

## Capítulo 13. TRANSACCIONES

### 1. CONCEPTO DE TRANSACCIÓN

**Transacción** → unidad de la ejecución de un programa que accede y posiblemente actualiza varios elementos de datos.

Delimitado por declaraciones inicio transacción y fin transacción (todas las operaciones que se ejecutan entre ellas).

**Asegurar integridad BD** → **Propiedades** que han de cumplir las transacciones (ACID):

1.- **Atomicidad** → o todas las operaciones o ninguna.

Responsabilidad del SGBD → Componente de gestión de transacciones.

2.- **Consistencia** → ejecución aislada(sin concurrencia)conserva la consistencia BD

Responsabilidad del programador de la transacción.

3.- **Aislamiento** → las T concurrentes no se afectan.

El resultado de ejecutar concurrentemente las transacciones es equivalente al que obtendríamos al ejecutarlas secuencialmente.

Responsabilidad SGBD→ Componente de control de concurrencia.

4.- **Durabilidad** → si finaliza con éxito cambios permanecen incluso si fallo.

**Se garantiza si** se asegura que:

1.- Las modificaciones realizadas por la transacción se guardan en disco antes de que finalice la transacción.

2.- La información de las modificaciones realizadas por la transacción guardada en disco es suficiente para permitir a la base de datos reconstruir dichas modificaciones cuando el sistema se reinicie después de un fallo.

Responsabilidad SGBD → Componente de gestión de recuperaciones.

**Estado inconsistente** → BD deja reflejar el estado real del mundo que modela.

La BD puede atravesar estados inconsistentes durante la ejecución de una transacción pero éstos no se verán nunca desde fuera de la transacción si se asegura la atomicidad.

## 2. ESTADOS DE UNA TRANSACCIÓN

Transacción **abortada** → que no termina con éxito.

Atomicidad → no efecto sobre BD; cualquier cambio debe deshacerse.

La transacción se ha **retrocedido** → se han deshecho los cambios de T abortada.

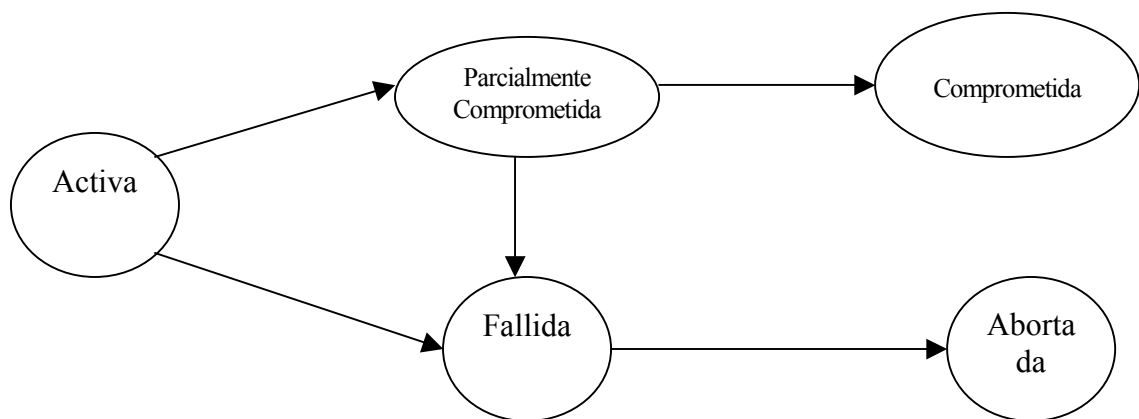
Transacción **comprometida** → T que termina con éxito.

Si ha hecho modificaciones lleva a la BD a un nuevo estado consistente que permanece incluso si se produce un fallo en el sistema.

No se pueden deshacer sus efectos abortándola.

Deshacer cambios → con una T **compensadora** :

- No siempre se puede crear
- No la gestiona el SBD



Estados:

**Activa** → estado inicial, cuando se está ejecutando.

**Parcialmente comprometida** → tras ejecutarse la última instrucción.

**Fallida** → tras descubrir que no puede continuar la ejecución normal.

**Abortada** → tras retroceder la T y reestablecer la BD al estado anterior a iniciar T.

**Comprometida** → tras completarse con éxito.

Una transacción ha **terminado** si se ha comprometido o se ha abortado.

Una **T fallida** se tiene que retroceder y luego pasa a abortada.

Hay **2 opciones** para **retroceder**:

- 1.- **Reiniciar** la T → si el error no lo ha provocado la lógica de la transacción. Una transacción reiniciada se considera una nueva transacción.
- 2.- **Cancelar** la T → si hay un error interno lógico en el programa de la transacción o se ha debido a una entrada incorrecta o no se han encontrado los datos deseados.

**¡OJO!** Con las **escrituras externas observables** como terminal o impresora, no pueden borrarse por lo que muchos sistemas las permiten sólo cuando la transacción se ha comprometido. En las T. de larga duración la muestra de datos al usuario es deseable pero compromete la atomicidad de la transacción.

### 3. IMPLEMENTACIÓN DE LA ATOMICIDAD Y LA DURABILIDAD

El soporte para la atomicidad y durabilidad lo implementa el componente de gestión de recuperaciones.

**SISTEMA DE LAS COPIAS SOMBRA** (simple pero ineficiente):

- Sólo una transacción está activa en cada momento, no hay concurrencia.
- La BD es un fichero en disco.
- Un puntero apunta a la copia actual de la BD.
- Las T copian la BD entera y realizan los cambios en la copia.
- La BD original **copia en la sombra** queda inalterada.
- Si la T se completa se actualiza el puntero y la copia en la sombra se borra.
- Una T se compromete en el momento en que el puntero actualizado se escribe a disco.
- Asegura la atomicidad y durabilidad.
- La implementación depende de que escribir el puntero sea una operación atómica, esto lo garantiza el SO.
- Ineficiente y no permite la concurrencia.

### 4. EJECUCIONES CONCURRENTES

Razones para permitir la concurrencia:

- Aumento de la **productividad del sistema** (nº transacciones en un tiempo dado) explotando el paralelismo E/S y UCP.
- Reducción del **tiempo medio de respuesta** (tiempo medio desde que una transacción comienza hasta que se completa) cuando hay una mezcla de transacciones unas cortas y otras largas.

Mantener consistencia → **esquemas de control de concurrencia**.

**Planificación** → representa el orden cronológico en el cual se ejecutan las instrucciones.

**Planificación secuencial** → secuencia de instrucciones de varias transacciones, en la cual las instrucciones pertenecientes a una única transacción están juntas en dicha planificación.

**¡OJO!** :

- Cuando se ejecutan concurrentemente varias transacciones, la planificación no tiene por qué ser secuencial.
- No todas las ejecuciones concurrentes producen un estado correcto. Algunas planificaciones posibles pueden llevar a un estado inconsistente.

**Componente de control de concurrencia** → componente del SGBD encargado de asegurar que cualquier planificación que se ejecute lleva a la BD a un estado consistente

**Se puede asegurar la consistencia si** se está seguro de que cualquier **planificación** que se ejecute tiene el mismo efecto que otra que se hubiese ejecutado sin concurrencia; es decir, la planificación debe ser en cierto modo **equivalente a** una planificación **secuencial**.

## 5. SECUENCIALIDAD

Las únicas **operaciones significativas** de la transacción son **leer** y **escribir**.

Dos formas diferentes de equivalencia con una planificación secuencial:

### 1.- SECUENCIALIDAD EN CUANTO A CONFLICTOS

Dos instrucciones consecutivas tienen **conflicto** (importa el orden de ejecución) si son de diferentes transacciones, actúan sobre el mismo elemento de datos y al menos una es una operación escribir.

Dos **planificaciones** P y P' son **equivalentes en cuanto a conflictos** si P se puede transformar en P' por medio de una serie de intercambios de instrucciones no conflictivas.

Una **planificación** es **secuenciable en cuanto a conflictos** si es equivalente en cuanto a conflictos a una planificación secuencial.

**¡OJO!** Es posible encontrar planificaciones que produzcan el mismo resultado (equivalentes) y que no son equivalentes en cuanto a conflictos → hay definiciones de equivalencia de planificaciones menos rigurosas.

### 2.- SECUENCIALIDAD EN CUANTO A VISTAS

Menos rigurosa que la equivalencia en cuanto a conflictos.

Dos **planificaciones** P y P' son **equivalentes en cuanto a vistas** si cumplen:

- Para todo elemento de datos Q, si la transacción T lee el valor inicial de Q de la planificación P, entonces T debe leer también el valor inicial de Q de la planificación P'.
- Para todo elemento de datos Q, si la transacción T ejecuta **leer**(Q) en la planificación P y el valor lo ha producido la transacción T' (si existe), entonces en la planificación P' la transacción T debe leer también el valor de Q que haya producido la transacción T'.
- Para todo elemento de datos Q, la transacción (si existe) que realice la última operación **escribir**(Q) en la planificación P debe realizar la última operación **escribir**(Q) en la planificación P'.

En otras palabras, que cada transacción en ambas planificaciones lea los mismos valores, realice los mismos cálculos y el resultado final sea el mismo.

Una **planificación** es **secuenciable en cuanto a vistas** si es equivalente en cuanto a vistas a una planificación secuencial.

**Escritura a ciegas** → cuando una transacción realiza una operación **escribir**(Q) sin haber realizado ninguna operación **leer**(Q)

**¡OJO!** Toda planificación secuenciable en cuanto a conflictos es secuenciable en cuanto a vistas, pero existen planificaciones secuenciables en cuanto a vistas que no son secuenciables en cuanto a conflictos (aquellas que contienen escrituras a ciegas).

## 6. RECUPERABILIDAD

Si la transacción T falla, hay que deshacer la transacción para asegurar la atomicidad. En un sistema concurrente, toda transacción que dependa de T (lea datos escritos por T) se aborta también.

Por estas razones es necesario poner ligaduras al tipo de planificaciones permitidas por el sistema:

### 1.- PLANIFICACIONES RECUPERABLES

¡OJO! Recordar que una transacción comprometida no puede abortarse.

**Planificación recuperable** → para todo par de transacciones T y T' tales que T' lee elementos de datos que ha escrito previamente T, la operación comprometer de T aparece antes que la de T'.

No deben permitirse planificaciones no recuperables.

### 2.- PLANIFICACIONES SIN CASCADA

**Retroceso en cascada** → fenómeno en el cual el fallo de una transacción provoca que haya que retroceder varias transacciones (las que dependen de la que falló).

No es deseable por el aumento del trabajo necesario en los retrocesos.

**Planificación sin cascada** → para todo par de transacciones T y T' tales que T' lee un elemento de datos que ha escrito previamente T, la operación comprometer de T aparece antes que la operación de lectura de T'.

Toda planificación sin cascada es también recuperable.

## 7. IMPLEMENTACIÓN DEL AISLAMIENTO

**Esquemas de control de concurrencia** que aseguren que, incluso si se ejecutan concurrentemente muchas transacciones, sólo se generen planificaciones aceptables, sin tener en cuenta la forma en que el SO comparte en los recursos entre las transacciones.

**Objetivo** de los esquemas de control de concurrencia:

- Proporcionar un elevado grado de concurrencia.
- Asegurar que las planificaciones son secuenciables en cuanto a conflictos o en cuanto a vistas.
- Asegurar que las planificaciones que se generan son sin cascada.

Ejemplo trivial → que la transacción bloquee la BD completa con lo que sólo se generan planificaciones secuenciales.

## 8. DEFINICIÓN DE TRANSACCIÓN EN SQL

**Commit work** → compromete la transacción actual y comienza una nueva.

**Rollback work** → provoca que la transacción actual aborte.

- La palabra clave work es opcional en ambas instrucciones.
- Si el programa termina sin ninguna de estas órdenes, la norma no especifica y depende de la implementación.
- La norma especifica que el sistema debe garantizar tanto la secuencialidad como la ausencia de retroceso en cascada.
- Definición de secuencialidad → la planificación debe tener el mismo efecto que tendría una planificación secuencial.
- Son aceptables tanto la secuencialidad en cuanto a conflictos como la secuencialidad en cuanto a vistas.

SQL-92 permite cierta laxitud con la secuencialidad, permite los **siguientes niveles de consistencia**:

- **Serializable (Secuenciable)** → nivel predeterminado.
- **Repeatable read (Lectura repetible)** → permite que se lean sólo los registros que se han comprometido y entre dos lecturas de un registro en una transacción no se permite que otra transacción actualice el registro. Sin embargo, la transacción puede no ser secuenciable respecto a otras T. (Ej ? registros que cumplan cierta condición)
- **Read Committed (Con compromiso de lectura)** → permite que se lean sólo los registros comprometidos pero no requiere también las lecturas repetidas.
- **Read Uncommitted (Sin compromiso de lectura)** → permite incluso que se lean registros sin comprometer. Nivel más bajo de consistencia.

## 9. COMPROBACIÓN DE LA SECUENCIALIDAD

### 1.- COMPROBACIÓN DE LA SECUENCIALIDAD EN CUANTO A CONFLICTOS

- Se construye un grafo dirigido, **grafo de precedencia** para la planificación.
- Los **vértices** son las **transacciones** que participan en la planificación.
- Hay una **arista** entre el vértice T y el T' (  $T \rightarrow T'$  ) si se cumple **alguna** de las 3 condiciones siguientes:
  - 1.- T ejecuta **escribir(Q)** antes de que T' ejecute **leer(Q)**.
  - 2.- T ejecuta **leer(Q)** antes de que T' ejecute **escribir(Q)**.
  - 3.- T ejecuta **escribir(Q)** antes de que T' ejecute **escribir(Q)**.
- Si existe un arco  $T \rightarrow T'$  en el grafo de precedencia, en toda planificación secuencial equivalente T debe aparecer antes que T'.
- Si el gra. tiene algún ciclo la planificación no es secuenciable en cuanto a conflictos.
- Si no tiene ciclos es secuenciable en cuanto a conflictos y el orden se puede obtener por una **ordenación topológica** que determina un orden lineal que consiste en el orden parcial del grafo (realizando un recorrido).
- Probar secuencialidad en cuanto a conflictos → algoritmo detección de ciclos ( $n^2$ ).

## 2.- COMPROBACIÓN DE LA SECUENCIALIDAD EN CUANTO A VISTAS

Es necesario construir un **grafo de precedencia etiquetado** en el que los nodos son las transacciones.

Sean  $P$  una planificación que consiste en las transacciones  $\{T_1, T_2, \dots, T_n\}$ .

Sean  $T_c$  y  $T_f$  dos transacciones ficticias:

$T_c \rightarrow$  ejecuta **escribir**( $Q$ ) para todo  $Q$  al que se accede en  $P$ .

$T_f \rightarrow$  ejecuta **leer**( $Q$ ) para todo  $Q$  al que se accede en  $P$ .

Se construye una nueva planificación  $P'$  a partir de  $P$  insertando  $T_c$  al comienzo de  $P$  y añadiendo  $T_f$  al final de  $P$ .

Las **reglas para insertar arcos etiquetados**: (Notación: En  $\rightarrow_0$  el cero irá sobre la flecha)

- 1) Añadir el arco  $T_i \rightarrow_0 T_j$  si la transacción  $T_j$  lee el valor de  $Q$  que ha escrito  $T_i$ .
- 2) Borrar los arcos que incidan en transacciones inútiles. Una **transacción** es **inútil** si no existe camino en el grafo de precedencia desde ella hasta la transacción  $T_f$ .
- 3) Para todo elemento de datos  $Q$  tal que  $T_j$  lee el valor de  $Q$  que ha escrito  $T_i$ , y  $T_k$  ejecuta **escribir**( $Q$ ) y  $T_k \neq T_c$ , hacer lo siguiente:
  - a) Si  $T_i = T_c$  y  $T_j \neq T_f$ , entonces insertar el arco  $T_j \rightarrow_0 T_k$  en el grafo.
  - b) Si  $T_i \neq T_c$  y  $T_j = T_f$ , entonces insertar el arco  $T_k \rightarrow_0 T_i$  en el grafo.
  - c) Si  $T_i \neq T_c$  y  $T_j \neq T_f$ , entonces insertar el par de arcos  $T_k \rightarrow_p T_i$  y  $T_j \rightarrow_p T_k$  en el grafo, siendo  $p$  un entero único mayor que 0 que no se haya utilizado antes para etiquetar arcos.

La regla 3c) indica que si  $T_i$  escribe un elemento de datos que lee  $T_j$ , entonces una transacción  $T_k$  que escriba el mismo elemento de datos debe ir antes que  $T_i$  o después de  $T_j$ . Se tiene una elección del lugar en que debe aparecer  $T_k$  en un orden secuencial equivalente.

Las reglas 3a) y 3b) son casos especiales que resultan de que  $T_c$  y  $T_f$  son la 1ª y última transacción.

Si el grafo **NO contiene ciclos**, la planificación **es secuenciable en cuanto a vistas**. Pero, si **contiene ciclos** puede ser o no secuenciable en cuanto a vistas (**no se sabe**).

**¿Cómo se determina si la planificación es secuenciable en cuanto a vistas?**

Si tenemos  $n$  pares de arcos con etiqueta distinta a 0, esto es, se ha aplicado  $n$  veces la regla 3c). Existen  $2^n$  grafos diferentes, donde cada uno contiene sólo uno de los arcos de cada par. Si alguno de estos grafos es acíclico, entonces la planificación es secuenciable en cuanto a vistas.

La secuencialidad se determina eliminando las transacciones ficticias  $T_c$  y  $T_f$  y haciendo la ordenación topológica del grafo acíclico restante.

Requiere la prueba exhaustiva de todos los posibles grafos distintos por lo que el **coste** es **exponencial**, es un problema NP-completo.

Por esto los **esquemas de control de concurrencia usan condiciones suficientes**. Si satisface las condiciones suficientes, la planificación es secuenciable en cuanto a vistas, pero puede haber planificaciones secuenciables en cuanto a vistas que no cumplan las condiciones suficientes.