

A decorative graphic on the left side of the slide, consisting of a network of light blue lines and circles, resembling a circuit board or a neural network, set against a dark blue background.

CONTROL DE CONCURRENCIA

SEGUNDA PARTE

ÍNDICE

- Introducción
 - Lock binario, shared lock
 - **Two phase locking**
 - Control de concurrencia por timestamp
 - Control de concurrencia por multiversion
- Niveles Aislamiento en SQL



RETOMEMOS EL SIGUIENTE EJEMPLO

TRANSACCION 1

WRITE LOCK (X)

READ(X)

WRITE (X)

UNLOCK (X)

TRANSACCION 2

WRITE LOCK (X)

READ(X)

WRITE (X)

UNLOCK (X)

READ LOCK (X)

READ(X)

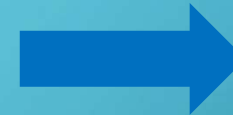
UNLOCK (X)

LOCK X

WRITE LOCK

WRITE LOCK

~~WRITE LOCK~~



RETOMEMOS EL SIGUIENTE EJEMPLO

TRANSACCION 1

WRITE LOCK (X)

READ(X)

WRITE (X)

UNLOCK (X)

READ LOCK (X)

READ(X)

UNLOCK (X)

TRANSACCION 2

WRITE LOCK (X)

READ(X)

WRITE (X)

UNLOCK (X)

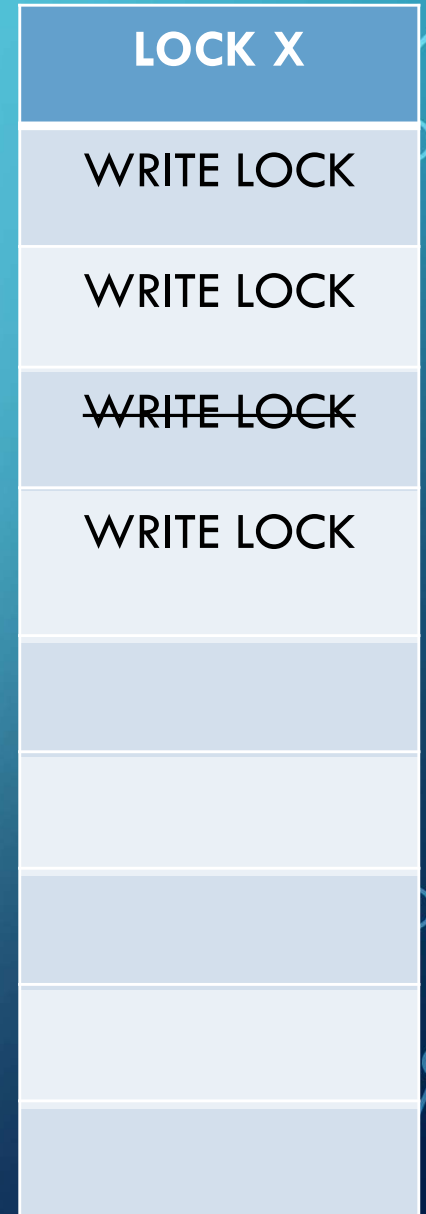
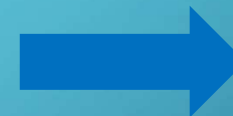
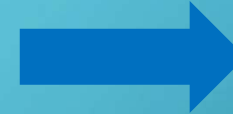
LOCK X

WRITE LOCK

WRITE LOCK

~~WRITE LOCK~~

WRITE LOCK



RETOMEMOS EL SIGUIENTE EJEMPLO

TRANSACCION 1

WRITE LOCK (X)

READ(X)

WRITE (X)

UNLOCK (X)

READ LOCK (X)

READ(X)

UNLOCK (X)

TRANSACCION 2

WRITE LOCK (X)

READ(X)

WRITE (X)

UNLOCK (X)

LOCK X

WRITE LOCK

WRITE LOCK

~~WRITE LOCK~~

WRITE LOCK

WRITE LOCK

~~WRITE LOCK~~



RETOMEMOS EL SIGUIENTE EJEMPLO

TRANSACCION 1

WRITE LOCK (X)

READ(X)

WRITE (X)

UNLOCK (X)

READ LOCK (X)

READ(X)

UNLOCK (X)

TRANSACCION 2

WRITE LOCK (X)

READ(X)

WRITE (X)

UNLOCK (X)

LOCK X

WRITE LOCK

WRITE LOCK

~~WRITE LOCK~~

WRITE LOCK

WRITE LOCK

~~WRITE LOCK~~

READ LOCK

RETOMEMOS EL SIGUIENTE EJEMPLO

TRANSACCION 1

WRITE LOCK (X)

READ(X)

WRITE (X)

UNLOCK (X)

READ LOCK (X)

READ(X)

UNLOCK (X)

TRANSACCION 2

WRITE LOCK (X)

READ(X)

WRITE (X)

UNLOCK (X)

LOCK X

WRITE LOCK

WRITE LOCK

~~WRITE LOCK~~

WRITE LOCK

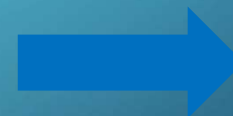
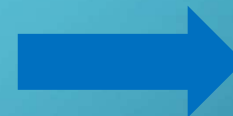
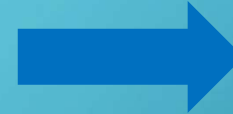
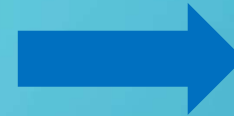
WRITE LOCK

~~WRITE LOCK~~

READ LOCK

READ LOCK

~~READ LOCK~~



RETOMEMOS EL SIGUIENTE EJEMPLO

TRANSACCION 1

WRITE LOCK (X)

READ(X)

WRITE (X)

UNLOCK (X)



El valor que lee no es el mismo que escribió

READ LOCK (X)

READ(X)

UNLOCK (X)

TRANSACCION 2

WRITE LOCK (X)

READ(X)

WRITE (X)

UNLOCK (X)

LOCK X

WRITE LOCK

WRITE LOCK

~~WRITE LOCK~~

WRITE LOCK

WRITE LOCK

~~WRITE LOCK~~

READ LOCK

READ LOCK

~~READ LOCK~~

TWO PHASE LOCKING -INTRO

- Es un mecanismo muy simple, en el cual cada data ítem tiene un lock que puede estar o no activo
- La notación es la misma que la utilizada para el shared lock
- Recibe este nombre porque tiene una fase de “crecimiento” (en la cual se adquieren locks) y una de “decrecimiento” (donde sólo se liberan)
- Resuelve el problema del ejemplo anterior



RETOMEMOS EL SIGUIENTE EJEMPLO

TRANSACCION 1

WRITE LOCK (X)

READ(X)

WRITE (X)

UNLOCK (X)

READ LOCK (X)

READ(X)

UNLOCK (X)

TRANSACCION 2

WRITE LOCK (X)

READ(X)

WRITE (X)

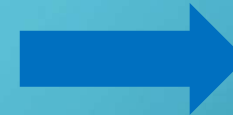
UNLOCK (X)

LOCK X

WRITE LOCK

WRITE LOCK

~~WRITE LOCK~~



RETOMEMOS EL SIGUIENTE EJEMPLO

TRANSACCION 1

WRITE LOCK (X)

READ(X)

WRITE (X)

UNLOCK (X)

READ LOCK (X)

READ(X)

UNLOCK (X)

TRANSACCION 2

WRITE LOCK (X)

READ(X)

WRITE (X)

UNLOCK (X)

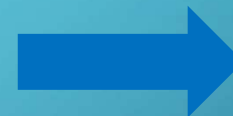
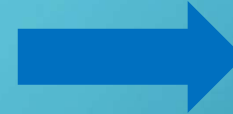
LOCK X

WRITE LOCK

WRITE LOCK

~~WRITE LOCK~~

WRITE LOCK



RETOMEMOS EL SIGUIENTE EJEMPLO

TRANSACCION 1

WRITE LOCK (X)

READ(X)

WRITE (X)

UNLOCK (X)

READ LOCK (X)

READ(X)

UNLOCK (X)

TRANSACCION 2

WRITE LOCK (X)

READ(X)

WRITE (X)

UNLOCK (X)

LOCK X

WRITE LOCK

WRITE LOCK

~~WRITE LOCK~~

WRITE LOCK

WRITE LOCK

~~WRITE LOCK~~



RETOMEMOS EL EJEMPLO

TRANSACCION 1

WRITE LOCK (X)

READ(X)

WRITE (X)

UNLOCK (X)

READ LOCK (X)

READ(X)

UNLOCK (X)

TRANSACCION 2

WRITE LOCK (X)

READ(X)

WRITE (X)

UNLOCK (X)

LOCK X

WRITE LOCK

WRITE LOCK

~~WRITE LOCK~~

WRITE LOCK

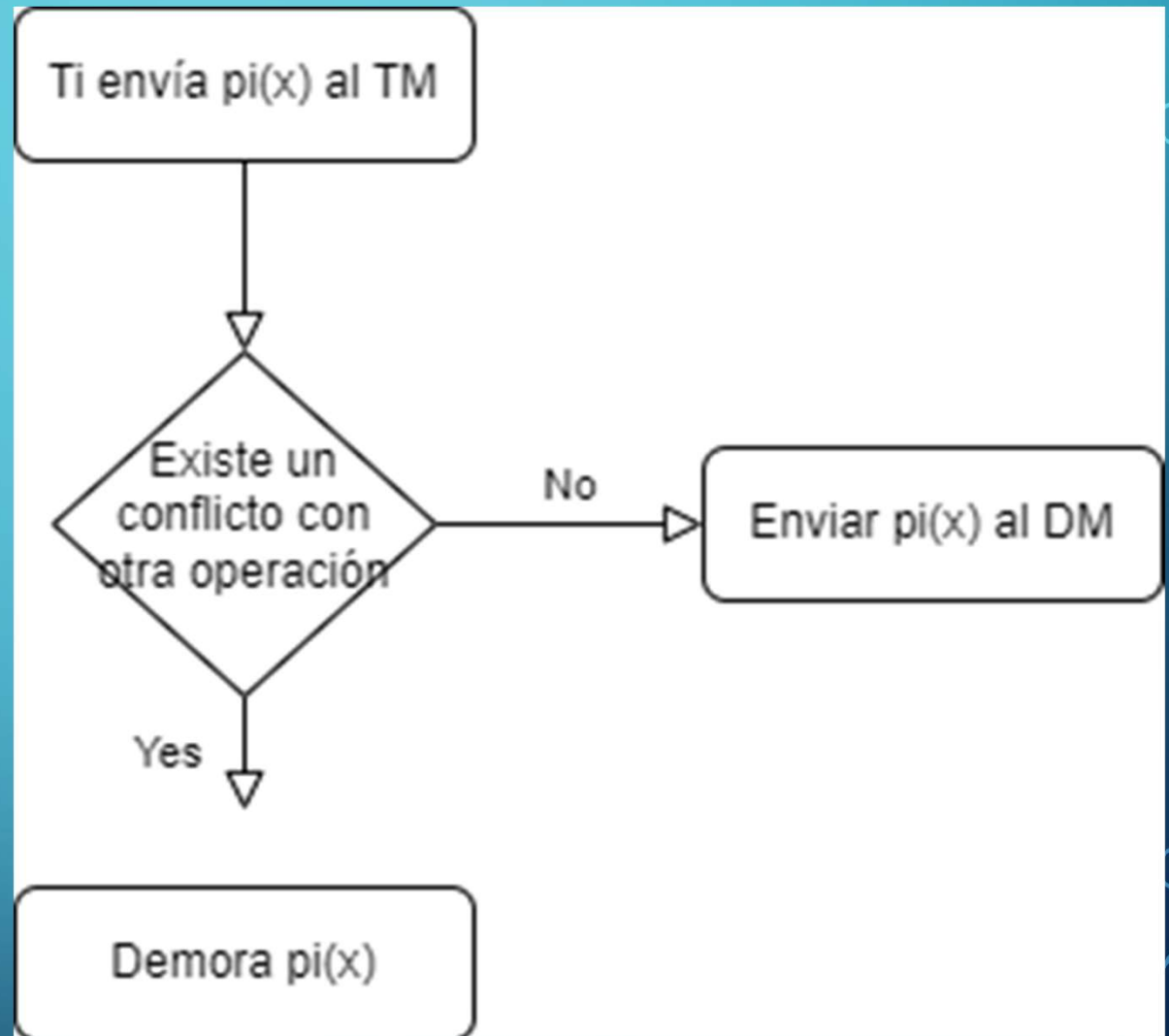
WRITE LOCK

~~WRITE LOCK~~

Operación Rechazada

TWO PHASE LOCKING (2 PL) - BÁSICO

- REGLA 1



TWO PHASE LOCKING (2 PL) - BÁSICO

- REGLA 2

Una vez que el scheduler estableció un lock para T_i , por ejemplo $pli[x]$, no lo va a liberar hasta que el DM informe que proceso la operación correspondiente al lock, $pi[x]$.

- REGLA 3

Una vez que el scheduler libero un lock de una transacción no va a adquirir ningún nuevo lock para esa transacción

TWO PHASE LOCKING - EJEMPLO

- La regla 3 es la que le da el nombre al mecanismo, dado que establece una fase de “crecimiento” (al tomar los locks) y una de “decrecimiento” (al liberar los locks)
- Tomemos el siguiente ejemplo
 - $H = r1(x), r1(x), u1(x), w12(y), w2(y), u2(y), r1(y), r1(y), u1(y), c2, c1$
 - ¿Cumple con las reglas del 2PL?

TWO PHASE LOCKING - EJEMPLO

- La regla 3 es la que le da el nombre al mecanismo, dado que establece una fase de “crecimiento” (al tomar los locks) y una de “decrecimiento” (al liberar los locks)
- Tomemos el siguiente ejemplo
 - $H = r1(x), r1(x), \underline{u1(x)}, w12(y), w2(y), u2(y), \underline{r1(y)}, r1(y), u1(y), c2, c1$
 - ¿Cumple con las reglas del 2PL?
 - NO
- Este tipo de mecanismos NO evitan el deadlock. ¿Pueden pensar un ejemplo?

TWO PHASE LOCKING - DEADLOCK

- La existencia de deadlocks implica que se necesita una estrategia para tratarlos. Una posibilidad es establecer un timeout, ¿pero cuánto es un buen timeout?
- Otra posibilidad es construir un grafo para representar que operaciones “esperan” de otras. Este grafo se llama Wait For Graph (WFG). Se genera un nodo por cada transacción y un arco entre 2 nodos cuando una transacción espera a otra. Un deadlock se produce cuando hay un ciclo en ese grafo
- El Scheduler va a agregar un arco de T_i a T_j cada vez hay un lock de T_i esperando a que se libere un lock de T_j . El arco se libera cuando se libera el ultimo lock de T_j que espera T_i
- ¿Cuál es el WFG del ejemplo que propusieron?
- ¿En qué momento se efectúa el control de ciclos del grafo?
- ¿Cómo se elige a la víctima?

TWO PHASE LOCKING - CORRECTITUD

- Para que el algoritmo sea correcto es necesario probar que las historias que produce son serializables
- De la regla 1 de la definición de 2PL se deduce que
 - Proposición 1: Sea H es una historia producida por un 2PL. Si $oi(x)$ esta en $C(H)$, entonces $oli(x)$ y $ui(x)$ están en $C(H)$ y $oli(x) < oi(x) < ui(x)$
 - Proposición 2: Sea H una historia producida por un 2PL. Si $pi(x)$ y $qj(x)$ ($i < j$) son dos operaciones en conflicto en $C(H)$, entonces $ui(x) < qlj(x)$ o $uj(x) < qli(x)$
- De la regla 3 de la definición se deduce que
 - Proposición 3: Sea H es una historia producida por un 2PL. Si $pi(x)$ y $qi(y)$ están en $C(H)$, entonces $pli(x) < uqi(y)$

TWO PHASE LOCKING – CORRECTITUD 2

- Usando esto vamos a demostrar que el grafo que produce el 2LPL es acíclico
- 1) Si $T_i \rightarrow T_j$ está en $SG(H)$, entonces una de las operaciones de T_i en un data ítem, por ejemplo x , se ejecuto antes y entro en conflicto con una de las operaciones de T_j . Entonces T_i debe liberar el lock de x antes de que T_j pueda tomarlo.
- 2) Supongamos $T_i \rightarrow T_j \rightarrow T_k$ es un camino en $SG(H)$, del primer paso se deduce que T_i libero algún lock antes de que T_j lo tomara y lo mismo vale entre T_j y T_k . Por lo tanto por transitividad T_i libero un lock antes de que T_k tomara uno. Por inducción esto puede extenderse arbitrariamente
- 3) Supongamos que $SG(H)$ tiene un ciclo, digamos que $T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_n \rightarrow T_1$. Por el punto anterior esto implicaría que T_1 libero un lock antes de que T_1 tomara uno, lo que contradice la tercera regla del protocolo.
- Por lo tanto el grafo no puede tener ciclos y H es serializable

ÍNDICE

- Introducción
 - Lock binario, shared lock
 - Two phase locking
 - **Control de concurrencia por timestamp**
 - Control de concurrencia por multiversion
- Niveles Aislamiento en SQL

CONTROL DE CONCURRENCIA POR TIMESTAMP

- Este es un mecanismo de control de concurrencia que no lockea
- El TM le asigna a cada operación un timestamp. Todas las operaciones que componen esa transacción llevan el mismo timestamp
- Regla: Si $p_i(x)$ y $q_j(x)$ son dos operaciones en conflicto, entonces $p_i(x) < q_j(x)$ si y solo si $TS(T_i) < TS(T_j)$
- Una forma de implementar el control por Timestamp es que el scheduler envía la operación al DM a menos que esta llegue “muy tarde”.
- Una operación $q_j(x)$ “llega tarde” cuando el scheduler ya procesó $o_i(x)$, con $TS(T_i) < TS(T_j)$ y ambas operaciones están en conflicto. En este caso el scheduler aborta T_i y la relanza, con lo cual su timestamp será posterior.

TIMESTAMP - CONT

- Para poder verificar si una operación llega tarde , el scheduler mantiene para cada data ítem su max-r-scheduler y su max-w-scheduler.
- Al recibir una operación compara su timestamp con los “maxs” correspondientes para indicar si la misma puede continuar o no.
- Si su timestamp es menor que algunos de los que corresponden a operaciones en conflicto la operación es cancelada
- ¿Qué operaciones controlan con que max? ¿Por qué?

TIMESTAMP - CONT

- ¿Qué operaciones controlan con que max? ¿Por qué?

TIMESTAMP - CONT

- Que operaciones controlan con que max? Por que?

	Max-r-scheduler	Max-w-scheduler
read	NO	si
write	SI	SI

- Si la operación supera el control, el scheduler actualiza el TS correspondiente y programa la operación
- Qué pasa si luego de esto la operación cuyo max-w-scheduler use para un read aborta?

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X		
Y		
Z		

R1 (x)

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	1	
Y		
Z		

R1 (x)

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	1	
Y		
Z		

$R1(x)$
 $W1(x)$

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	1	1
Y		
Z		

$R1(x)$
 $W1(x)$

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	1	1
Y		
Z		

R1 (x)
W1(x)
R2(y)
W2(y)

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	1	1
Y	2	2
Z		

R1 (x)
W1(x)
R2(y)
W2(y)

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	1	1
Y	2	2
Z		

R1 (x)
W1(x)
R2(y)
W2(y)
R3(x)

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	3	1
Y	2	2
Z		

R1 (x)
W1(x)
R2(y)
W2(y)
R3(x)

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	3	1
Y	2	2
Z	3	3

R1 (x)
W1(x)
R2(y)
W2(y)
R3(x)
R3(z)
W3(z)

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	3	1
Y	2	2
Z	3	3

R1 (x)
W1(x)
R2(y)
W2(y)
R3(x)
R3(z)
W3(z)
R2(x)

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	3	1
Y	2	2
Z	3	3

R1 (x)
W1(x)
R2(y)
W2(y)
R3(x)
R3(z)
W3(z)
R2(x)
R2(z)

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp,
- Tengo 3 data ítems

	Max read	Max write
X	3	1
Y	2	2
Z	3	3

R1 (x)
W1(x)
R2(y)
W2(y)
R3(x)
R3(z)
W3(z)
R2(x)
R2(z)

TIMESTAMP – THOMAS WRITE RULE

- Supongamos que tenemos dos transacciones $T1$ y $T2$ con $TS(T1) < TS(T2)$, y supongamos que se produce la siguiente situación
 - $w2(x) \ w1(x)$
- ¿Este plan es válido para el control por timestamp?
- ¿Existe alguna solución?
- Una posibilidad es “ignorar” el $w1(x)$. ¿Por qué es válida esta posibilidad?
- La “Thomas Write Rule” establece :
 - Si $\text{max-w-scheduler}(x) = TS(Tj)$ y se recibe la operación $w_i(x)$ con $TS(Ti) < TS(Tj)$ no procesar $w_i(x)$, darla simplemente por “cumplida”

THOMAS WRITE RULE (CONT)

- Supongamos ahora que se produce la siguiente situación
 - $R2(x) \ w2(x) \ w1(x)$
- ¿En este plan puede aplicarse la TWR?
- Si, puede aplicarse, sin embargo, el resultado no es correcto dado que T2 lee antes de escribir y lo que escribe seguramente este influido por el valor de x. En este caso al no haber ejecutado $w1(x)$ previamente a esa lectura el resultado puede ser diferente.
- En realidad, el control por timestamp no permite que esta situación se produzca, dado que el $w1(x)$ va a ser rechazado por que “alguien más nuevo leyó x1 antes que el”

ÍNDICE

- Introducción
 - Lock binario, shared lock
 - Two phase locking
 - Control de concurrencia por timestamp
 - **Control de concurrencia por multiversion**
- Niveles Aislamiento en SQL

CONTROL DE CONCURRENCIA POR MULTIVERSIÓN

- La idea básica que está detrás de este mecanismo de control de concurrencia es la de darle a cada transacción la versión de los data ítems que estaban vigentes al momento de su inicio.
- ¿Cuántas versiones es necesario guardar? En principio tantas como sean necesarias para atender a la transacción “ más vieja” vigente. Solo se guardan las versiones correspondientes a transacciones activas o comiteadas (¿si una transacción abortó no es necesario guardar su versión, por qué ?)

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp y que el valor inicial de cada data item es 0 y que cada transacción suma 1
- Tengo 3 data ítems

	Desde	Hasta	Valor
X	0		0
Y	0		0
Z	0		0

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp y que el valor inicial de cada data item es 0 y que cada transacción suma 1.
- Tengo 3 data ítems

$R1(x)$, lee 0
 $W1(x)$, escribe 1

	Desde	Hasta	Valor
X	0	1	0
X	1		1
Y	0		0
Z	0		0

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp y que el valor inicial de cada data item es 0 y que cada transacción suma 1.
- Tengo 3 data ítems

$R1(x)$, lee 0
 $W1(x)$, escribe 1
 $R2(y)$, lee 0
 $W2(y)$, escribe 1

	Desde	Hasta	Valor
X	0	1	0
X	1		1
Y	0	2	0
Y	2		1
Z	0		0

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp y que el valor inicial de cada data item es 0 y que cada transacción suma 1.
- Tengo 3 data ítems

$R1(x)$, lee 0
 $W1(x)$, escribe 1
 $R2(y)$, lee 0
 $W2(y)$, escribe 1
 $r3(x)$, lee 1

	Desde	Hasta	Valor
X	0	1	0
X	1		1
Y	0	2	0
Y	2		1
Z	0		0

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp y que el valor inicial de cada data item es 0 y que cada transacción suma 1.
- Tengo 3 data ítems

$R1(x)$, lee 0
 $W1(x)$, escribe 1
 $R2(y)$, lee 0
 $W2(y)$, escribe 1
 $r3(x)$, lee 1
 $R3(z)$, lee 0
 $W3(z)$, escribe 1

	Desde	Hasta	Valor
X	0	1	0
X	1		1
Y	0	2	0
Y	2		1
Z	0	3	0
Z	3		1

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp y que el valor inicial de cada data item es 0 y que cada transacción suma 1.
- Tengo 3 data ítems

$R1(x)$, lee 0
 $W1(x)$, escribe 1
 $R2(y)$, lee 0
 $W2(y)$, escribe 1
 $r3(x)$, lee 1
 $R3(z)$, lee 0
 $W3(z)$, escribe 1
 $r2(x)$, lee 1

	Desde	Hasta	Valor
X	0	1	0
X	1		1
Y	0	2	0
Y	2		1
Z	0	3	0
Z	3		1

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp y que el valor inicial de cada data item es 0 y que cada transacción suma 1.
- Tengo 3 data ítems

$R1(x)$, lee 0
 $W1(x)$, escribe 1
 $R2(y)$, lee 0
 $W2(y)$, escribe 1
 $r3(x)$, lee 1
 $R3(z)$, lee 0
 $W3(z)$, escribe 1
 $r2(x)$, lee 1
 $r2(z)$,

	Desde	Hasta	Valor
X	0	1	0
X	1		1
Y	0	2	0
Y	2		1
Z	0	3	0
Z	3		1

VEAMOS UN EJEMPLO

- $R1(x), w1(x), r2(y), w2(y), r3(x), r3(z), w3(z), r2(x), r2(z)$
- Por simplicidad supongamos que el subíndice de cada transacción se corresponde con su timestamp y que el valor inicial de cada data item es 0 y que cada transacción suma 1.
- Tengo 3 data ítems

$R1(x)$, lee 0

$W1(x)$, escribe 1

$R2(y)$, lee 0

$W2(y)$, escribe 1

$r3(x)$, lee 1

$R3(z)$, lee 0

$W3(z)$, escribe 1

$r2(x)$, lee 1

$r2(z)$, lee 0 y puede procesarlo

	Desde	Hasta	Valor
X	0	1	0
X	1		1
Y	0	2	0
Y	2		1
Z	0	3	0
Z	3		1

CONTROL DE CONCURRENCIA POR MULTIVERSIÓN -CONT

- Sea $\{T_0, T_1, T_2 \dots T_N\}$ un conjunto de transacciones, para procesar las operaciones de este conjunto el scheduler necesita transformarlas en operaciones sobre versiones.
- Es necesario definir una función h tal que
- $h(w_i(x)) = w_i(x_i)$
- $h(r_i(x)) = r_i(x_j)$, donde T_j es la última transacción no abortada que escribió x .
- $h(a_j) = a_j$
- $h(c_j) = c_j$

MULTIVERSIÓN - CONT

- Una historia completa del tipo multiversión (MV) sobre H es un orden parcial con una relación de orden $<$ tal que
 - 1. $H = h(\bigcup_{i=1}^n T_i)$ para alguna función h ;
 - 2. para cada T_i y todas las operaciones p_i, q_i en T_i , si $p_i <_i q_i$, entonces $h(p_i) < h(q_i)$;
 - 3. si $h(r_i[x]) = r_i(x_i)$, entonces $w_i(x) < r_i(x)$;
 - 4. si $w_i[x] <_i r_i[x]$, entonces $h(r_i(x)) = r_i[x_i]$; y
 - 5. si $h(r_i[x]) = r_i(x_i)$, $i \neq j$, y $c_j \in H$, entonces $c_i < c_j$.

MULTIVERSIÓN

- ¿Cuándo se rechaza una operación?

OTRO PROBLEMA DE CONCURRENCIA

- Hay un problema más sofisticado
 - Phantom Problem

Transacción 1

Select count(*)

From ventas

Where monto > 10.000;

Transacción 2

Insert into ventas (
...,monto) values (
....., 15.000)

PHANTOM PROBLEM

- Si el Schedule es T2 , T1 no hay problema
- ¿Qué pasa si es T1, T2?
- ¿Cómo funcionan los mecanismos de control de concurrencia en este caso ?

PHANTOM PROBLEM

- Si el Schedule es T2 , T1 no hay problema
- ¿Qué pasa si es T1, T2?
- ¿Cómo funcionan los mecanismos de control de concurrencia en este caso ?
- Los mecanismos de control de concurrencia no pueden actuar sobre registros que no existen , de ahí el nombre de “phantom problema”

PHANTOM PROBLEM

- Para resolverlo se pueden usar diferentes alternativas
- Lockeo del predicado (no está implementado en la práctica, es NP-hard determinar si dos predicados son compatibles)
- Index locking
 - Lockea en forma compartida las “hojas” del índice que contiene los punteros a los datos que se leen
 - Permite lecturas, pero no borrados ni inserciones
- Si no se usa un índice para acceder se bloquean todas las hojas de un índice existente

REPASEMOS

	Share Lock	Timestamp	MultiVersion
Falsa modificación	1	1	1
Modificación Perdida	2	2	2
Falsa Sumarización	3	3	3

ÍNDICE

- Introducción
 - Lock binario, shared lock
 - Two phase locking
 - Control de concurrencia por timestamp
 - Control de concurrencia por multiversion
- **Niveles Aislamiento en SQL**

NIVELES DE AISLAMIENTO EN SQL

Isolation level	Lost updates	Dirty reads	Non-repeatable reads	Phantoms
Read Uncommitted	don't occur	may occur	may occur	may occur
Read Committed	don't occur	don't occur	may occur	may occur
Repeatable Read	don't occur	don't occur	don't occur	may occur
Serializable	don't occur	don't occur	don't occur	don't occur

CONTROL DE CONCURRENCIA

- Esta presentación fue armada utilizando, además de material propio, material de
 - "Concurrency Control and Recovery in Database Systems" de Bernstein, Hadzilacos y Goodman



CONTROL DE CONCURRENCIA

¡MUCHAS GRACIAS!