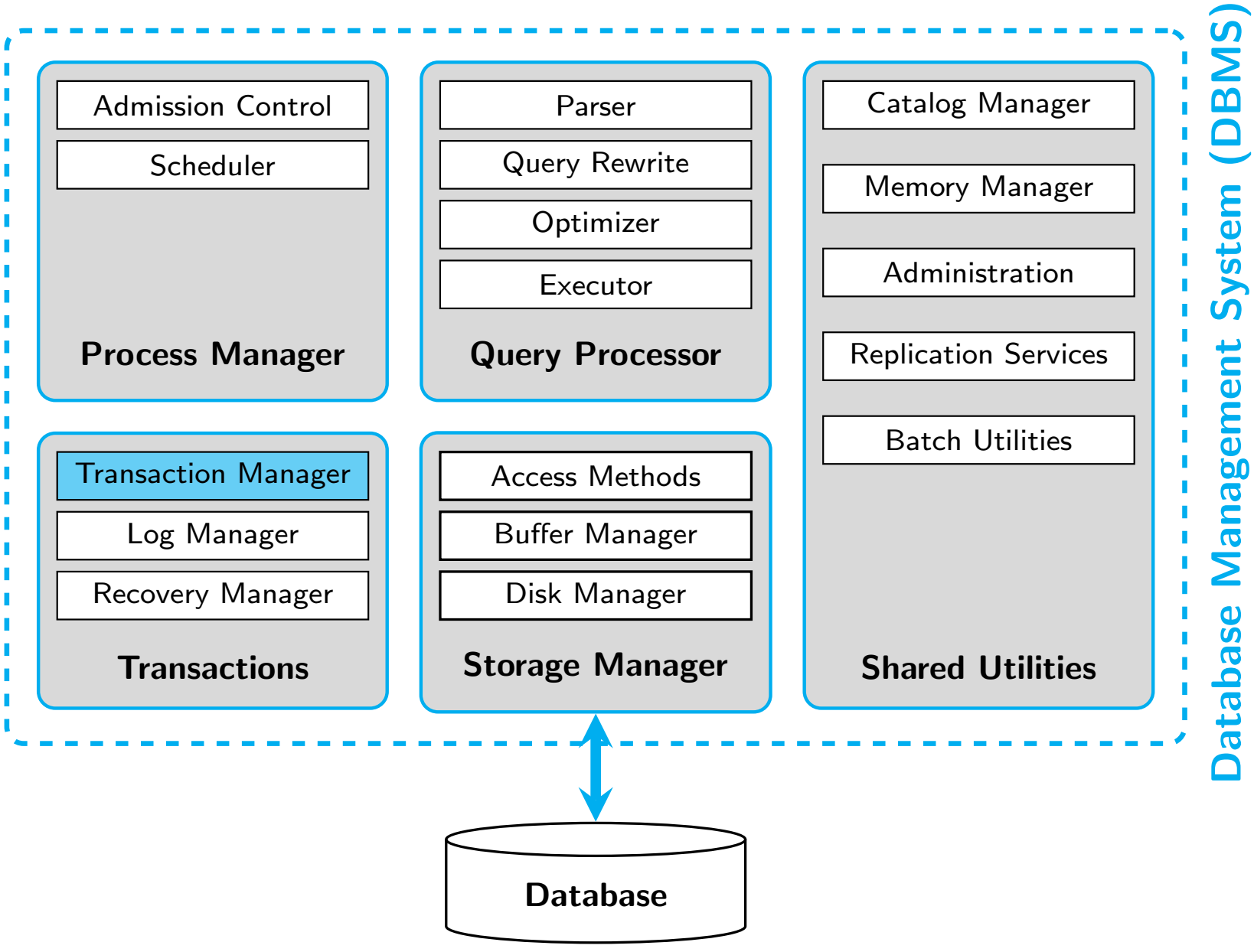
Control de concurrencia optimista

Clase 19

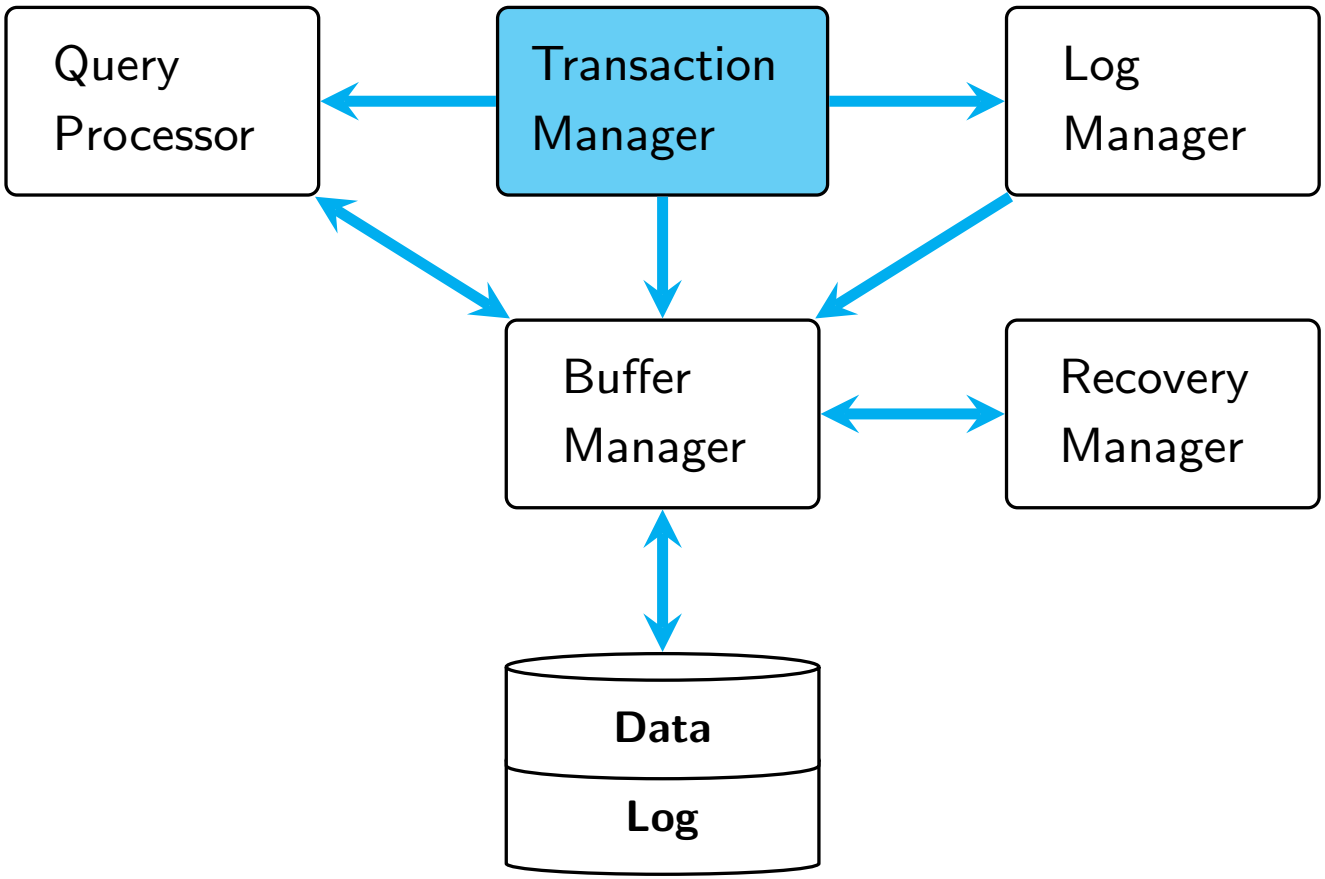
IIC 3413

Prof. Cristian Riveros

Control de concurrencia optimista



Zoom a la arquitectura de las transacciones



¿qué es control de concurrencia optimista?

Observaciones:

1. la mayoría de las transacciones son read-only.
2. la mayoría de las transacciones acceden datos disjuntos.

Suposiciones:

- Comportamiento no-serializable es menor.
- Abortar transacciones si resulta un schedule no-serializable.

Esto se conoce como control de concurrencia **optimista**.

Locking vs optimismo

- **Locking** previene comportamiento no-serializable.
- **Optimismo** permite ejecución hasta que schedule sea no-serializables.
  - En caso de llegar a un schedule no-serializable, usa `ROLLBACK`.
- **Locking** es eficiente para muchas transacciones de escritura.
- **Optimismo** es eficiente para muchas transacciones `READ-ONLY`.

Protocolos **optimista** de control de concurrencia

1. Timestamping.
2. Snapshot isolation.

Outline

Timestamps

Snapshot isolation

Outline

Timestamps

Snapshot isolation

Timestamps

Cada transacción **T** recibe un **timestamp** TS(**T**) que puede ser asignado:

- por clock del sistema.
- por un contador.

Transacciones que empiezan mas tarde tienen un timestamp mayor.

Invariante:

Orden de **timestamps** define el orden de serialización.

Orden de timestamps define el orden de serialización

Un schedule S de la forma:

- $\ldots\ w_i(X)\ \ldots\ r_j(X)\ \ldots$
- $\ldots\ r_i(X)\ \ldots\ w_j(X)\ \ldots$
- $\ldots\ w_i(X)\ \ldots\ w_j(X)\ \ldots$

es valido ssi $\mathrm{TS}(\mathbf{T}_i) < \mathrm{TS}(\mathbf{T}_j)$.

¿cómo verificamos que esta invariante se cumpla?

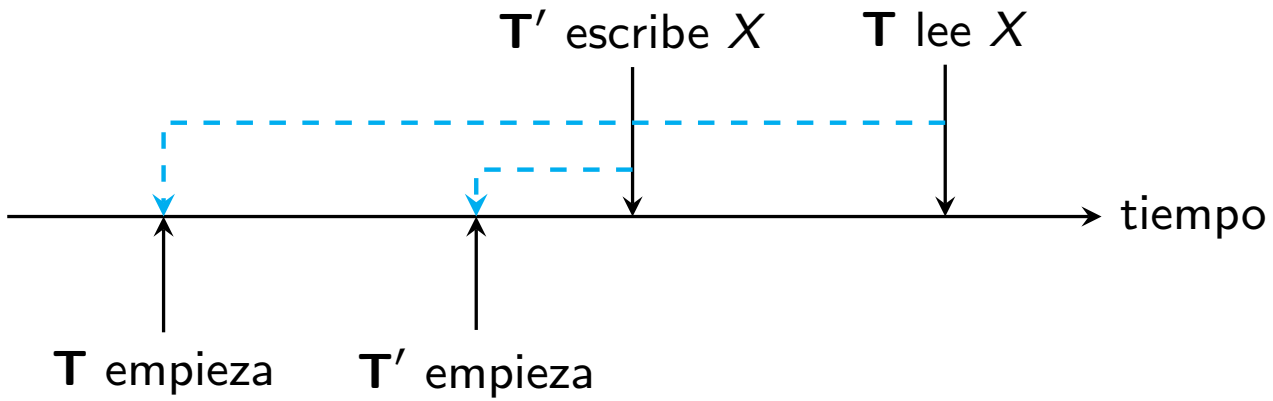
Timestamps y elementos de la BD

Para cada elemento X de la BD asociamos:

- $\mathrm{RT}(X)$: el timestamp mayor de todas las transacciones que han leído X (**read time** de X).
- $\mathrm{WT}(X)$: el timestamp mayor de todas las transacciones que han escrito X (**write time** de X).
- $\mathrm{C}(X)$: es 1 si, y solo si, la última transacción que escribió X hizo commit (**commit** de X).

¿qué puede salir mal?

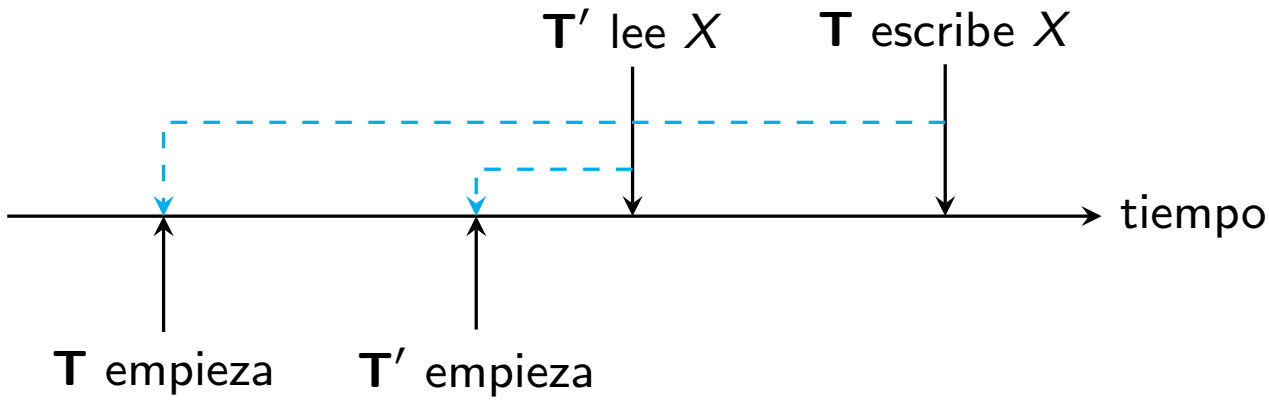
- **Lectura** tardía: **T** desea leer X pero $\mathrm{TS}(\mathbf{T}) < \mathrm{WT}(X)$. (R1)



Hacemos `ROLLBACK` de **T**.

¿qué puede salir mal?

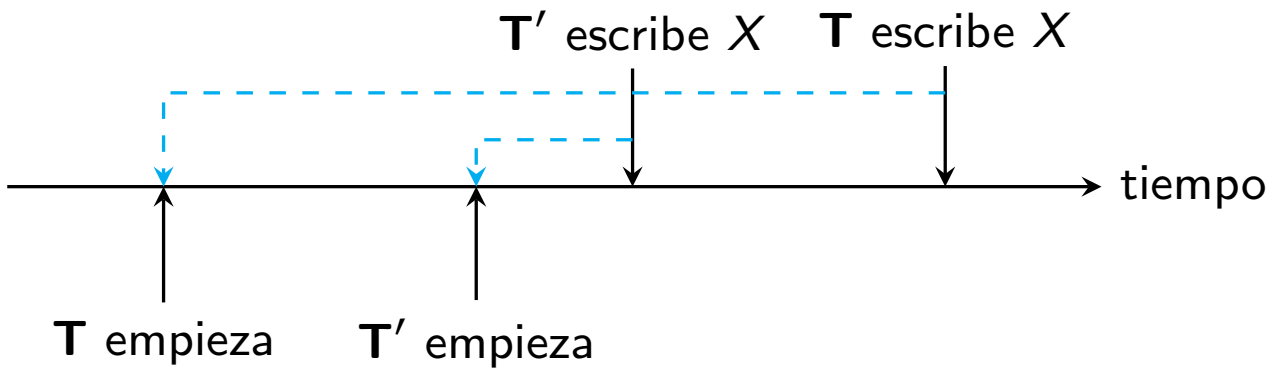
- **Lectura** tardía: **T** desea leer X pero $\mathrm{TS}(\mathbf{T}) < \mathrm{WT}(X)$. (R1)
- **Escritura** tardía: **T** desea escribir X
  - pero $\mathrm{TS}(\mathbf{T}) < \mathrm{RT}(X)$. (W1)



Hacemos `ROLLBACK` de **T**.

¿qué puede salir mal?

- **Lectura** tardía: $\mathbf{T}$ desea leer X pero $\mathrm{TS}(\mathbf{T}) < \mathrm{WT}(X)$. (R1)
- **Escritura** tardía: $\mathbf{T}$ desea escribir X
  - pero $\mathrm{TS}(\mathbf{T}) < \mathrm{RT}(X)$. (W1)
  - y $\mathrm{RT}(X) \leq \mathrm{TS}(\mathbf{T})$ pero $\mathrm{TS}(\mathbf{T}) < \mathrm{WT}(X)$. (W2)



No hacemos nada!! (*Thomas write rule*)

Control de concurrencia basado en Timestamps

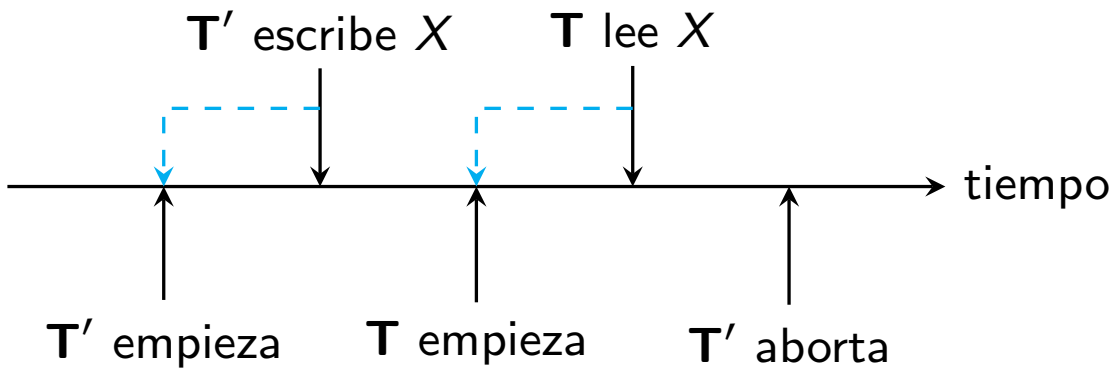
Versión 1.0

1. Transacción $\mathbf{T}$ quiere leer X.
   - Si $\mathrm{TS}(\mathbf{T}) < \mathrm{WT}(X)$, entonces abortamos $\mathbf{T}$. (R1)
   - Si no, $\mathbf{T}$ lee X y $\mathrm{RT}(X) := \max\{\mathrm{RT}(X), \mathrm{TS}(\mathbf{T})\}$.
2. Transacción $\mathbf{T}$ quiere escribir X.
   - Si $\mathrm{TS}(\mathbf{T}) < \mathrm{RT}(X)$, entonces abortamos $\mathbf{T}$. (W1)
   - Si $\mathrm{TS}(\mathbf{T}) < \mathrm{WT}(X)$, entonces ignoramos la escritura. (W2)
   - Si no, $\mathbf{T}$ escribe X y $\mathrm{WT}(X) := \mathrm{TS}(\mathbf{T})$.

¿algún bug en nuestra implementación?

Mas problemas: transacciones que abortan

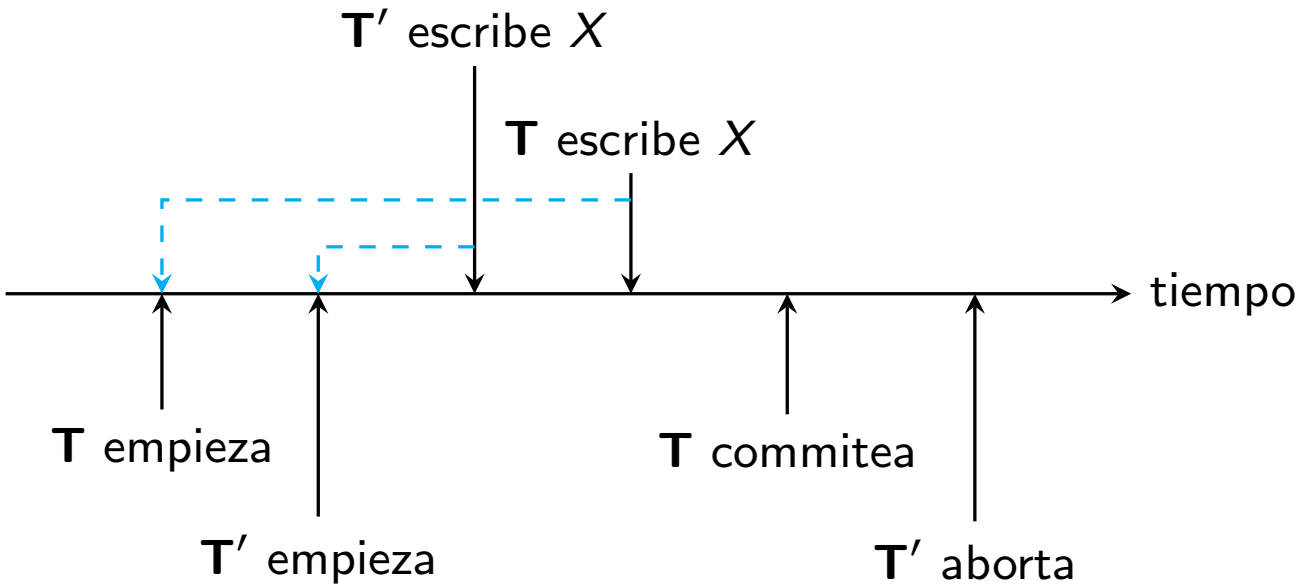
- **T** desea leer X y $\mathrm{WT}(X) < \mathrm{TS}(\mathbf{T})$, pero $\mathrm{C}(X) = 0$. (RC)



Debemos hacer que **T** espere hasta que $\mathrm{C}(X) = 1$.

Mas problemas: transacciones que abortan

- **T** desea leer X y $\mathrm{WT}(X) < \mathrm{TS}(\mathbf{T})$, pero $\mathrm{C}(X) = 0$. (RC)
- **T** desea escribir X y $\mathrm{TS}(\mathbf{T}) < \mathrm{WT}(X)$, pero $\mathrm{C}(X) = 0$. (WC)



Debemos hacer que **T** espere hasta que $\mathrm{C}(X) = 1$.

Control de concurrencia basado en Timestamps
Versión 2.0

Cuando **T** selecciona $r(x)$ o $w(x)$, el control de concurrencia decide si:

- otorgar el permiso de X a **T**.
- `ROLLBACK` de **T** (y empezar **T** con un nuevo timestamp).
- retrasar **T** hasta que $C(X) = 1$.

Control de concurrencia basado en Timestamps
Versión 2.0

Transacción **T** desea **leer** X:

1. Si $\mathrm{TS}(\mathbf{T}) \geq \mathrm{WT}(X)$:
   - Si $\mathrm{C}(X) = 1$: permitir la lectura de X.
   - Si $\mathrm{C}(X) = 0$: retrasar **T** hasta que $\mathrm{C}(X) = 1$. (RC)
2. Si $\mathrm{TS}(\mathbf{T}) < \mathrm{WT}(X)$: `ROLLBACK` de **T**. (R1)

Control de concurrencia basado en Timestamps

Versión 2.0

Transacción $\mathbf{T}$ desea **escribir** X:

1. Si $\mathrm{TS}(\mathbf{T}) \geq \mathrm{RT}(X)$ y $\mathrm{TS}(\mathbf{T}) \geq \mathrm{WT}(X)$:
   - Permitir la escritura de X y actualiza $\mathrm{C}(X) := 0$.
2. Si $\mathrm{TS}(\mathbf{T}) \geq \mathrm{RT}(X)$ pero $\mathrm{TS}(\mathbf{T}) < \mathrm{WT}(X)$:
   - Si $\mathrm{C}(X) = 1$: ignoramos la escritura de X. (W2)
   - Si $\mathrm{C}(X) = 0$: retrasar $\mathbf{T}$ hasta que $\mathrm{C}(X) = 1$. (WC)
3. Si $\mathrm{TS}(\mathbf{T}) < \mathrm{RT}(X)$: `ROLLBACK` de $\mathbf{T}$. (W1)

Control de concurrencia basado en Timestamps
Versión 2.0

Transacción **T** desea hacer `COMMIT`.

- buscar todos los elementos X modificados por **T** y
- actualizar $C(X) := 1$.

Transacción **T** desea hacer `ROLLBACK`.

- buscar todas las transacciones $\mathbf{T}'$ que están a la espera de **T**,
- otorgarles el permiso, y
- verificar si la acción de $\mathbf{T}'$ es válida para el schedule.

Control de concurrencia basado en Timestamps

Versión 2.0

Ejemplo

$\mathbf{T}_1$	$\mathbf{T}_2$	$\mathbf{T}_3$	A	B	C
200	150	175	RT = 0	RT = 0	RT = 0
			WT = 0	WT = 0	WT = 0
$r_1(B)$				RT = 200	
	$r_2(A)$		RT = 150		
		$r_3(C)$			RT = 175
$w_1(B)$				WT = 200	
$w_1(A)$			WT = 200		
COMMIT					
	$w_2(C)$				
	ABORT				
		$w_3(A)$			

Multiversión timestamps

Idea: mantener **múltiples versiones** de un elemento según su timestamp.

Para un elemento X y un tiempo t:

- X_t, X_{t-1}, X_{t-2}, ...
- $\mathrm{TS}(X_t)$, $\mathrm{TS}(X_{t-1})$, $\mathrm{TS}(X_{t-2})$, ...

¿para qué nos puede servir las múltiples versiones?

(ahorrarnos los `ROLLBACK` del tipo (R1))

Timestamps sin multiversión

Ejemplo

$\mathbf{T}_1$	$\mathbf{T}_2$	$\mathbf{T}_3$	$\mathbf{T}_4$	A
150	200	175	225	RT = 0
				WT = 0
$r_1(A)$				RT = 150
$w_1(A)$				WT = 150
	$r_2(A)$			RT = 200
	$w_2(A)$			WT = 200
		$r_3(A)$		
		ABORT		
			$r_4(A)$	RT = 225

Timestamps con multiversión

Ejemplo

$\mathbf{T}_1$	$\mathbf{T}_2$	$\mathbf{T}_3$	$\mathbf{T}_4$	A_0	A_{150}	A_{200}
150	200	175	225			
$r_1(A)$				READ		
$w_1(A)$					CREATE	
	$r_2(A)$				READ	
	$w_2(A)$					CREATE
		$r_3(A)$			READ	
			$r_4(A)$			READ

Control de concurrencia con múltiples versiones

Timestamps con multiversión es también conocido como:

Multiversion concurrency control (MVCC)

Outline

Snapshot isolation

- Un tipo de control de concurrencia optimista y **multiversión**.
- Muy eficiente y popular en los DBMS comerciales:
  - Oracle, SQL Server, PostgreSQL, ...
- Previene la anomalías clásicas, pero aún así **NO** es serializable.

Snapshot isolation: idea principal

Cada transacción utiliza, durante toda su ejecución,
la última versión disponible de los datos cuando empezó.

Cada transacción mira un **snapshot** o **copia** instantánea.

Snapshot isolation: formalización

Soporte:

- Cada transacción **T** recibe un timestamp TS(**T**).
- Mantenemos multiversiones de un mismo elemento.

Reglas:

1. Si **T** desea leer X, entonces **T** accede a la versión X_t donde:
   - t es un timestamp y
   - X_t es la última versión estable de X tal que $t < \text{TS}(\mathbf{T})$.
2. Si **T** desea escribir X, entonce funciona "*First commiter wins*".
   - la transacción que pierde hace `ROLLBACK`.

Snapshot isolation: formalización

Ejemplo

$\mathbf{T}_1$	$\mathbf{T}_2$	$\mathbf{T}_3$	$\mathbf{T}_4$
$r_1(X)$			
$w_1(X)$			
COMMIT			
	$w_2(X)$		
	ABORT		
		$r_3(X)$	
		$r_3(Y)$	
		$w_3(X)$	
			$r_4(X)$
			$r_4(Y)$
		$w_3(Y)$	
		$r_3(X)$	
		COMMIT	
			COMMIT

Snapshot isolation: formalización

Ejemplo (con versiones)

$\mathbf{T}_1$	$\mathbf{T}_2$	$\mathbf{T}_3$	$\mathbf{T}_4$
$r_1(X_0)$			
$w_1(X_1)$			
COMMIT			
	$w_2(X_2)$		
	ABORT		
		$r_3(X_1)$	
		$r_3(Y_0)$	
		$w_3(X_3)$	
			$r_4(X_1)$
			$r_4(Y_0)$
		$w_3(Y_3)$	
		$r_3(X_3)$	
		COMMIT	
			COMMIT

Snapshot isolation: formalización

Ejemplo ("*First commiter wins*")

$\mathbf{T}_1$	$\mathbf{T}_2$	$\mathbf{T}_3$
$r_1(X_0)$		
$w_1(X_1)$		
`COMMIT`		
	$r_2(X_1)$	
	$w_2(X_2)$	
		$r_3(X_1)$
		$w_3(X_3)$
	`COMMIT`	
		`ROLLBACK`
		$r_3(X_2)$
		$w_3(X_3)$
		`COMMIT`

Snapshot Isolation

Ventajas:

- Transacciones `READ-ONLY` nunca son retrasadas.
- Solo transacciones de escritura pueden ser abortadas.
- **Generalmente** tiene un comportamiento serializable.

¿qué anomalía NO-serializable podrían ocurrir?

Anomalías comunes

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

¿qué podría andar mal con **snapshot** isolation?

Anomalía write-skew

Ejemplo

Suponga dos balances, X e Y, tal que siempre se debe satisfacer $X + Y \geq 0$.

$\mathbf{T}_1$	$\mathbf{T}_2$
$r_1(X_0)$	
$r_1(Y_0)$	
	$r_2(X_0)$
	$r_2(Y_0)$
$w_1(X_1)$	
	$w_2(Y_2)$
`COMMIT`	
	`COMMIT`

Schedule es válido bajo snapshot isolation, pero NO es serializable.

Snapshot isolation: usuarios demasiado confiados

- SI apareció como una crítica al `ANSI SQL-92 standard`.
- Implementado por la mayoría de los motores de BD.
  - Ofrecido como `SNAPSHOT ISOLATION`.
- PostgreSQL y Oracle lo implementaban como `SERIALIZABLE`.
  - Actualmente como `SERIALIZABLE SNAPSHOT ISOLATION`.

OJO: muchos usuarios todavía creen que es serializable.

¿qué usa mi motor de BD?

- DB2: Strict 2PL.
- SQL Server:
  - Strict 2PL para `SERIALIZATION`.
  - MVCC para `SNAPSHOT ISOLATION`
- PostgreSQL:
  - Snapshot isolation mejorado para `SERIALIZATION`.
  - MVCC para `SNAPSHOT ISOLATION`
- Oracle:
  - Multiversion timestamps para `SERIALIZATION`.
  - MVCC para `SNAPSHOT ISOLATION`.

OJO: revisar esta información!