BDII 3 – Gestión de Transacciones

Concepto de transacción

- Una TRANSACCIÓN es un **conjunto de operaciones** (de acceso y actualización de datos) que forman una unidad lógica de **trabajo** y por tanto deben ser ejecutadas como una única unidad de **ejecución**.
  - Las operaciones que forman una transacción se delimitan entre una secuencia ***begin transaction*** y una ***end transaction.***
- Las transacciones deben cumplir las **propiedades ACID** (*Atomicity, Consistency, Isolation* y *Durability*):
  - ATOMICIDAD → Asegura que, o bien todas las operaciones de la transacción se realizan adecuadamente en la bd o bien no lo hace ninguna de ellas.
    - Si una transacción comienza, pero algo falla (o en la transacción o en algo externo a ella), se deben **deshacer** todos los cambios que la transacción haya podido hacer en la bd.
    - ↳ Es responsabilidad del **sgbd**.
  - CONSISTENCIA → Asegura que, tras ejecutar de forma aislada una transacción, si la bd era consistente antes de ella, lo seguirá siendo.
    - ↳ Es responsabilidad del **programador**.
  - AISLAMIENTO → Asegura que, a pesar de que se ejecuten varias transacciones concurrentemente, una dará la impresión de haberse ejecutado antes.
    - Esta característica afecta al rendimiento, pero es necesaria en ciertas ocasiones.
    - ↳ Es responsabilidad del **sgbd**.
  - DURABILIDAD → Asegura que, tras la finalización con éxito de una transacción, los cambios realizados en la bd persistirán incluso aunque se produzca un fallo en el sistema.
    - ↳ Es responsabilidad del **sgbd**.

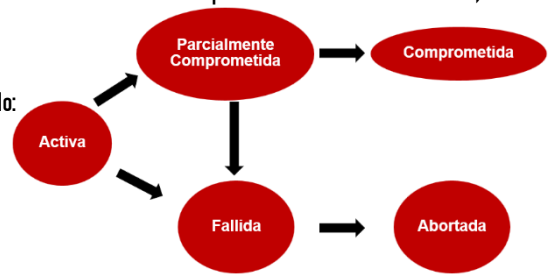
Estructura de almacenamiento

Los medios de almacenamiento se pueden clasificar en 3 tipos:

- Almacenamiento VOLÁTIL → la información almacenada **no sobrevive** a las caídas del sistema, pero acceder a ella es muy **rápido** debido a su velocidad de acceso y la capacidad de acceder de manera directa a cualquier tipo de datos (acceso aleatorio).
  - ↳ Memoria principal, memoria caché.
- Almacenamiento NO VOLÁTIL → la información almacenada **sí sobrevive** a las caídas del sistema, pero acceder a ella es mucho más **lento** debido a que no tiene acceso aleatorio.
  - Es susceptible a **fallos**, lo que puede generar pérdidas de información.
  - ↳ Discos magnéticos, almacenamiento flash, medios ópticos, cintas magnéticas.
- Almacenamiento ESTABLE → la información almacenada **nunca se pierde**, pero su implementación es teóricamente imposible. Se aproxima reduciendo la probabilidad de pérdida de datos mediante la **replicación** de la información en varios medios de almacenamiento **no volátil** con **modos de fallo independientes**.
  - Para que una transacción sea **duradera**, se deben escribir sus cambios en memoria estable.
  - Para que una transacción sea **atómica**, se debe escribir un registro en memoria estable antes de que se escriba en disco cualquier cambio a la bd.
  - ▸ Entonces, el grado en el que el sgbd asegura la durabilidad y atomicidad depende de **cuán estable sea el almacenamiento estable**.

Atomicidad y durabilidad

- Si una transacción **no** termina **correctamente** → los cambios que realizó en la bd se deben **deshacer**, para lo que se mantiene un **registro en memoria estable** donde se lleva cuenta de todas las modificaciones que realiza una transacción antes de escribirlas al disco.
  - En él se almacena, entre otras cosas, el **valor anterior** de los datos modificados, permitiendo retroceder la bd al estado que tenía antes de comenzar.
- Si una transacción termina **correctamente** → se **compromete**, por lo que los cambios que esta haya realizado en la bd no se pueden deshacer abortándola, sólo se pueden deshacer ejecutando una **transacción compensadora**.
  - ↳ La creación de transacciones compensadoras es trabajo del **programador** y a veces no existen.



Existen diversos **estados** que nos indican en qué situación se encuentra una transacción en un momento dado:

- **ACTIVA** (estado inicial) → la transacción permanece en este estado durante toda su ejecución.
- **PARCIALMENTE COMPROMETIDA** → se alcanza cuando finaliza su última operación, pero como aún no ha escrito en el disco, todavía puede fallar.
- **FALLIDA** → se alcanza cuando se produce un error que no permite continuar la ejecución normal.
- **ABORTADA** → se alcanza tras pasar por el estado de **fallida** y **retroceder** la transacción restableciendo el estado de la bd antes de comenzar su ejecución. Una vez una transacción alcanza este estado, el sgbd tiene 2 opciones:
  - **Reiniciar** la transacción → sólo cuando se abortó por un error de hardware o software no provocado por la lógica interna de la transacción.
    - ↳ Una transacción reiniciada se considera una nueva transacción.
  - **Cancelar** la transacción:
    - Cuando hay un error **interno lógico** que sólo se puede corregir reescribiendo el programa de aplicación.
    - Debido a una **entrada incorrecta**.
    - Debido a que no se encontraran los **datos** deseados en la bd.
- **COMPROMETIDA** → se alcanza una vez la transacción se completa correctamente, incluyendo la escritura en el disco.

- Se dice que una transacción está **terminada** cuando alcanza el estado abortada o comprometida.

- Las ESCRITURAS EXTERNAS OBSERVABLES son aquellas que producen una salida al usuario (p. e. por pantalla o por correo) y que entonces pueden haber sido vistas por el usuario fuera del sgbd, por lo que no pueden ser borradas.
  - Por ello, muchos sistemas las almacenan en memoria no volátil y esperan a que la transacción esté **comprometida** para mostrarlas
    - ↳ Si el sistema falla entre que se comprometa la transacción y se muestren las escrituras externas, el sgbd podrá mostrarlas leyéndolas de la memoria no volátil cuando se reinicie el sistema.

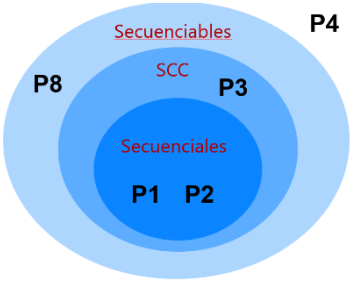
AISLAMIENTO

- Existen varias razones para **permitir la concurrencia** en la ejecución de transacciones:
  - ✓ **Mejora el rendimiento y uso de recursos** → permite explotar el paralelismo entre la CPU y del sistema de E/S, incrementando el rendimiento (es decir, el número de transacciones que puede realizar en un tiempo dado) y la utilización de la CPU y del disco, pues pasarán menos tiempo desocupados
  - ✓ **Reducción del tiempo de espera** → si sólo se permite ejecución secuencial, las transacciones cortas tendrían que esperar a que una larga anterior acabase, lo cual puede llevar a un **retardo impredecible** en la ejecución. Al operar en paralelo, se reducen los estos retardos y el **tiempo medio de espera**
- × Sin embargo, la ejecución concurrente implica que, aunque cada transacción individual sea correcta, puede que se incumpla la propiedad de aislamiento y se lleguen a **estados inconsistentes** en la bd.
- Para evitar esto, usa el sistema de CONTROL DE CONCURRENCIA para controlar la interacción entre transacciones concurrentes.
- Este sistema crea PLANIFICACIONES, que representan el orden cronológico en el que se ejecutan las instrucciones de las distintas transacciones.
  - ↳ Siempre se mantendrá el orden entre las operaciones de cada transacción.
  - Planificaciones SECUENCIALES → las instrucciones pertenecientes a una única transacción aparecen juntas.
    - ↳ Para un conjunto de N transacciones hay $N!$ posibles planificaciones secuenciales.
    - × No aprovechan la ejecución concurrente.
  - Planificaciones SECUENCIABLES → se obtiene el mismo resultado que con una planificación secuencial, pero puede realizarse una ejecución concurrente.

SECUENCIALIDAD

Consideraremos que las únicas operaciones que se pueden realizar sobre un elemento de datos Q son $leer(Q)$ y $escribir(Q)$.

- Consideremos una planificación S en la que hay dos instrucciones consecutivas I y J pertenecientes a las transacciones T_i y T_j, respectivamente.
  - Si I y J se refieren a distintos elementos de datos → el **orden** de I y J **no importa**, se pueden intercambiar sin afectar al resultado de cualquier instrucción de la planificación.
  - Si I y J se refieren al mismo elemento de datos, Q:
    - Si $I = leer(Q)$ y $J = leer(Q)$ → el **orden** de I y J **no importa**.
    - Si $I = leer(Q)$ y $J = escribir(Q)$ o viceversa → el **orden** de I y J **sí importa** para T_i
    - Si $I = escribir(Q)$ y $J = escribir(Q)$ → el **orden** de I y J **afecta** a la siguiente instrucción $leer(Q)$.
- Se dice que I y J están EN CONFLICTO si son operaciones de **diferentes transacciones** sobre el **mismo elemento** de datos y al menos una de ellas es $\boldsymbol{escribir}$.
  - Si I y J no están en conflicto, se pueden cambiar de orden para obtener una nueva planificación S' que será equivalente a S, pues todas las instrucciones tienen el mismo orden excepto I y J, cuyo orden no importa.
- Se dice que S y S' son EQUIVALENTES EN CUANTO A CONFLICTOS si S se puede transformar en S' mediante intercambios de instrucciones no conflictivas.
- Se dice que una planificación S es SECUENCIABLE EN CUANTO A CONFLICTOS si es equivalente en cuanto a conflictos a una planificación secuencial.
- El GRAFO DE PRECEDENCIA de una planificación es un grafo dirigido cuyos nodos representan las transacciones y cuyos arcos son todas las secuencias $T_i \rightarrow T_j$ que representan que, teniendo dos instrucciones conflictivas, T_i ejecuta la suya antes que T_j.
  - Si el grafo de precedencia de una planificación tiene algún **ciclo**, entonces la planificación **no es secuenciable en cuanto a conflictos** (pues alguna instrucción de una transacción tiene que ir antes que una instrucción de otra transacción y viceversa). Si **no hay ciclos, es secuenciable**.
- Es posible encontrar planificaciones que produzcan el **mismo resultado** y que **no sean equivalentes en cuanto a conflictos**.
  - Por tanto, pueden existir **planificaciones secuenciables** (pues producen el mismo resultado que el que se obtiene al ejecutar las transacciones secuencialmente) **pero que no son secuenciables en cuanto a conflictos** (pues tienen ciclos en su grafo de precedencia).



AISLAMIENTO Y ATOMICIDAD

Existen dos tipos de planificaciones que son **aceptables** desde el punto de vista de **recuperación del fallo de una transacción**:

- PLANIFICACIÓN RECUPERABLE → aquella en la que todo par de transacciones A y B donde B lee elementos que ha escrito antes A, la operación $commit$ de A aparece antes que la de B.
  - ↳ Así, se asegura que los elementos leídos por B son válidos y si A se tiene que abortar, también lo hará B.
- RETROCESO EN CASCADA → fenómeno en el cual un fallo en una única transacción provoca una serie de retrocesos en varias transacciones.
  - × No son deseables pues provocas un **aumento** significativo del **trabajo** necesario para **deshacer transacciones**.
- PLANIFICACIÓN SIN CASCADA → aquella en la que todo par de transacciones A y B donde B lee elementos que ha escrito antes A, la operación $commit$ de A aparece antes que la $leer$ de B.

Naturalmente, todas las planificaciones sin cascadas son también recuperables.

NIVELES DE AISLAMIENTO DE TRANSACCIONES

- La **secuencialidad** asegura que la ejecución concurrente **mantiene** el aislamiento y por tanto la **consistencia**, pero los protocolos necesarios para alcanzarla pueden provocar un grado de concurrencia muy bajo, **disminuyendo** el **rendimiento**.
- Por tanto, el diseñador de la aplicación puede decidir usar un **nivel de aislamiento** más débil para mejorar el rendimiento del sistema:
  - SECUENCIABLE ⟶ normalmente asegura la ejecución secuencial.
  - LECTURA REPETIBLE ⟶ solo permite leer datos escritos por transacciones comprometidas e impide que entre dos lecturas de un dato en una transacción otra lo actualice.
  - LECTURA COMPROMETIDA ⟶ solo permite leer datos escritos por transacciones comprometidas, pero entre dos lecturas de un dato en una transacción este puede ser actualizado por otra.
  - LECTURA NO COMPROMETIDA ⟶ permite leer datos no comprometidos.
  - Para cambiar el nivel de aislamiento de una transacción en **SQL** se usa la sentencia: $\boldsymbol{set\ transaction\ isolation}\ level.$
    - ↳ Debe ser la primera sentencia de la transacción.
  - Para cambiar el nivel de aislamiento de una transacción en **JDBC** se usa la función: $Connection.\boldsymbol{setTransactionIsolationLevel}(nivel)$

IMPLEMENTACIÓN DE NIVELES DE AISLAMIENTO

- Las DIRECTIVAS DE CONTROL DE CONCURRENCIA tienen los siguientes objetivos:
  - Proporcionar un **elevado grado de concurrencia**
  - Asegurar planificaciones **secuenciables en cuanto a conflictos**
  - Asegurar planificaciones **sin cascada**.

BLOQUEO

- Consiste en bloquear los elementos de datos cuando una transacción acceda a ellos.
- El bloqueo se debe mantener el **tiempo necesario** para asegurar la **secuencialidad**, pero **sin excederse** para no **dañar demasiado el rendimiento**.
- Pueden dar problemas en las transacciones en las que se accede a elementos en función del resultado de una cláusula $\boldsymbol{where}$.

Se añaden dos técnicas al concepto para asegurarse de que cumple los dos primeros objetivos:

- **Bloqueo en dos fases** ⟶ en la primera fase se adquieren bloqueos, pero no se libera ninguno, y en la segunda se liberan, pero no se adquieren.
- **Modos de bloqueo**:
  - Bloqueos compartidos ⟶ se usan en datos que la transacción lee, de manera que varias transacciones pueden mantener bloqueos compartidos sobre un mismo elemento de datos.
  - Bloqueos exclusivos ⟶ se usan en datos que la transacción escribe, de manera que sólo una transacción puede mantener un bloqueo exclusivo sobre cada elemento de datos.

MARCAS DE TIEMPO

- Consiste en asignarle a cada transacción una marca de tiempo (normalmente cuando comienza).
- El sgbd mantiene dos marcas de tiempo para cada elemento de datos { **marca de lectura** ⟶ marca de la transacción que lo leyó en último lugar. **marca de escritura** ⟶ marca de la transacción que escribió el valor actual.
- Las transacciones acceden a los datos en el **orden establecido por las marcas**, de manera que si una intenta acceder fuera de orden se aborta y reinicia.

VERSIONES MÚLTIPLES

- Consiste en almacenar varias versiones de cada dato.
- Así, una transacción puede leer un **valor antiguo** en lugar de otro escrito por una transacción comprometida o que debería haberse ejecutado después.

AISLAMIENTO DE INSTANTÁNEAS

- Consiste en asignar a cada transacción su propia versión o "instantánea" de la bd cuando comienza.
- Así, cada transacción trabajará con los datos de su versión privada de la bd y por tanto estará aislada de las actualizaciones de otras transacciones. Las **actualizaciones** realizadas se reflejarán en la bd "real" cuando se **comprometa** la transacción.
  - ↳ Una transacción no podrá comprometerse si alguna otra ha modificado alguno de los datos que estaba intentando actualizar, en su lugar será abortada.

TRANSACCIONES COMO SECUENCIAS DE SQL

- El FENÓMENO FANTASMA sucede cuando una planificación incluye **inserciones** o **eliminaciones** de datos sobre una relación sobre la cual también se realiza una **consulta**, de manera que si se realiza primero la inserción/eliminación, el resultado de la consulta será distinto.
- Entonces, para el control de concurrencia no es suficiente considerar las tuplas a las que accede una transacción, también hay que considerar la **información que se usa para encontrar esas tuplas**.
- Diremos que una inserción, borrado o actualización es CONFLICTIVA CON UN PREDICADO sobre una relación si pueden afectar al conjunto de tuplas seleccionadas por dicho predicado.
  - ↳ El bloqueo que se basa en esa idea se denomina **bloqueo de predicado**, pero es muy costoso y en la práctica no se usa.
    - ↳ Una solución alternativa son los protocolos de **bloqueo de índices**.