    - T1 pide el dato A, se lo dan.
    - T2 pide el dato B, se lo dan.
    - T1 pide el dato B, no se lo pueden dar xq T2 lo está acaparando.
    - T2 pide el dato A, no se lo pueden dar xq T1 lo está acaparando.
    - No se libera nada xq T1 espera de T2 y viceversa.
- Es el efecto secundario que ocurre de bloquear la información al principio, de usarla y liberarla.
- *Conclusiones*
    - Si los datos son liberados pronto → se evitan posibles deadlock.
    - Si los datos se mantienen bloqueados → se evitan inconsistencias.
    - Tengo que lidiar con deadlock o inconsistencia, cuál prefiero?
        - Se prefiere lidiar con el deadlock ya que el mismo no genera inconsistencia sobre la BD.
        - ¿Cómo soluciono un deadlock?
            - La solución es elegir a una víctima, elijo a una de las dos transacciones y la ABORTO.
            - La otra transacción puede empezar con sus instrucciones.

Protocolos de bloqueo

- Dos fases
    - Requiere que las transacciones hagan bloqueos en dos fases para evitar inconsistencias:
        - Fase de crecimiento: se obtienen datos.
        - Fase de decrecimiento: se liberan los datos.
        - Pido todo al principio, uso y libero al final.
    - Garantiza seriabilidad en conflictos, pero no evita situaciones de deadlock.
    - Como se consideran las operaciones
        - Fase de crecimiento: se piden bloqueos en orden: compartido, exclusivo.
        - Fase de decrecimiento: se liberan datos o se pasa de exclusivo a compartido.

Protocolo basado en hora de entrada

- El orden de ejecución es determinado por adelantado, no depende de quien llega primero.
- Cada transacción recibe una HDE (Hora de entrada)
  - Hora del servidor
  - Un contador
- Sí HDE(Ti) < HDE(Tj), Ti es anterior a Tj
- Cada dato
  - Hora en que se ejecutó el último WRITE.
  - Hora en que se ejecutó el último READ.
  - Las operaciones READ y WRITE que pueden entrar en conflicto se ejecutan y eventualmente fallan por HDE.
    - ¿Qué significa?
- Algoritmo de ejecución
  - Ti solicita READ(Q)
    - HDE(Ti) < HW(Q): se rechaza ya que Ti solicita un dato que fue escrito por una transacción posterior, estaría queriendo leer un dato que ya fue reemplazado.
    - HDE(Ti) >= HW(Q): ejecuta por que mi transacción quiere leer un dato que fue modificado antes que mi hora de entrada. Se establece HR(Q) = Max{HDE(Ti), HR(Ti)} (saca el máximo entre esos dos datos).
  - Ti solicita WRITE(Q)
    - HDE(Ti) < HR(Q): rechaza ya que Q es utilizado por otra transacción anteriormente y ésta supuso que Q no cambiaba.
    - HDE(Ti) < HW(q): rechaza por que Ti intenta escribir un valor viejo y obsoleto.
    - HDE(Ti) > [HW(Q) y HR(Q)]: ejecuta y HW(Q) se establece con HDE(Ti)
  - Sí Ti falla, y se rechaza entonces puede recomenzar con una nueva hora de entrada.

Casos de concurrencia. Granularidad

- A registros caso más normal de bloqueo.
- Otros casos de bloqueo
  - BD completa (quien puede querer bloquear la BD completa?)
  - Áreas (y un conjunto de áreas?)
  - Tablas (y un conjunto de tablas?)
  - Algunos DBMS permiten un bloqueo de atributos:
    - Se llama nivel de granularidad.
    - El nivel de granularidad de bloqueo depende de cada DBMS.

Otras operaciones conflictivas

- Delete(Q) requiere un uso completo del registro, ya que borra una tupla de la tabla.
- Insert(Q) el dato permanece bloqueado hasta que la operación finalice.
    - ¿Por qué bloquearía un dato que todavía nadie puede leer?
        - Puede que necesite insertar en más de una tabla a la vez.
            - Ejemplo: insertar una factura en una tabla facturas.

Registro histórico en entornos concurrentes

Consideraciones del protocolo basado en bitácora

- Existe un único buffer de datos compartidos y uno para la bitácora.
- Cada transacción tiene un área donde lleva sus datos.
- El retroceso de una transacción puede llevar al retroceso de otras transacciones.

Retroceso en cascada

- Falla una transacción → puede llevar a abortar otras.
    - Se abortan otras cuando hay datos en común usados en varias transacciones.
    - Se ejecuta T1, se ejecuta T2 y se ejecuta T3.
    - T1 falla, como T2 usa datos de T1 y T3 también. Entonces se abortan las 3.
- Puede llevar a deshacer una gran cantidad de trabajo.
    - Me la tengo que bancar.
    - Puede ocurrir pero no es muy habitual.

Durabilidad

- Puede ocurrir que Ti falle, y que Tj deba retrocederse, pero que Tj ya terminó.
- Transacción que alcance el estado de cometido, NO PUEDE RETROCEDER.
- Como debería actuar para cumplir la condición de durabilidad en Tj?.
    - Protocolo de bloqueo en dos fases: los bloqueos exclusivos deben conservarse hasta que Ti termine (cuando Ti alcance el estado de cometido libera los bloqueos exclusivos).
    - HDE, agrega un bit, para escribir el dato, además de lo analizado, revisar el bit: sí está en 0 procedemos, si está en 1 la transacción anterior no terminó, debo esperar.
    - Esto implica que las transacciones que conflictúan se hagan en SERIE, de forma necesaria.
        - No ocurre habitualmente.

- También evito el retroceso en cascada.

Bitácora

- Es similar a los sistemas monousuario.
- Como proceder con checkpoints
  - Colocarlo cuando ninguna transacción esté activa. Puede que no exista el momento.
  - Checkpoint <L> L es una lista de transacciones activas al momento del checkpoint.
  - Básicamente: de acá para atrás está todo bien, pero revisá la lista de transacciones activas.
- Ante un fallo
  - UNDO y REDO según el caso.
  - Debemos buscar antes del checkpoint solo aquellas transacciones que estén en la lista.