

BASES DE DATOS

The background features several overlapping, wavy lines in orange, light blue, and lime green, creating a dynamic and modern aesthetic.

CLASE 9

Entornos concurrentes

- En un **entorno monousuario** es posible mantener la BD **consistente** mientras los usuarios acceden, agregan, borran o modifican los datos de la BD.
- Mediante el uso de **transacciones** y un conjunto de **métodos de recuperación de integridad**, el SGBD garantiza las **propiedades ACID** (atomicidad, consistencia, aislamiento y durabilidad)

Entornos concurrentes

- Sin embargo, aunque es más sencillo que las transacciones se ejecuten secuencialmente, existen **razones para permitir la concurrencia**
- Una transacción está constituida por una serie de pasos:
 - Pasos que implican **operaciones de E/S**
 - Pasos que implican **operaciones de CPU**

Entornos concurrentes

- Es posible realizar las operaciones de E/S en **paralelo** con el procesamiento de CPU → **varias transacciones simultáneas**
 - Se aumenta la **productividad** del sistema → hay menos dispositivos desocupados
 - Se mejora el **tiempo de respuesta promedio** de una transacción

Entornos concurrentes

- Varias transacciones ejecutándose simultáneamente **compartiendo recursos**
 - Transacciones correctas en un entorno monousuario → pueden llevar a **fallos** en un ambiente concurrente
 - Se necesita un **mecanismo de control de concurrencia** que asegure que las transacciones concurrentes no interfieren entre sí
 - Permite mantener la **consistencia** de la BD

Entornos concurrentes

T0: **READ**(A)
 A = A – 50
 WRITE(A)
 READ(B)
 B = B + 50
 WRITE(B)

T1: **READ**(A)
 temp = A * 0.1
 A = A – temp
 WRITE(A)
 READ(B)
 B = B + temp
 WRITE(B)

- Secuencializando: A + B → ¿OK?
 - T0, T1
 - T1, T0
- T0-T1 <> T1-T0: A + B → ¿OK?

Entornos concurrentes

- Caso 1: $A + B$

T0: **READ(A)**
 $A = A - 50$
 WRITE(A)

READ(B)
 $B = B + 50$
 WRITE(B)

T1: **READ(A)**
 $\text{temp} = A * 0.1$
 $A = A - \text{temp}$
 WRITE(A)

READ(B)
 $B = B + \text{temp}$
 WRITE(B)

Entornos concurrentes

- Caso 2: $A + B$

T0: **READ**(A)
 $A = A - 50$

WRITE(A)
READ(B)
 $B = B + 50$
WRITE(B)

T1:

READ(A)
temp = $A * 0.1$
 $A = A - \text{temp}$
WRITE(A)
READ(B)

$B = B + \text{temp}$
WRITE(B)

Entornos concurrentes

- Problemas de concurrencia
 - Si se produce un fallo en el sistema, o **aún sin que se produzca**, una transacción correcta por sí misma puede producir un resultado incorrecto por **interferencia con otra transacción**
 - **Tres posibles problemas**
 - El problema de la actualización perdida
 - El problema de la dependencia no confirmada
 - El problema del análisis inconsistente

Entornos concurrentes

- Problemas de concurrencia: **actualización perdida**
 - *Una actualización “pisa” a otra (que se pierde)*
 - La transacción A **recupera** la tupla **T** en el tiempo **t1**
 - La transacción B **recupera** la misma tupla **T** en **t2**
 - La transacción A **actualiza** la tupla **T** en **t3**
 - La transacción B **actualiza** la misma tupla **T** en **t4**
 - La actualización de la transacción A **se pierde** en **t4**

Entornos concurrentes

- Problemas de concurrencia: **dependencia no confirmada**
 - *Una transacción recupera o actualiza una tupla actualizada por otra transacción (aún no finalizada)*
 - La transacción B **actualiza** la tupla **T** en el tiempo **t1**
 - La transacción A **recupera** la tupla **T** en el tiempo **t2**
 - La transacción B **deshace la actualización** de la tupla **T** en el tiempo **t3** (undo)
 - La transacción A está operando bajo **suposición falsa**

Entornos concurrentes

- Problemas de concurrencia: **análisis inconsistente**
 - *Una transacción realiza cálculos con tuplas que otra transacción está modificando → resultado incorrecto*
 - La transacción A **acumula saldos** de tres cuentas: **c1**, **c2** y **c3**
 - La transacción B **transfiere** \$1000 de la cuenta **c3** a la cuenta **c1** y se confirma. Pero esto sucede después de que A procesó **c1**, pero antes de que A procesase **c3**.
 - Si luego A finaliza con éxito → **resultado erróneo**

Entornos concurrentes

- Problemas de concurrencia
 - Es importante controlar que las transacciones **no interfieran entre sí**, para asegurar que se mantiene la **consistencia en la BD**
 - Las instrucciones **READ** y **WRITE** son las que pueden generar un conflicto y son las que deben considerarse
 - El **orden en el que se ejecutan** este tipo de instrucciones es **determinante**

Seriabilidad

- Una **planificación** es una secuencia de ejecución de transacciones
 - Representa el **orden cronológico** en el cual se ejecutan las instrucciones en el sistema
 - Una **planificación serial** (secuencial) consiste en la secuencia de instrucciones de varias transacciones, en la cual las instrucciones pertenecientes a una misma transacción **están juntas**

Seriabilidad

- Un entorno concurrente con N transacciones en ejecución puede generar $N!$ planificaciones seriales diferentes
- Sin embargo, cuando se ejecutan concurrentemente varias transacciones, la planificación **no necesariamente será serial**
- Las instrucciones de distintas transacciones se pueden intercalar → **son posibles muchas más secuencias de ejecución**

Seriabilidad

- Asumiendo que las transacciones involucradas son **correctas** (transforman un estado correcto de la BD en otro estado correcto):
 - La ejecución de las transacciones en cualquier orden serial → **es correcta**
 - Si una ejecución intercalada equivale a alguna ejecución serial → **es correcta**
 - Se dice entonces que es una **planificación serializable**

Seriabilidad

- En el ejemplo visto al inicio de la clase, suponiendo que inicialmente $A=100$ y $B=100$:
 - Ejecución serial $T0, T1 \rightarrow A=45$ y $B=155 \rightarrow$ correcta
 - Ejecución serial $T1, T0 \rightarrow A=40$ y $B=160 \rightarrow$ correcta
 - Cualquier ejecución intercalada con un resultado equivalente a alguna de las ejecuciones seriales también es correcta

Seriabilidad

- Conflictos en planificaciones
 - I_1, I_2 instrucciones de las transacciones **T1** y **T2** respectivamente
 - Si operan sobre datos distintos → **no hay conflicto**
 - Si operan sobre el mismo dato **Q**:
 - $I_1 = \text{READ}(Q) = I_2 \rightarrow$ **no importa el orden de ejecución**
 - $I_1 = \text{READ}(Q), I_2 = \text{WRITE}(Q) \rightarrow$ depende del orden de ejecución (I_1 leerá valores distintos)
 - $I_1 = \text{WRITE}(Q), I_2 = \text{READ}(Q) \rightarrow$ depende del orden de ejecución (I_2 leerá valores distintos)
 - $I_1 = \text{WRITE}(Q) = I_2 \rightarrow$ depende del orden de ejecución (estado final de la BD diferentes)
 - I_1, I_2 **están en conflicto** si actúan sobre el **mismo dato** y al menos una de ellas es una operación de **escritura**

Seriabilidad

- Conflictos en planificaciones
 - Dos planificaciones $S1$ y $S2$ se denominan **equivalentes en cuanto a conflictos** cuando $S2$ se logra a partir del intercambio de **instrucciones no conflictivas** de $S1$
 - $S2$ es **serializable en cuanto a conflictos** si existe otra planificación $S1$ equivalente en cuanto a conflictos con $S2$, y además $S1$ es una planificación serial

Seriabilidad

- Pruebas de seriabilidad
 - Existen métodos para demostrar la **seriabilidad de planificaciones concurrentes**. Se trata de algoritmos diseñados para **detectar conflictos** en la ejecución concurrente de dos o más transacciones
 - Uno de estos métodos es el **grafo de precedencia de transacciones**. En este grafo, cada **vértice** o nodo representa una **transacción** de la planificación. Una **arista** dirigida de un nodo a otro indica una **dependencia** entre las transacciones involucradas

Seriabilidad

- Pruebas de seriabilidad
 - Dadas dos transacciones T_i , T_j pertenecientes a una planificación, existirá una arista dirigida de T_i a T_j si:
 - T_i ejecuta un $WRITE(Q)$ antes que T_j un $READ(Q)$
 - T_i ejecuta un $READ(Q)$ antes que T_j un $WRITE(Q)$
 - T_i ejecuta un $WRITE(Q)$ antes que T_j un $WRITE(Q)$
 - Si el grafo tiene **ciclos** → la planificación **no es serializable en cuanto a conflictos**

Control de concurrencia

- Métodos de control de concurrencia
 - Existen diferentes tipos de técnicas de control de concurrencia. Algunas de estas técnicas son las basadas en bloqueos, las basadas en marcas temporales, las optimistas (validación), entre otras.
 - Las técnicas de control de concurrencia que se analizarán son:
 - Protocolo de **bloqueo**
 - Protocolo basado en **hora de entrada**

Control de concurrencia

- Protocolo de bloqueo
 - Las transacciones adquieren **bloqueos** sobre los objetos de datos que utilizarán
 - Se asegura que los datos que necesita para su ejecución no cambiarán
 - Dos tipos de bloqueos:
 - Exclusivos
 - Compartidos

Control de concurrencia

- Protocolo de bloqueo
 - Si una transacción pone un **bloqueo exclusivo** sobre la tupla **T**:
 - Se **rechaza** el pedido de cualquier otra transacción para un bloqueo **de cualquier tipo** sobre **T**
 - Si una transacción pone un **bloqueo compartido** sobre la tupla **T**:
 - Se **rechaza** el pedido de cualquier otra transacción para un bloqueo **exclusivo** sobre **T**
 - Se **acepta** el pedido de cualquier otra transacción para un bloqueo **compartido** sobre **T**

Control de concurrencia

- Protocolo de bloqueo
 - Una transacción que desea **recuperar** una tupla **T**, antes debe adquirir un bloqueo **compartido** sobre **T**
 - Una transacción que desea **actualizar** una tupla **T**, antes debe adquirir un bloqueo **exclusivo** sobre **T**
 - Cada transacción debe solicitar el tipo de bloqueo que necesite. Si obtiene el dato → debe **utilizarlo** y luego **liberarlo**

Control de concurrencia

- Protocolo de bloqueo
 - Si el bloqueo pedido por una **transacción B** se rechaza porque **conflictúa** con un bloqueo de la **transacción A**, la transacción B puede:
 - **Esperar por el dato y reintentar**. Durante el tiempo de espera, la transacción puede realizar otras operaciones que no requieran el recurso bloqueado
 - **Fallar y abortar**. Se debe deshacer las operaciones que haya realizado la transacción y luego reiniciar desde el principio como una nueva transacción
 - **Notificar al usuario/aplicación**. Notificar sobre la imposibilidad de continuar la ejecución para que el usuario decida qué hacer

Control de concurrencia

- Protocolo de bloqueo
 - Sean dos transacciones **T1** y **T2**:
 - **T1** requiere utilización **exclusiva** de los datos
 - **T2** requiere utilización **compartida** de los datos

| | |
|---|---|
| T1 leer(cuentaA) cuentaA = cuentaA - 1000; escribir (cuentaA) Leer (cuentaB) cuentaB = cuentaB + 1000; escribir(cuentaB) | T2 leer(cuentaA) leer (cuentaB) presentar (cuentaA + cuentaB) |
|---|---|

Control de concurrencia

- Protocolo de bloqueo
 - Las mismas transacciones con las instrucciones necesarias para efectuar el bloqueo de datos:

| | |
|---|---|
| <p>T1</p> <p>bloqueo_exclusivo(cuentaA)</p> <p>leer(cuentaA)</p> <p>cuentaA = cuentaA - 1000;</p> <p>escribir (cuentaA)</p> <p>Liberar_bloqueo(cuentaA)</p> <p>bloqueo_exclusivo(cuentaB)</p> <p>Leer (cuentaB)</p> <p>cuentaB = cuentaB + 1000;</p> <p>escribir(cuentaB)</p> <p>Liberar_bloqueo(cuentaB)</p> | <p>T2</p> <p>bloqueo_compartido(cuentaA)</p> <p>leer(cuentaA)</p> <p>Liberar_bloqueo(cuentaA)</p> <p>bloqueo_compartido(cuentaB)</p> <p>leer (cuentaB)</p> <p>Liberar_bloqueo(cuentaB)</p> <p>presentar (cuentaA + cuentaB)</p> |
|---|---|

Control de concurrencia

- Protocolo de bloqueo

- Ejemplo de una secuencia de ejecución posible.
Estado inicial: $\text{cuentaA} = 4000$, $\text{cuentaB} = 1000$

1. T1 → bloqueo_exc(A) → éxito
2. T2 → bloqueo_comp(A) → fracaso (T2 en espera)
3. T1 → leer(A) (A = 4000)
4. T1 → A = A - 1000 (A = 3000)
5. T1 → escribir(A) (A = 3000, en BD)
6. T1 → liberar_bloqueo(A)
7. T2 → bloqueo_comp(A) → éxito (T2 continúa)
8. T2 → leer(A) (A = 3000)
9. T2 → liberar_bloqueo(A)
10. T2 → bloqueo_comp(B), → éxito

11. T1 → bloqueo_exc(B), → fracaso (T1 en espera)
12. T2 → leer(B) (B = 1000)
13. T2 → liberar_bloqueo(B)
14. T1 → bloqueo_exc(B), → éxito (T1 continúa)
15. T1 → leer(B) (B = 1000)
16. T1 → B = B + 1000, (B = 2000)
17. T1 → escribir (B) (B = 2000, en BD)
18. T1 → liberar_bloqueo(B)
19. T2 → presentar(A+B) → presenta 4000

Control de concurrencia

- Protocolo de bloqueo
 - Como se puede ver en el ejemplo, aun realizando los bloqueos pertinentes **no se garantiza el aislamiento**
 - Una **solución** a este problema es que las transacciones soliciten todos los bloqueos de datos que necesitarán al **inicio de su ejecución**
 - Se deben liberar los bloqueos una vez que ya se utilizaron los datos
 - Esta solución genera un **nuevo problema** denominado interbloqueo o **deadlock**

Control de concurrencia

- Deadlock
 - Los **interbloqueos** son situaciones en las que dos o más transacciones se encuentran en **estado simultáneo de espera**
 - El sistema debe poder detectarlos y romperlos
 - La **detección** implica descubrir un **ciclo** en el grafo de espera (grafo que indica “quién está esperando a quién”)
 - La **ruptura** implica seleccionar una de las transacciones bloqueadas como **víctima** y **deshacerla** (liberando así su bloqueo sobre el dato en conflicto)

Control de concurrencia

- Deadlock

- Ejemplo:

T1:

bloqueo_exc(**A**)

bloqueo_exc(**B**)

T2:

bloqueo_comp(**B**)

bloqueo_comp(**A**)

- La transacción **T1** no progresará hasta que no se libere **B** y la transacción **T2** no progresará hasta que no se libere **A**
→ **deadlock**
 - Una de las dos transacciones **debe retroceder**, liberando así sus bloqueos de datos

Control de concurrencia

- Deadlock
 - Para realizar la **selección de la víctima** se tiene en cuenta el **menor costo** posible:
 - La última que empezó
 - La que haya realizado menos trabajo
 - La que haya realizado menos escrituras
 - La de menor prioridad
 - Si una transacción **siempre es elegida como víctima** nunca progresará → **inanición**
 - Para evitarlo es posible asignar **prioridades** → cuando una transacción es seleccionada como víctima se sube su prioridad

Control de concurrencia

- Inanición → otra posibilidad
 - Un conjunto de transacciones solicita un **bloqueo compartido** sobre el mismo elemento de datos
 - Cada una de ellas libera el bloqueo poco después de que sea concedido
 - Otra transacción necesita un **bloqueo exclusivo** sobre este elemento de datos, pero **nunca lo obtiene** → **no progresa**

Control de concurrencia

- Inanición → otra posibilidad
 - Para evitar estos casos de inanición, cuando la transacción T_i pide un bloqueo sobre un elemento de datos Q en un modo particular M , se concede el bloqueo siempre que:
 - No exista otra transacción que posea un bloqueo sobre Q en un modo que conflictúe con M
 - No exista otra transacción que esté esperando un bloqueo sobre Q y que lo haya solicitado antes que T_i

Control de concurrencia

- Protocolo de bloqueo de **dos fases**
 - Agrega reglas a la definición convencional para **evitar situaciones de pérdida de consistencia**
 - Requiere que las transacciones realicen sus bloqueos en **dos fases**:
 - **Fase de crecimiento**: se obtienen datos. Se piden bloqueos en orden: compartido, exclusivo. No se liberan bloqueos
 - **Fase de decrecimiento**: se liberan bloqueos o se pasa de exclusivo a compartido (conversiones de bloqueos). No se puede obtener ningún bloqueo nuevo

Control de concurrencia

- Protocolo de bloqueo de **dos fases**
 - Mas allá de que existen otras alternativas, es uno de los protocolos **más utilizados comercialmente**
 - **Ventaja:**
 - Proporciona un buen equilibrio entre concurrencia y rendimiento
 - Garantiza seriabilidad en cuanto a conflictos
 - **Desventajas:**
 - No evita situaciones de interbloqueo
 - Mucho uso de bloqueos exclusivos → provoca serie

Control de concurrencia

- Protocolo basado en hora de entrada
 - A cada transacción **T_i** del sistema se le asocia una **hora de entrada fija única (HDE)**, antes de que la transacción **T_i** empiece su ejecución
 - Hora del servidor
 - Un contador (contador lógico que se incrementa después de asignar una nueva hora de entrada)
 - **HDE(T_i) < HDE(T_j)** → **T_i** es anterior y se ejecuta primero
 - Las operaciones **READ** y **WRITE** que se ejecutan pueden entrar en **conflicto** y eventualmente **fallar** por **HDE**

Control de concurrencia

- Protocolo basado en hora de entrada
 - El método además asigna a cada elemento de dato **D** de la BD dos **marcas temporales**:
 - La hora de última lectura: **HL(D)**
 - La hora de última escritura : **HE(D)**
 - Estas marcas corresponden a la **HDE** de la **última transacción** que **leyó** o **escribió** el dato

Control de concurrencia

- Protocolo basado en hora de entrada
 - Sea una transacción **T1** que desea leer el dato **D**:
 - Para poder leer el dato, se debe cumplir que: $HDE(T1) > HE(D)$
 - Esta acción asegura que cualquier transacción que intente leer un dato, debe haberse iniciado en un momento posterior a la última escritura de ese dato

Control de concurrencia

- Protocolo basado en hora de entrada
 - Sea una transacción **T1** que desea leer el dato **D**:
 - ¿Qué pasa si $HDE(T1) < HE(D)$?
 - El dato **D** es demasiado nuevo para que **T1** pueda leerlo. Una transacción posterior a **T1** pudo escribir el dato
 - Si se acepta la lectura del dato **D** por parte de **T1**, es posible que **T1** plantee una situación de pérdida de consistencia de información

Control de concurrencia

- Protocolo basado en hora de entrada
 - Notar que para leer un dato de la BD, el protocolo basado en HDE **no necesita chequear la HL** del dato → **las lecturas pueden ser compartidas**
 - Varias transacciones pueden leer el mismo dato sin generar conflicto entre ellas
 - En caso de que la operación de lectura del dato **D** tenga éxito, se debe establecer la **HL(D)** como el **máximo** entre **HDE(T1)** y **HL(D)**

Control de concurrencia

- Protocolo basado en hora de entrada
 - Sea una transacción **T1** que desea **escribir** el dato **D**:
 - Si **HDE(T1) < HL(D)** la operación **falla**. No es posible que **T1** escriba el dato **D** si **fue leído por una transacción posterior** a ella. Si se aceptara esa operación, la transacción que leyó resultaría incorrecta
 - Si **HDE (T1) < HE (D)** la operación **falla**. No es posible que **T1** escriba el dato **D**, cuando dicho dato ya **fue escrito por una transacción posterior**
 - En cualquier otro caso, **T1** puede ejecutar la operación de escritura, y establece **HE(D)** con el valor de **HDE (T1)**

Recuperación ante fallos

- Protocolo basado en bitácora → concurrencia
- **Nuevo tipo de fallo:** una transacción que no puede continuar su ejecución por problemas de **bloqueos** o **acceso** a la BD
- Retroceso en cascada
 - Falla una transacción → puede llevar a abortar otras
 - Puede implicar **deshacer gran cantidad de trabajo**

Recuperación ante fallos

- Protocolo basado en bitácora → concurrencia
 - Puede ocurrir que una transacción **T_i** falle, y debido a ello otra transacción **T_j** deba retrocederse. ¿Pero que pasa si esto **no es posible** porque **T_j** ya finalizó?
 - **Nueva condición:** **T_j** no puede finalizar su ejecución si **T_i** utiliza datos que **T_j** necesita, y **T_i** no está en estado de finalizada
 - Si **T_i** **falla** → **T_j** también **falla**

Recuperación ante fallos

- Protocolo basado en bitácora → **concurrency**
- **Checkpoints**: se deberían colocar cuando ninguna transacción esta activa → en entornos concurrentes ese momento **puede no existir**
 - **<Checkpoint (L)>**, donde **L** es la **lista de transacciones activas** al momento del checkpoint
 - Ante un fallo:
 - **UNDO** y **REDO** según el caso
 - Debemos buscar antes del checkpoint sólo aquellas transacciones que estén en la lista **L**

Conclusiones

- Integridad → protección ante pérdidas accidentales de consistencia
 - Problemas durante el procesamiento de transacciones
 - Fallos de hardware
 - Interrupción del suministro de energía
 - Acceso concurrente a los datos
 - Anomalías causadas por la distribución de datos sobre varias computadoras

Conclusiones

- Seguridad → protección contra intentos malintencionados para modificar datos
 - Nivel físico: control sobre robos, catástrofes, etc.
 - Nivel humano: control de acceso a usuarios
 - Nivel SO: control de acceso a más bajo nivel, control de recursos, etc.
 - SGBD: autorización de operaciones sobre los datos
 - Admin. BD: autorización de operaciones sobre el esquema, índices, etc.