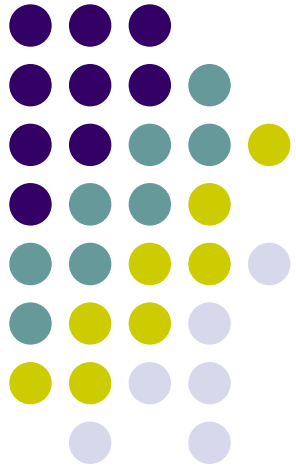
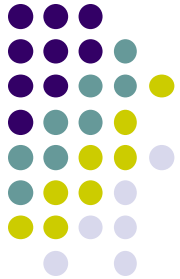
Recuperación y Concurrencia



Índice

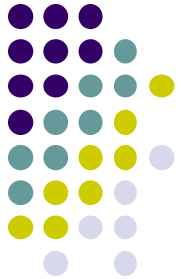
- Introducción
- Estructura del almacenamiento y sus operaciones
- Problemas asociados a fallos
- Transacciones
- Recuperación ante fallos
- Concurrencia

Introducción



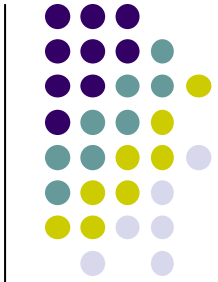
- Recuperación: Técnicas que proporcionan los SGBD para poder recuperar la información cuando se produce un fallo en el SGBD.
- Concurrencia: Técnicas que proporcionan los SGBD para permitir el acceso concurrente de varios usuarios a los mismos datos.

Índice



- Introducción
- Estructura del almacenamiento y sus operaciones
- Problemas asociados a fallos y concurrencia
- Transacciones
- Recuperación ante fallos
- Concurrencia

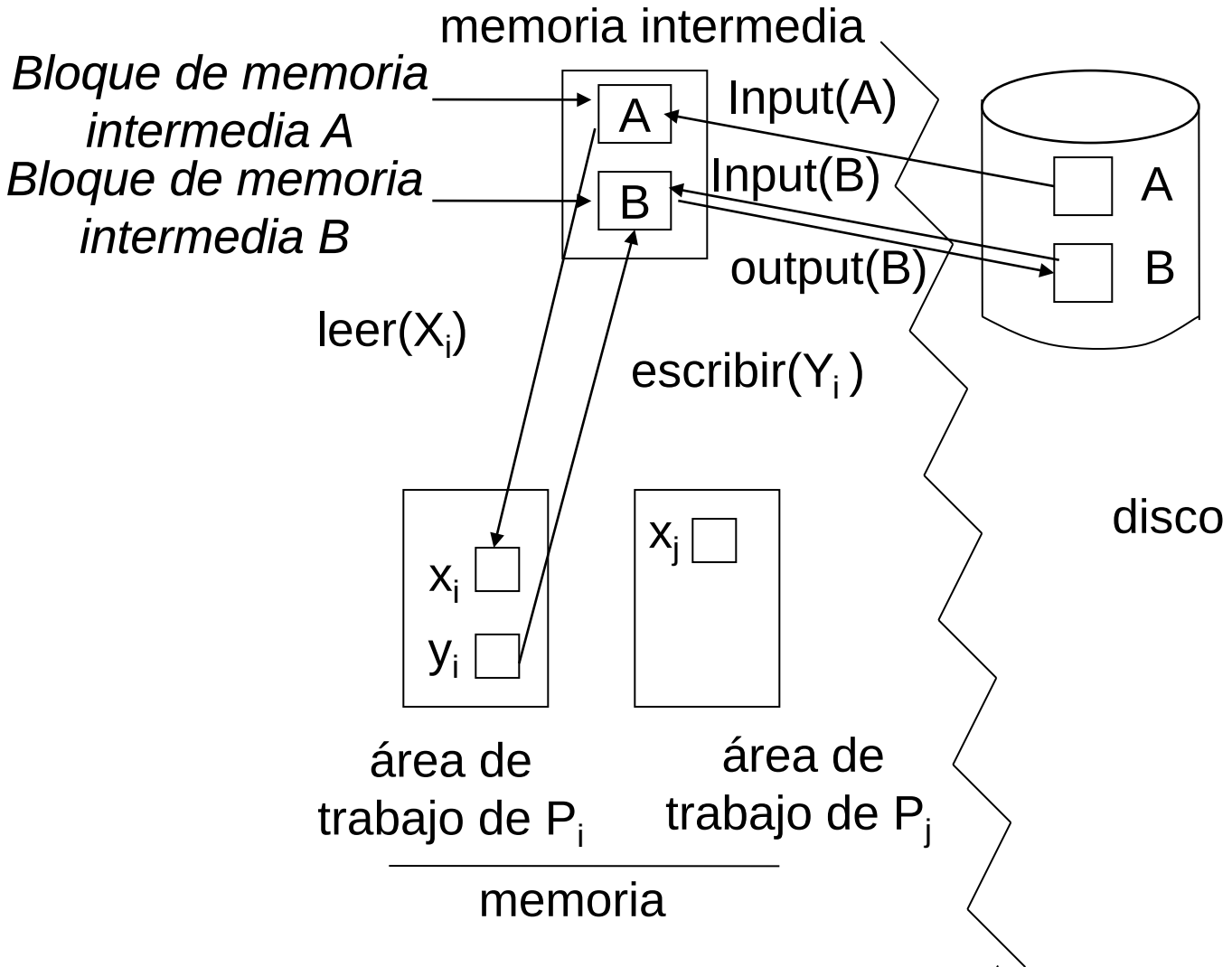
Estructura del almacenamiento



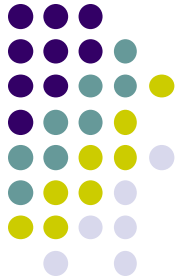
- **Almacenamiento volátil**:
  - no sobrevive a la caída del sistema
  - ejemplos: memoria principal, memoria caché
- **Almacenamiento no volátil**:
  - sobrevive a las caídas de los sistemas
  - ejemplos: disco, cinta, memoria flash, no volátil (batería de reserva) RAM
- **Almacenamiento estable**:
  - una forma mítica de almacenamiento que sobrevive a todos los fallos
  - aproximada en el mantenimiento de muchas copias en distintos medios no volátiles

Estructura op de almacenamiento

- Cada proceso tiene su área de memoria

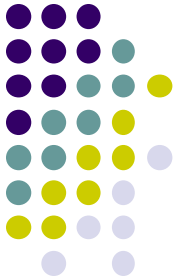


Estructura op de almacenamiento



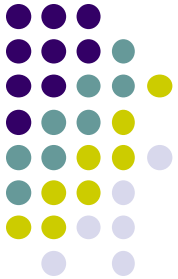
- Leer (X_i)
  - Input(A): Lee el bloque físico (A) donde está X a no ser que ya esté en la memoria intermedia
  - Trasladar el valor X al área de memoria de P_i
- Escribir (Y_i)
  - Input(B)
  - Trasladar el valor Y a la memoria intermedia.
  - No necesita un Output(B), ya se ocupará el sistema de paginación.

Índice



- Introducción
- Estructura del almacenamiento y sus operaciones
- Problemas asociados a fallos
- Transacciones
- Recuperación ante fallos
- Concurrencia

Problemas: Fallos

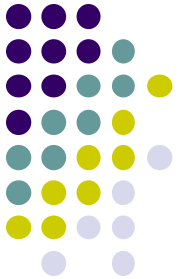


1. **leer**(X_i)
2. $X_i := X_i - 50$
3. **escribir**(X_i)
4. **leer**(Y_i)
5. $Y_i := Y_i + 50$
6. **escribir**(Y_i)

Momento Fallo		Resultado esperado	
X	Y	X	Y
950	100	950	150

- ¿Qué pasa si ocurre un fallo entre los pasos 3 y 4? Si X= 1000 y Y=100 inicialmente
- O se hace todo o no se hace nada.

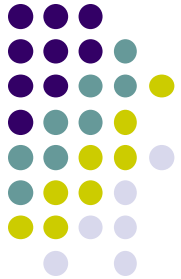
Índice



- Introducción
- Estructura del almacenamiento y sus operaciones
- Problemas asociados a fallos
- Transacciones
  - Definición
  - Propiedades ACID
  - Commit y Rollback
  - Estados
- Recuperación ante fallos
- Concurrencia

Transacción

Definición



- Tanto la recuperación ante fallos como el control del acceso concurrente se apoyan en el concepto de *Transacción.*
- Una ***transacción*** es una *unidad* de ejecución de programa que accede, y posiblemente actualiza, a varios elementos de datos.
- Una transacción debe ver una base de datos consistente.
- Durante la ejecución de la transacción la base de datos puede ser inconsistente.
- Cuando se compromete una transacción la base de datos deber ser consistente.
- Se pueden ejecutar múltiples transacciones en paralelo.

Índice

- Introducción
- Estructura del almacenamiento y sus operaciones
- Problemas asociados a fallos
- Transacciones
  - Definición
  - Propiedades ACID
  - Commit y Rollback
  - Estados
- Recuperación ante fallos
- Concurrencia

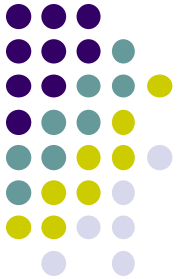
Transacción: Propiedades ACID

Atomicity, Consistency, Isolation and Durability

Para preservar la integridad de los datos, el sistema de bases de datos debe asegurar:

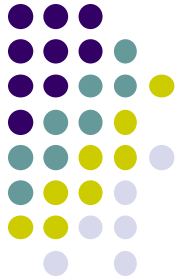
- **Atomicidad.** O todas las operaciones de la transacción se reflejan correctamente en la base de datos, o ninguna.
- **Consistencia.** La ejecución de una transacción en aislamiento preserva la consistencia de la base de datos..
- **Aislamiento.** Aunque varias transacciones se pueden ejecutar concurrentemente, cada transacción debe ignorar a las otras transacciones que se ejecutan concurrentemente con ella. Los resultados de las transacciones intermedias deben ocultarse de otras transacciones ejecutadas concurrentemente.
  - Es decir, por cada par de transacciones T_i y T_j, a T_i le parece que, o bien T_j ha terminado su ejecución antes de que comience T_i, o que T_j ha comenzado su ejecución después de que T_i terminara.
- **Durabilidad.** Tras la finalización con éxito de una transacción permanecen los cambios realizados en la base de datos, incluso si hay fallos en el sistema.

Índice



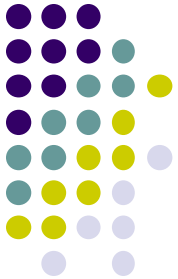
- Introducción
- Estructura del almacenamiento y sus operaciones
- Problemas asociados a fallos
- Transacciones
  - Definición
  - Propiedades ACID
  - Commit y Rollback
  - Estados
- Recuperación ante fallos
- Concurrencia

Transacciones: Commit y Rollback



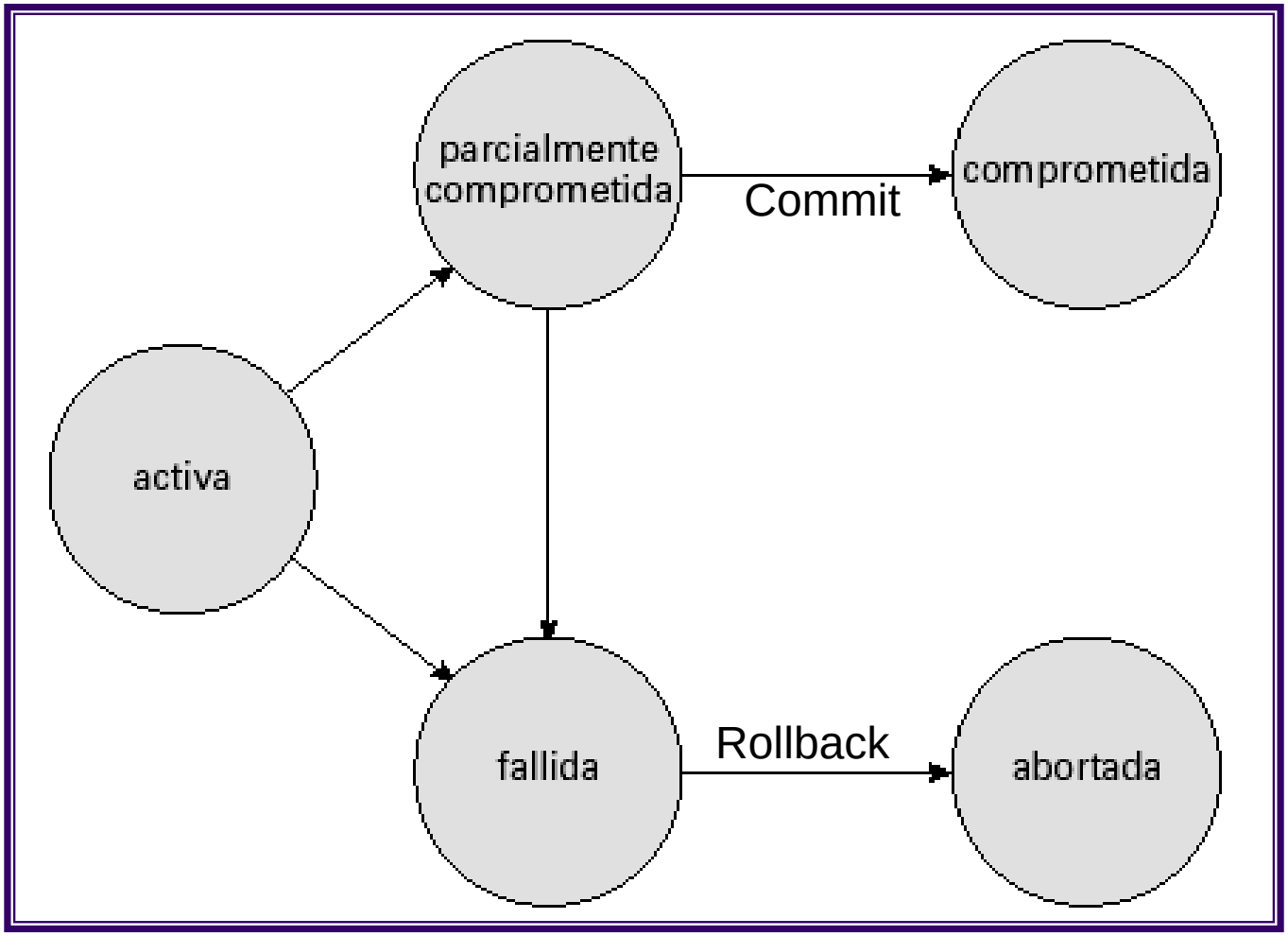
- Commit: Señala una *finalización satisfactoria* de la transacción, por lo que los cambios (actualizaciones) ejecutados por la transacción se pueden enviar con seguridad a la BD y no se desharán.
- Rollback: Señala que la transacción *no ha terminado satisfactoriamente*, por lo que deben deshacerse los cambios o efectos que la transacción pudiera haber aplicado a la BD

Índice

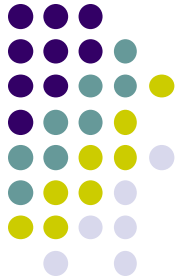


- Introducción
- Estructura del almacenamiento y sus operaciones
- Problemas asociados a fallos
- Transacciones
  - Definición
  - Propiedades ACID
  - Commit y Rollback
  - Estados
- Recuperación ante fallos
- Concurrencia

Estados de una transacción



Estados de una transacción

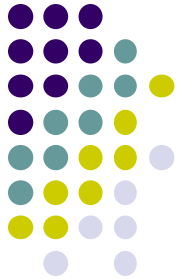


- **Activa**, el estado inicial; la transacción permanece en este estado mientras se está ejecutando
- **Parcialmente comprometida**, después que se ha ejecutado la instrucción final.
- **Fallida**, después de descubrir que la ejecución normal ya no puede llevarse a cabo.
- **Abortada**, después que la transacción se ha retrocedido y la base de datos restaurado a su estado anterior al inicio de la transacción. Después de que haya abortado se podría reiniciar la transacción – sólo si no hay errores lógicos internos.
- **Comprometida**, después de terminación con éxito.

Índice

- Introducción
- Estructura del almacenamiento y sus operaciones
- Problemas asociados a fallos
- Transacciones
- Recuperación ante fallos
  - Introducción
  - Tipos de fallos
  - Log
- Concurrencia

Recuperación ante fallos

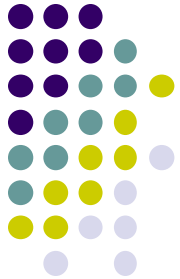


- Parte de todo SGBD es su subsistema de Recuperación que detecta fallos y restaura la BD al estado anterior a la ocurrencia de un fallo.

Índice

- Introducción
- Estructura del almacenamiento y sus operaciones
- Problemas asociados a fallos
- Transacciones
- Recuperación ante fallos
  - Introducción
  - Tipos de fallos
  - Log
- Concurrencia

Tipos de Fallos

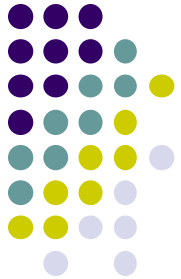


- Fallos de medios de almacenamiento: *caídas duras*. Ejemplo: Aterrizaje de cabezas del disco duro.
  - Hay que recurrir al almacenamiento estable.
- Fallos del sistema: *caídas suaves.* Ejemplo: Se va la luz. Estos fallos no dañan físicamente la BD.
- Errores de software: puede hacer que la BD pierda la consistencia. Ejemplo: errores debidos a la concurrencia.

Índice

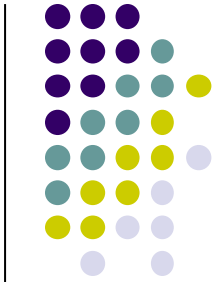
- Introducción
- Estructura del almacenamiento y sus operaciones
- Problemas asociados a fallos
- Transacciones
- Recuperación ante fallos
  - Introducción
  - Tipos de fallos
  - Log
- Concurrencia

Recuperación ante fallos: Log



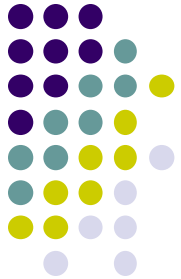
- El **registro histórico (LOG)** es una **secuencia de registros** y mantiene un registro de las actividades de actualización de la base de datos.
- Un registro histórico se mantiene en almacenamiento estable.

Índice



- Introducción
- Estructura del almacenamiento y sus operaciones
- Problemas asociados a fallos y concurrencia
- Transacciones
- Recuperación ante fallos
- Concurrencia
  - Introducción
  - Protocolos basados en bloqueos
  - Protocolo de bloqueo de 2 fases
  - Interbloqueo
  - Esquemas multiversión

Concurrencia



- Hablamos de concurrencia cuando dos o más transacciones acceden a los mismos datos en instantes de tiempo idénticos o próximos.

Concurrencia

Problema de pérdida de actualización

Correcto	Incorrecto
900	900
880	980

¿Cuál es el valor de X si inicialmente valía 1000?

T_1	T_2
Leer(X_1)	
	Leer(X_2)
$X_1=X_1-100$	
	$X_2=X_2-20$
Escribir(X_1)	
	Escribir(X_2)
commit	
	commit

Concurrencia

T_1	T_2
Leer(X_1)	
$X_1=X_1+100$	
Escribir(X_1)	
	Leer(X_2)
	$X_2=X_2*1.03$
	Escribir(X_2)
rollback	
	commit

Problema de lectura sucia

- Si X=100 y Y=100 inicialmente.

- Cuál es el resultado después de ejecutar las transacciones con esta planificación y cuál sería el correcto?

Correcto	Incorrecto
X	X
103	206

Concurrencia

T_1	T_2
	Leer(X_2)
Leer(X_1)	
$X_1=X_1+100$	
Escribir(X_1)	
commit	
	Leer(X_2)
	commit

Problema de lectura no repetible

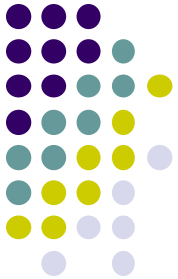
- Si X=100 y Y=100 inicialmente.

- Cuál es el resultado después de ejecutar las transacciones con esta planificación y cuál sería el correcto?

Correcto		Incorrecto	
X_1	X_2	X_1	X_2
100	100	100	200

Concurrencia

T_1	T_2
`select * from t`	
`where col=5`	
	`insert into t(col) values (5)`
	`commit`
`select * from t`	
`where col=5`	



Problema de lectura fantasma

- Los dos selects de T_1 dan resultados distintos

Definiciones

- **Planificación:** Una secuencia de las operaciones realizadas por un conjunto de transacciones concurrentes que preserva el orden de las operaciones en cada una de las transacciones individuales.
- **Planificación serie:** Una planificación en la que las operaciones de cada transacción se ejecutan consecutivamente sin que se entrelacen operaciones de otras transacciones.

Planificación

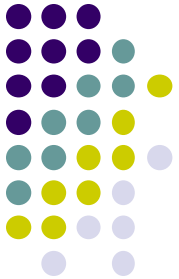
T_1	T_2
leer(A)	
$A := A - 50$	
	leer(A)
	$temp := A * 0.1$
	$A := A - temp$
	escribir(A)
	leer(B)
escribir(A)	
leer(B)	
$B := B + 50$	
escribir(B)	
	$B := B + temp$
	escribir(B)

Planificación

T_1	T_2
leer(A)	
$A := A - 50$	
escribir(A)	
leer(B)	
$B := B + 50$	
escribir(B)	
	leer(A)
	$temp := A * 0.1$
	$A := A - temp$
	escribir(A)
	leer(B)
	$B := B + temp$
	escribir(B)

Planificación serie

Serializable



- Una ejecución concurrente se considera correcta si y sólo si es serializable.
- Una ejecución de un conjunto de transacciones es serializable si y sólo si producen el mismo resultado que alguna planificación serie.
- Sin embargo, dos planificaciones serie no tienen porque dar el mismo resultado final.

Serializable

T_1	T_2
leer(A)	
$A := A - 50$	
	leer(A)
	$temp := A * 0.1$
	$A := A - temp$
	escribir(A)
	leer(B)
escribir(A)	
leer(B)	
$B := B + 50$	
escribir(B)	
	$B := B + temp$
	escribir(B)

T_1	T_2
leer(A)	
$A := A - 50$	
escribir(A)	
leer(B)	
$B := B + 50$	
escribir(B)	
	leer(A)
	$temp := A * 0.1$
	$A := A - temp$
	escribir(A)
	leer(B)
	$B := B + temp$
	escribir(B)

NO son equivalentes

La suma de (A+B) no se preserva

Serializable

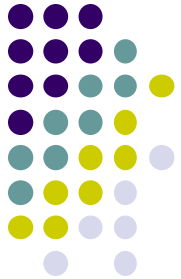
T_1	T_2
leer(A)	
$A := A - 50$	
escribir(A)	
leer(B)	
$B := B + 50$	
escribir(B)	
	leer(A)
	$temp := A * 0.1$
	$A := A - temp$
	escribir(A)
	leer(B)
	$B := B + temp$
	escribir(B)

T_1	T_2
leer(A)	
$A := A - 50$	
escribir(A)	
	leer(A)
	$temp := A * 0.1$
	$A := A - temp$
	escribir(A)
leer(B)	
$B := B + 50$	
escribir(B)	
	leer(B)
	$B := B + temp$
	escribir(B)

Son equivalentes

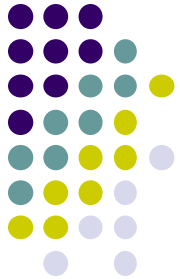
Observa que si ejecutas primero T2 y después T1, el resultado no es el mismo

Índice



- Introducción
- Estructura del almacenamiento y sus operaciones
- Problemas asociados a fallos y concurrencia
- Transacciones
- Recuperación ante fallos
- Concurrencia
  - Introducción
  - Protocolos basados en bloqueos
  - Protocolo de bloqueo de 2 fases
  - Interbloqueo
  - Esquemas multiversión

Protocolos basados en bloqueos



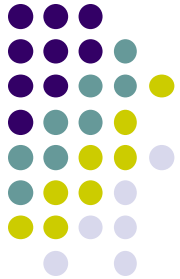
- Un bloqueo es un mecanismo para controlar el acceso concurrente a un elemento de datos
- Los elementos de datos se pueden bloquear de dos maneras:

  1. *modo exclusivo (X)*. El elemento de datos además de leerse se puede escribir. Un bloqueo de este tipo se solicita con la instrucción **bloquear-X**.

  2. *modo compartido (S)*. Los elementos de datos sólo se pueden leer. Un bloqueo de este tipo se solicita con la instrucción **bloquear-S**.

- Las solicitudes de bloqueo se dirigen al gestor de control de concurrencia. La transacción puede realizar la operación sólo después de que se conceda la solicitud.

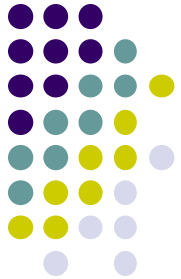
Protocolos basados en bloqueos



- A una transacción se le puede garantizar un bloqueo en un elemento si el bloqueo solicitado es compatible con los bloqueos que ya tengan otras transacciones sobre ese mismo elemento
- Cualquier número de transacciones puede tener bloqueos compartidos sobre un elemento,
  - pero si una de ellas tiene uno exclusivo sobre un determinado elemento, ninguna otra puede tener ningún otro bloqueo sobre dicho elemento.
- Si no se puede garantizar un bloqueo, la transacción que lo solicita tiene que esperar hasta que los bloqueos incompatibles que tienen otras transacciones se hayan liberado. A continuación se autoriza el bloqueo.

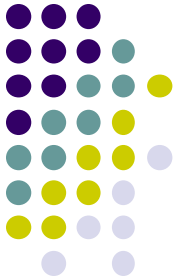
	S	X
S	Compatible	Incomp
X	Incomp	Incomp

Granularidad de los bloqueos



- Se puede bloquear:
- Atributo de una tupla.
- Tuplas.
- Tablas.
- Bloques de disco.

Protocolos basados en bloqueos

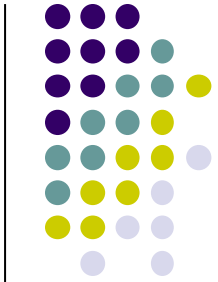


- Ejemplo de una transacción que realiza un bloqueo:

  T_2: **bloquear-S**(*A*)
  leer *(A)*;
  desbloquear(*A*)
  bloquear-S(*B*)
  leer *(B)*;
  desbloquear(*B*)
  visualizar*(A+B)*

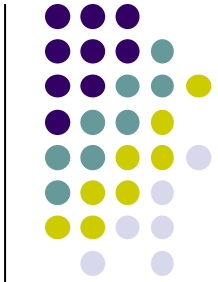
- El bloqueo anterior no es suficiente para garantizar la secuencialidad — si se actualizasen *A* y *B* entre la lectura de *A* y *B*, la suma mostrada sería errónea.
- Un **protocolo de bloqueo** es un conjunto de reglas que siguen todas las transacciones cuando se solicitan o se liberan bloqueos. Los protocolos de bloqueo restringen el número de planificaciones posibles.

Índice



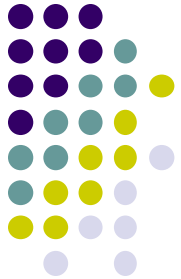
- Introducción
- Estructura del almacenamiento y sus operaciones
- Problemas asociados a fallos y concurrencia
- Transacciones
- Recuperación ante fallos
- Concurrencia
  - Introducción
  - Protocolos basados en bloqueos
  - Protocolo de bloqueo de 2 fases
  - Interbloqueo
  - Esquemas multiversión

El protocolo de bloqueo de dos fases



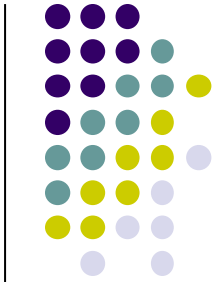
- Fase 1: Fase de crecimiento
  - las transacciones pueden conseguir bloqueos
  - las transacciones pueden aumentar bloqueos (de s a X).
  - las transacciones no pueden liberar bloqueos
- Fase 2: Fase de decrecimiento
  - las transacciones pueden liberar bloqueos
  - las transacciones pueden disminuir bloqueos (de X a s)
  - las transacciones no pueden conseguir bloqueos
- Esto garantiza la seriabilidad

El protocolo de bloqueo *riguroso* de dos fases



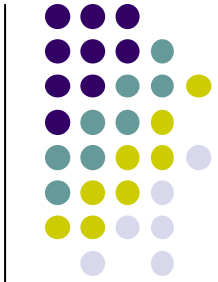
- Fase 1: Fase de crecimiento
  - las transacciones pueden conseguir bloqueos
  - las transacciones pueden aumentar bloqueos (de s a X).
  - las transacciones no pueden liberar bloqueos
- Fase 2: Fase de decrecimiento al terminar la transacción
  - las transacciones pueden liberar bloqueos

Protocolo de bloqueo *riguroso* de 2 fases



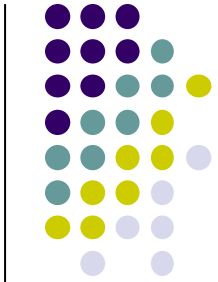
	T_1	T_2	
$s(X)$	**Leer**(X_1)	**Leer**(Y_2)	$s(Y)$
$s(Y)$	**Leer**(Y_1)	$Y_2=Y_2+100$	
	$Y_1=Y_1+X_1$		
$x(Y)$	**escribir**(Y, Y_1)	**Leer**(Z_2)	$s(Z)$
.	.	$Z_2=Z_2+Y_2$	
.	.	**Escribir**(Z_2)	$x(Z)$
.	.	**Fin**(T_2)	
$x(Y)$	**Escribir**(Y_1)		
	Fin(T_1)		

Índice



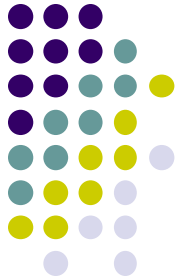
- Introducción
- Estructura del almacenamiento y sus operaciones
- Problemas asociados a fallos y concurrencia
- Transacciones
- Recuperación ante fallos
- Concurrencia
  - Introducción
  - Protocolos basados en bloqueos
  - Protocolo de bloqueo de 2 fases
  - Interbloqueo
  - Esquemas multiversión

Interbloqueo



	T_1	T_2	
s(X)	**Leer**(X_1)	**Leer**(Y_2)	*s(Y)*
s(Y)	**Leer**(Y_1)	$Y_2=Y_2+100$	
	$Y_1=Y_1+X_1$		
x(Y)	**Escribir**(Y_1)	**Leer**(X_2)	*s(X)*
.	.	$X_2=X_2+100$	
.	.	**Escribir**(X_2)	*x(X)*
.	.	.	.
	.	.	.
	.	.	.

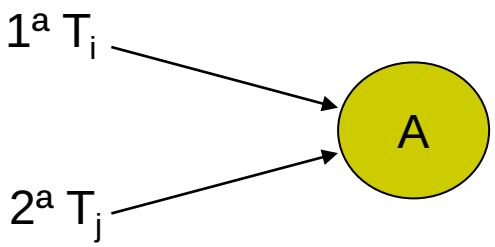
Prevención de interbloqueos



- Esquemas basados en límite de tiempo:
  - una transacción espera un bloqueo sólo durante un período de tiempo especificado. Después, la espera expira y se retrocede la transacción.
  - por lo que los interbloqueos no son posibles
  - fácil de implementar; pero existe la posibilidad de que aparezca la inanición. También es complicado determinar un buen valor para el intervalo de límite de tiempo.

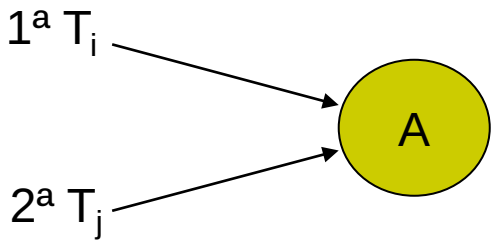
Prevención de interbloqueos

- Los siguientes esquemas utilizan marcas temporales de transacción sólo para la prevención de interbloqueos.



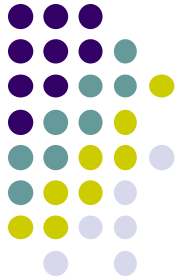
- esquema **esperar-morir**
  - Si la transacción que llega T_j es más antigua que la que ya está (T_i), la transacción que llega espera.
  - Si la transacción que llega T_j es más joven que la que está (T_i), T_j muere (Rollback) y libera los bloqueos.
  - T_j se vuelve a lanzar con el mismo timestamp

Prevención de interbloqueos



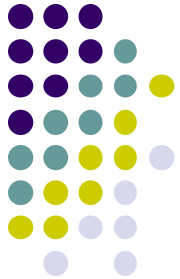
- esquema **herir-esperar**
  - Si la transacción que llega T_j es más joven espera.
  - Si la transacción que llega T_j es más antigua que la que ya está (T_i), hiere a la más reciente.
  - T_i se vuelve a lanzar con el mismo timestamp

Detección de interbloqueos



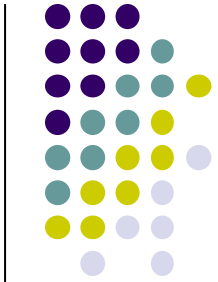
- Cuando se detecta un interbloqueo:
  - Se tendrán que retroceder algunas transacciones (selección de una víctima) para romper el interbloqueo. Se debe seleccionar como víctima aquella transacción que incurra en un coste mínimo.
  - Retroceso -- determinar hasta dónde se retrocederá dicha transacción
    - Retroceso total: Se aborta la transacción y luego vuelve a comenzar.
    - Sin embargo, es más efectivo retroceder la transacción sólo lo necesario para romper el interbloqueo.
  - Se produce la inanición cuando siempre se elige a la misma transacción como víctima. Para impedir la inanición se debe incluir en el factor de coste el número de retrocesos.

Índice



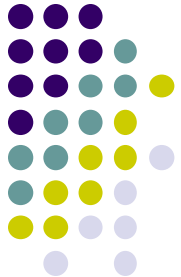
- Introducción
- Estructura del almacenamiento y sus operaciones
- Problemas asociados a fallos y concurrencia
- Transacciones
- Recuperación ante fallos
- Concurrencia
  - Introducción
  - Protocolos basados en bloqueos
  - Protocolo de bloqueo de 2 fases
  - Interbloqueo
  - Esquemas multiversión

Esquemas multiversión



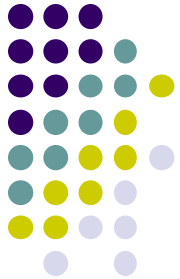
- El protocolo de 2 fases riguroso es demasiado restrictivo, y no permitiría demasiada concurrencia. Existen múltiples métodos de control de concurrencia, aquí veremos uno, los esquemas multiversión
- Los esquemas multiversión mantienen las versiones anteriores de los elementos de datos para aumentar la concurrencia.
- Cada **escribir(Q)** con éxito tiene como resultado la creación de una nueva versión de **Q**.
- Cuando se genera una operación **leer**(*Q*), se debe seleccionar la versión apropiada de *Q* de modo de que se asegure la seriabilidad.
- Las operaciones **leer** no tienen que esperar nunca ya que se devuelve la versión apropiada inmediatamente.
- Es crucial, por motivos de rendimiento, que una transacción sea capaz de determinar rápida y fácilmente la versión del elemento de datos que se va leer.

Esquemas multiversión



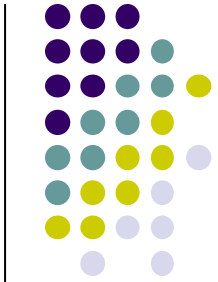
- La técnica más frecuente es la de marcas temporales.
- A cada transacción T_i se le asocia una única marca temporal denominada MT(T_j).
- Cada elemento de datos Q tiene una secuencia de versiones <Q_1, Q_2,...., Q_m>. Cada versión Q_k contiene tres campos de datos:
  - **contenido** -- es el valor de la versión Q_k.
  - **marca_temporal-E**(Q_k) -- es la marca temporal de la transacción que haya creado la versión Q_k
  - **marca_temporal-L**(Q_k) es la mayor marca temporal de todas las transacciones que hayan leído con éxito la versión Q_k

Esquemas multiversión



- cuando una transacción T_i crea una versión nueva Q_k de Q, la marca_temporal-E y la marca_temporal-L de Q_k se inician con el valor MT(T_i).
- La marca_temporal-L de Q_k se actualiza cada vez que una transacción T_j lee Q_k, y MT(T_j) > marca_temporal-L (Q_k).

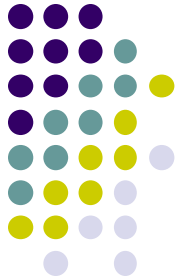
Esquemas Multiversión



Ordenación por marcas temporales

- Supóngase que la transacción T_i genera una operación **leer**(Q) o **escribir**(Q). Sea Q_k la versión de Q cuya marca temporal de escritura es la mayor marca temporal menor o igual que MT(T_i).
  1. Si la transacción T_i ejecuta **leer**(Q) entonces el valor que se devuelve es el contenido de la versión Q_k.
  2. Si la transacción T_i ejecuta **escribir**(Q)
     1. si MT(T_i) < marca_temporal-L(Q_k) entonces la transacción T_i se retrocede.
     2. si MT(T_i) = marca_temporal-E(Q_k) se sobrescribe el contenido de Q_k,
     3. en otro caso se crea una nueva versión de Q.
- Obsérvese que:
  - Las lecturas siempre tienen éxito
  - Se rechaza una escritura de T_i si alguna otra transacción T_j que (en el orden de secuencialidad definido por los valores de la marca temporal) debiera leer la escritura de T_i ya ha leído una versión creada por una transacción anterior a T_i.
- El protocolo asegura la seriabilidad.

Esquemas multiversión Ejemplo



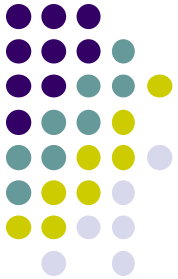
T_1: esc g, T_4: lee g, T_5 lee g, T_2 lee g

Versión g	**ML**	**ME**	**ML**	**ME**	**ML**	**ME**	**ML**	**ME**
Q_1	1	1	4	1	5	1	5	1
	T_1: esc		T_4: lee		T_5: lee		T_2: lee	

T_1: esc g, T_4: lee g, T_5 lee g, T_2 esc g

Versión g	**ML**	**ME**	**ML**	**ME**	**ML**	**ME**		
Q_1	1	1	4	1	5	1	Aborta!	
	T_1: esc		T_4: lee		T_5: lee		T_2: esc	

Esquemas multiversión Ejemplo



T_1: esc g, T_4: esc g, T_5 lee g, T_2 lee g

Versión g	**ML**	**ME**	**ML**	**ME**	**ML**	**ME**	**ML**	**ME**
Q_1	1	1	4	1	4	1	4	1
Q_2			4	4	5	4	5	4
	T_1: esc		T_4: esc		T_5: lee		T_2: lee	

Este método permite leer siempre.
Tiene dos problemas
-Que los conflictos se resuelven retrocediendo transacciones
-La lectura requiere actualizar la marca de lectura, lo que implica dos accesos potenciales a disco.

Muchos sistemas de SGBD reales usan una mezcla de esquemas multiversión con bloqueos.