

BASES DE DATOS

The background of the slide features several overlapping, wavy lines in orange, light blue, and lime green, creating a modern, abstract design.

CLASE 8

Transacciones

- Hasta ahora se ha visto:
 - Cómo diseñar una BD
 - Cómo manipular una BD
- Para poder **confiar en la información** almacenada en una base de datos es necesario asegurar que la misma **no contenga errores**
 - ¿Qué sucede si las operaciones **fallan** antes de completarse?
 - ¿Qué sucede si varios usuarios quieren trabajar **al mismo tiempo con los mismos datos**?

Transacciones

- Para las operaciones de **lectura**:
 - Una operación de lectura se interrumpe → **no afecta a los datos guardados**
 - Varios usuarios concurrentemente realizan lecturas sobre los mismos datos → **no causa problemas**
- Para las operaciones de **escritura**:
 - Una operación compleja se interrumpe en un momento indeterminado → **puede dejar inconsistencias**
 - Varios usuarios concurrentemente intentan realizar operaciones sobre los mismos datos → **puede generar inconsistencias**

Transacciones

- Es necesario **garantizar la integridad y consistencia de los datos** para cualquier situación posible, teniendo en cuenta que las bases de datos pueden ser utilizadas por múltiples usuarios simultáneamente
- El mecanismo que surge para garantizar que las operaciones realizadas por uno o más usuarios no provoquen una pérdida de consistencia en los datos almacenados es la **transacción**

Transacciones

- Una **transacción** es una secuencia de operaciones que forman una **unidad lógica de trabajo**
 - **Objetivo:** garantizar la **consistencia** de la BD a pesar de los fallos del sistema y de la ejecución concurrente
 - Para poder asegurar este objetivo, una transacción se debe realizar enteramente o no realizarse → **no puede quedar incompleta**
 - Si finaliza → sus efectos quedan en la BD
 - Si se anula → sus efectos no quedan en la BD

Transacciones

- Ejemplo de transacción: transferencia bancaria entre dos cuentas

READ (cuenta2.saldo)

cuenta2.saldo = cuenta2.saldo – importe

WRITE (cuenta2.saldo)

READ (cuenta1.saldo)

cuenta1.saldo = cuenta1.saldo + importe

WRITE (cuenta1.saldo)

Transacciones

- Ejemplo de transacción: transferencia bancaria entre dos cuentas
 - Saldos iniciales: **cuenta1** = \$2000, **cuenta2** = \$1000
 - Se realiza una transferencia de \$100 y el proceso se efectúa sin problemas; los saldos finales son correctos:
 - **cuenta1** = \$2100, **cuenta2** = \$900
 - Si se produjera un **fallo** luego de actualizar el saldo de la **cuenta2** pero antes de actualizar el saldo de la **cuenta1**; se “pierden” \$100 → **inconsistencia**
 - No se puede permitir que una transacción se ejecute de forma parcial

Transacciones

- El concepto moderno de transacciones fue formalizado por **Edgar F. Codd** en la década de 1970, introduciendo el concepto de **propiedades ACID** (Atomicidad, Consistencia, Aislamiento <Isolation>, Durabilidad)
- Estas propiedades, que deben cumplir las transacciones, son **esenciales para garantizar la fiabilidad y la coherencia de los datos** en las bases de datos, especialmente en entornos donde múltiples usuarios pueden acceder y manipular los datos simultáneamente.

Transacciones

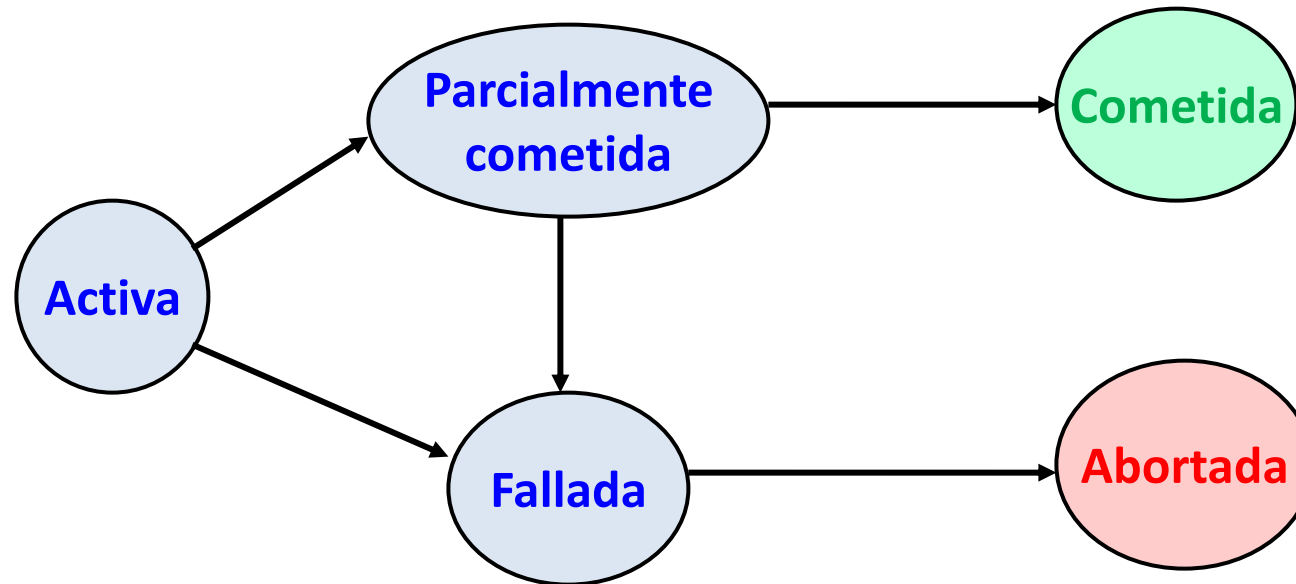
- Propiedades **ACID**
 - **Atomicidad**: una transacción debe ser una unidad atómica de trabajo. O todas las operaciones de la transacción se ejecutan o no lo hacen ninguna de ellas
 - **Consistencia**: la ejecución aislada de la transacción conserva la consistencia de la BD (lleva a la BD de un estado consistente a otro consistente)
 - **Aislamiento**: cada transacción ignora el resto de las transacciones que se ejecutan concurrentemente en el sistema, c/u actúa como única
 - **Durabilidad**: una transacción terminada con éxito realiza cambios permanentes en la BD, incluso si hay fallos en el sistema

Transacciones

- Una transacción puede pasar por varios **estados** durante su ciclo de vida.
 - **Activa**: estado inicial, estado normal durante la ejecución
 - **Parcialmente cometida**: después de ejecutarse la última 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) se ha cometido parcialmente y se garantiza que nunca abortará

Transacciones

- Diagrama de estados de una transacción



Transacciones

- Si una transacción falla → **debe retroceder**
 - Se debe tener especial cuidado con las **escrituras externas**
 - Ejemplo: una transacción emite mensajes por pantalla / impresora, pero luego se produce un fallo:
 - Las escrituras externas realizadas no pueden borrarse → **quedan mensajes fuera del sistema de BD**
 - Deben aplazarse este tipo de escrituras hasta que la transacción haya **cometido**

Transacciones

- En el caso en el que una transacción falla y retrocede, llega entonces al estado de **abortada**
- Una transacción abortada se puede:
 - **Reiniciar**: se intenta volver a ejecutar la transacción cuando **el error no depende de la lógica de la misma** (ya sea de hardware o de software)
 - **Cancelar**: se cancela la transacción si el fallo se debió a un error interno lógico de la misma

Transacciones

- Posibles entornos de transacciones
 - **Sistemas monousuario**
 - Un sólo usuario con una única transacción
 - **Sistemas concurrentes**
 - Varios usuarios simultáneos, c/u con múltiples transacciones a la vez
 - **Sistemas distribuidos**
 - BD residente en varios servidores → mayor complejidad

Fallos

- Causas de los fallos
 - Caída del sistema de BD
 - Software
 - Hardware
 - Catástrofes
 - Problemas con la ejecución de transacciones
 - Lógica de la transacción
 - Bloqueos entre transacciones

Fallos

- Causas de los fallos
 - Se debe conocer con precisión las posibilidades de **pérdida de información** del sistema de BD → se necesita una correcta elección de los **tipos de almacenamiento**
 - **Volátil**: la información no sobrevive a las caídas del sistema
 - **No volátil** → la información sobrevive a las caídas del sistema. Puede perderse ante fallas de hardware
 - **Estable** → en teoría, la información nunca se pierde
 - Sistemas UPS
 - Replicación en varios medios no volátiles independientes

Fallos

- Clasificación de los fallos
 - Cuando el fallo sucede en una transacción que **sólo ejecuta operaciones de lectura** → **fallo sin pérdida de información**
 - La transacción se aborta fácilmente, y la consistencia de la BD no se ve afectada
 - Cuando el fallo sucede en una transacción que **involucra escrituras de datos** → **fallo con pérdida de información**
 - Se debe asegurar la consistencia de la BD

Fallos

- Procedimiento ante fallos
 - ¿Reiniciar o cancelar?
 - De acuerdo con el **momento** y **tipo de error** que se produzca durante la ejecución de una transacción, se puede resolver **volver a ejecutar o no** la transacción
 - Pero en cualquiera de los casos se debe asegurar que la transacción es **abortada con anterioridad**, para mantener la **consistencia de datos**
 - ¿Cómo abortar una transacción?
 - ¿Qué parte de la transacción se había ejecutado al momento del error?

Fallos

- Procedimiento ante fallos
 - La consistencia de datos se puede perder si se efectúan cambios en la BD sin la seguridad de que la transacción va a finalizar correctamente
 - La solución implica **llevar el control de las modificaciones** que se efectúan en la BD
 - Esta tarea es **responsabilidad del SGBD**. Ante un fallo, una serie de **métodos de recuperación de integridad** permite la restauración de la consistencia de la BD

Fallos

- Procedimiento ante fallos
 - Los **métodos de recuperación de integridad** llevan a cabo diferentes tareas, que se pueden dividir en:
 - Acciones llevadas a cabo durante el procesamiento normal de la transacción → permite la recuperación ante fallos
 - Acciones llevadas a cabo después de ocurrir el fallo para restablecer el contenido de la BD a un estado que asegure las propiedades ACID
 - Se va a analizar los siguientes métodos:
 - Bitácora (log)
 - Doble paginación (paginación en sombra)

Bitácora

- La recuperación basada en bitácora mantiene un **registro histórico** en donde almacena la secuencia de actividades realizadas sobre la BD
- El **gestor de recuperación** del **SGBD** es el responsable de:
 - Identificar las transacciones → nro correlativo único
 - Controlar la ejecución de transacciones
 - Administrar el archivo de registro (bitácora)
 - Usar los algoritmos de recuperación

Bitácora

- Contenido de la bitácora
 - **<Ti Start>** se escribe antes de que empiece a ejecutar la transacción i. Indica que la transacción está activa
 - **<Ti, E, Va, Vn>**
 - **Ti**: Identificador de la transacción
 - **E**: Identificador del elemento de datos
 - **Va**: Valor anterior
 - **Vn**: Valor nuevo
 - **<Ti Commit>** → transacción parcialmente cometida
 - **<Ti Abort>** → transacción abortada

Bitácora

- Las operaciones deben almacenarse **antes en la bitácora**, y luego si realizarse sobre la BD
- ¿Qué pasa si una transacción comienza y se produce un fallo antes de que se escriba en la bitácora la sentencia **<Ti Commit>** o la sentencia **<Ti Abort>**?
- ¿Cómo se asegura que la bitácora se guarda efectivamente en disco?

Bitácora

- Alternativas de recuperación con bitácora
 - **Modificación diferida de la BD**
 - La base de datos se cambia recién cuando la transacción finaliza
 - **Modificación inmediata de la BD**
 - Ante un cambio, primero se guarda en la bitácora, y luego inmediatamente se lleva a la BD

Bitácora

- Modificación **diferida** de la BD
 - Demora todas las escrituras
 - Si el sistema se cae antes de que la transacción finalice su ejecución → se ignora la bitácora
 - Si la transacción se aborta → se ignora la bitácora
 - Si la transacción está parcialmente cometida → se usa la bitácora para las escrituras diferidas
 - No es necesario guardar los valores anteriores

Bitácora

- Modificación **diferida** de la BD
- Ejecución de una transacción **Ti**
 - Antes de empezar: $\langle Ti \text{ Start} \rangle$
 - Para cada escritura: $\langle Ti, E, Vn \rangle$
 - Al finalizar: $\langle Ti \text{ Commit} \rangle$
 - Escritura de la bitácora en **memoria estable**
 - Escritura diferida de las modificaciones en la BD

Bitácora

- Modificación **diferida** de la BD
 - Para el ejemplo de la transferencia entre dos cuentas:
 - <T0 Start>
 - <T0, cuenta2.saldo, 900>
 - <T0, cuenta1.saldo, 2100>
 - <T0 Commit>

Bitácora

- Modificación **diferida** de la BD
- Si se produjera un fallo, al recuperarse:
 - Toda transacción iniciada pero sin **<Ti Commit>** no realizó ningún cambio → **es ignorada**
 - Toda transacción iniciada y con **<Ti Commit>** puede haber hecho cambios, e incluso haber dejado una inconsistencia → **es ejecutada nuevamente**
 - **REDO(Ti) → IDEMPOTENCIA**

Bitácora

- 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 anterior, ya que los cambios se van efectuando sobre la BD
 - $\langle T_i, E, V_a, V_n \rangle$

Bitácora

- Modificación **inmediata** de la BD
 - Para el ejemplo de la transferencia entre dos cuentas:
 - <T0 Start>
 - <T0, cuenta2.saldo, 1000, 900>
 - <T0, cuenta1.saldo, 2000, 2100>
 - <T0 Commit>
 - Durante la ejecución se irá almacenando la bitácora y actualizando la BD

Bitácora

- Modificación **inmediata** de la BD
- Si se produjera un fallo, al recuperarse:
 - Toda transacción iniciada pero sin **<Ti Commit>** puede haber hecho cambios, e incluso haber dejado una inconsistencia → **debe deshacerse**
 - **UNDO(Ti)**
 - Toda transacción iniciada y con **<Ti Commit>** puede haber hecho cambios, e incluso haber dejado una inconsistencia → **es ejecutada nuevamente**
 - **REDO(Ti)**

Bitácora

- Puntos de verificación
 - Revisar la bitácora desde el comienzo
 - Lleva mucho tiempo
 - Probablemente gran porcentaje esté correcto y terminado
 - Es posible agregar puntos de verificación de forma periódica → desde allí hacia atrás está todo correctamente realizado
 - <checkpoint>
 - ¿Periodicidad?

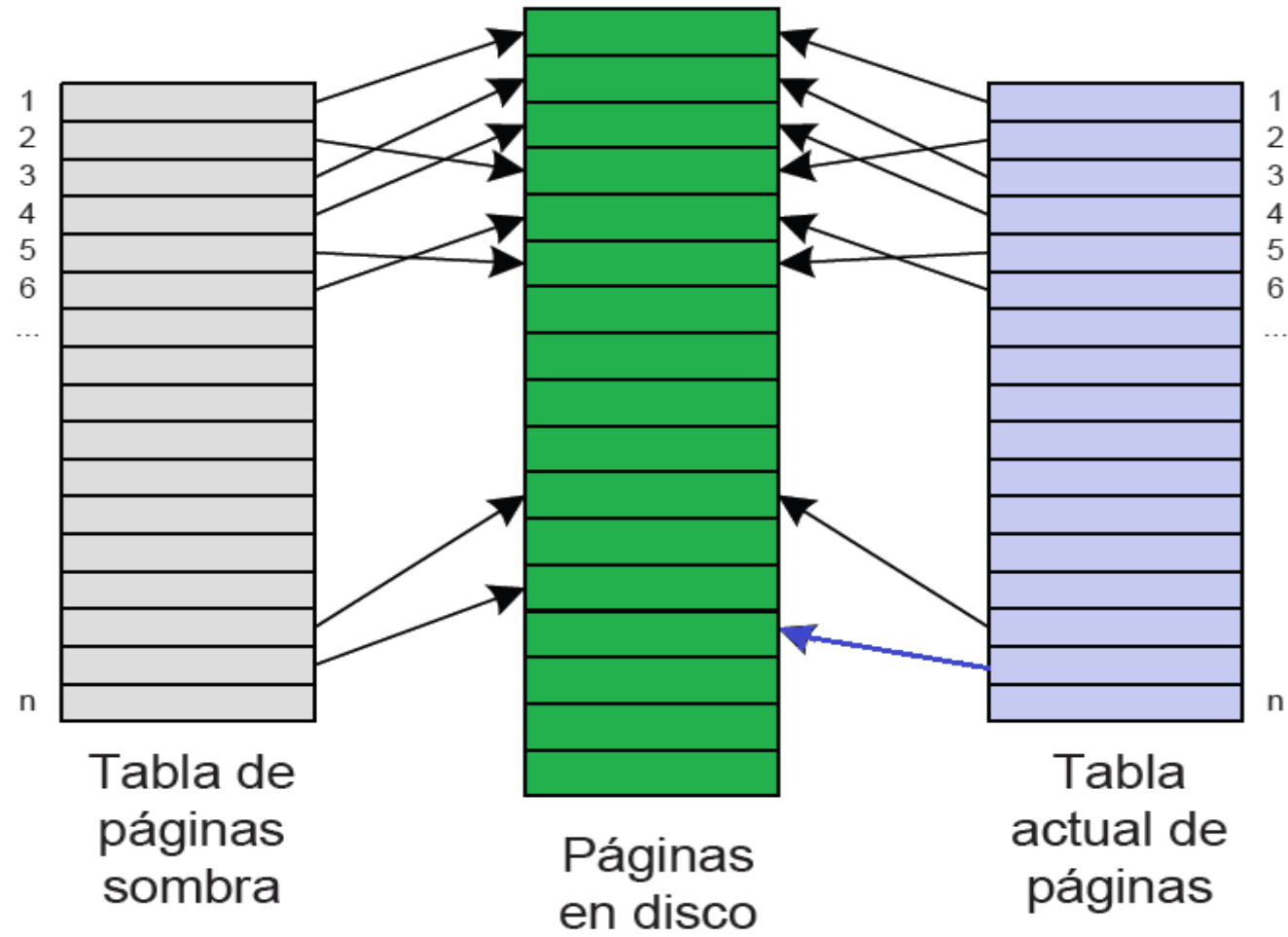
Bitácora

- Puntos de verificación
 - Ante un fallo se revisa sólo desde el último `<checkpoint>` en adelante
 - ¿Cuándo se colocan?
 - Entornos monousuario → períodos de inactividad
 - Entornos concurrentes → más complejo

Doble paginación

- La técnica de **doble paginación** consiste en dividir la BD en una determinada cantidad de bloques de longitud fija
- Estos bloques se numeran y se denominan nodos o **páginas**
- Durante la vida de una transacción se mantienen dos tablas de páginas:
 - Tabla actual → **memoria volátil**
 - Tabla sombra → **almacenamiento estable**

Doble paginación



Doble paginación

- Método de trabajo
 - Al iniciar una nueva transacción se genera una **nueva página** apuntada por la **tabla actual**
 - La **tabla de páginas en sombra** mantiene un puntero a la página que contiene el **estado anterior** de los datos involucrados en la transacción
 - Al comienzo, antes de ejecutar la primera operación de la transacción, **ambas tablas son iguales**
 - Durante la ejecución de la transacción sólo **se modifica la tabla actual**

Doble paginación

- Método de trabajo
 - Dada una transacción que realiza una escritura sobre la BD:
 - Se obtiene desde la BD el nodo a escribir
 - Se modifica el dato en memoria principal y se graba en disco, en un nuevo nodo → la tabla actual de páginas referencia al nuevo nodo
 - Si la operación finaliza correctamente:
 - Se modifica la referencia de la tabla de páginas en sombra para que apunte al nuevo nodo
 - Se libera el nodo viejo

Doble paginación

- Método de trabajo
 - Dada una transacción que realiza una escritura sobre la BD:
 - ¿Qué sucede si se produce un fallo o la caída del sistema?
 - No es posible garantizar que la escritura del nuevo nodo fue exitosa
 - Se recupera el estado anterior de la BD desde la tabla de páginas en sombra, que seguro es correcta

Doble paginación

- Método de trabajo
 - La **tabla de páginas en sombra** tiene el **estado anterior** de cualquier transacción que estuviera **activa** en el momento de la caída del sistema
 - Así, no es necesario disponer de una operación **UNDO(Ti)**
→ se cuenta con la dirección de la página anterior sin las modificaciones
 - **ABORT automáticos**

Doble paginación

- **Ventajas**

- Elimina la sobrecarga de escrituras de la bitácora
- Recuperación más rápida → no existe REDO/UNDO

- **Desventajas**

- División de la BD en páginas
- Sobrecarga
- Fragmentación
- Garbage Collector
- Complicada en ambientes concurrentes / distribuidos