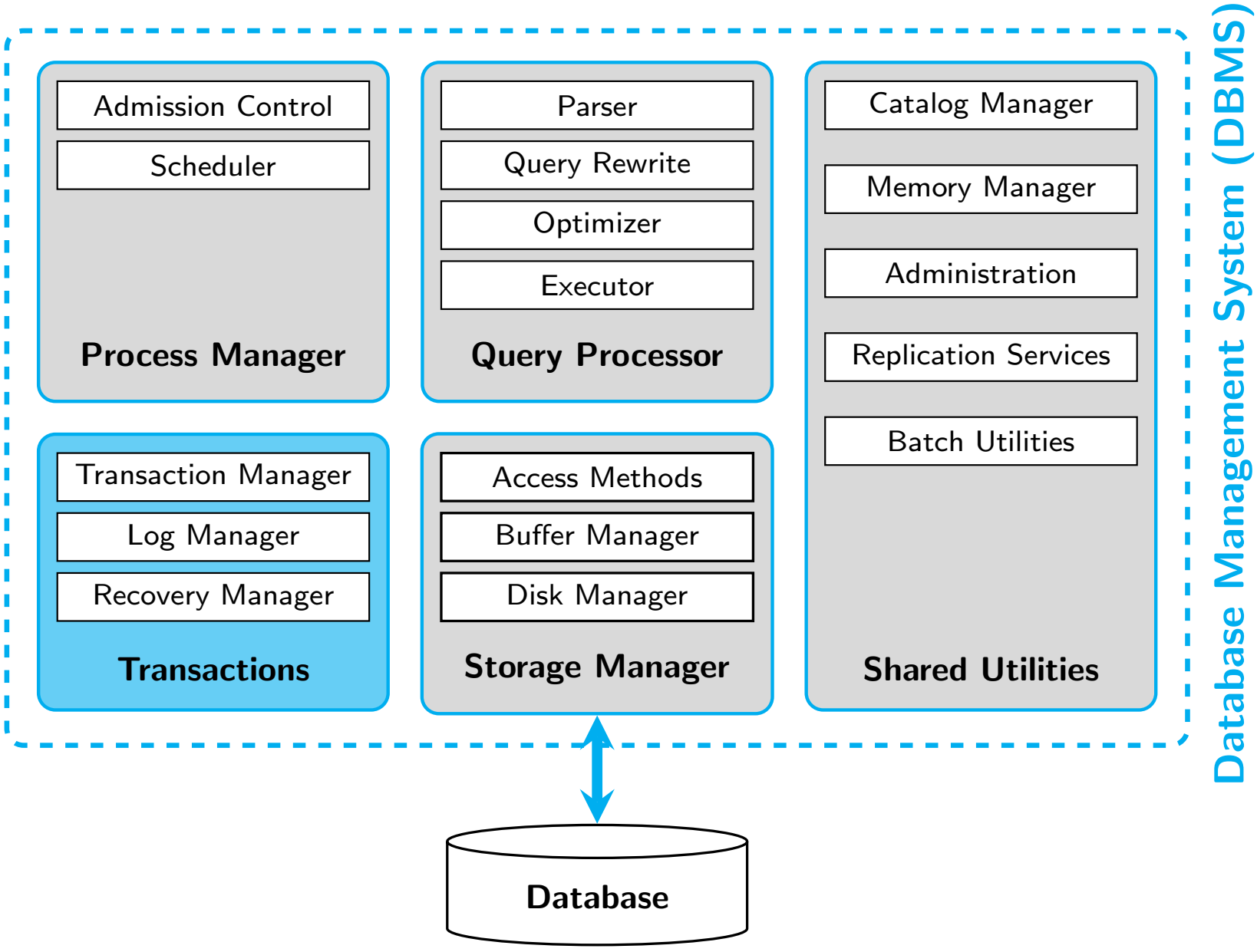
Gestión de transacciones

Clase 17

IIC 3413

Prof. Cristian Riveros

Gestión de transacciones



Acceso a los datos

Hasta el momento hemos supuesto:

1. un solo proceso accede los datos.
2. cada consulta o actualización es atómica.
3. no hay fallas en la administración de los datos.

¿qué tan razonables son estas suposiciones?

Ejemplo

Traspaso de fondos de cuenta 1 a cuenta 2.

```
UPDATE   Accounts
SET      balance = balance − v
WHERE    Aid = 1;

UPDATE   Accounts
SET      balance = balance + v
WHERE    Aid = 2;
```

Acceso a los datos

Ejemplo

Traspaso de fondos de cuenta 1 a cuenta 2.

- Yo traspaso 100 pesos de una cuenta a otra.
- Mi señora traspasa 200 pesos de una cuenta a otra.

Yo	**Mi señora**	B_1	B_2	
READ(B_1, x)		1000	1000	
WRITE$(B_1, x - 100)$		900		
READ(B_2, x)				
WRITE$(B_2, x + 100)$			1100	
	READ(B_1, y)			
	WRITE$(B_1, y - 200)$	700		
	READ(B_2, y)			
	WRITE$(B_2, y + 200)$	700	1300	✓

Acceso a los datos

Ejemplo

Traspaso de fondos de cuenta 1 a cuenta 2.

- Yo traspaso 100 pesos de una cuenta a otra.
- Mi señora traspasa 200 pesos de una cuenta a otra.

Yo	**Mi señora**	B_1	B_2	
READ(B_1, x)		1000	1000	
WRITE$(B_1, x - 100)$		900		
	READ(B_1, y)			
	WRITE$(B_1, y - 200)$	700		
	READ(B_2, y)			
	WRITE$(B_2, y + 200)$		1200	
READ(B_2, x)				
WRITE$(B_2, x + 100)$		700	1300	✓

Acceso a los datos

Ejemplo

Traspaso de fondos de cuenta 1 a cuenta 2.

- Yo traspaso 100 pesos de una cuenta a otra.
- Mi señora traspasa 200 pesos de una cuenta a otra.

Yo	**Mi señora**	B_1	B_2	
READ(B_1, x)		1000	1000	
WRITE$(B_1, x - 100)$		900		
READ(B_2, x)				
	READ(B_1, y)			
	WRITE$(B_1, y - 200)$	700		
	READ(B_2, y)			
	WRITE$(B_2, y + 200)$		1200	
WRITE$(B_2, x + 100)$		700	1100	✗

Acceso a los datos

Ejemplo

Traspaso de fondos de cuenta 1 a cuenta 2.

- Yo traspaso 100 pesos de una cuenta a otra.
- Mi señora traspasa 200 pesos de una cuenta a otra.

Yo	**Sistema**	B_1	B_2	
`READ`(B_1, x)		1000	1000	
`WRITE`$(B_1, x - 100)$		900		
	`ERROR`	900	1000	✕

Necesitamos transacciones

Definición

Una **transacción** es una secuencia de una o más operaciones que modifican o consultan la base de datos.

Ejemplo

- Transferencias de dinero entre cuentas de banco.
- Compra de un grupo de bienes.
- Registrarse para un curso.
- etc.

Transacciones en SQL

```
START    TRANSACTION

UPDATE   Accounts
SET      balance = balance − v
WHERE    Aid = 1;

UPDATE   Accounts
SET      balance = balance + v
WHERE    Aid = 2;

COMMIT
```

- `START TRANSACTION` y `COMMIT`
  nos permiten agrupar operaciones en una sola transacción.

Sobre transacciones

- Unos de los componentes fundamentales de una DBMS.
- Fundamental para muchas aplicaciones.
- Uno de los premios **Turing Award** en BD:

Charles Bachman	1973	primeros cimientos para DBMS.
Edgar Codd	1981	por inventar el modelo relacional.
Jim Gray	1998	por inventar las **transacciones**.
Michael Stonebracker	2015	por desarrollar Ingres.

Propiedades “acidas”

ACID = **A**tomicity
Consistency
Isolation
Durability

Propiedades “acidas”

Atomicity: Se ejecuta todos los pasos de una transacción o no se ejecuta nada.

Consistency: Al terminar una transacción los datos deben estar en un estado consistente.

Isolation: Cada transacción se ejecuta sin ser interferida por otras transacciones.

Durability: Si un transacción hace commit, sus cambios sobreviviran cualquier tipo de falla.

¿qué puede salir mal para mantener ACID?

1. Transacciones **concurrentes**.
   - Isolation.
   - De esto se encarga el **Transaction** Manager.

2. Recuperación a fallas.
   - Atomicity.
   - Durability.
   - De esto se encarga el **Log** Manager y el **Recovery** Manager.

¿quién asegura **Consistency**?

Problemas con transacciones concurrentes

1. Conflictos Write-Read (WR): lecturas sucias.

T_1	T_2	A	B	
READ(A, x)		1000	1000	
WRITE$(A, x - 100)$		900		
	READ(A, y)			
	WRITE$(A, y * 1.1)$	990		
	READ(B, y)			
	WRITE$(B, y * 1.1)$		1100	
READ(B, x)				
WRITE$(B, x + 100)$		990	1200	✗

Problemas con transacciones concurrentes

1. Conflictos Write-Read (WR): lecturas sucias.
2. Conflictos Read-Write (RW): lecturas irrepetibles.

T_1	T_2	A	
READ(A, x)		1	
IF$(x > 0)$			
	READ(A, y)		
	IF$(y > 0)$		
	WRITE$(A, y - 1)$	0	
	ENDIF		
WRITE$(A, x - 1)$		-1	
ENDIF		-1	✗

Problemas con transacciones concurrentes

1. Conflictos Write-Read (WR): lecturas sucias.
2. Conflictos Read-Write (RW): lecturas irrepetibles.
3. Conflictos Write-Write (WW): reescritura de datos temporales.

T_1	T_2	A	B	
`WRITE`$(A, 1000)$		1000		
`WRITE`$(B, 1000)$			1000	
	`WRITE`$(A, 2000)$	2000		
	`WRITE`$(B, 2000)$	2000	2000	✓

Problemas con transacciones concurrentes

1. Conflictos Write-Read (WR): lecturas sucias.
2. Conflictos Read-Write (RW): lecturas irrepetibles.
3. Conflictos Write-Write (WW): reescritura de datos temporales.

T_1	T_2	A	B	
	`WRITE`$(A, 2000)$	2000		
	`WRITE`$(B, 2000)$		2000	
`WRITE`$(A, 1000)$		1000		
`WRITE`$(B, 1000)$		1000	1000	✓

Problemas con transacciones concurrentes

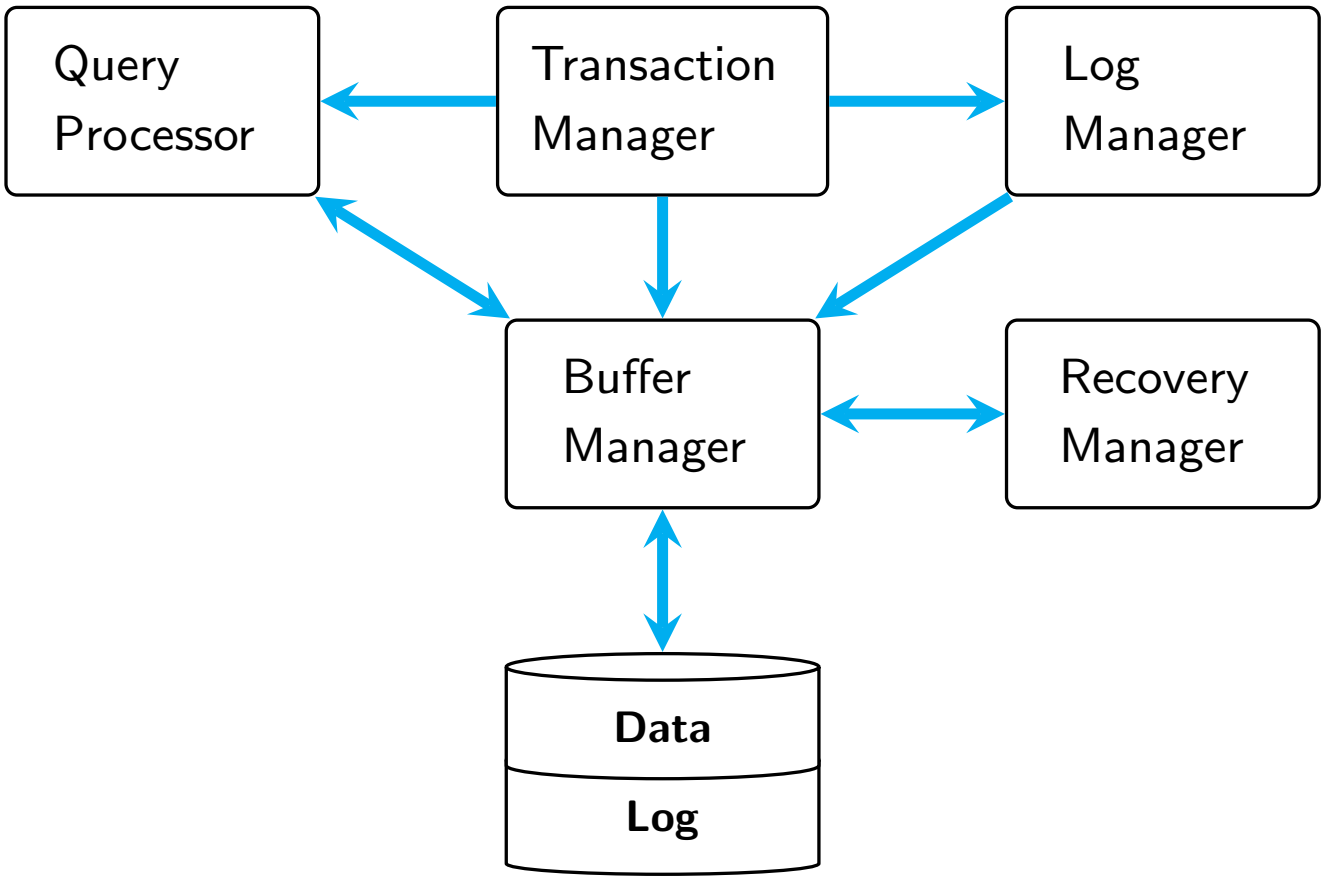
1. Conflictos Write-Read (WR): lecturas sucias.
2. Conflictos Read-Write (RW): lecturas irrepetibles.
3. Conflictos Write-Write (WW): reescritura de datos temporales.

T_1	T_2	A	B	
WRITE(A, 1000)		1000		
	WRITE(A, 2000)	2000		
	WRITE(B, 2000)		2000	
WRITE(B, 1000)		2000	1000	✗

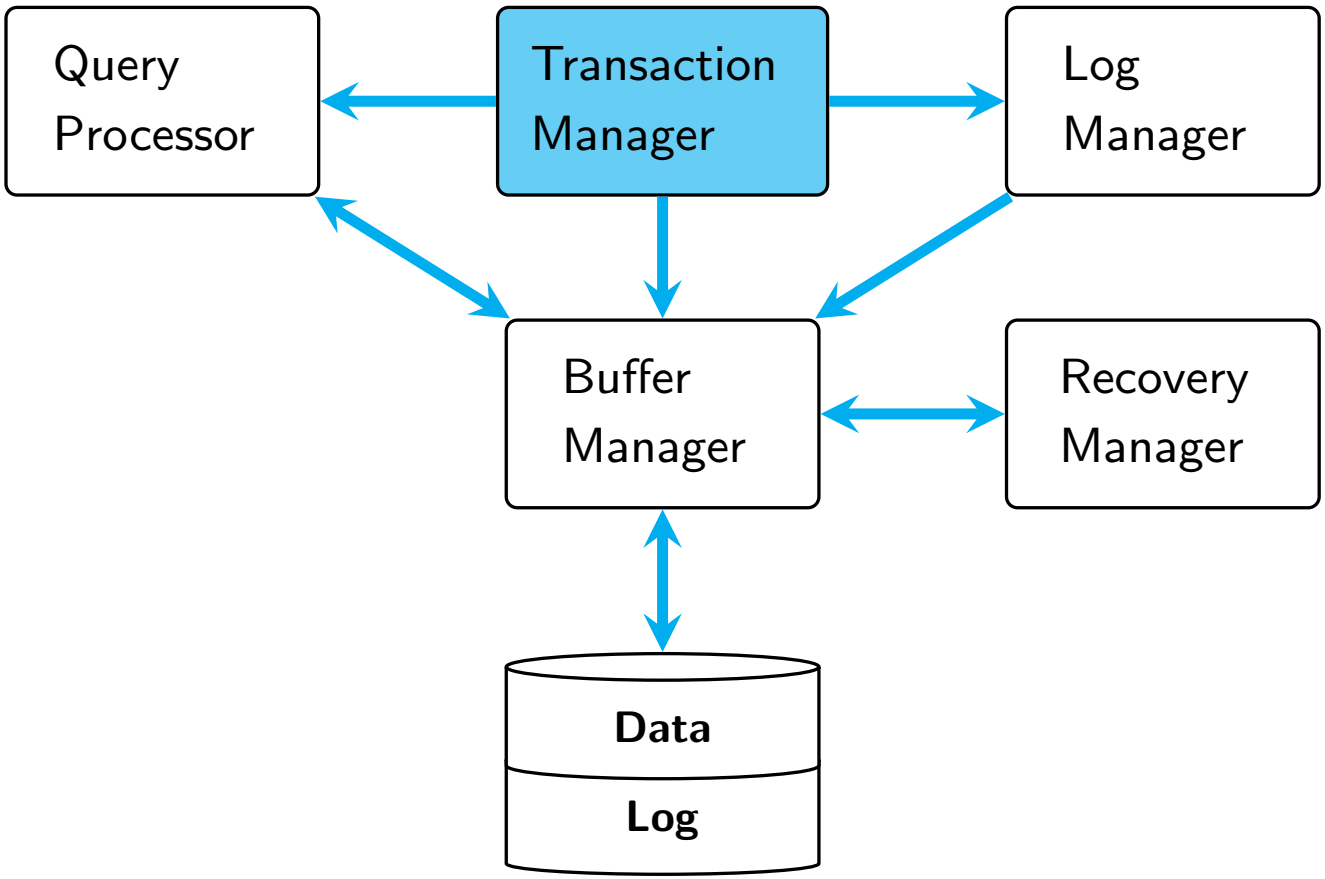
Posibles fallas en la ejecución

1. Datos erróneos.
   - Solución: Restricciones de integridad, data cleaning.
2. Fallas del disco duro.
   - Solución: RAID, copias redundantes.
3. Catástrofes.
   - Solución: Copias distribuidas.
4. Fallas del sistema.
   - Solución: **Log** manager, **Recovery** manager.

Zoom a la arquitectura de las transacciones



Zoom a la arquitectura de las transacciones



Outline

Transacciones

Transaction Manager

Outline

Transacciones

Definición

- Una **transacción** es una secuencia de 1 o más operaciones que modifican o consultan la bases de datos.

- Cada transacción esta compuesta por:
  - una secuencia de instrucciones.
  - un estado.

- **Estado** incluye posición actual en código y variables temporales.

Transacciones

Ejemplo

Acciones	x	B_1	B_2
		800	500
`READ`(B_1, x)	800		
$x := x - 100$	700		
`WRITE`(B_1, x)		700	
`READ`(B_2, x)	500		
$x := x * 3$	1500		
`WRITE`(B_2, x)			1500

Elementos de una BD

Definición

> *"Un elemento de una BD es un valor o conjunto de valores que puede ser accedido o modificado por una transacción."*

Un elemento puede ser (depende del DBMS):

- Atributo de una tupla.
- Tupla/record.
- Página.
- Relación.

¿qué granularidad de elemento debieramos preferir?

Estado de una BD

Definición

- El **estado** de una BD es el valor para cada uno de sus elementos.
- Un estado de una BD es **consistente** si satisface:
  - restricciones de integridad.
  - restricciones **implicitas**.

Principio de correctitud

"Si una transacción comienza con una BD en estado consistente y se ejecuta en ausencia de otras transacciones o errores de sistema, entonces la BD estará en un estado consistente cuando la transacción termine."

¿es razonable? ¿qué sucede con las restricciones **implicitas**?

Operaciones de una transacción

Transacciones interactuán con tres espacios de memoria:

- almacenamiento no-volátil (disco duro).
- memoria RAM (buffer manager).
- variables locales (estado de una transacción).

Operaciones de una transacción

Operaciones primitivas:

- `PIN`(X).
- `READ`(X, t).
- `WRITE`(X, t).
- `UNPIN`(X).
- `COMMIT`.
- `ABORT`.

donde X es un elemento de la BD y t es una variable local.

¿por qué podría una transacción hacer `ABORT`?

Operaciones de una transacción

Ejemplo (completo)

Acciones	x	B_1	B_2
`PIN`(B_1)		800	
`READ`(B_1, x)	800		
$x := x - 100$	700		
`WRITE`(B_1, x)		700	
`UNPIN`(B_1)			
`PIN`(B_2)			500
`READ`(B_2, x)	500		
$x := x * 3$	1500		
`WRITE`(B_2, x)			1500
`UNPIN`(B_2)			

Outline

Transaction Manager

"Propiedad isolation asegura que, si bien las acciones de varias transacciones pueden ser intercaladas, el resultado final es idéntico a ejecutar todas las transacciones una después de la otra en algún orden secuencial."

Transaction Manager es el encardo de asegurar isolation.

Concurrencia de transacciones

Definición

- Un **schedule** S es una secuencia de operaciones primitivas de una o más transacciones, tal que, para toda transacción T_i, las acciones de T_i aparecen en S en el mismo orden de su definición.

Concurrencia de transacciones

Ejemplo

Dos transacciones T_1 y T_2:

T_1	T_2
READ(A, x)	READ(A, y)
$x := x + 100$	$y := y * 2$
WRITE(A, x)	WRITE(A, y)
READ(B, x)	READ(B, y)
$x := x + 200$	$y := y * 3$
WRITE(B, x)	WRITE(B, y)

Importante: Para este análisis omitimos las acciones PIN y UNPIN.

Concurrencia de transacciones

Ejemplo

Un posible schedule para T_1 y T_2:

T_1	T_2
READ(A, x)	
$x := x + 100$	
WRITE(A, x)	
READ(B, x)	
$x := x + 200$	
WRITE(B, x)	
	READ(A, y)
	$y := y * 2$
	WRITE(A, y)
	READ(B, y)
	$y := y * 3$
	WRITE(B, y)

Concurrencia de transacciones

Ejemplo

Otro posible schedule para T_1 y T_2:

T_1	T_2
READ(A, x)	
$x := x + 100$	
WRITE(A, x)	
	READ(A, y)
	$y := y * 2$
	WRITE(A, y)
READ(B, x)	
$x := x + 200$	
WRITE(B, x)	
	READ(B, y)
	$y := y * 3$
	WRITE(B, y)

Concurrencia de transacciones

Definición

- Un **schedule** S es una secuencia de operaciones primitivas de una o más transacciones, tal que, para toda transacción T_i, las acciones de T_i aparecen en S en el mismo orden de su definición.

- Un schedule es **serial** si todas las acciones de cada transacción son ejecutas en grupo y no hay intercalación de acciones.

Concurrencia de transacciones

Ejemplo

¿és este schedule **serial** para T_1 y T_2?

T_1	T_2	A	B
READ(A, x)		100	50
$x := x + 100$			
WRITE(A, x)		200	
READ(B, x)			
$x := x + 200$			
WRITE(B, x)			250
	READ(A, y)		
	$y := y * 2$		
	WRITE(A, y)	400	
	READ(B, y)		
	$y := y * 3$		
	WRITE(B, y)	400	750

Concurrencia de transacciones

Ejemplo

¿és este otro schedule **serial** para T_1 y T_2?

T_1	T_2	A	B
	$\texttt{READ}(A, y)$	100	50
	$y := y * 2$		
	$\texttt{WRITE}(A, y)$	200	
	$\texttt{READ}(B, y)$		
	$y := y * 3$		
	$\texttt{WRITE}(B, y)$		150
$\texttt{READ}(A, x)$			
$x := x + 100$			
$\texttt{WRITE}(A, x)$		300	
$\texttt{READ}(B, x)$			
$x := x + 200$			
$\texttt{WRITE}(B, x)$		300	350

Concurrencia de transacciones

Ejemplo

¿y este otro?

T_1	T_2	A	B
READ(A, x)		100	50
$x := x + 100$			
WRITE(A, x)		200	
	READ(A, y)		
	$y := y * 2$		
	WRITE(A, y)	400	
READ(B, x)			
$x := x + 200$			
WRITE(B, x)			250
	READ(B, y)		
	$y := y * 3$		
	WRITE(B, y)	400	750

Concurrencia de transacciones

Definición

- Un **schedule** S es una secuencia de operaciones primitivas de una o más transacciones, tal que, para toda transacción T_i, las acciones de T_i aparecen en S en el mismo orden de su definición.

- Un schedule S es **serial** si todas las acciones de cada transacción son ejecutas en grupo y no hay intercalación de acciones.

- Un schedule S es **serializable** si existe algún serial schedule S' tal que el resultado de S y S' es el mismo para todo estado inicial de la BD.

Concurrencia de transacciones

Ejemplo

¿és este schedule **serializable** para T_1 y T_2?

T_1	T_2	A	B
READ(A, x)		100	50
$x := x + 100$			
WRITE(A, x)		200	
	READ(A, y)		
	$y := y * 2$		
	WRITE(A, y)	400	
READ(B, x)			
$x := x + 200$			
WRITE(B, x)			250
	READ(B, y)		
	$y := y * 3$		
	WRITE(B, y)	400	750

Concurrencia de transacciones

Ejemplo

¿y este otro? ¿es **serializable**?

T_1	T_2	A	B
READ(A, x)		100	50
$x := x + 100$			
WRITE(A, x)		200	
	READ(A, y)		
	$y := y * 2$		
	WRITE(A, y)	400	
	READ(B, y)		
	$y := y * 3$		
	WRITE(B, y)		150
READ(B, x)			
$x := x + 200$			
WRITE(B, x)		400	350

Concurrencia de transacciones

Definición

- Un **schedule** S es una secuencia de operaciones primitivas de una o más transacciones, tal que, para toda transacción T_i, las acciones de T_i aparecen en S en el mismo orden de su definición.
- Un schedule S es **serial** si todas las acciones de cada transacción son ejecutas en grupo y no hay intercalación de acciones.
- Un schedule S es **serializable** si existe algún serial schedule S' tal que el resultado de S y S' es el mismo para todo estado inicial de la BD.

La tarea del **Transaccion Manager**
es permitir solo schedules que son **serializables**!

¿qué tan posible es decidir serializabilidad?

Ejemplo

Este schedule NO es **serializable**:

T_1	T_2	A	B
READ(A, x)		100	50
$x := x + 100$			
WRITE(A, x)		200	
	READ(A, y)		
	$y := y * 2$		
	WRITE(A, y)	400	
	READ(B, y)		
	$y := y * 3$		
	WRITE(B, y)		150
READ(B, x)			
$x := x + 200$			
WRITE(B, x)		400	350

¿qué tan posible es decidir serializabilidad?

Ejemplo

¿y este schedule? ¿es **serializable**?

T_1	T_2	A	B
READ(A, x)		100	50
$x := x + 100$			
WRITE(A, x)		200	
	READ(A, y)		
	$y := y + 300$		
	WRITE(A, y)	500	
	READ(B, y)		
	$y := y + 400$		
	WRITE(B, y)		450
READ(B, x)			
$x := x + 200$			
WRITE(B, x)			650

¿qué tan posible es decidir serializabilidad?

- Es muy difícil verificar si un schedule es serilizable (indecidible?).
- Desde ahora, solo nos enfocaremos en los `READ` y `WRITE` de un objeto.

Simplificación

Solo consideramos como operaciones primitivas `READ` y `WRITE` de un objeto:

- $T_i : \texttt{READ}(X, t) \Rightarrow r_i(X)$.
- $T_i : \texttt{WRITE}(X, t) \Rightarrow w_i(X)$.

¿qué tan posible es decidir serializabilidad?

Ejemplo

S antes	S simplificado
T_1 : READ(A, x)	$r_1(A)$
T_1 : $x := x + 100$	
T_1 : WRITE(A, x)	$w_1(A)$
T_2 : READ(A, y)	$r_2(A)$
T_2 : $y := y * 2$	
T_2 : WRITE(A, y)	$w_2(A)$
T_2 : READ(B, y)	$r_2(B)$
T_2 : $y := y * 3$	
T_2 : WRITE(B, y)	$w_2(B)$
T_1 : READ(B, x)	$r_1(B)$
T_1 : $x := x + 200$	
T_1 : WRITE(B, x)	$w_1(B)$

$$S: \quad r_1(A)\ w_1(A)\ r_2(A)\ w_2(A)\ r_2(B)\ w_2(B)\ r_1(B)\ w_1(B)$$