UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática etsinf

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Diseño y Gestión de Bases de Datos

Tema 4

Control de la concurrencia en bases de datos relacionales

Profesoras: Laura Mota y Pedro Valderas

Control de la concurrencia.

Objetivos:

- ✓ Estudiar la ejecución concurrente de transacciones: problemas y soluciones.
- ✓ Estudiar el tratamiento de este problema en SQL.
- ✓ Analizar la relación entre concurrencia y recuperación.

2

En los temas anteriores, en los que se ha estudiado la ejecución de transacciones en un entorno monousuario (Tema 2), se pretendía analizar lo que significa ejecutar correctamente una transacción y las técnicas que lo aseguran: técnicas de recuperación de transacciones (Tema 3).

En este tema, se va a estudiar la ejecución de transacciones en un entorno multiusuario, es decir, se va a estudiar la ejecución concurrente (intercalada) de transacciones y los problemas que esta nueva situación introduce.

Control de la concurrencia.

1 Ejecución concurrente de transacciones: anomalías.

2 Control de la concurrencia en SQL.

3 Planes serializables por conflictos.

4 Protocolos de bloqueo.

5 Protocolos de ordenamiento por marcas de tiempo.

6 Protocolos multiversión.

7 Concurrencia y recuperación.

1 Ejecución concurrente de transacciones: anomalías.

Las operaciones de acceso a una BD se organizan en transacciones

TRANSACCIÓN ➡ Secuencia de operaciones de acceso a la base de datos (consulta o actualización) que constituyen una unidad de ejecución.

Procesar correctamente una transacción significa:

(a) todas las operaciones de la transacción se ejecutan con éxito y sus cambios (actualizaciones) quedan grabados permanentemente en la base de datos (transacciones confirmadas)

o bien

(b) la transacción no tiene ningún efecto en la base de datos (transacciones anuladas o interrumpidas).

Principio ACID: **A**tomicidad, **C**onsistencia, **Aislamiento**, **P**ersistencia

4

Propiedad de **Aislamiento**: una transacción debe ejecutarse como si se ejecutase de forma aislada (en solitario), sobre un estado de la base de datos.

El cumplimiento estricto de esta propiedad significaría que no se puedan ejecutar simultáneamente varias transacciones sobre un estado de la BD.

Sin embargo, como el acceso concurrente es una de las (buenas) características de la tecnología de bases de datos (Tema 1), los SGBD deben permitir la ejecución concurrente (intercalada) de transacciones y al mismo tiempo cumplir con la propiedad de **aislamiento**.

Para ello, los SGBD implementan **protocolos** que admiten la concurrencia, pero **controlan** que el efecto de la ejecución intercalada de varias transacciones sea el mismo que el que se produciría si las transacciones se ejecutaran de forma aislada, es decir, una a continuación de otra (**en serie**).

1 Ejecución concurrente de transacciones: anomalías.

Ejecución concurrente de transacciones: ejecución intercalada de las operaciones de dos o más transacciones.

La ejecución concurrente de transacciones (no controlada) puede generar **anomalías**:

Resultado inesperado o no deseado de la ejecución concurrente de varias transacciones

- ✓ Anomalías de Lectura:
  - Lectura sucia.
  - Lectura no repetible.
  - Lectura de fantasmas.
- ✓ Anomalía de Escritura:
  - Pérdida de actualizaciones.

5

Se entiende por anomalía un resultado inesperado o no deseado de la ejecución concurrente de varias transacciones.

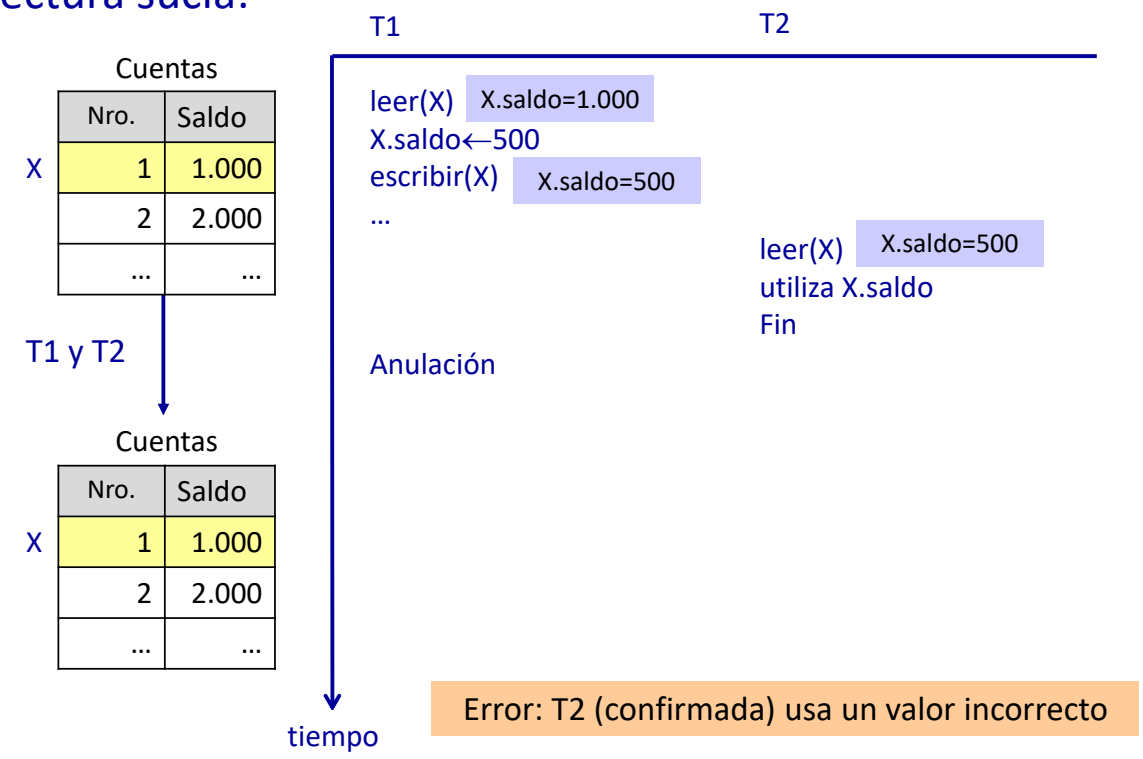
Las anomalías más frecuentes están tipificadas y reciben un nombre.

Si el efecto que se persigue, en la ejecución concurrente de transacciones, es el que se obtendría si las transacciones se ejecutaran de forma aislada, es decir si se ejecutase una transacción a continuación de otra (**en serie**), entonces una transacción debe ver siempre el mismo valor de los elementos de datos que no actualiza (el mismo estado) y debe ver sólo los cambios (actualizaciones) que ella realiza.

A continuación, se muestran ejemplos de estas cuatro anomalías.

1 Ejecución concurrente de transacciones: anomalías.

Lectura sucia:



6

¿Cuál es el problema?

T2 lee un elemento de datos actualizado por T1 que todavía no ha finalizado, sus actualizaciones no han sido confirmadas. La ejecución concurrente es correcta, cada transacción ve un único estado de la base de datos. Se respeta la propiedad de **aislamiento**.

El problema surge cuando T1 es anulada (lo que es imprevisible): T2 ha leído un valor del elemento de datos que nunca ha existido (ha sido anulado). Para **corregir** la anomalía, es decir, el efecto indeseado de la lectura sucia de T2, ésta deberá ser anulada después de la anulación de T1 (anulación en cascadas).

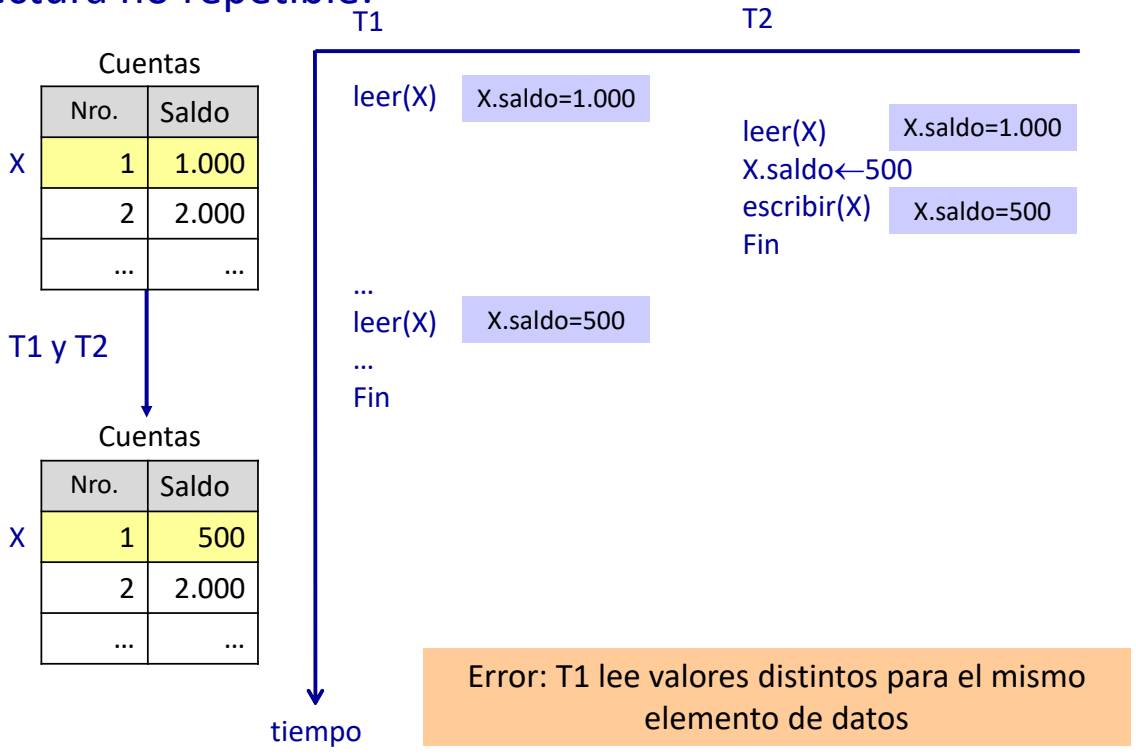
La forma de **evitar** la anomalía es permitir sólo planes concurrentes estrictos (punto 7), es decir, planes en los que no se permita que una transacción lea un dato actualizado por otra transacción hasta que ésta última se haya confirmado.

Es importante observar que, en la anulación en cascada, se puede anular una transacción que ya ha sido confirmada.

Esta anomalía de la "lectura sucia" se considera más bien una *seudoanomalía*, si se permite la lectura sucia se puede corregir su efecto indeseable, haciendo una anulación en cascada. Esta anomalía se estudiará con más detalle en el punto 7 del tema.

1 Ejecución concurrente de transacciones: anomalías.

Lectura no repetible:



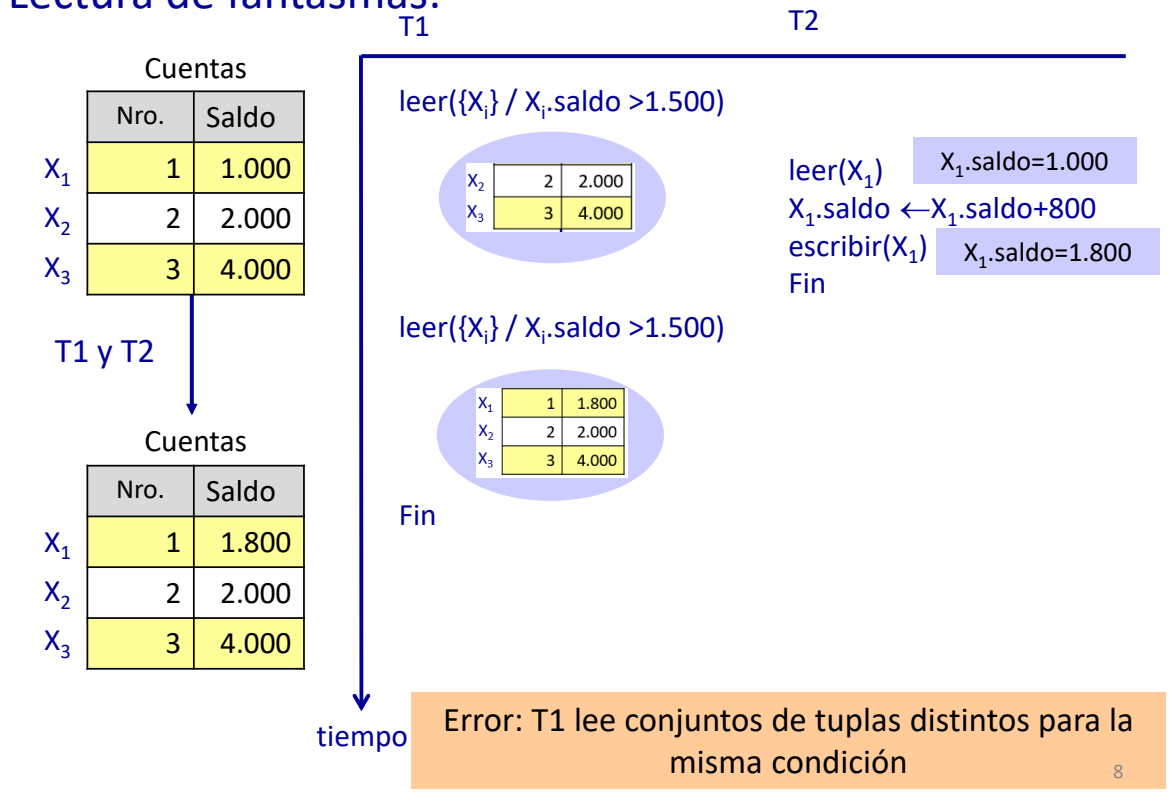
¿Cuál es el problema?

T2 actualiza un elemento de datos que ha sido leído por T1, que todavía no ha finalizado. El problema surge cuando T1 vuelve a acceder (en lectura) al mismo elemento de datos y **lee un valor distinto**.

De acuerdo a la propiedad de **aislamiento**, el efecto final de la ejecución concurrente de estas dos transacciones debería ser el mismo que el que se obtendría si las dos transacciones se ejecutasen de forma aislada (**en serie**), primero T1 y luego T2, o primero T2 y luego T1, en ambos casos T1 leería, en las dos lecturas, el mismo valor (en el primer orden 1000 y en el segundo 500). Como se puede observar, en la ejecución concurrente del ejemplo, no es así.

1 Ejecución concurrente de transacciones: anomalías.

Lectura de fantasmas:



Esta anomalía podría verse como un caso específico de la lectura no repetible y para ilustrarla con un ejemplo hay que considerar una lectura un poco distinta a las anteriores ya que tiene que ver con el hecho de que una transacción solicita la lectura de las tuplas (registros) que cumplan una determinada condición.

¿Cuál es el problema?

De acuerdo a la propiedad de **aislamiento**, una transacción debe leer siempre en un mismo estado de la base de datos y, por lo tanto, sólo debe ver los cambios que ella realiza.

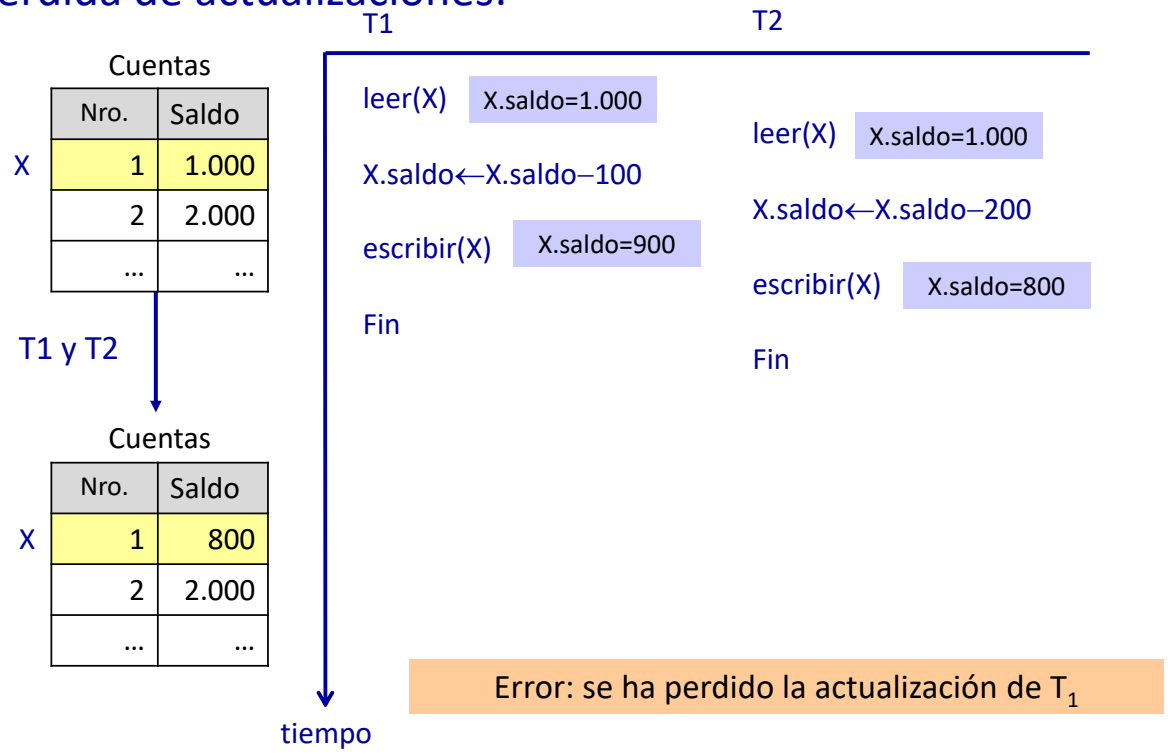
T1 (que sólo lee) no está haciendo sus lecturas en el mismo estado de la base de datos. Las dos lecturas le proporcionan conjuntos distintos de valores.

La causa de la anomalía es la misma que en la anomalía de "lectura no repetible", pero el efecto es distinto.

T1 no se ejecuta en un único estado de la base de datos, no se cumple la propiedad de **aislamiento**.

1 Ejecución concurrente de transacciones: anomalías.

Pérdida de actualizaciones:



9

¿Cuál es el problema?

T1 actualiza un elemento de datos que ha sido leído por T2 que todavía no ha finalizado. El problema surge cuando T2 vuelve a acceder al elemento de datos para actualizarlo (actualización en función de su lectura), **se pierde la actualización de T1**.

De acuerdo a la propiedad de **aislamiento**, el efecto final de la ejecución concurrente de estas dos transacciones debería ser el mismo que el que se obtendría si las dos transacciones se ejecutasen de forma aislada (**en serie**), primero T1 y luego T2, o primero T2 y luego T1. Como se puede observar, en la ejecución concurrente del ejemplo, no es así.

Nota: en SQL la operación UPDATE con asignaciones SET donde aparecen referencias a columnas en la parte derecha de las asignaciones, es realmente una lectura-escritura, que se realizan de manera consecutiva. Es decir, en SQL la escritura (UPDATE) de T2, realmente sería una lectura (el valor ya actualizado por T1) seguida de la actualización. Esto provoca que a veces pueda pensarse que este problema no existe, cuando de hecho sí que se da.

Recuperación en bases de datos.

1 Ejecución concurrente de transacciones: anomalías.

2 Control de la concurrencia en SQL.

3 Planes serializables por conflictos.

4 Protocolos de bloqueo.

5 Protocolos de ordenamiento por marcas de tiempo.

6 Protocolos multiversión.

7 Concurrencia y recuperación.

2 Control de la concurrencia en SQL.

Control del procesamiento de transacciones en SQL:
(operaciones de usuario)

SET TRANSACTION modo [, modo] …

modo:= **nivel de aislamiento**

| modo de acceso

| área de diagnostico

modo de acceso:= READ ONLY

| READ WRITE

área de diagnostico:= DIAGNOSTICS SIZE *número*

11

Las distintas versiones de la instrucción SQL **SET TRANSACTION** permiten que el usuario, durante su sesión de trabajo, pueda dar al SGBD directrices sobre su funcionamiento.

El argumento *nivel de aislamiento* permite indicar el nivel de control de la concurrencia en el que debe funcionar el sistema.

2 Control de la concurrencia en SQL.

Control del procesamiento de transacciones en SQL:

SET TRANSACTION modo [, modo] …

- ✓ El nivel de aislamiento hace referencia al control de la ejecución concurrente.
- ✓ El área de diagnostico especifica el *numero* máximo de condiciones de diagnóstico (errores o excepciones) que pueden registrarse relativas a la ejecución de las últimas instrucciones SQL.
- ✓ El modo de acceso restringe el tipo de operaciones que se pueden ejecutar en una transacción: READ ONLY prohíbe operaciones de actualización de la base de datos.

12

2 Control de la concurrencia en SQL.

Control del procesamiento de transacciones en SQL:

nivel de aislamiento:= ISOLATION LEVEL {READ UNCOMMITTED
| READ COMMITTED
| REPETEABLE READ
| SERIALIZABLE}

READ UNCOMMITTED (lectura no confirmada)

READ COMMITED (lectura confirmada)

REPETEABLE READ (lectura repetible)

SERIALIZABLE (serializable)

13

El lenguaje SQL propone cuatro niveles de control de la concurrencia. Los niveles son gradualmente más exigentes en el control de la concurrencia, es decir, en las anomalías que no se van a permitir, como se indica en la transparencia siguiente.

Esta flexibilidad, en el control de la concurrencia, tiene sentido porque el usuario conoce el entorno en el que está teniendo lugar su sesión de trabajo, y por lo tanto, puede saber si hace falta un control estricto de la concurrencia.

En un SGBD, el control de la concurrencia conlleva un trabajo añadido a la propia la ejecución de las operaciones de las transacciones.

Control del procesamiento de transacciones en SQL:

Si una transacción se ejecuta en un nivel de aislamiento distinto a **serializable**, entonces pueden darse algunas de las anomalías estudiadas.

	Anomalía			
Nivel de aislamiento	Lectura sucia	Lectura no repetible	Lectura de fantasmas	Pérdida de actualizaciones
READ UNCOMMITED	**SÍ**	**SÍ**	**SÍ**	**SÍ**
READ COMMITED	**NO**	**SÍ**	**SÍ**	**SÍ**
REPETIBLE READ	**NO**	**NO**	**SÍ**	**SÍ**
SERIALIZABLE	**NO**	**NO**	**NO**	**NO**

Control de la concurrencia.

1 Ejecución concurrente de transacciones: anomalías.

2 Control de la concurrencia en SQL.

3 Planes serializables por conflictos.

4 Protocolos de bloqueo.

5 Protocolos de ordenamiento por marcas de tiempo.

6 Protocolos multiversión.

7 Concurrencia y recuperación.

3 Planes serializables por conflictos.

¿Cómo se deben intercalar las operaciones de las transacciones para asegurar la propiedad de **aislamiento**?

El objetivo del control de la ejecución concurrente de transacciones es asegurar que el efecto final de la ejecución concurrente sea el mismo que el que se obtendría si las transacciones se ejecutasen de forma aislada, es decir una a continuación de otra (**en serie**).

16

El objetivo del control de la ejecución concurrente de transacciones es asegurar que la ejecución concurrente respeta la propiedad de **aislamiento**, es decir, que el efecto final de la ejecución concurrente sea el mismo que el que se obtendría si las transacciones se ejecutasen de forma aislada, es decir una a continuación de otra (**en serie**). Se trata de poder intercalar las operaciones de transacciones distintas sin que se produzcan anomalías (interferencias no deseables).

En este punto del tema, se va a intentar caracterizar los planes concurrentes que pueden ser aceptados como buenos porque no presentan anomalías, es decir, porque son "*equivalentes*" a un **plan en serie** para el mismo conjunto de transacciones.

3 Planes serializables por conflictos.

- ✓ ¿Qué operaciones provocan las anomalías?
- ✓ ¿Qué se entiende por ejecución concurrente "correcta"?
- ✓ ¿Qué se puede hacer?

- ✓ Plan de ejecución de un conjunto de transacciones.
- ✓ Operaciones en conflicto en un plan.
- ✓ Plan serializable por conflictos.

- ✓ $r_i(x)$: la transacción T_i lee *x*.
- ✓ $w_i(x)$: la transacción T_i escribe *x*.
- ✓ c_i: la transacción T_i se confirma.
- ✓ a_i: la transacción T_i se anula.

17

Una ejecución concurrente de varias transacciones es correcta si respeta la propiedad de **aislamiento**, es decir, si el efecto final de la ejecución concurrente es el mismo que el que se obtendría si las transacciones se ejecutasen de forma aislada, es decir una a continuación de otra (**en serie**).

Para analizar **qué** planes concurrentes deben ser aceptados como buenos (sin anomalías), y, en consecuencia, **cómo** se debe controlar la ejecución concurrente de transacciones, para aceptar sólo esos planes, se van a hacer algunas definiciones previas que faciliten el estudio.

3 Planes serializables por conflictos.

Plan de ejecución de un conjunto de transacciones:

Ordenamiento de las operaciones de las transacciones en el que las operaciones de cada transacción aparecen en el plan en el mismo orden que aparecen en la transacción.

Plan en serie: plan en el que, para cada transacción que participa en el plan, todas sus operaciones se ejecutan consecutivamente en el plan (las operaciones no se intercalan).

Plan concurrente: plan en el que las operaciones de cualquier transacción que participa en el plan pueden aparecer intercaladas entre las operaciones de cualquier otra transacción del plan.

18

En primer lugar, se define el concepto de **plan de ejecución de un conjunto de transacciones**. Un plan es un **plan concurrente** si las operaciones de las distintas transacciones están intercaladas en el plan. El plan es **un plan en serie** si las operaciones de cada transacción se ejecutan consecutivamente en el plan (una transacción después de otra).

- Todo plan en serie es correcto, no presenta anomalías. Dado un conjunto de *n* transacciones, hay *n!* planes en serie posibles.
- Un plan concurrente puede presentar anomalías.

Nota: En esta definición, “ordenamiento” se entiende como “ordenamiento total”.

3 Planes serializables por conflictos.

Plan de ejecución de un conjunto de transacciones:

Ordenamiento de las operaciones de las transacciones en el que las operaciones de cada transacción aparecen en el plan en el mismo orden que aparecen en la transacción.

T_1
O_{11}
O_{12}
O_{13}

T_2
O_{21}
O_{22}

No es un plan
O_{21}
O_{12}
O_{11}
O_{22}
O_{13}

P_1: Plan en serie
O_{11}
O_{12}
O_{13}
O_{21}
O_{22}

P_2: Plan en serie
O_{21}
O_{22}
O_{11}
O_{12}
O_{13}

P_3: Plan concurrente
O_{21}
O_{11}
O_{22}
O_{12}
O_{13}

...

P_n: Plan concurrente
O_{21}
O_{11}
O_{12}
O_{22}
O_{13}

O_{ij}: Operación *j* de la transacción *i*

19

3 Planes serializables por conflictos.

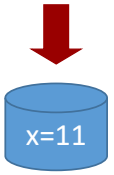
Plan de ejecución de un conjunto de transacciones:

- Un plan **en serie** siempre es correcto.
- Dos planes en serie del mismo conjunto de transacciones pueden dejar la base de datos en estados distintos.

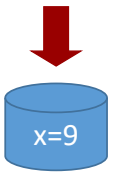
T_1
$r_1(x)$;
si x < 3
entonces x ← x+2
$w_1(x)$;
c_1;

T_2
$r_2(x)$
x← x+7
$w_2(x)$;
c_2;

P_1: Plan en serie
$r_1(x)$;
si x < 3
entonces x ← x+2
$w_1(x)$;
c_1;
$r_2(x)$
x← x+7
$w_2(x)$;
c_2;



P_2: Plan en serie
$r_2(x)$
x← x+7
$w_2(x)$;
c_2;
$r_1(x)$;
si x < 3
entonces x ← x+2
$w_1(x)$;
c_1;



20

3 Planes serializables por conflictos.

Lectura sucia:

$r_1(x), w_1(x), r_2(x), c_2, a_1$

Lectura no repetible:

$r_1(x), r_2(x), w_2(x), c_2, r_1(x), c_1$

Lectura de fantasmas: siendo x un conjunto de filas

$r_1(x), r_2(x), w_2(x), c_2, r_1(x), c_1$

Pérdida de actualizaciones:

$r_1(x), r_2(x), w_1(x), w_2(x), c_1, c_2$

¿Qué operaciones provocan las anomalías?

21

En la transparencia, se muestran los planes de ejecución concurrentes de los cuatro ejemplos de anomalías que se han presentado anteriormente.

3 Planes serializables por conflictos.

Operaciones en conflicto:

Dos operaciones de un plan están en conflicto si satisfacen las siguientes tres condiciones:

1) pertenecen a distintas transacciones,
2) acceden al mismo elemento de datos X, y
3) al menos una de ellas es una operación escribir(X).

P: $r_1(x)$, $r_2(x)$, $w_1(x)$, $w_2(x)$, c_1, c_2

- $r_1(x)$ y $w_2(x)$ están en conflicto
- $r_2(x)$ y $w_1(x)$ están en conflicto
- $w_1(x)$ y $w_2(x)$ están en conflicto

22

En el análisis de los planes de ejecución concurrentes, lo primero que hay que hacer es identificar **la causa de las anomalías**.

Analizando los ejemplos de las cuatro anomalías, se puede observar que, para que se produzca una anomalía (una interferencia), las transacciones deben estar accediendo al mismo elemento de datos y además, al menos una de ellas, debe estar actualizándolo. Si las transacciones sólo leen, aunque sea el mismo elemento de datos, no pueden interferir entre sí, no puede haber anomalías.

La introducción del concepto de **par de operaciones en conflicto** va a permitir caracterizar este hecho, e identificar **la causa** de las anomalías.

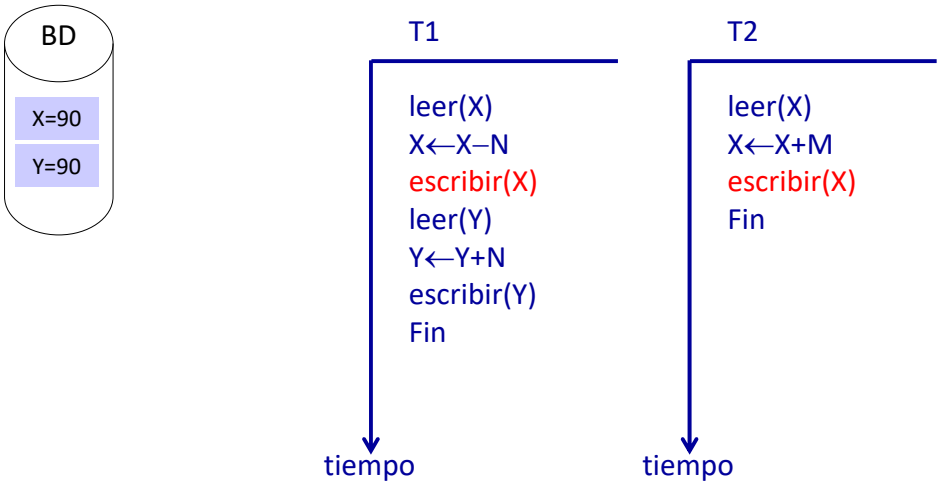
La aparición de pares de operaciones en conflicto en un plan va a ser una condición necesaria pero no suficiente para que se produzca una anomalía. Es decir, si hay anomalía habrá pares de operaciones en conflicto, pero el hecho de que aparezcan pares de operaciones en conflicto no significa que vaya a producirse una anomalía, el plan: P: $w_1(x)$, $w_2(x)$, c_1, c_2, tiene un par de operaciones en conflicto pero no produce ninguna anomalía.

El problema va a residir en el orden en el que aparecen las operaciones en conflicto dentro de cada par, como se analizará a continuación.

3 Planes serializables por conflictos.

Ejemplo 1

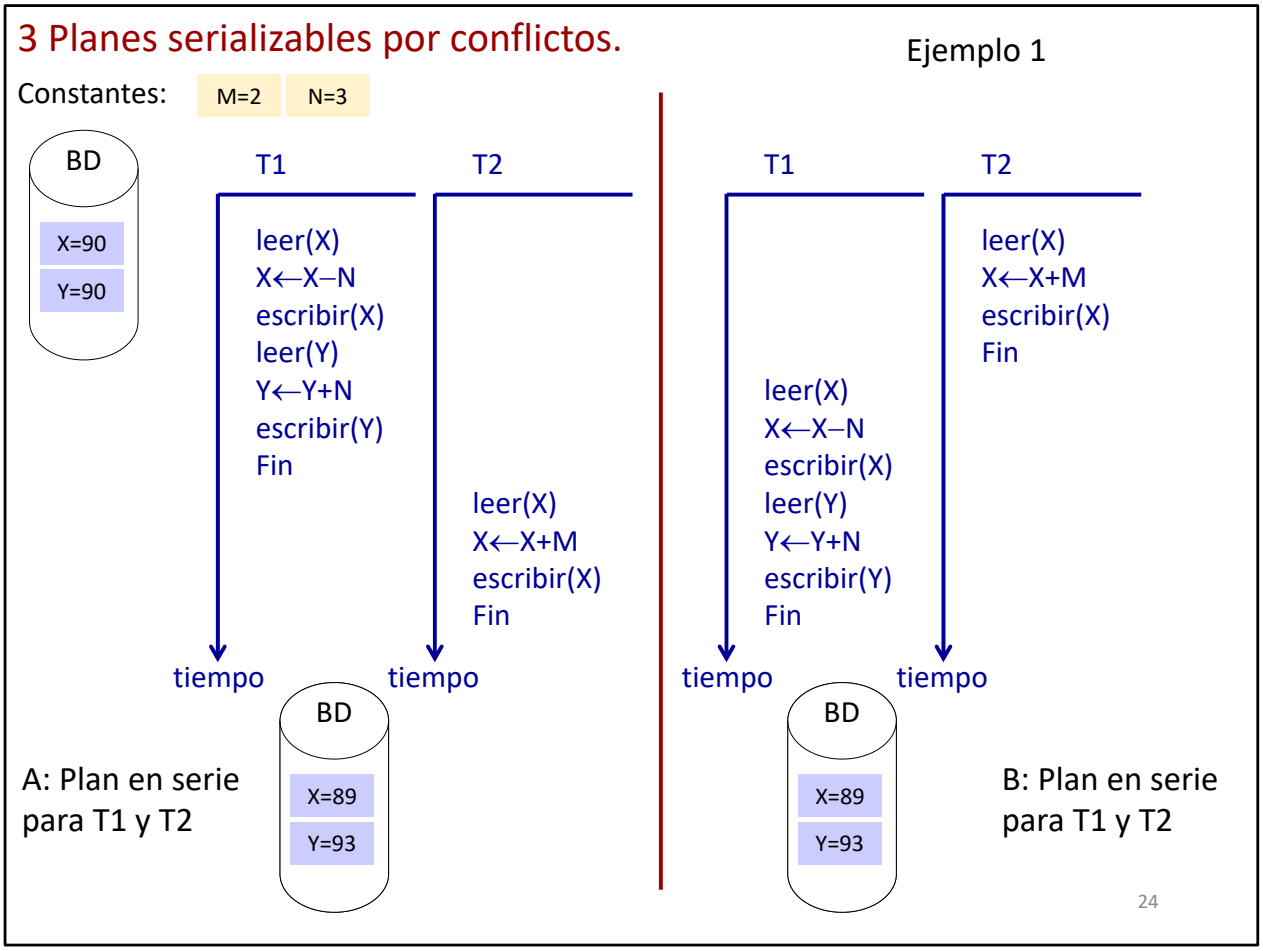
Constantes: M=2 N=3



23

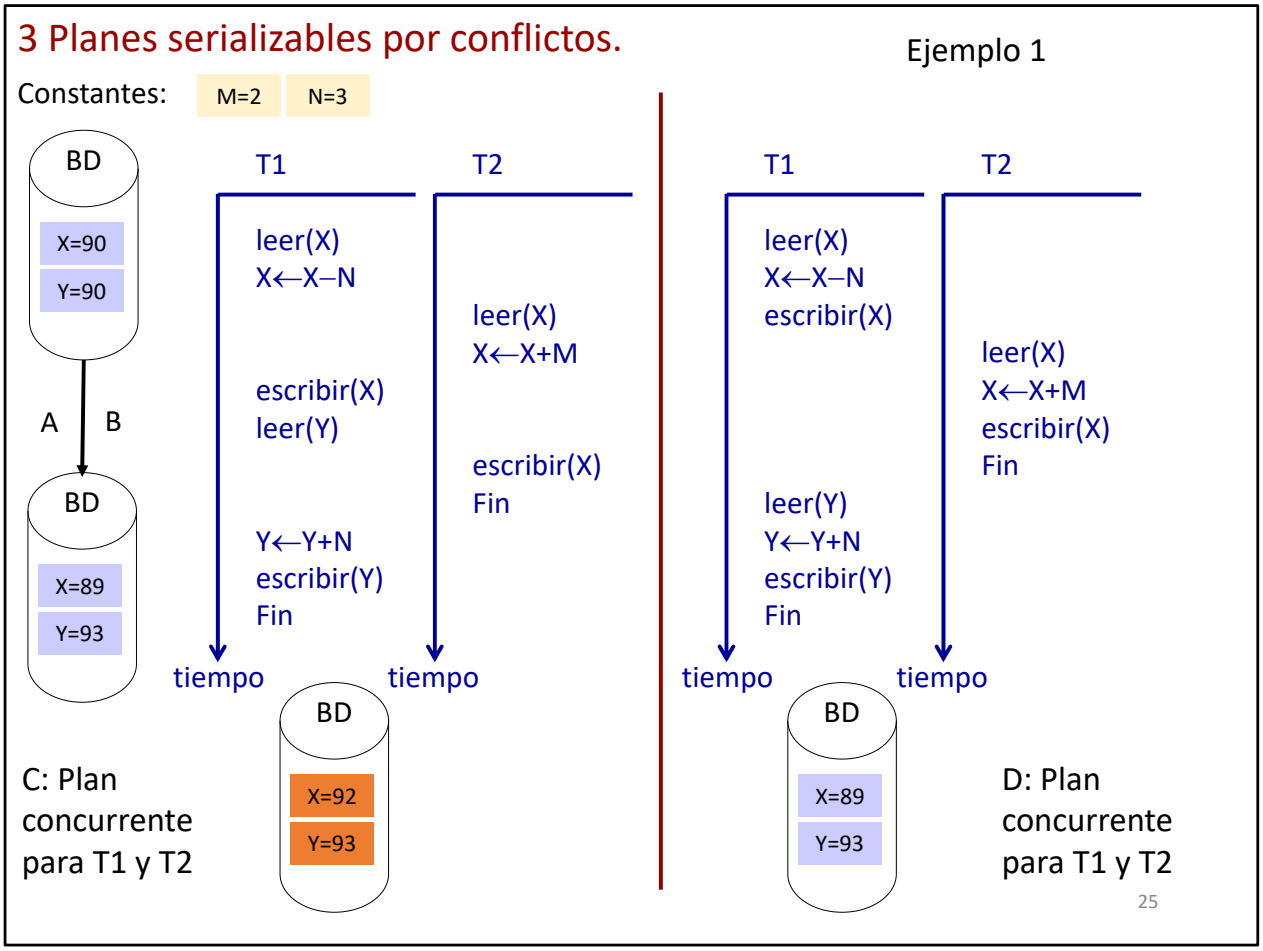
En esta transparencia, se presentan dos transacciones: T1 y T2.

En las transparencias siguientes, se presentan los dos planes en serie para el par de transacciones y dos posibles planes de ejecución concurrente.



Es importante observar que planes en serie distintos, para el mismo conjunto de transacciones, no tienen que producir el mismo estado de la base de datos.

En el ejemplo, el resultado de los dos planes en serie es el mismo porque las operaciones que se ejecutan sobre el elemento de datos X son conmutativas. Si las operaciones sobre X de T1 y T2 fuesen una operación de suma y una operación de producto, el valor final de X en ambos planes en serie no sería el mismo.



En la ejecución concurrente de transacciones, se espera que el efecto final de la ejecución sea el mismo que el que se produciría si se ejecutase una transacción a continuación de otra (propiedad de **aislamiento**).

En el ejemplo, se observa:

- En el plan concurrente **C**, la intercalación de las operaciones sobre el elemento de datos X no es correcta, se pierde la actualización de T1, es decir, se produce la anomalía de la pérdida de actualizaciones.

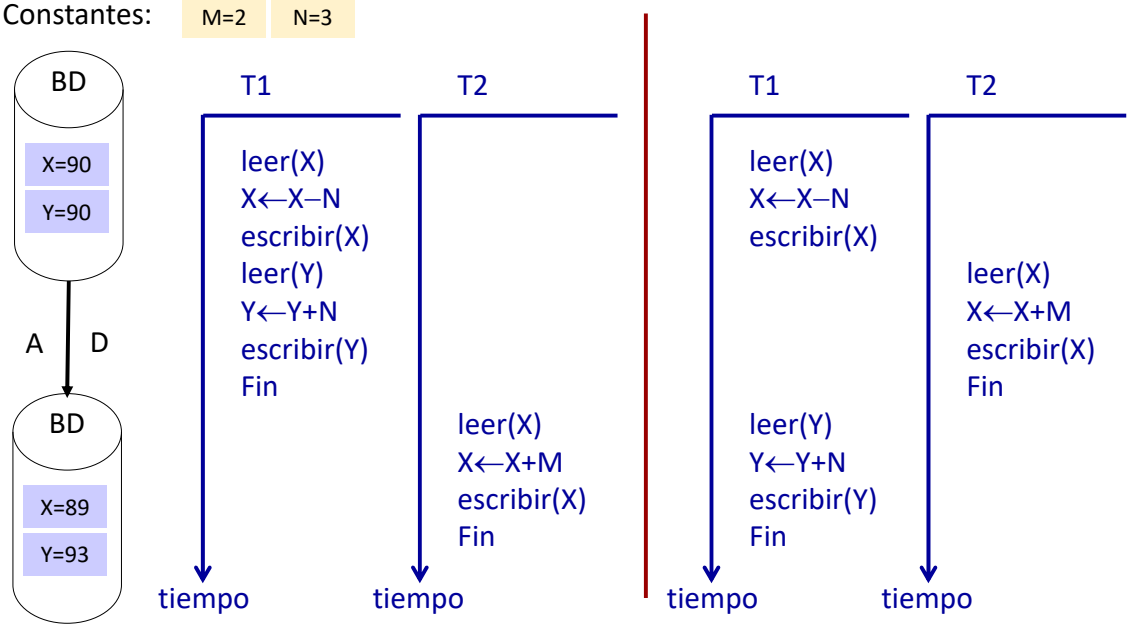
- En el plan concurrente **D**, la intercalación de las operaciones sobre el elemento de datos X es correcta, las operaciones de una transacción (T1), sobre X, ocurren antes que las de la otra transacción (T2). El efecto es el mismo que si se hubiese ejecutado primero T1 y a continuación T2 (**plan en serie**), es decir, como si las dos transacciones se hubiesen ejecutado de forma aislada.

Obsérvese que, en el plan D, existe concurrencia, es decir, se intercalan las operaciones de ambas transacciones, pero las operaciones en conflicto (sobre X) ocurren en el mismo orden que en el plan en serie T1-T2.

3 Planes serializables por conflictos.

Ejemplo 1

Constantes: M=2 N=3

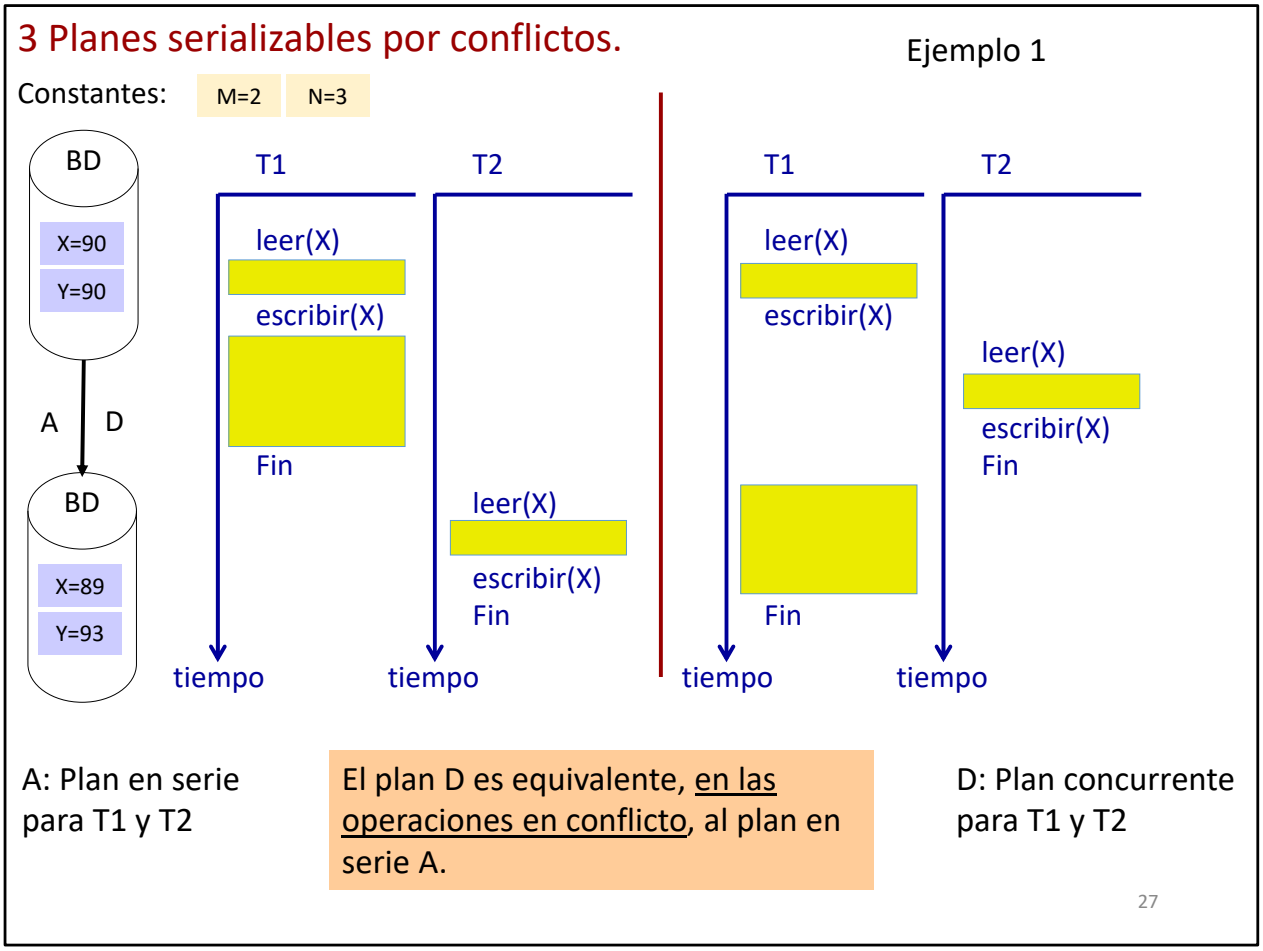


A: Plan en serie para T1 y T2

El plan D es equivalente, en las operaciones en conflicto, al plan en serie A.

D: Plan concurrente para T1 y T2

26



En el plan **D,** en el que no se produce ninguna anomalía, las operaciones en conflicto aparecen exactamente en el mismo orden que en el plan en serie T1-T2.

El dibujo muestra que lo que es igual, en ambos planes, es la "*radiografía*" de las operaciones en conflicto, en este caso, las operaciones sobre el elemento de datos X.

El plan D no es un plan en serie, pero en las operaciones en conflicto se comporta como el plan en serie T1-T2. La concurrencia se produce en las operaciones que no entran en conflicto (sobre el elemento de datos Y)

Nota: en el plan D se produce la anomalía de la "lectura sucia", pero como ya se ha comentado, y se estudiará más adelante (punto 7), esta seudoanomalía se puede corregir o evitar.

El ejemplo sugiere una idea para el **control** de la ejecución concurrente de transacciones: "la intercalación de las operaciones en el plan se debe hacer de forma que, para cada par de transacciones del plan, los pares de operaciones en conflicto deben incluir las operaciones de ambas transacciones en el mismo orden, primero la operación de una de las transacciones y después la de la otra transacción" (efecto similar al de un plan en serie para el mismo par de transacciones).

Esta idea va a estar en el centro de todos los protocolos que se han desarrollado para controlar la ejecución concurrente de transacciones.

3 Planes serializables por conflictos.

Objetivo: permitir planes concurrentes que sean equivalentes a un plan en serie.

⬇

Un plan concurrente de un conjunto de transacciones es **serializable** si es equivalente a un plan en serie para las mismas transacciones.

⬇

¿Qué significa ser equivalente?

⬇

Cómo la causa de las anomalías está en las operaciones en conflicto, se va introducir una definición de equivalencia basada en las operaciones en conflicto: **equivalencia por conflictos**.

28

Si el efecto que se espera de la ejecución concurrente de un conjunto de transacciones es el mismo que el que se produciría con un plan en serie para las mismas transacciones, se va a introducir el **concepto de equivalencia entre un plan concurrente y un plan en serie**. Un plan concurrente equivalente a un plan en serie se denomina **plan serializable.**

Aunque, en el estudio anterior, ya se ha sugerido cuál es la idea de equivalencia entre un plan concurrente y un plan en serie, se debe hacer una definición más precisa de este concepto.

3 Planes serializables por conflictos.

Dos planes son **equivalentes por conflictos** si el orden de dos operaciones cualesquiera en conflicto es el mismo en ambos planes.

Un plan es **serializable por conflictos** si es equivalente por conflictos a un **plan en serie**.

29

De acuerdo a las ideas sugeridas en los ejemplos anteriores, el concepto de equivalencia entre planes que interesa definir es el de **equivalencia por conflictos**.

Dos **planes** (para el mismo conjunto de transacciones) son **equivalentes por conflictos**, si el orden de dos operaciones cualesquiera en conflicto es el mismo en ambos planes. El efecto sobre los elementos de datos en conflicto es el mismo en ambos planes. En los pares de operaciones que no están en conflicto, el orden de las operaciones no importa.

Respecto a los elementos de datos en conflicto, los dos planes son exactamente iguales.

Dos planes concurrentes pueden ser equivalentes por conflictos y no ser serializables, es decir pueden presentar anomalías (no ser buenos).

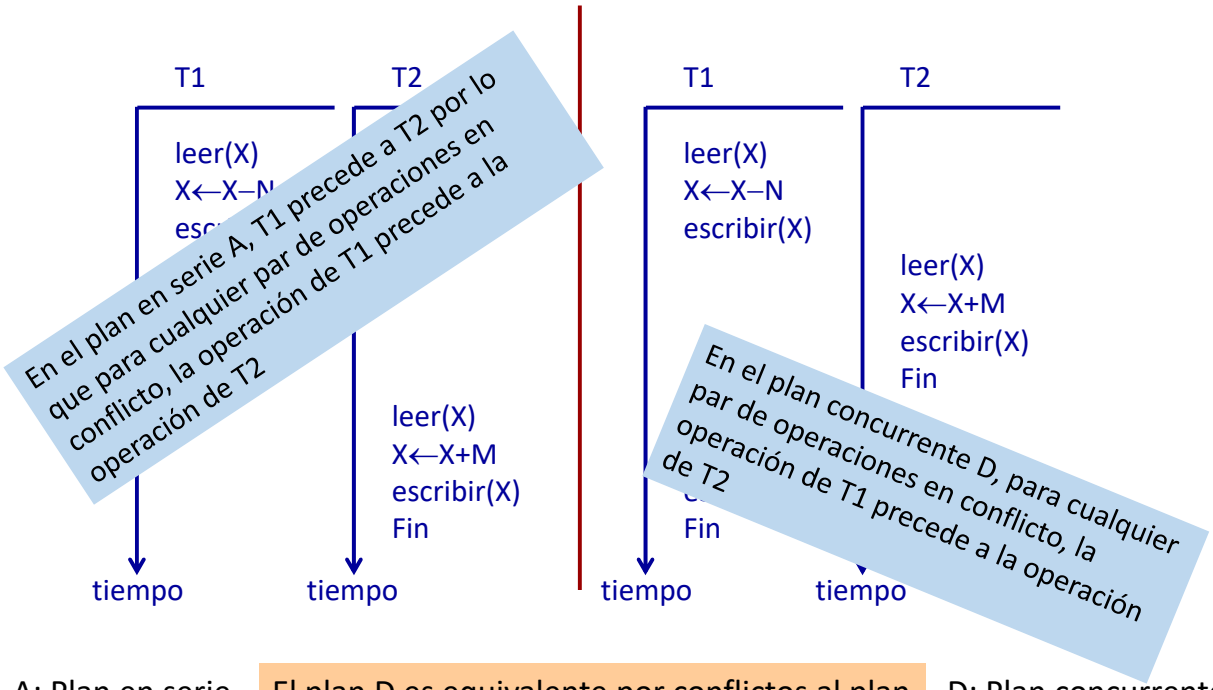
Como el efecto esperado, en la ejecución concurrente de transacciones, es el que se obtendría ejecutando una transacción a continuación de otra (**en serie**), en los SGBD, sólo se van a admitir planes **serializables por conflictos**, es decir, equivalentes por conflictos a un plan en serie.

En un **plan serializable por conflictos,** se van a poder intercalar en cualquier orden las operaciones no conflictivas de las transacciones, pero las operaciones en conflicto se deberán intercalar en el mismo orden que en un plan en serie para el mismo conjunto de transacciones.

El estado final de la BD en un plan concurrente serializable (por conflictos) es el mismo que el de su plan en serie equivalente.

3 Planes serializables por conflictos.

Ejemplo 1



A: Plan en serie para T1 y T2

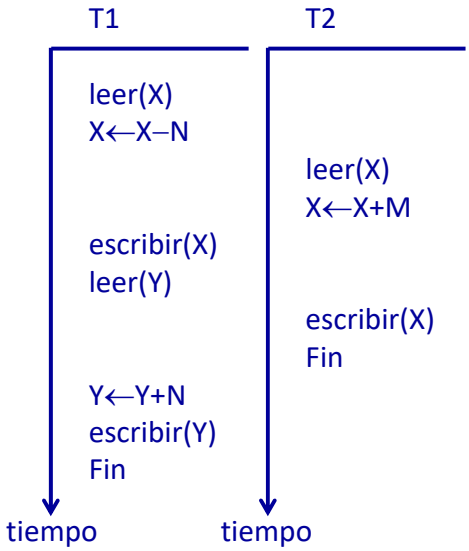
El plan D es equivalente por conflictos al plan en serie A por lo que es seriabilizable por conflictos

D: Plan concurrente para T1 y T2

30

3 Planes serializables por conflictos.

Ejemplo 1



C: Plan concurrente para T1 y T2

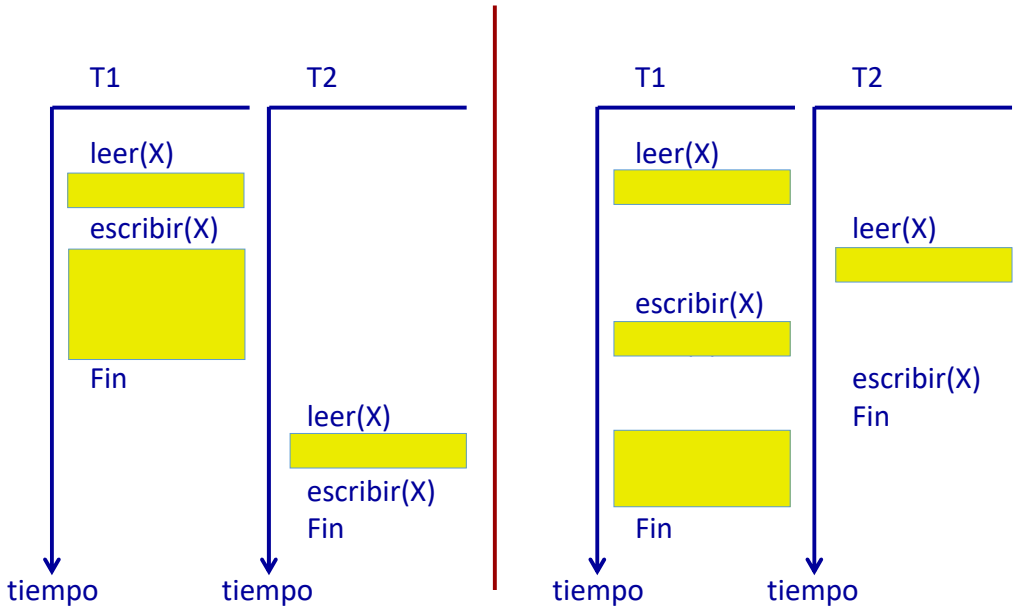
¿Es el plan C serializable por conflictos?

31

El efecto de este plan, en los elementos de datos en conflicto, no es el mismo que el de la ejecución del plan en serie T1-T2, ni el de la ejecución del plan en serie T2-T1.

3 Planes serializables por conflictos.

Ejemplo 1



A: Plan en serie para T1 y T2

El plan C **no** es equivalente por conflictos, al plan en serie A.

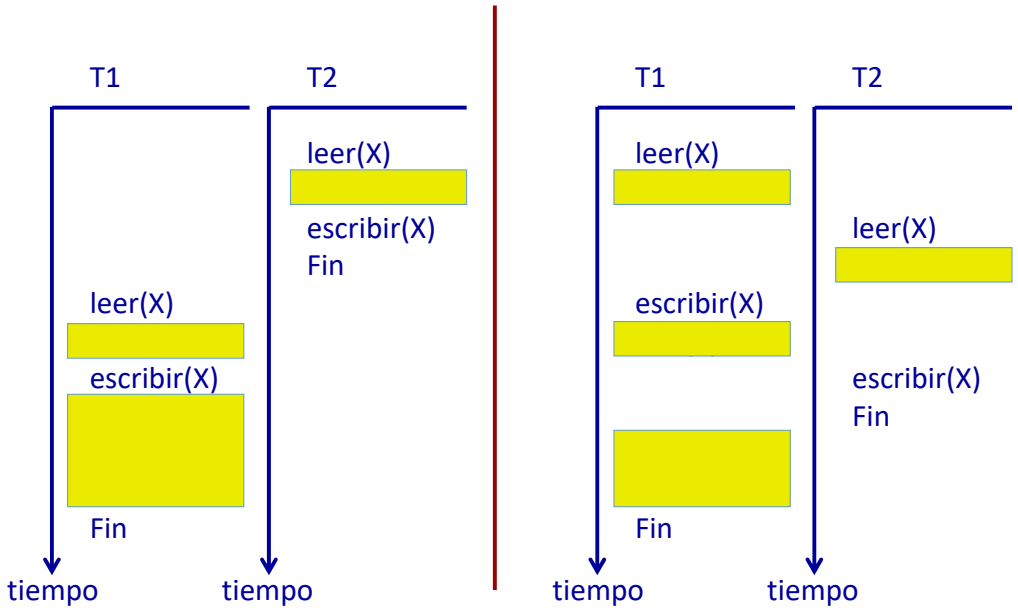
C: Plan concurrente para T1 y T2

32

El efecto de este plan no es el mismo que el de la ejecución del plan en serie T1-T2.

3 Planes serializables por conflictos.

Ejemplo 1



B: Plan en serie para T1 y T2

El plan C **no** es equivalente por conflictos, al plan en serie B.

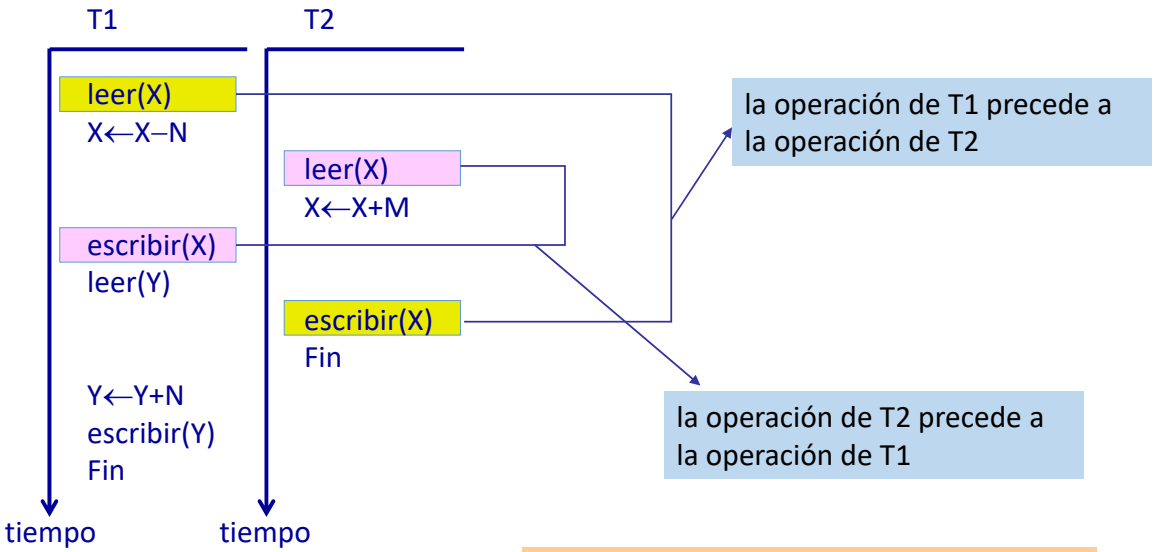
C: Plan concurrente para T1 y T2

33

El efecto de este plan no es el mismo que el de la ejecución del plan en serie T2-T1.

3 Planes serializables por conflictos.

Ejemplo 1



Es imposible la equivalencia del plan C con un plan en serie para T1 y T2

C: Plan concurrente para T1 y T2

34

Como se puede observar, en el plan C, aparecen dos pares de operaciones en conflicto, en los que el orden de las operaciones de ambas transacciones está invertido:

- En el par $[r_1(X), w_2(X)]$, primero ocurre la operación de T1 y a continuación la operación de T2.
- En el par $[r_2(X), w_1(X)]$, primero ocurre la operación de T2 y a continuación la operación de T1.

Es imposible la equivalencia del plan C con un plan en serie para T1 y T2.

3 Planes serializables por conflictos.

Prueba de seriabilidad por conflicto de un plan:

Grafo de serialización: grafo dirigido G=(N, A), que consiste en un conjunto de nodos N y un conjunto de arcos A.

- ✓ El grafo contiene un nodo por cada transacción que participa en el plan.
- ✓ Se crea un arco $a_{ij} = (T_i \rightarrow T_j)$ si una operación de T_i aparece en el plan antes de una operación en conflicto de T_j.

Un plan concurrente no es serializable por conflictos si su grafo de serialización tiene un ciclo.

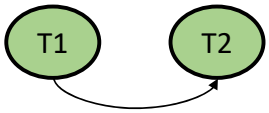
35

El grafo de serialización es una herramienta para determinar si un plan concurrente es serializable por conflictos.

Cuando, en el grafo de serialización de un plan, aparece un ciclo, el plan ya no es serializable y debe ser rechazado.

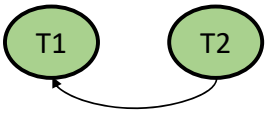
3 Planes serializables por conflictos.

Ejemplo 1



X

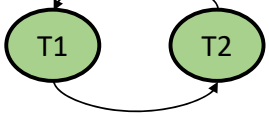
Grafo para el plan A
(plan en serie)



X

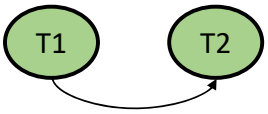
Grafo para el plan B
(plan en serie)

X



X

Grafo para el plan C
(plan no serializable)



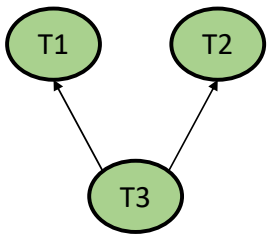
X

Grafo para el plan D
(plan seriabilizable)

36

3 Planes serializables por conflictos.

Puede haber más de un plan en serie equivalente a un plan serializable dado:



Planes en serie equivalentes:

T3-T1-T2

T3-T2-T1

3 Planes serializables por conflictos.

En la práctica comprobar la seriabilidad de un plan una vez ha terminado no es un comportamiento adecuado:

- ✓ En un entorno concurrente es posible que no haya un instante de finalización del plan.
- ✓ En caso de que sí que haya finalizado el plan, y después se comprueba si el plan es serializable, en el caso de que no lo sea, se deberá deshacer todo el plan.

Los sistemas reales no comprueban la seriabilidad de los planes. Aplican protocolos que aseguran, durante la ejecución del plan, que éste será serializable sin tener que comprobarlo una vez finalizado.

38

Control de la concurrencia.

Técnicas de control de la concurrencia → PROTOCOLOS

↓

Objetivo: Asegurar que los planes son **serializables por conflictos.**

39

Como se anunció al principio del tema, el objetivo del control de la ejecución concurrente de transacciones, es poder intercalar las operaciones de transacciones distintas sin que se produzcan anomalías (interferencias no deseables). Se trata de asegurar que la ejecución concurrente respeta la propiedad de **aislamiento**, es decir, que el efecto final sea el mismo que el que se obtendría si las transacciones se ejecutasen una a continuación de otra (**en serie**).

En el punto 3 del tema, se **han caracterizado** los planes concurrentes que pueden ser aceptados como buenos porque no presentan anomalías, éstos son los **planes serializables por conflictos**, es decir, planes que son equivalentes, en las operaciones en conflicto, a un plan en serie para el mismo conjunto de transacciones.

Todos los protocolos (algoritmos) desarrollados para el control de la concurrencia en los SGBD admiten sólo planes serializables por conflictos.

Los protocolos aseguran **dinámicamente** que el plan concurrente que se está ejecutando es un plan serializable por conflictos, es decir, rechazan, durante la ejecución del plan, aquellas transacciones (u operaciones) que puedan conducir a una efecto indeseado (anomalías).

Control de la concurrencia.

Trabajar con planes completos significaría esperar a que todas las transacciones finalizasen, lo que no es posible en un sistema real.

Los protocolos de control de la concurrencia analizan dinámicamente (durante la ejecución del plan) la aparición de operaciones en conflicto y actúan para evitar anomalías.

40

Planes completos: una vez finalizadas todas las transacciones, se analiza si el plan es serializable (por conflictos), por ejemplo, usando el grafo de serialización. Si el plan es serializable se acepta, si no lo es se rechazan todas las transacciones del plan.

Ejemplo: en el plan P: $r_1(x)$, $r_2(x)$, $w_1(x)$, $w_2(x)$, c_1, c_2, para las transacciones:

T1: $r_1(x)$, $w_1(x)$, c_1 T2: $r_2(x)$, $w_2(x)$, c_2

se ha producido la anomalía de "pérdida de actualizaciones". El plan debe ser rechazado.

En un sistema real, en el que se inician transacciones continuamente, nunca se tiene un plan completo. Para tener planes completos, se debería impedir, en un momento determinado, la entrada de nuevas transacciones y esperar a que las transacciones activas finalizasen, antes de analizar el plan que se ha producido.

Esto es inadmisible en un sistema real. Los usuarios podrían estar mucho tiempo en espera. En la práctica los sistemas no analizan los planes completos.

Los protocolos, durante la ejecución de las transacciones, van analizando los potenciales conflictos para evitarlos, rechazando sólo la transacción que entra en conflicto con las otras transacciones que se están ejecutando concurrentemente.

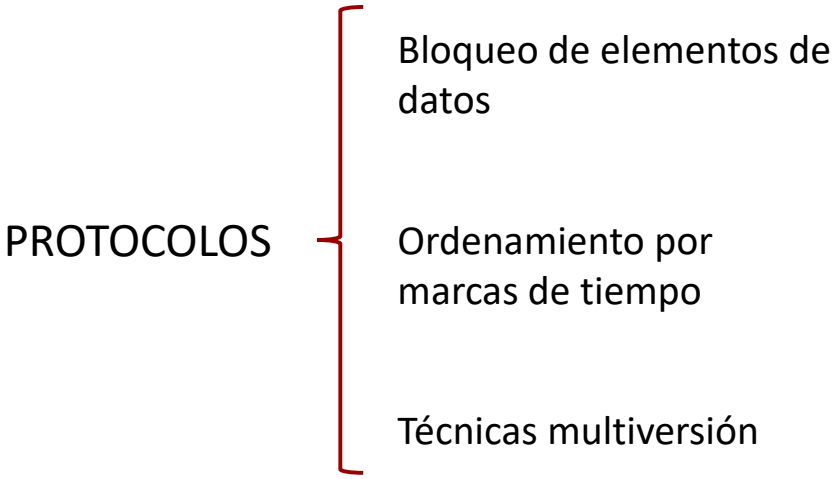
Por ejemplo en el plan anterior P, el sistema, ante la petición $w_1(X)$, detecta una potencial anomalía (par de operaciones en conflicto) y debe actuar de alguna manera:

- dejar en espera (bloquear) T1 a la espera de lo que haga T2
- rechazar (anular) T1.
- …

Según el protocolo de control de la concurrencia, utilizado en el SGBD, la respuesta del sistema será distinta.

Los sistemas previenen las anomalías sin llegar a comprobar que se han producido: política conservadora.

Control de la concurrencia.



41

Cada protocolo de control de la concurrencia se basa en una estrategia distinta, aunque el objetivo sea el mismo. Los protocolos se pueden clasificar, según su estrategia, en tres

- Bloqueo de elementos de datos.
- Ordenación de las operaciones por la marca de tiempo de las transacciones.
- Mantenimientos de varias versiones para los elementos de datos.

Los SGBD comerciales implementan en sus protocolos ideas de una o varias de estas estrategias.

Control de la concurrencia.

1 Ejecución concurrente de transacciones: anomalías.

2 Control de la concurrencia en SQL.

3 Planes serializables por conflictos.

4 Protocolos de bloqueo.

5 Protocolos de ordenamiento por marcas de tiempo.

6 Protocolos multiversión.

7 Concurrencia y recuperación.

4 Protocolos de bloqueo.

PROTOCOLOS

- Bloqueo de elementos de datos
- Ordenamiento por marcas de tiempo
- Técnicas multiversión

4 Protocolos de bloqueo.

Protocolos de bloqueos de elementos de datos:

- ✓ Los protocolos de bloqueo de elementos de datos se basan en la idea de restringir el acceso al mismo elemento de datos por varias transacciones.
- ✓ Las transacciones **solicitan el bloqueo** de un elemento de datos para acceder a él, y lo **liberan** posteriormente.
- ✓ Cuando una transacción solicita el bloqueo de un elemento de datos y las **reglas de bloqueo** se lo impiden, la transacción queda en **espera**.

BLOQUEO ➡ Estado de un elemento de datos, provocado por una transacción, que determina las operaciones que otras transacciones pueden realizar sobre él.

44

La idea básica de los **protocolos de bloqueo** consiste en restringir (temporalmente) el acceso al mismo elemento de datos, por parte de varias transacciones.

Aparentemente, esta estrategia impide la concurrencia, pero las reglas del bloqueo aceptarán o prohibirán el acceso al mismo elemento de datos, según el tipo de operación solicitada (lectura o escritura).

El efecto de estos protocolos de bloqueo en las transacciones es que éstas se mantienen en espera hasta que el acceso al elemento de datos solicitado está permitido.

Existen distintos tipos de protocolos de bloqueo. Uno de los más populares es el **protocolo de lectura/escritura** que es el que se presenta a continuación.

4 Protocolos de bloqueo.

Bloqueo de lectura/escritura (compartido/exclusivo):

- ✓ En el bloqueo de lectura/escritura un elemento de datos puede estar en tres estados: 'bloqueado para lectura', 'bloqueado para escritura' o 'desbloqueado'.
- ✓ Una transacción debe **'bloquear para lectura'** el elemento de datos X antes de realizar una operación **leer(X): bloquear_lectura(X).**
- ✓ Una transacción debe **'bloquear para escritura'** el elemento de datos X antes de realizar una operación **escribir(X).** En este caso también podrá realizar la operación **leer(X): bloquear_escritura(X).**
- ✓ Si **bloqueo(X)='bloqueado para lectura',** otras transacciones, distintas a la que realizó el bloqueo de X, pueden 'bloquear para lectura' el elemento de datos (**bloqueo compartido**).
- ✓ Si **bloqueo(X)='bloqueado para escritura'**, ninguna otra transacción, distinta a la que realizó el bloqueo de X, puede bloquear el elemento de datos (**bloqueo exclusivo**).
- ✓ Una transacción debe '**desbloquear'** el elemento de datos X, que tiene bloqueado, cuando ya no necesite acceder a él: **desbloquear(X).**

45

En la transparencia, se presentan las reglas del protocolo de bloqueo de lectura/escritura.

Al final de este punto del tema, se presentan los algoritmos para las operaciones: bloquear_lectura, bloquear_escritura, desbloquear.

4 Protocolos de bloqueo.

Bloqueo de lectura/escritura (compartido/exclusivo):

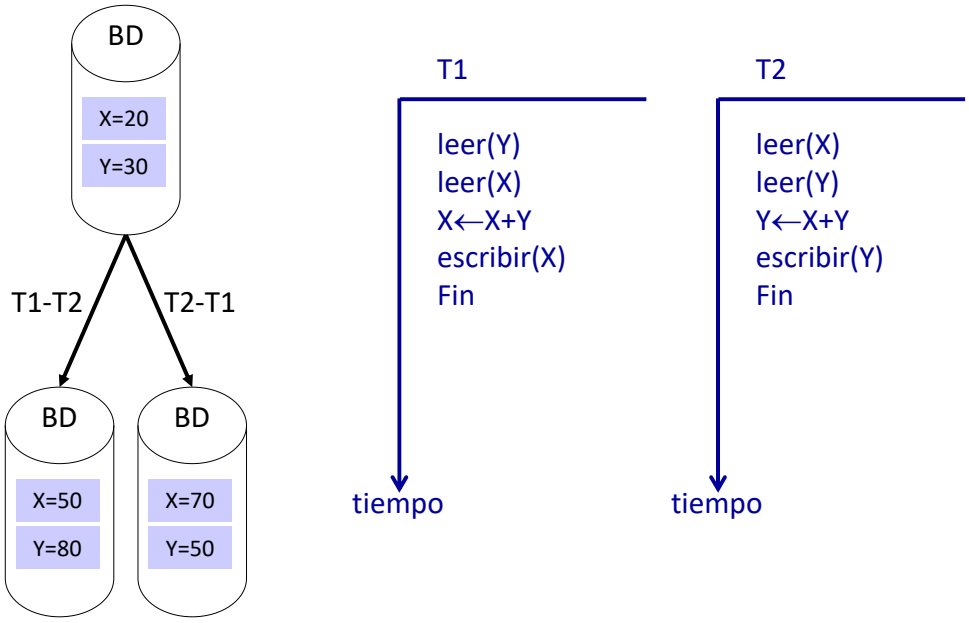
En estos protocolos, se pueden flexibilizar las operaciones de bloqueo y desbloqueo, con la **conversión de bloqueos**.

- ✓ **Promoción del bloqueo**: una transacción T que tiene un bloqueo de lectura sobre el elemento de datos X puede solicitar conseguir el bloqueo de escritura (si T es la única transacción con bloqueo de lectura sobre X, es posible promover el bloqueo, de lo contrario T debe esperar).
- ✓ **Degradación del bloqueo**: una transacción T que tiene un bloqueo de escritura sobre el elemento X puede solicitar bajar el bloqueo a lectura liberando el bloqueo de escritura.

Con esto, se evita el paso intermedio de desbloqueo del elemento de datos.

4 Protocolos de bloqueo.

Ejemplo 2



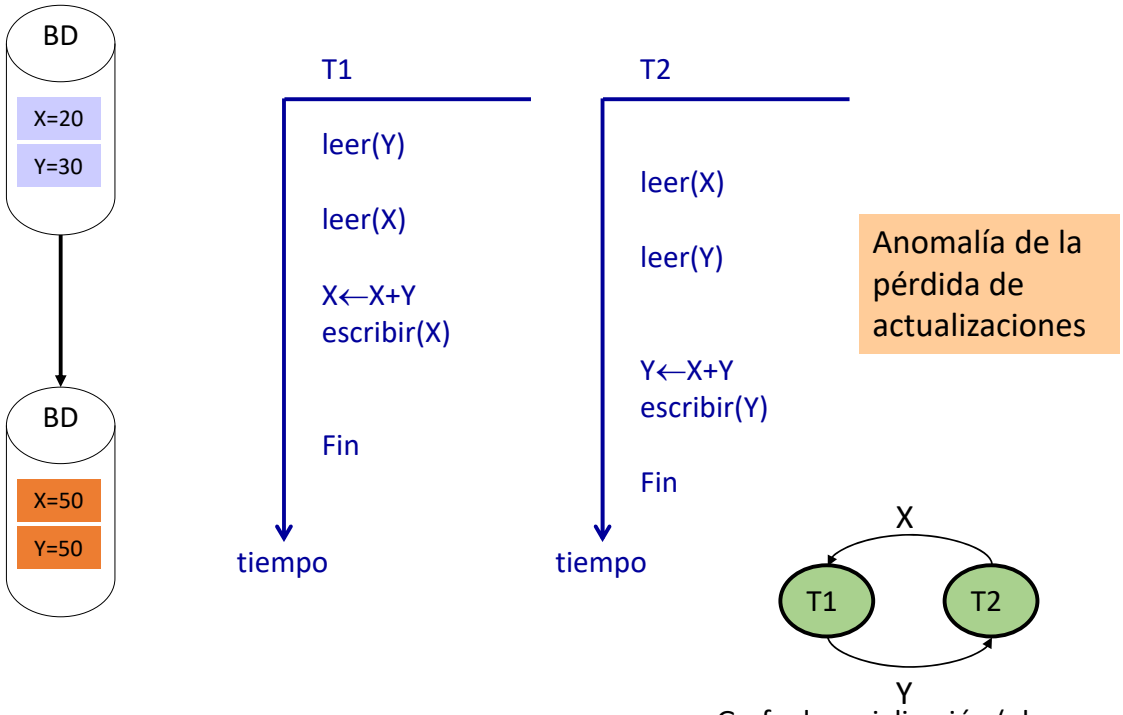
47

Es interesante observar que los dos planes en serie para estas dos transacciones no producen el mismo estado de la base de datos.

4 Protocolos de bloqueo.

Plan concurrente para T1 y T2

Ejemplo 2



48

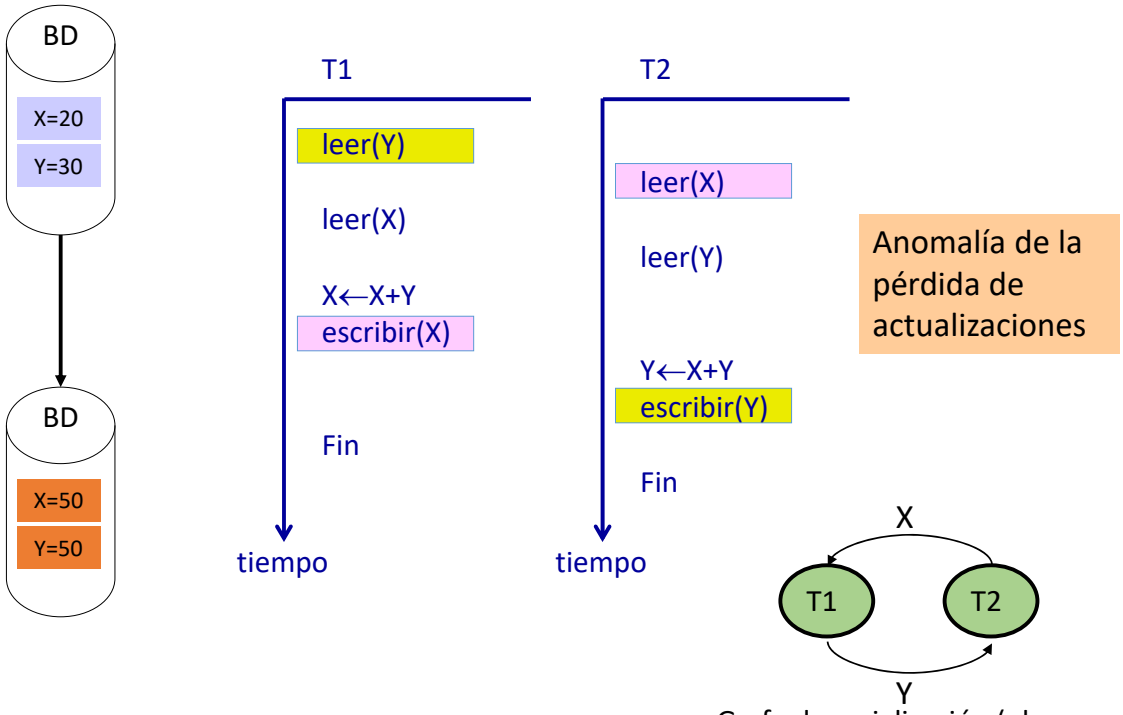
En el plan concurrente que se presenta para T1 y T2, ¿qué anomalía se produce?: la "pérdida de actualizaciones", cada transacción pierde las actualizaciones que la otra transacción hace.

Como demuestra el grafo de serialización, el plan no es serializable por conflictos.

4 Protocolos de bloqueo.

Plan concurrente para T1 y T2

Ejemplo 2



Grafo de serialización (plan no serializable)

49

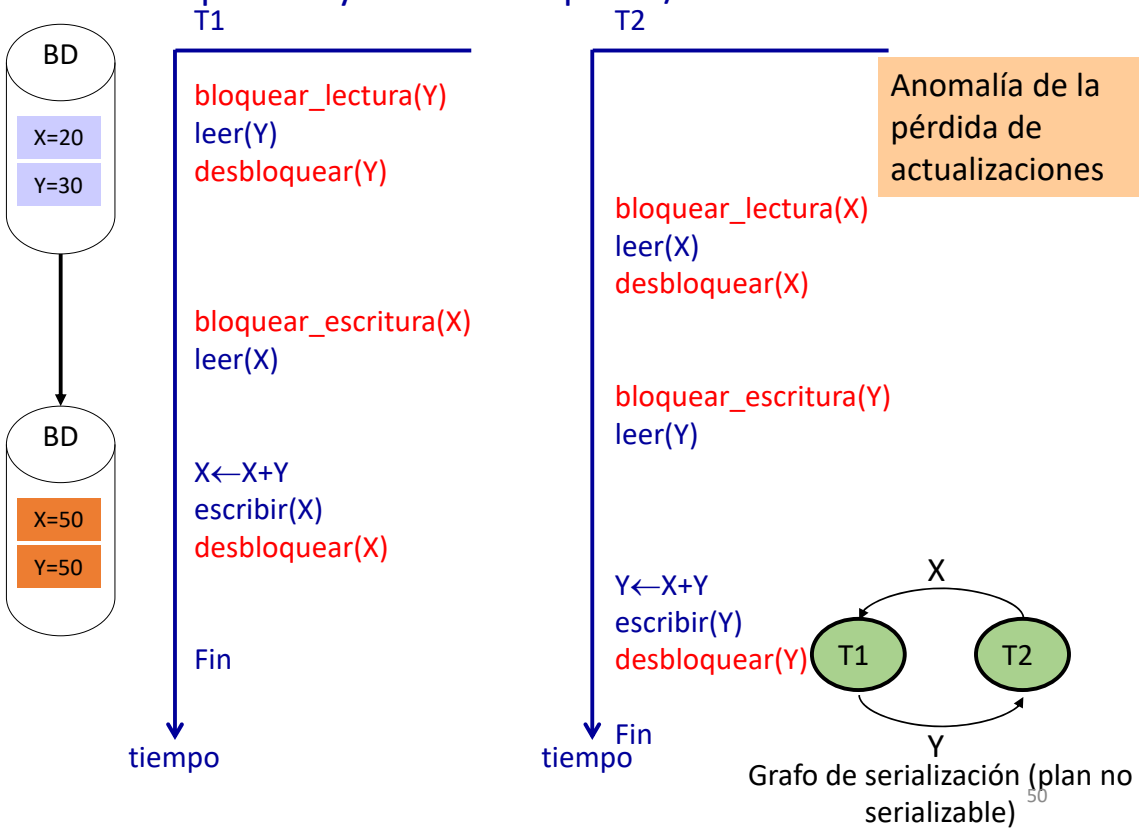
En la transparencia, se destacan dos pares de operaciones en conflicto que aparecen en el plan, y que causan la anomalía.

Es importante observar que, en ambos pares, el orden de las operaciones es contrario. En un par, primero sucede la operación de T1 y luego la operación de T2, y en el otro par primero sucede la operación de T2 y luego la de T1. Es imposible la equivalencia con alguno de los dos planes en serie para T1 y T2.

4 Protocolos de bloqueo.

Ejemplo 2

Plan concurrente para T1 y T2: con bloqueo L/E



En un intento de usar el protocolo de bloqueo de lectura/escritura, se obtiene el plan concurrente que se muestra en la transparencia.

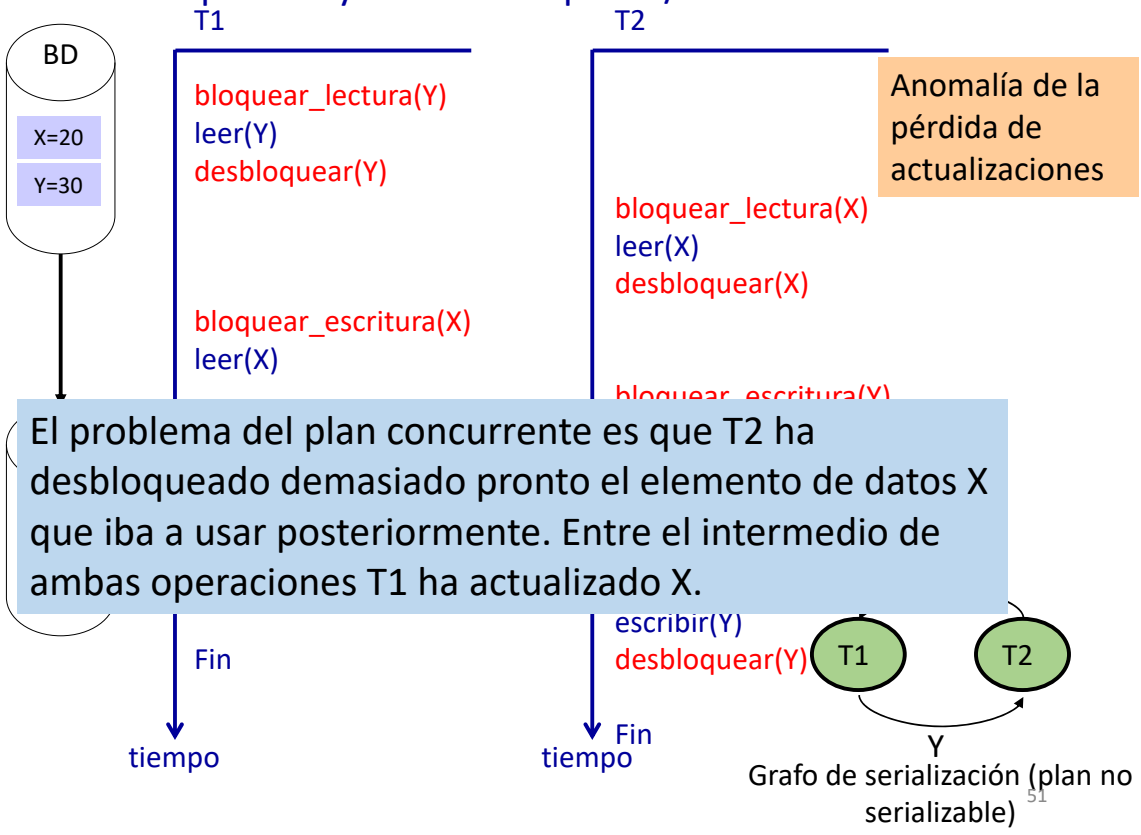
En aplicación del protocolo, antes de solicitar una operación, se solicita el correspondiente bloqueo del elemento de datos y, después de realizar la operación, el elemento de datos se desbloquea, para que sea accesible para otras transacciones.

Se puede observar que el nuevo plan concurrente sigue presentando la anomalía de la "pérdida de actualizaciones", no es serializable por conflictos. **¿Dónde está el problema?**

4 Protocolos de bloqueo.

Ejemplo 2

Plan concurrente para T1 y T2: con bloqueo L/E

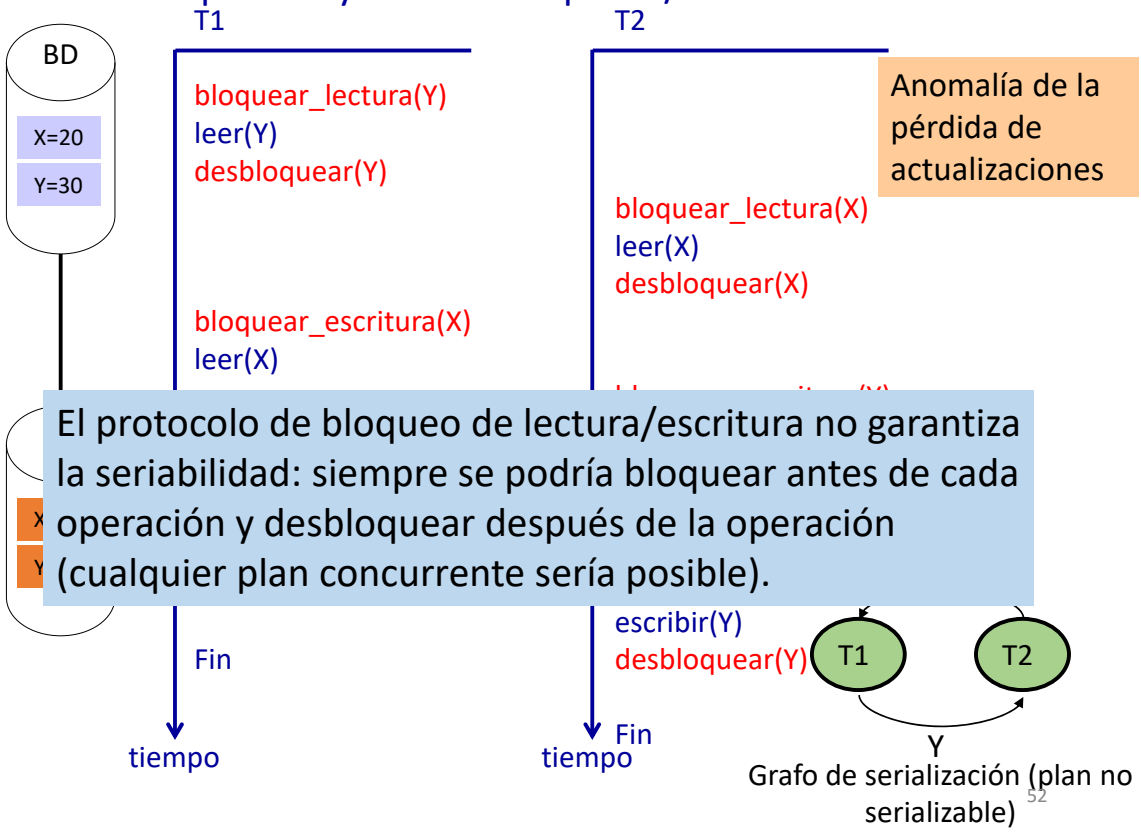


Las operaciones de bloqueo de las transacciones deben estar limitadas por alguna restricción adicional que evite este problema.

4 Protocolos de bloqueo.

Ejemplo 2

Plan concurrente para T1 y T2: con bloqueo L/E



52

Las operaciones de bloqueo de las transacciones deben estar limitadas por alguna restricción adicional que evite este problema.

4 Protocolos de bloqueo.

El protocolo de **bloqueo de lectura/escritura** no garantiza la seriabilidad (por conflictos) de los planes: hace falta una regla adicional.

↓

Bloqueo en dos fase **(B2F)**

↓

Regla de las dos fases (2F): En una transacción todas las operaciones de bloqueo (*bloquear_lectura, bloquear_escritura*) **preceden** a la primera operación de desbloqueo (*desbloquear*) de la transacción.

53

La restricción (regla) adicional que se va a añadir al protocolo de bloqueo lectura/escritura es la **regla de las dos fases (2F).**

Regla 2F: cuando una transacción desbloquea algún elemento de datos ya no puede emitir ninguna operación de bloqueo.

4 Protocolos de bloqueo.

Se puede demostrar que si un plan concurrente es acorde con el protocolo B2F entonces el plan es serializable por conflictos.

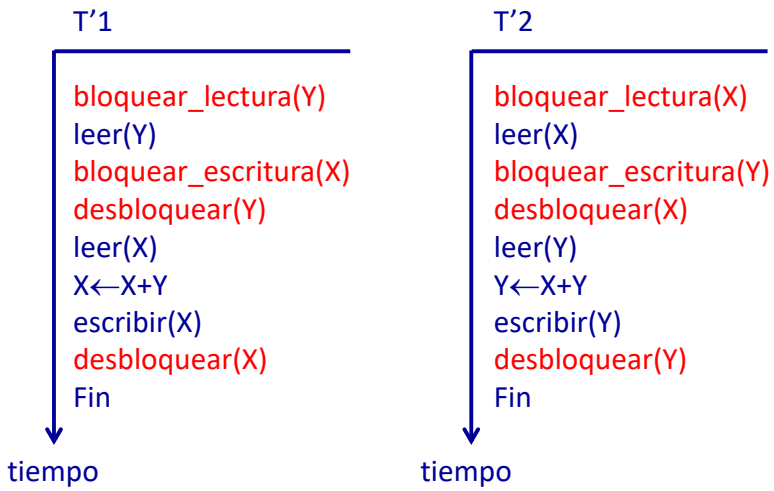
El protocolo de bloqueo lectura/escritura en dos fases (B2F) limita la concurrencia, pero asegura la seriabilidad (por conflictos) de los planes sin tener que examinarlos

54

Si se completa un plan, controlado por el protocolo B2F, se garantiza que el plan es serializable por conflictos, si no se completa es porque el plan se ha quedado en situación de bloqueo mortal, el protocolo deberá resolver esta situación, pero habrá evitado una posible anomalía.

4 Protocolos de bloqueo.

Ejemplo 2



T'1 y T'2 cumplen la regla 2F

55

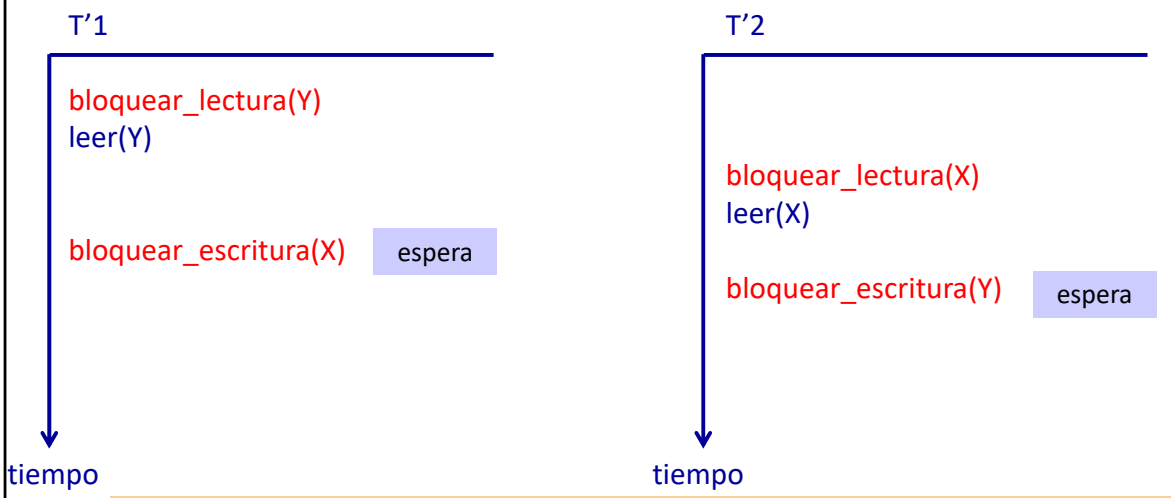
Las dos transacciones cumplen la regla de las dos fases **(2F)**.

¿Cómo podemos intercalar las operaciones de estas dos transacciones que cumplen la regla 2F?: respetando las reglas del bloqueo de lectura/escritura.

4 Protocolos de bloqueo.

Ejemplo 2

Plan concurrente para T'1 y T'2 (con B2F)



- ✓ El SGBD **evita** la anomalía.
- ✓ El plan no es posible: no se puede atender a ninguna de las dos transacciones porque la operación solicitada por ambas (bloqueo) no está permitida por las reglas del bloqueo B2F.
- ✓ Estrategia conservadora.

56

En la transparencia, se muestra un plan concurrente para T'1 y T'2, controlado por el protocolo B2F. Se observa que sólo se pueden intercalar las dos primeras operaciones de ambas transacciones, al intentar ejecutar la segunda operación de cualquiera de ellas se entra en la situación de bloqueo mortal, las dos transacciones se quedan en espera. El protocolo B2F ha evitado la anomalía.

Obsérvese que el protocolo deja en espera a T'1, sin saber todavía qué va a hacer T'2, es decir, sin saber si va a producirse una anomalía,.

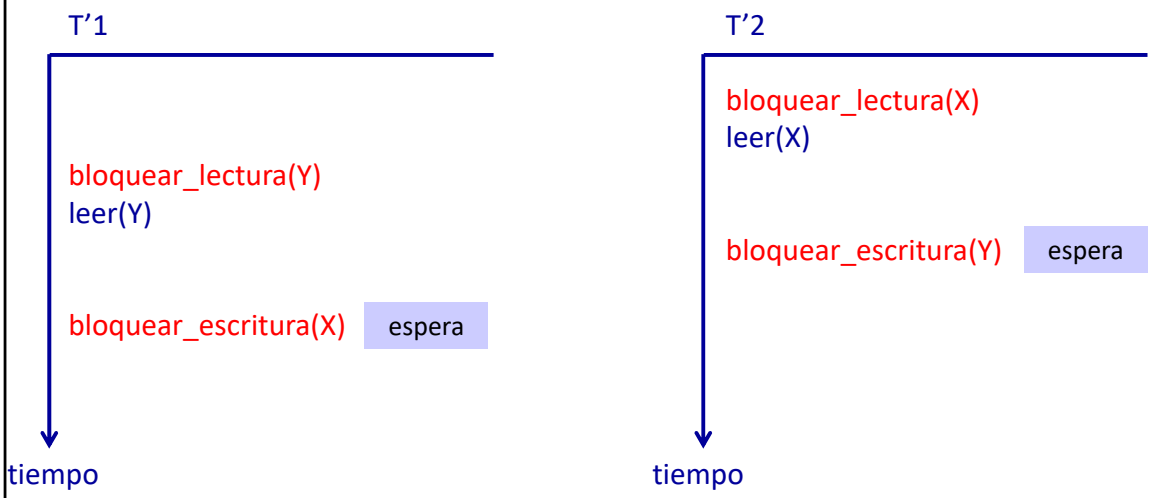
El protocolo B2F evita que en el plan aparezca un primer par de operaciones en conflicto. ¡Pero las operaciones en conflicto no siempre generan anomalías!

Los sistemas previenen las anomalías sin llegar a comprobar que se han producido: política **conservadora**.

4 Protocolos de bloqueo.

Ejemplo 2

Plan concurrente para T'1 y T'2 (con B2F)



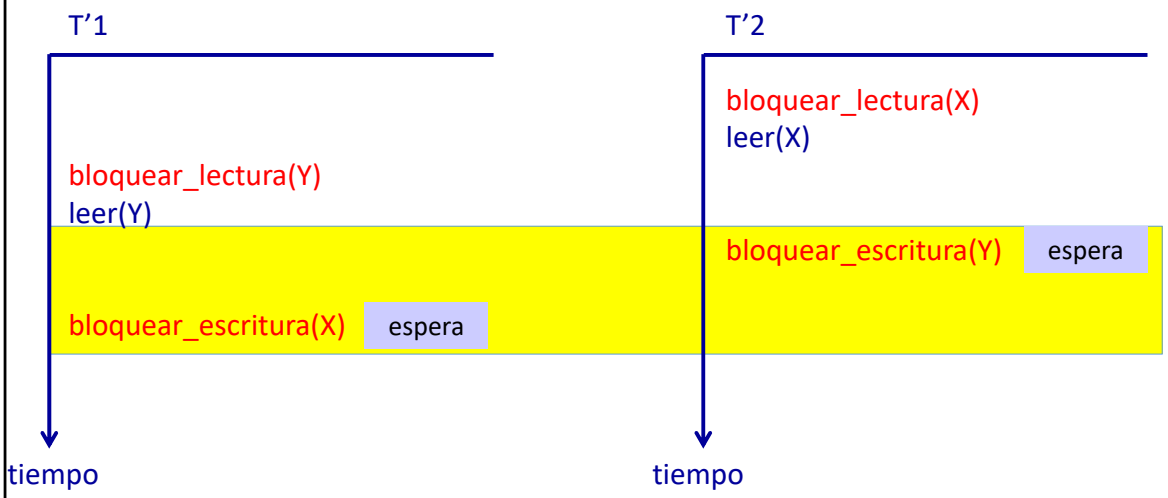
Problema de **bloqueo mortal**: no terminación del plan.

57

4 Protocolos de bloqueo.

Ejemplo 2

Plan concurrente para T'1 y T'2 (con B2F)



El protocolo de bloqueo de lectura/escritura B2F necesita una **regla adicional** para evitar o resolver el problema del bloqueo mortal.

58

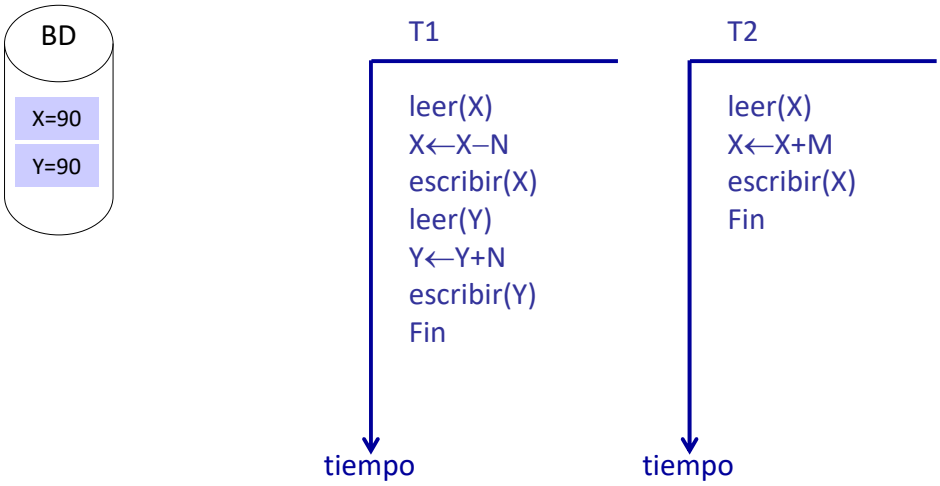
Obviamente, el protocolo B2F debe incluir una **regla adicional** para resolver las situaciones de **bloqueo mortal**.

Existen numerosas soluciones a esta situación. Más adelante, se presentará una de ellas.

4 Protocolos de bloqueo.

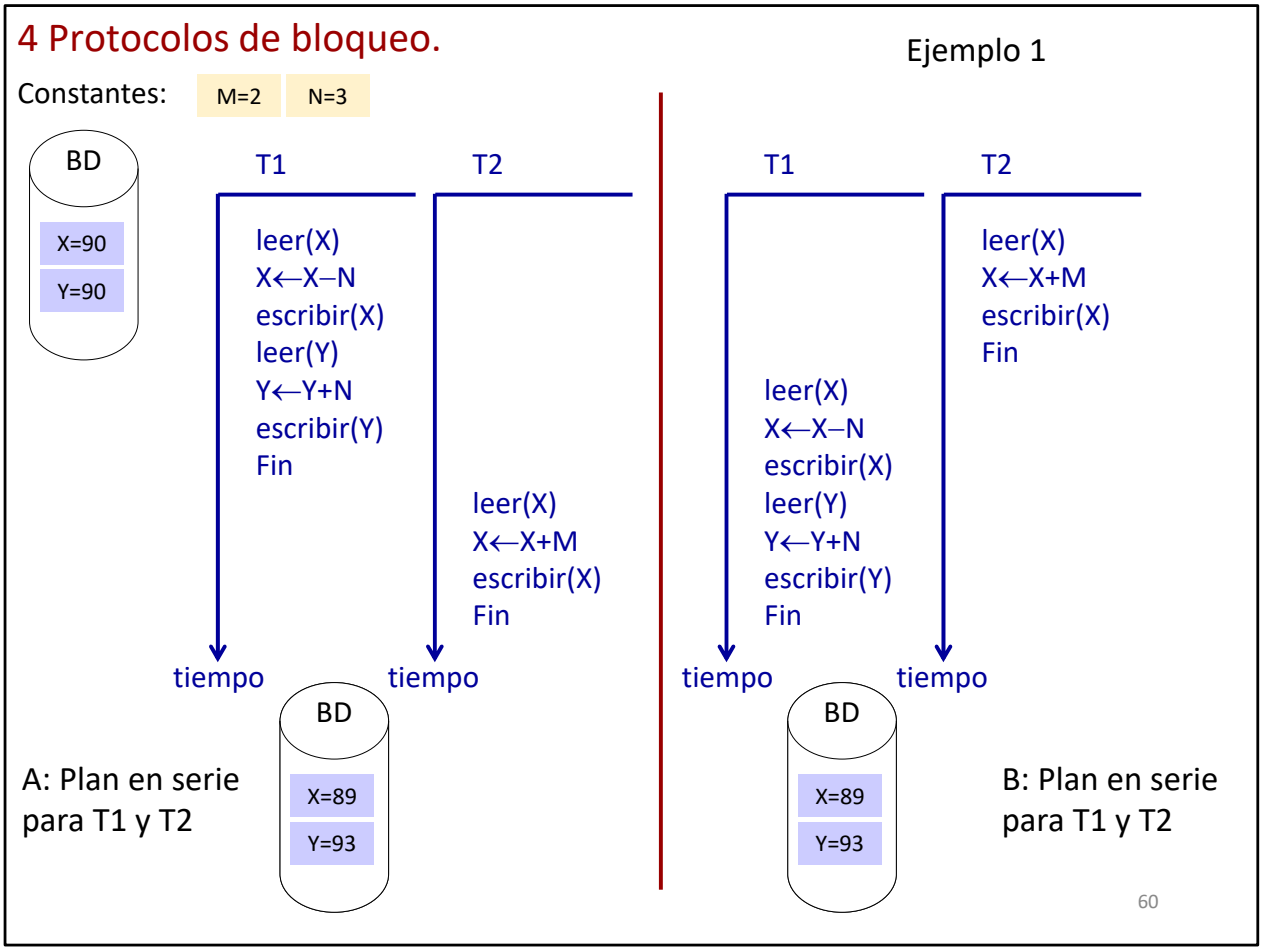
Ejemplo 1

Constantes: M=2 N=3



59

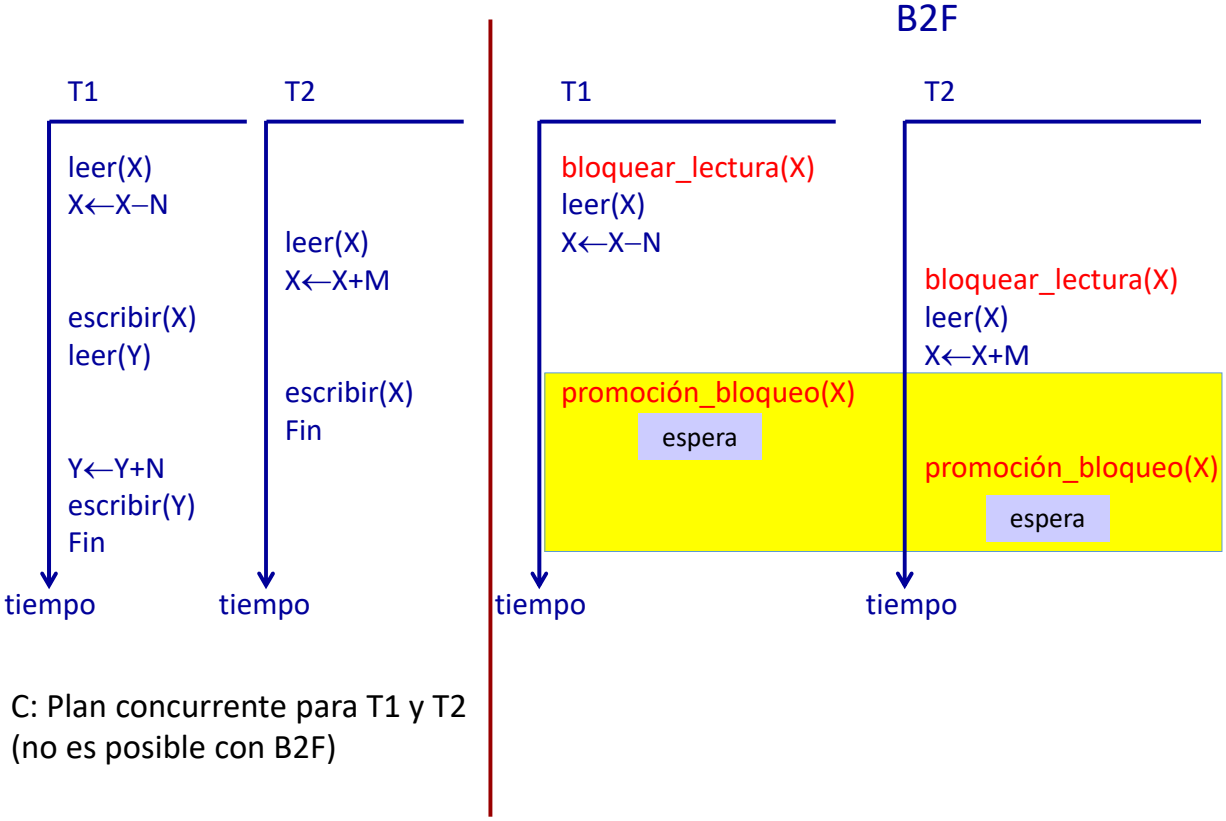
Sean las transacciones T1 y T2 del primer ejemplo usado en el tema.



Estos son los dos planes en serie para T1 y T2.

4 Protocolos de bloqueo.

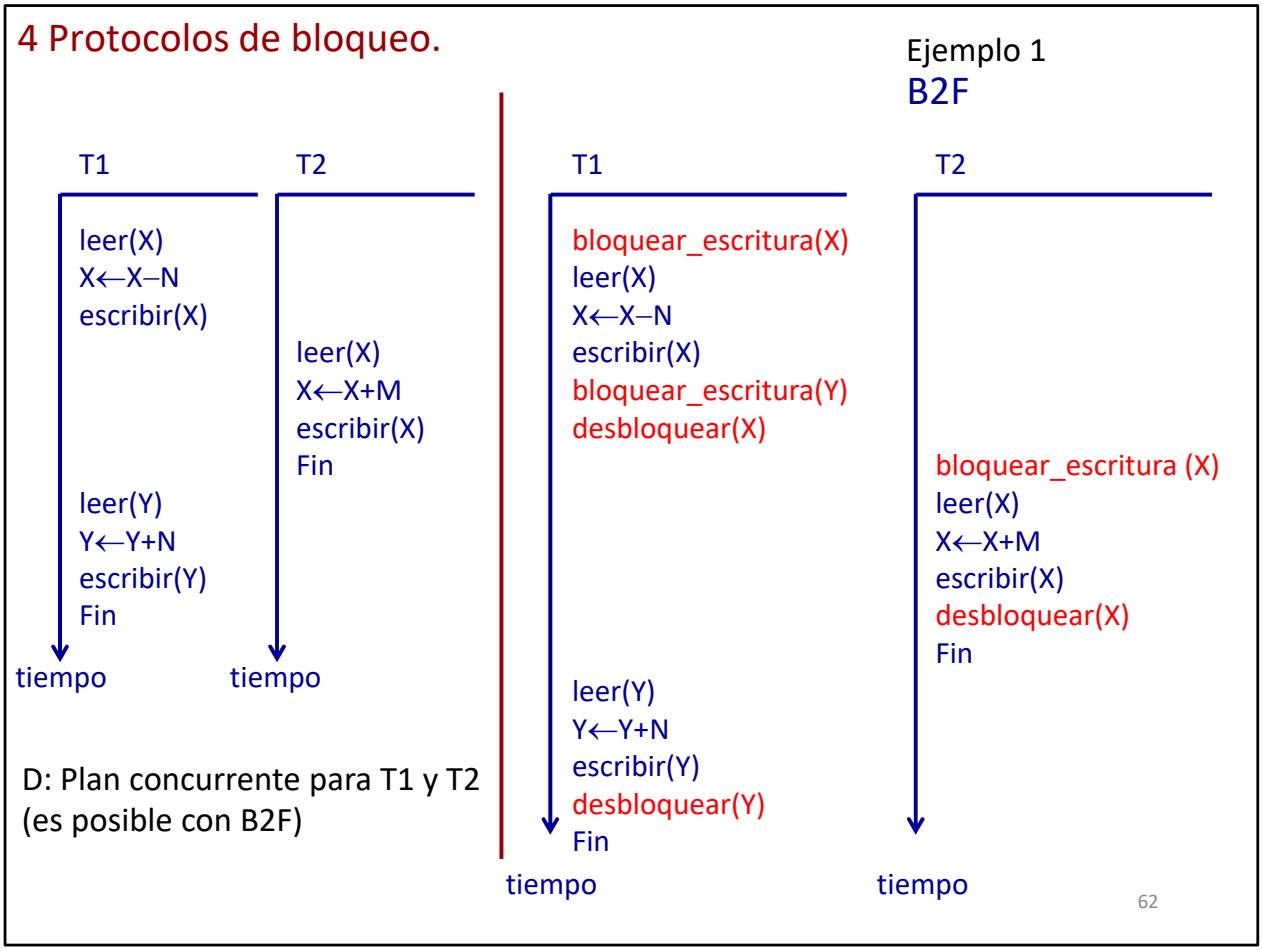
Ejemplo 1
B2F



C: Plan concurrente para T1 y T2
(no es posible con B2F)

61

El plan concurrente C está prohibido por el protocolo B2F. Las reglas de bloqueo dejan en espera a las dos transacciones, el protocolo evita la anomalía de la “pérdida de actualizaciones”, y conduce a una situación de bloqueo mortal.



El plan D, serializable por conflictos, está permitido por el protocolo B2F.

En el plan D, se produce la anomalía de la “lectura sucia”, pero, como ya se ha dicho, ésta es una seudoanomalía que se estudiará más adelante (punto 7).

4 Protocolos de bloqueo.

B2F Implícito: evita tener que escribir las instrucciones de bloqueos, desbloqueos y promociones.

El SGBD se encarga de generar implícitamente los bloqueos de lectura y de escritura y los desbloqueos sobre elementos de datos:

- ✓ Leer(X) genera un bloqueo para lectura sobre X.
- ✓ Escribir(X) genera un bloqueo para escritura sobre X.
- ✓ La finalización de la transacción (anulación o confirmación) genera el desbloqueo de todos los elementos de datos bloqueados por la transacción.

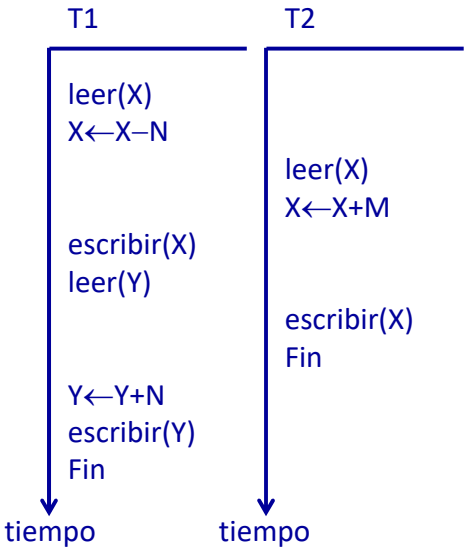
63

En el lenguaje SQL no existen operaciones de bloqueo y desbloqueo, se hacen de forma implícita.

En los SGBD comerciales, se pueden ofrecer operadores de bloqueo para casos especiales, pero también se trabaja con bloqueo implícito.

El bloqueo implícito reduce la concurrencia, ya que impide que una transacción libere sus bloqueos antes de finalizar, pero facilita la escritura de transacciones.

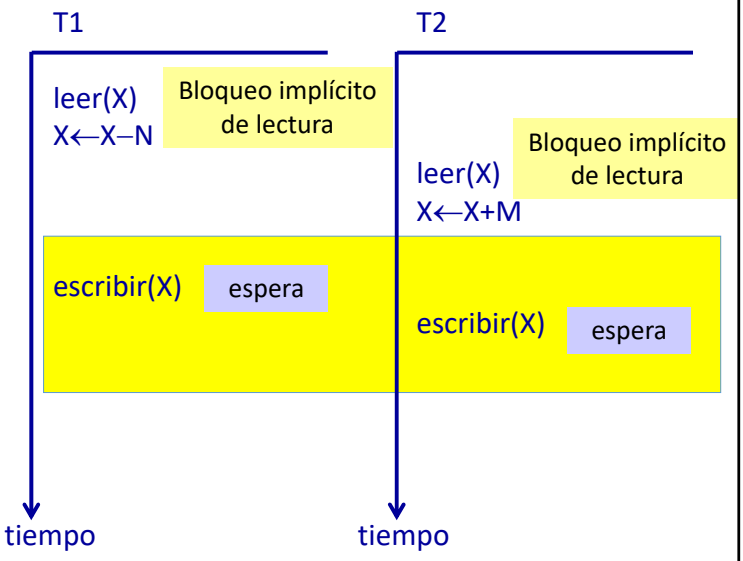
4 Protocolos de bloqueo.



C: Plan concurrente para T1 y T2
(no es posible con B2F implícito)

Ejemplo 1

B2F Implícito



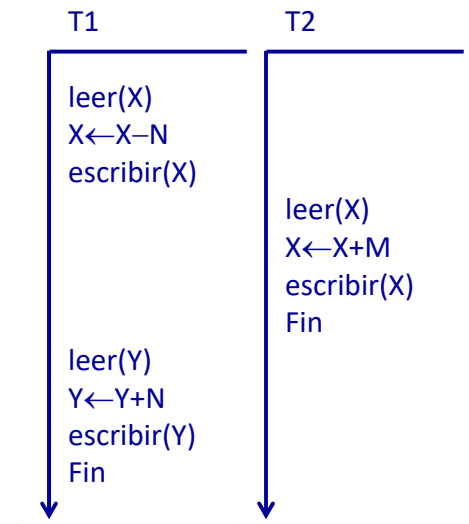
64

El plan C está prohibido por el protocolo B2F Implícito.

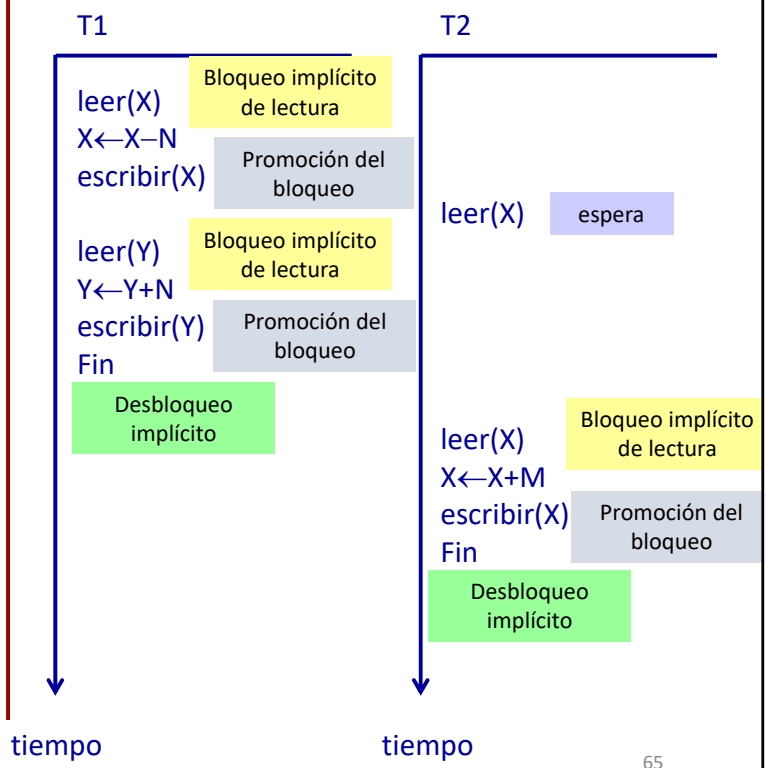
4 Protocolos de bloqueo.

Ejemplo 1

B2F Implícito



D: Plan concurrente para T1 y T2

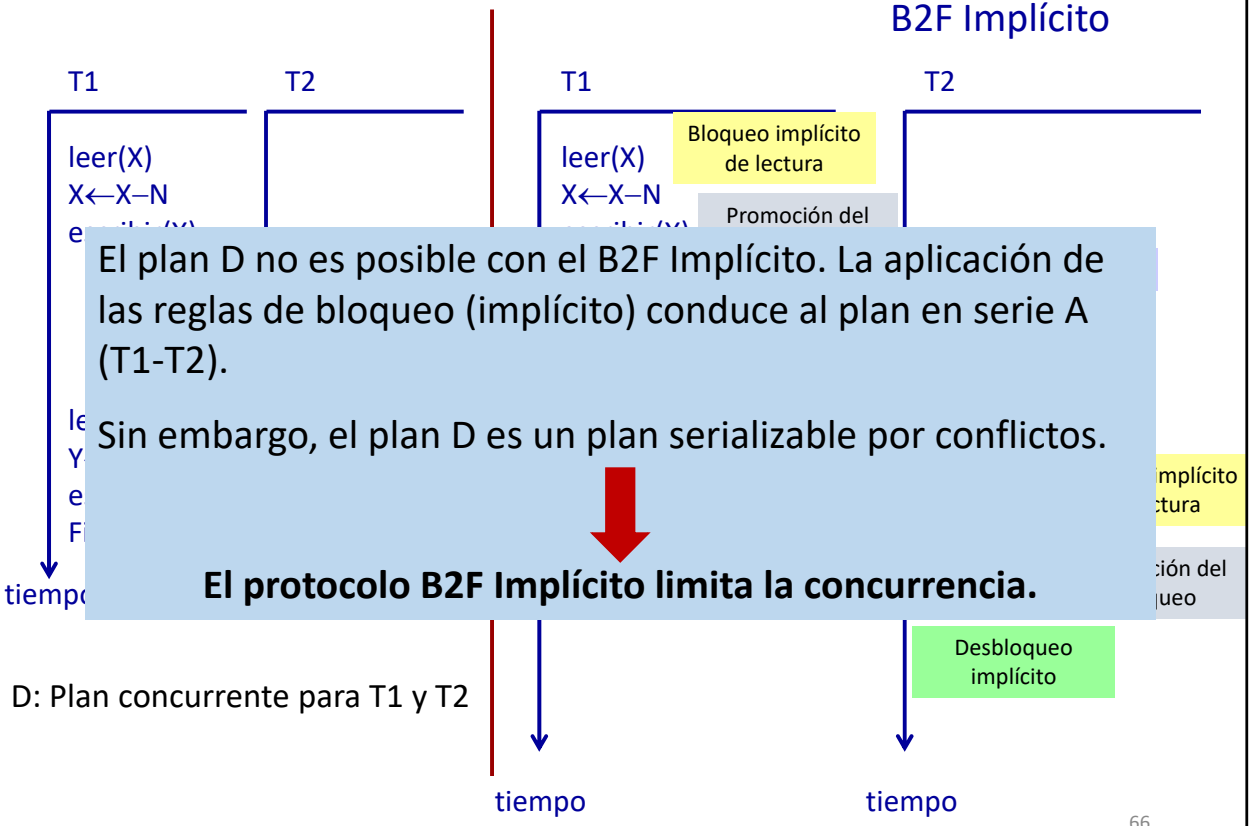


65

4 Protocolos de bloqueo.

Ejemplo 1

B2F Implícito



El plan D no es posible con el B2F Implícito. La aplicación de las reglas de bloqueo (implícito) conduce al plan en serie A (T1-T2).

Sin embargo, el plan D es un plan serializable por conflictos.

El protocolo B2F Implícito limita la concurrencia.

Desbloqueo implícito

D: Plan concurrente para T1 y T2

tiempo

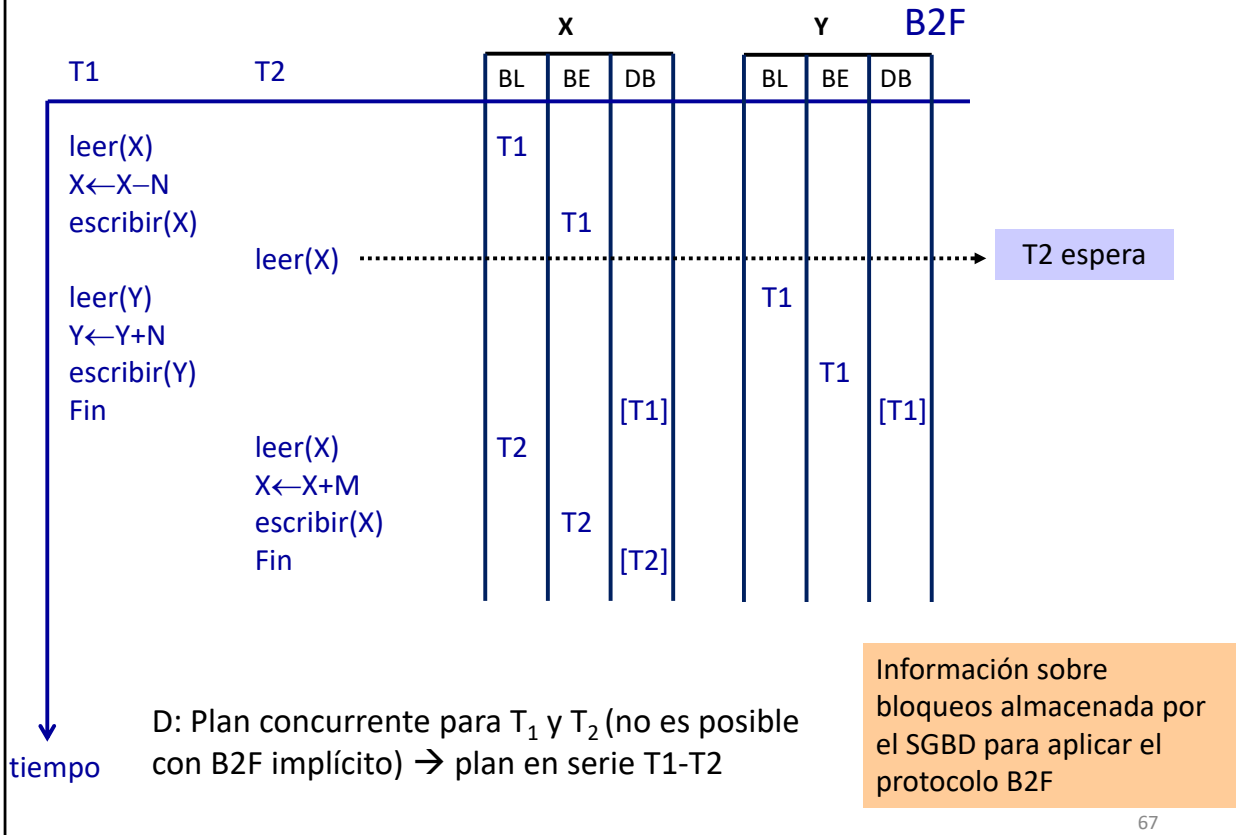
tiempo

66

4 Protocolos de bloqueo.

Ejemplo 1

B2F



D: Plan concurrente para T_1 y T_2 (no es posible con B2F implícito) → plan en serie T1-T2

Información sobre bloqueos almacenada por el SGBD para aplicar el protocolo B2F

67

En la transparencia:

- BL: Bloqueo en Lectura
- BE: Bloqueo en Escritura
- DB: Desbloqueo

Los SGBD aplican los protocolos de control de la concurrencia dinámicamente, durante la ejecución del plan. Los protocolos no analizan el plan completo sino que responden (aceptando o rechazando) a cada nueva solicitud (lectura o escritura) de las transacciones que participan en el plan.

Por ello, para poder aplicar los protocolos, los SGBD guardan toda la información, relevante para el protocolo, que se va producido durante la ejecución del plan, sin necesidad de guardar el plan completo.

En el protocolo B2F, la única información, que el SGBD necesita para aplicar las reglas de bloqueo, es el estado de bloqueo de los elementos de datos accedidos por las transacciones del plan.

La forma en que el SGBD guarda esta información no es relevante. En la transparencia se muestra cuál es esta información.

4 Protocolos de bloqueo.

Bloqueo mortal: cada transacción T_i en un conjunto de dos o más transacciones está esperando un elemento de datos que está bloqueado por otra transacción T_j de dicho conjunto.

Algoritmos de detección del bloqueo mortal: el sistema verifica si el plan está en un estado de bloqueo mortal.

68

Como se ha dicho anteriormente, en los protocolos de control de la concurrencia, basados en el bloqueo de elementos de datos, se puede presentar la situación de bloqueo mortal.

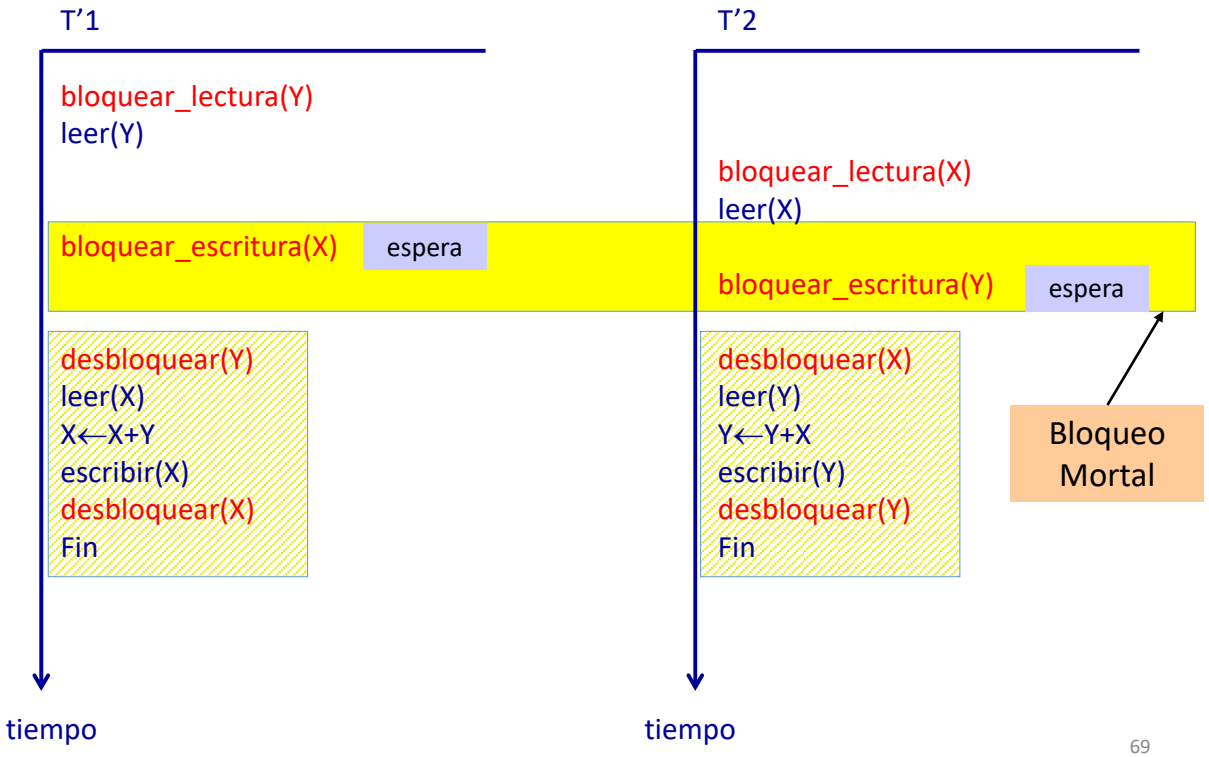
Por lo tanto, todo protocolo de bloqueo debe incorporar una regla adicional para resolver estas situaciones.

4 Protocolos de bloqueo.

Ejemplo 2

Plan concurrente para T'1 y T'2 con bloqueo mortal

B2F Explícito



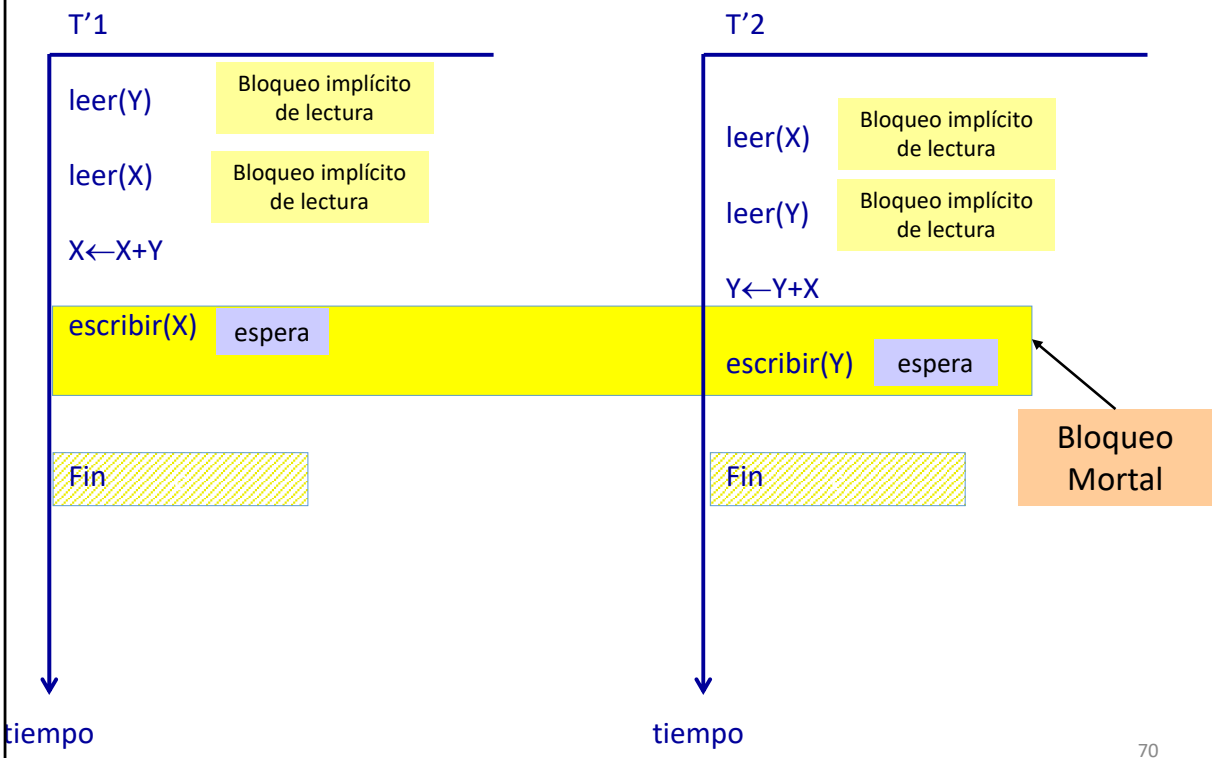
69

4 Protocolos de bloqueo.

Ejemplo 2

Plan concurrente para T'1 y T'2 con bloqueo mortal

B2F Implícito



70

4 Protocolos de bloqueo.

Uso del grafo de espera:

Grafo de espera: grafo dirigido G=(N,A), que consiste en un conjunto de nodos N y un conjunto de arcos A.

- ✓ El grafo contiene un nodo por cada transacción que se está ejecutando actualmente.
- ✓ El arco $a_{ij} = (T_i \rightarrow T_j)$ se crea si la transacción T_i está esperando bloquear un elemento X que está bloqueado por la transacción T_j (uno de los dos bloqueos es exclusivo). Cuando T_j libera el elemento X se borra el arco.

↓

Un plan concurrente, controlado por un protocolo de bloqueo, está en una situación de bloqueo mortal si su grafo de espera tiene un ciclo.

↓

Si el plan está en un estado de bloqueo mortal será preciso abortar alguna de las transacciones que lo están provocando.

71

En esta técnica de detección del bloqueo mortal, el sistema debe mantener el grafo de espera. Además, el sistema debe tener un criterio para decidir **cuándo** verifica si el grafo tiene un ciclo.

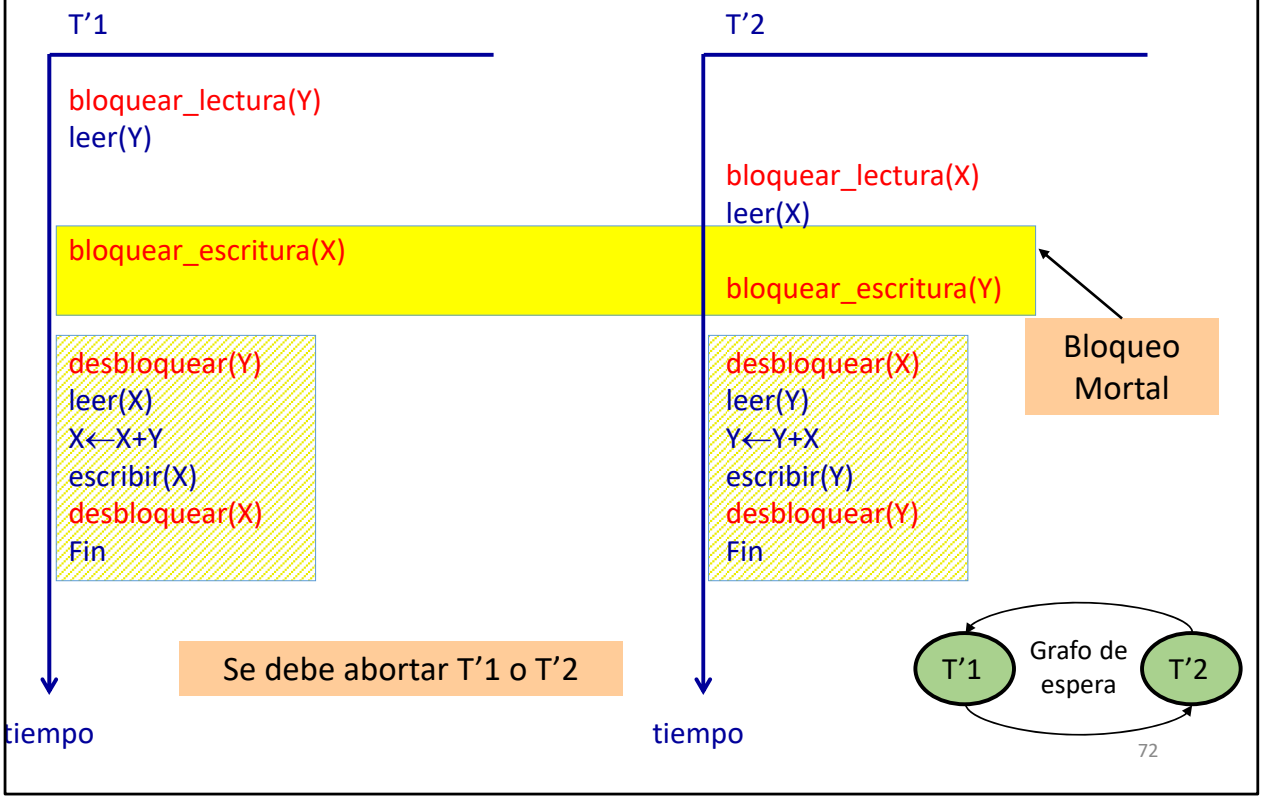
Un criterio podría ser considerar el número de transacciones que están esperando bloquear un elemento de datos, durante un tiempo mayor que un tiempo fijado por el sistema. Otro criterio podría ser verificar el grafo cada vez que se modifica.

4 Protocolos de bloqueo.

Ejemplo 2

Plan concurrente para T'1 y T'2 con bloqueo mortal

B2F Explícito



72

Operaciones para usar bloqueo de Lectura/Escritura:

```
Bloquear_escritura (X):
B: IF bloqueo(X)='desbloqueado' (el elemento está desbloqueado)
    THEN  bloqueo(X) ← 'bloqueado para escritura'
          añadir T a la lista de transacciones del elemento X
     ELSE  WAIT (hasta que bloqueo(X)='desbloqueado' y el
                 gestor de bloqueos considere a T)
           GOTO B
     END IF.
```

4 Protocolos de bloqueo.

Operaciones para usar bloqueo de Lectura/Escritura:

```
Bloquear_lectura (X):
B: IF bloqueo(X) = 'desbloqueado' (el elemento está desbloqueado)
    THEN  bloqueo(X) ← 'bloqueado para lectura'
          nro_lecturas ← 1
          añadir T a la lista de transacciones del elemento X
     ELSE
          IF bloqueo(X) = 'bloqueado para lectura'
          THEN  nro_lecturas ← nro_lecturas + 1
                añadir T a la lista de transacciones del
                  elemento X
          ELSE WAIT (hasta que bloqueo(X) = 'desbloqueado' y el
                      gestor de bloqueos considere a T)
               GOTO B
          END IF
     END IF.
```

Operaciones para usar bloqueo de Lectura/Escritura:

```
Desbloquear (X):

IF bloqueo(X)='bloqueado para escritura'
THEN   bloqueo(X) ← 'desbloqueado'
       eliminar T de la lista de transacciones del elemento X
         --considerar alguna transacción que espera para bloquear X--
ELSE
    IF bloqueo(X) = 'bloqueado para lectura'
    THEN  nro_lecturas ← nro_lecturas – 1
          eliminar T de la lista de transacciones del elemento X
          IF nro_lecturas=0
          THEN bloqueo(X) ← 'desbloqueado'
               -- considerar alguna transacción que espera para
                            bloquear X --
          END IF
       END IF
END IF
```

4 Protocolos de bloqueo.

Información de bloqueo de elementos de datos: dos opciones

En cada elemento de datos.

<elemento_datos, bloqueo, nro_lecturas, transacciones >

En una tabla con los elementos de datos bloqueados.

elemento_datos1,	bloqueo1,	nro_lecturas1,	transacciones1
elemento_datos2,	bloqueo2,	nro_lecturas2,	transacciones2
elemento_datos3,	bloqueo3,	nro_lecturas3,	transacciones3
…			

76

En los registros anteriores:

- **Bloqueo**: es el estado de bloqueo del elemento de datos.
- **Nro_lecturas:** es el número de transacciones que poseen un bloqueo de lectura sobre el elemento de datos (sólo tiene sentido cuando el bloqueo es de lectura).
- **Transacciones:** es una lista con las transacciones que poseen un bloqueo sobre el elemento de datos. Si el bloqueo es de escritura esta lista contiene un único elemento.

Control de la concurrencia.

1 Ejecución concurrente de transacciones: anomalías.

2 Control de la concurrencia en SQL.

3 Planes serializables por conflictos.

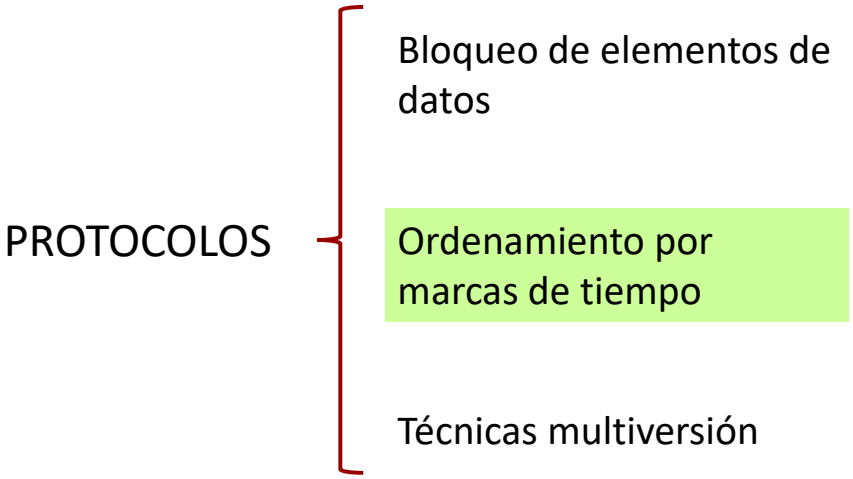
4 Protocolos de bloqueo.

5 Protocolos de ordenamiento por marcas de tiempo.

6 Protocolos multiversión.

7 Concurrencia y recuperación.

5 Protocolos de ordenamiento por marcas de tiempo.



78

5 Protocolos de ordenamiento por marcas de tiempo.

Protocolos de ordenamiento por marcas de tiempo (OMT):

Los protocolos de ordenamiento por marcas de tiempo se basan en la idea de usar las marcas de tiempo de las transacciones para **controlar el orden de ejecución de las operaciones en conflicto** dentro de un plan.

79

Los protocolos de ordenamiento por marcas de tiempo (OMT) sólo admiten planes concurrentes que sean equivalentes por conflictos al plan en serie en el que las transacciones se ejecutan en el orden en que se inician en el sistema (plan en serie cronológico).

Para alcanzar este objetivo, los protocolos controlan, dinámicamente, el **orden** de ejecución **de las operaciones en conflicto,** dentro del plan, usando para ello las **marcas de tiempo de las transacciones** a las que pertenecen dichas operaciones.

A continuación, se presenta el concepto de marca de tiempo de una transacción.

5 Protocolos de ordenamiento por marcas de tiempo.

Protocolos de ordenamiento por marcas de tiempo (OMT):

Marca de tiempo de una transacción: Valor que el SGBD asigna a cada transacción (representa el instante de tiempo en que se inicia la transacción).

- ✓ Las marcas de tiempo están ordenadas según el orden en que las transacciones se inician en el sistema. (Si T empieza **antes** que T' entonces la marca de tiempo de la transacción T es **menor** que la marca de tiempo de la transacción T'). Se dice que T es más vieja que T' o que T' es más joven que T.
- ✓ Las marcas de tiempo se pueden generar:
  - Usando un contador (entero) que se incrementa cada vez que se inicia una nueva transacción.
  - Utilizando el reloj del sistema.
- ✓ **La marca de tiempo de una transacción no cambia nunca.**

80

5 Protocolos de ordenamiento por marcas de tiempo.

Protocolos de ordenamiento por marcas de tiempo (OMT):

Plan en serie cronológico para un plan concurrente: Plan en serie en el que las transacciones se procesan siguiendo el orden de sus marcas de tiempo en el plan concurrente.

Un plan concurrente, admitido por el protocolo de ordenamiento por marcas de tiempo, es **equivalente por conflictos** a su plan en serie cronológico.

Para cualquier par de operaciones en conflicto en el plan, se debe ejecutar primero la perteneciente a la transacción más vieja (con marca de tiempo menor).

81

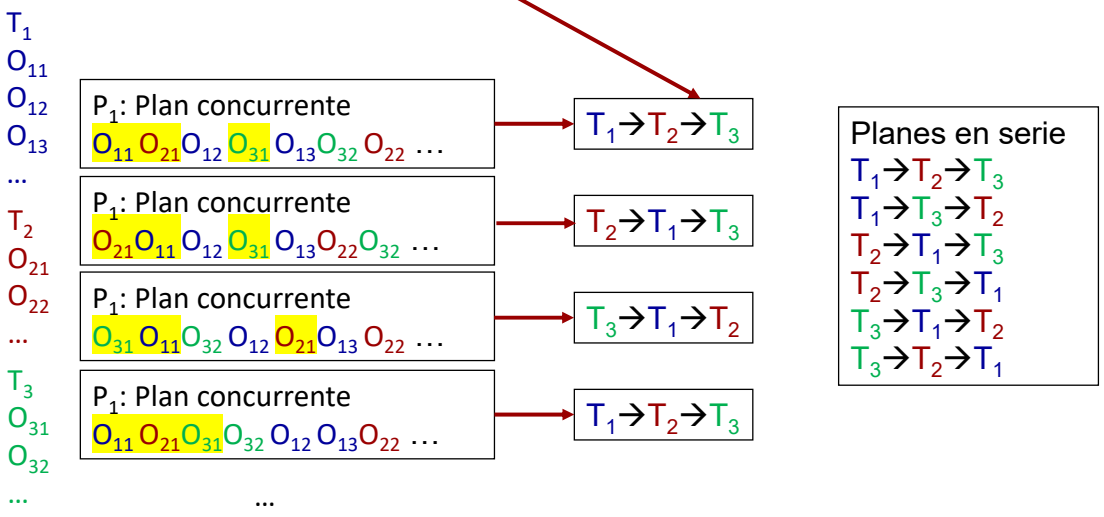
Para que un plan concurrente pueda ser admitido según este protocolo, no puede ser equivalente por conflictos a cualquier plan en serie sino sólo a uno, a su plan en serie cronológico.

Por ello, cuando aparece un par de operaciones en conflicto en el plan, el protocolo exige que la primera operación del par sea la de la transacción más vieja (con marca de tiempo menor).

3 Planes serializables por conflictos.

Protocolos de ordenamiento por marcas de tiempo (OMT):

Plan en serie cronológico para un plan concurrente: Plan en serie en el que las transacciones se procesan siguiendo el orden de sus marcas de tiempo en el plan concurrente.



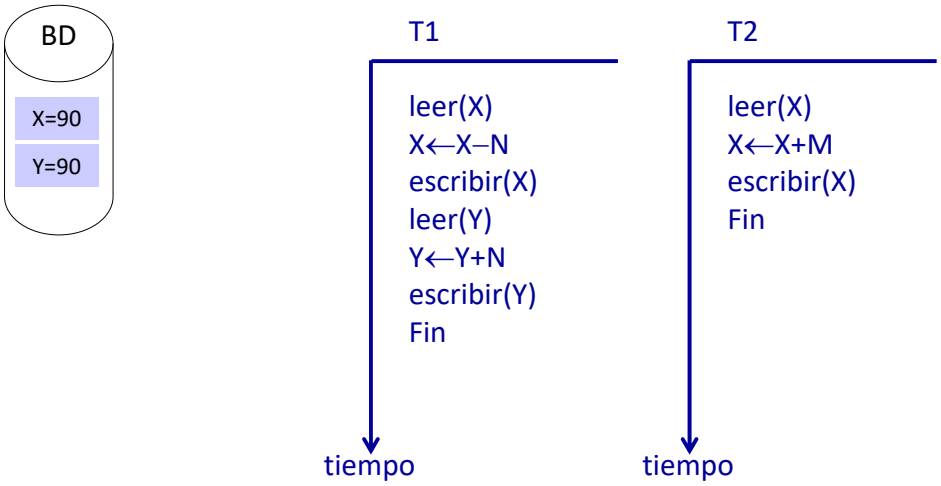
O_{ij}: Operación *j* de la transacción *i*

82

5 Protocolos de ordenamiento por marcas de tiempo.

Ejemplo 1

Constantes: M=2 N=3

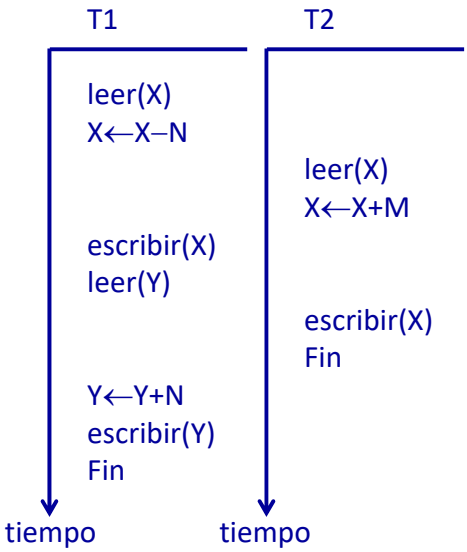


83

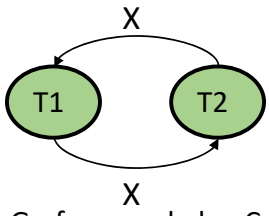
Sean las dos transacciones T1 y T2 del ejemplo 1, utilizado anteriormente en el tema.

5 Protocolos de ordenamiento por marcas de tiempo.

Ejemplo 1



C: Plan concurrente para T1 y T2



Grafo para el plan C

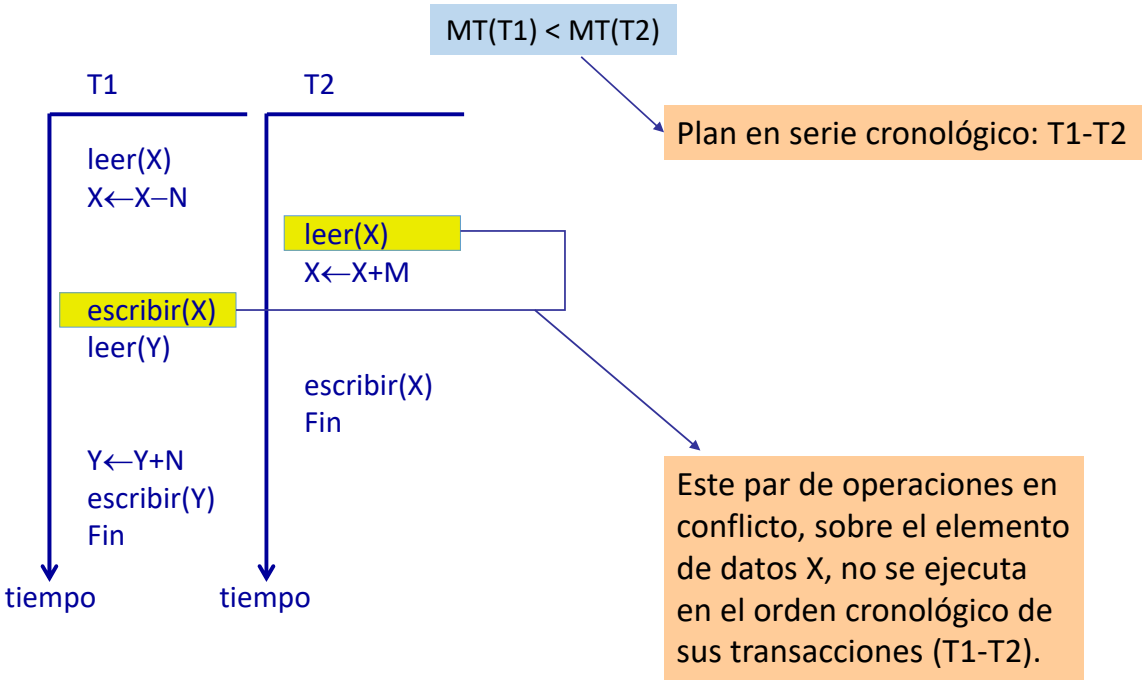
El plan C no es serializable por conflictos: se produce la anomalía de la pérdida de actualizaciones.

84

Como ya se vio anteriormente, el plan concurrente C no es serializable por conflictos, como demuestra el grafo de serialización. Se produce la anomalía de “pérdida de actualizaciones”.

5 Protocolos de ordenamiento por marcas de tiempo.

Ejemplo 1



Plan C: no es admitido por el protocolo OMT

85

En el plan C, MT(T1)<MT(T2), es decir, la transacción T1 se ha iniciado en el sistema antes que la transacción T2. El plan en serie cronológico asociado a ese plan concurrente es T1-T2.

El protocolo OMT controla el orden de aparición de las operaciones en conflicto en el plan.

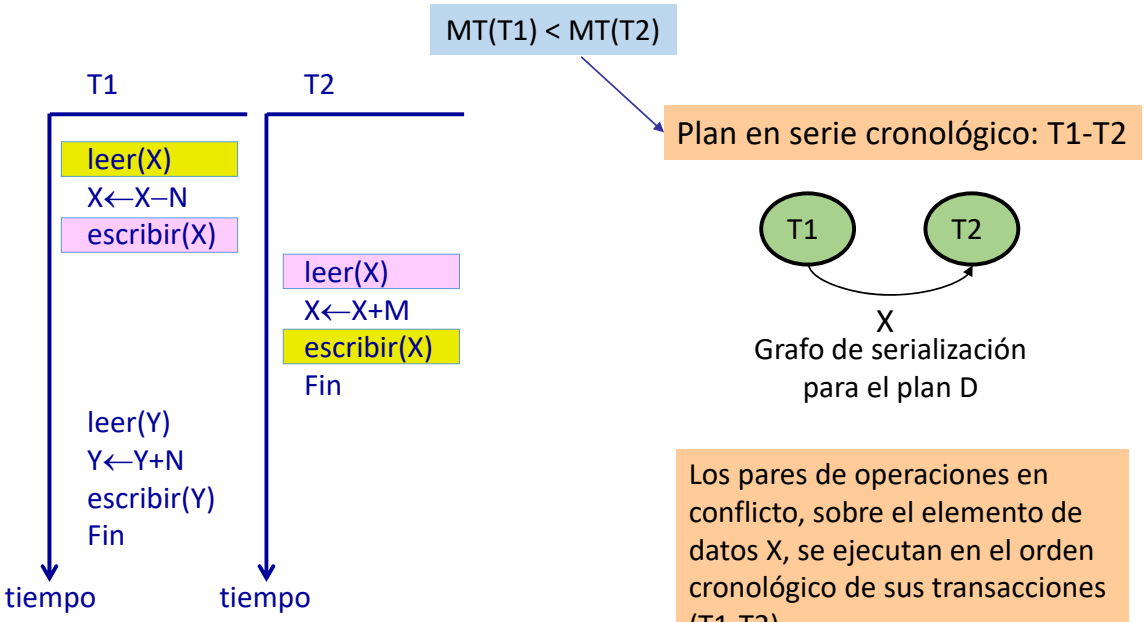
La operación escribir(X) de T1 entra en conflicto con la operación leer(X) de T2 que ya se ha producido. El protocolo controla el **orden** de este par de operaciones y, como no coincide con el orden cronológico de las transacciones (primero la operación de T1 y luego la de T2), rechazará la transacción T1.

Es importante observar que, cuando se rechaza T1, no se ha producido ninguna anomalía. El protocolo previene la aparición de anomalías: política conservadora.

La guía de referencia para controlar el plan es el plan en serie cronológico T1-T2.

5 Protocolos de ordenamiento por marcas de tiempo.

Ejemplo 1



Los pares de operaciones en conflicto, sobre el elemento de datos X, se ejecutan en el orden cronológico de sus transacciones (T1-T2).

Plan D: admitido por el protocolo OMT

86

En el plan D, MT(T1)<MT(T2), es decir, la transacción T1 se ha iniciado, en el sistema, antes que la transacción T2. El plan en serie cronológico asociado a ese plan concurrente es T1-T2.

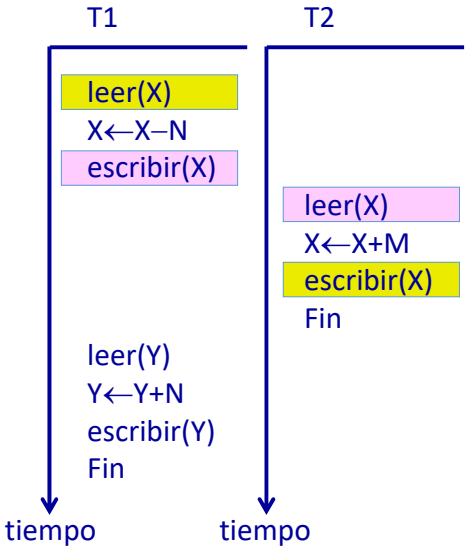
El protocolo OMT controla el orden de aparición de las operaciones en conflicto en el plan.

En el plan D, en todos los pares de operaciones en conflicto, primero ocurre la operación de T1 y luego la operación de T2, el plan es equivalente por conflictos al plan en serie cronológico T1-T2 El plan es admitido por el protocolo OMT.

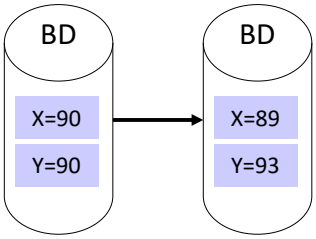
5 Protocolos de ordenamiento por marcas de tiempo.

Ejemplo 1

MT(T1) < MT(T2)



Constantes: M=2 N=3



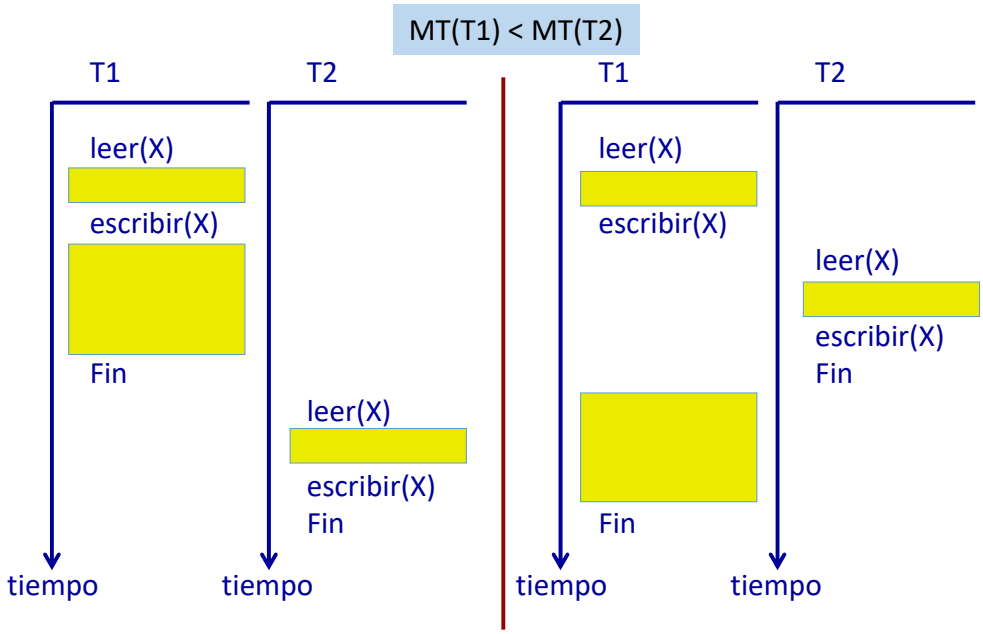
El plan concurrente D es equivalente por conflictos al plan en serie (**cronológico**) T1-T2, determinado por las marcas de tiempo de las transacciones.

Plan D: admitido por el protocolo OMT

87

5 Protocolos de ordenamiento por marcas de tiempo.

Ejemplo 1



A: Plan en serie T1-T2

D: Plan concurrente para T1 y T2

88

El plan D se comporta, en las operaciones en conflicto, igual que el plan en serie T1-T2.

5 Protocolos de ordenamiento por marcas de tiempo.

Protocolos de ordenamiento por marcas de tiempo (OMT): información mantenida por el sistema

Marcas de tiempo de un elemento de datos X: información que el SGBD necesita guardar para controlar el orden de ejecución de las operaciones en conflicto.

- ✓ **MT_lectura (X):** es la **mayor** de todas las marcas de tiempo de las transacciones que han leído con éxito el elemento X: marca de tiempo de la transacción más joven que ha leído con éxito el elemento de datos X.
- ✓ **MT_escritura (X):** es la **mayor** de todas las marcas de tiempo de las transacciones que han escrito con éxito el elemento X: marca de tiempo de la transacción más joven que ha escrito con éxito el elemento de datos X.

89

Como ya se ha comentado anteriormente, los protocolos aplican sus reglas dinámicamente, es decir, durante la ejecución del plan, no esperan a que el plan finalice (plan completo).

Ante cualquier solicitud de una transacción (lectura, escritura), los protocolos deben decidir si la operación es aceptada o rechazada, sin analizar el plan construido hasta ese momento.

Por este motivo, los **protocolos necesitan guardar** la **información** necesaria para aplicar su estrategia. Esta información varía de un protocolo a otro.

En el caso del protocolo OMT, esta información es la **marca de tiempo de la transacción más joven** que ha **leído** (marca de tiempo de lectura) y que ha **escrito** (marca de tiempo de escritura) cada elemento de datos accedido durante la ejecución del plan.

Es importante observar que el sistema no guarda información sobre las operaciones que las transacciones han realizado sobre los elementos de datos, es suficiente guardar las marcas de tiempo de lectura y de escritura de cada elemento de datos accedido.

La transacción más joven, que ha leído un elemento de datos no es necesariamente la última transacción que ha leído el elemento de datos.

Si el plan en serie equivalente debe ser el cronológico, es suficiente conocer cuál es la transacción más joven que ha leído y escrito un elemento de datos, si se solicita una nueva operación en conflicto sobre el elemento de datos, no será permitida si pertenece a una transacción anterior.

5 Protocolos de ordenamiento por marcas de tiempo.

Protocolo de ordenamiento por marcas de tiempo (OMT):

a) Sea T' la transacción más joven que ha escrito un dato X actualizando la marca de escritura de X: MT_escritura(X) ← MT(T'). Si una transacción T ejecuta después de esa escritura una operación de lectura de X, el protocolo OMT compara la marca de tiempo de esa transacción (MT(T)) con la marca de tiempo de escritura del elemento de datos X (MT_escritura(X)):

   1. Si **MT(T) < MT_escritura(X)** → MT(T) < MT(T'):
      - T es cronológicamente anterior a T'.
      - La operación de lectura de T va a entrar en conflicto con la operación de escritura de la transacción T'.
      - El orden de esas dos operaciones es el inverso al orden cronológico (la escritura de X por T' sería anterior a la lectura de X por T).
      - La lectura de X es rechazada y la transacción T es anulada por el SGBD.

T quiere leer X
T' transacción más joven que ha escrito X

90

5 Protocolos de ordenamiento por marcas de tiempo.

Protocolo de ordenamiento por marcas de tiempo (OMT):

a) Sea T' la transacción más joven que ha escrito un dato X actualizando la marca de escritura de X: MT_escritura(X) ← MT(T'). Si una transacción T ejecuta después de esa escritura una operación de lectura de X, el protocolo OMT compara la marca de tiempo de esa transacción (MT(T)) con la marca de tiempo de escritura del elemento de datos X (MT_escritura(X)):

2. Si **MT(T) > MT_escritura(X)** → MT(T) > MT(T'):
   - T es cronológicamente posterior a T'.
   - La operación de lectura de T va a entrar en conflicto con la operación de escritura de la transacción T'.
   - El orden de estas dos operaciones es el mismo que en el orden cronológico (la escritura de X por T' es anterior a la lectura de X).
   - La lectura de X es aceptada.
3. Si **MT(T) = MT_escritura(X)** → MT(T) = MT(T'): T y T' son la misma transacción y no hay par de operaciones en conflicto. La lectura de X es aceptada.

T quiere leer X
T' transacción más joven que ha escrito X

91

5 Protocolos de ordenamiento por marcas de tiempo.

Protocolo de ordenamiento por marcas de tiempo (OMT):

b) Sea T' la transacción más joven que ha leído un dato X actualizando la marca de lectura de X: MT_lectura(X) ← MT(T'). Y sea T'' la transacción más joven que ha escrito X actualizando la marca de escritura de X: MT_escritura(X) ← MT(T''). Si una transacción T ejecuta después de esa lectura y de esa escritura una operación de escritura de X, el protocolo OMT compara la marca de tiempo de esa transacción (MT(T)) con las marca de tiempo de escritura y de lectura del elemento de datos X (MT_escritura(X) y MT_lectura(X)):

1. Si **MT(T) < MT_lectura(X) o MT(T) < MT_escritura(X)→** MT(T) < MT(T') o MT(T) < MT(T''):
   - T es cronológicamente anterior a T' o a T''.
   - La operación de escritura de T va a entrar en conflicto con la operación de lectura de la transacción T' o con la operación de escritura de T''.
   - El orden de al menos un par de esas operaciones en conflicto es el inverso al orden cronológico.
   - La escritura de X es rechazada y la transacción T es anulada por el SGBD.

T quiere escribir X
T' transacción más joven que ha leído X
T" transacción más joven que ha escrito X

92

5 Protocolos de ordenamiento por marcas de tiempo.

Protocolo de ordenamiento por marcas de tiempo (OMT):

b) Sea T' la transacción más joven que ha leído un dato X actualizando la marca de lectura de X: MT_lectura(X) ← MT(T'). Y sea T'' la transacción más joven que ha escrito X actualizando la marca de escritura de X: MT_escritura(X) ← MT(T''). Si una transacción T ejecuta después de esa lectura y de esa escritura una operación d[*T quiere escribir X* marca de tiempo de esa tr *T' transacción más joven que ha leído X* e escritura y de lectura del el *T" transacción más joven que ha escrito X* ctura(X)):

2. Si **MT(T) > = MT_lectura(X) y MT(T) >= MT_escritura(X)→** MT(T) >= MT(T') y MT(T) >= MT(T'') (con al menos una de las desigualdades estricta):
   - T es cronológicamente posterior a T' y a T''.
   - La operación de escritura de T va a entrar en conflicto con la operación de lectura de la transacción T' o con la operación de escritura de T'' o con ambas.
   - El orden de esos dos pares de operaciones en conflicto es el mismo que en el orden cronológico.
   - La escritura de X es aceptada por el SGBD.
3. Si **MT(T) = MT_lectura(X) y MT(T) = MT_escritura(X)→** MT(T) = MT(T') y MT(T) = MT(T''): T, T' y T'' son la misma transacción y no hay pares de operaciones en conflicto. La escritura de X es aceptada por el SGBD.

93

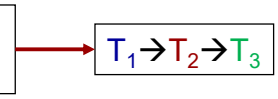
3 Planes serializables por conflictos.

Protocolos de ordenamiento por marcas de tiempo (OMT):

Plan en serie cronológico para un plan concurrente: Plan en serie en el que las transacciones se procesan siguiendo el orden de sus marcas de tiempo en el plan concurrente.

T_1
O_{11}
O_{12}
O_{13}
…
T_2
O_{21}
O_{22}
…
T_3
O_{31}
O_{32}
…

P_1: Plan concurrente
$O_{11}\ O_{21}\ O_{12}\ O_{31}\ O_{13}\ O_{32}\ O_{22}$ …



T_1 no puede leer X si ya la ha escrito T_2 o T_3
T_1 no puede escribir X si ya la ha escrito T_2 o T_3 o si X ya la ha leído T_2 o T_3

T_2 no puede leer X si ya la ha escrito T_3
T_2 no puede escribir X si ya la ha escrito T_3 o si X ya la ha leído T_3

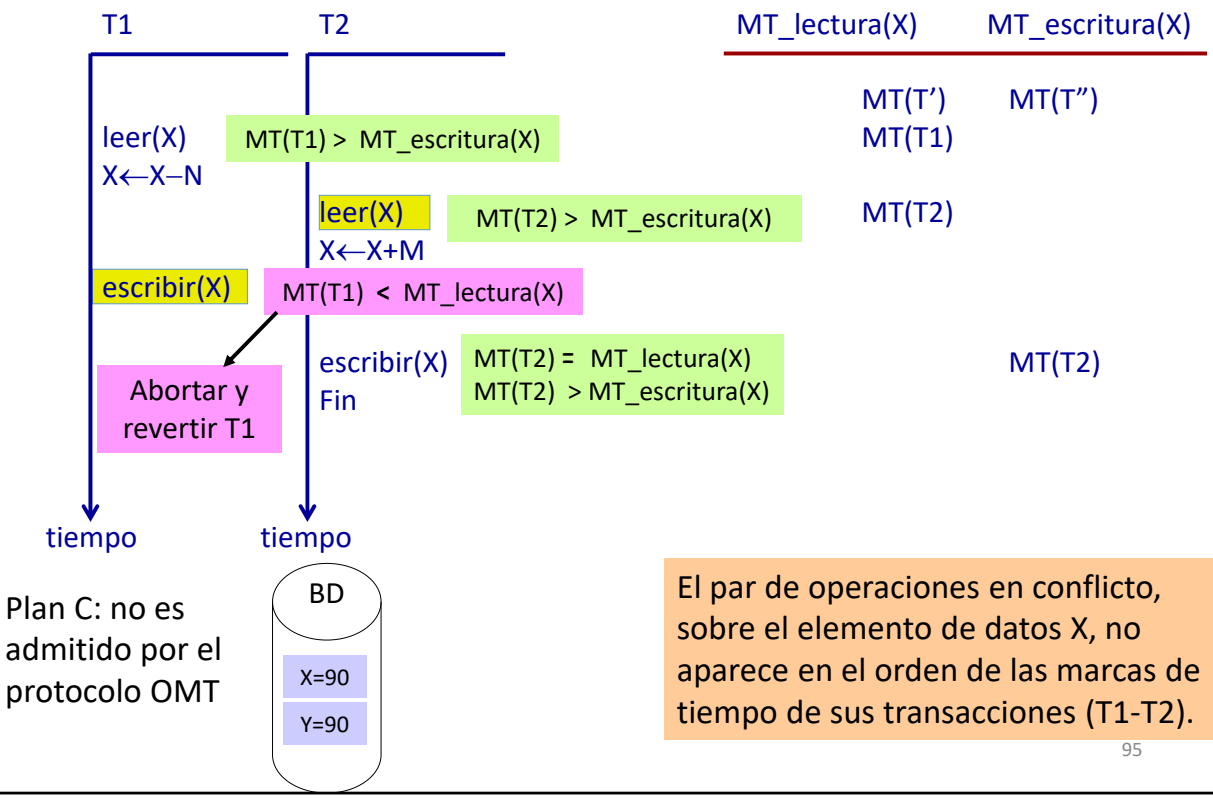
T_3 siempre puede leer y escribir porque es la más joven de las tres

O_{ij}: Operación *j* de la transacción *i*

94

5 Protocolos de ordenamiento por marcas de tiempo.

Constantes: M=2 N=3 MT(T'')<MT(T') < MT(T1) < MT(T2)



Plan C: no es admitido por el protocolo OMT

El par de operaciones en conflicto, sobre el elemento de datos X, no aparece en el orden de las marcas de tiempo de sus transacciones (T1-T2).

95

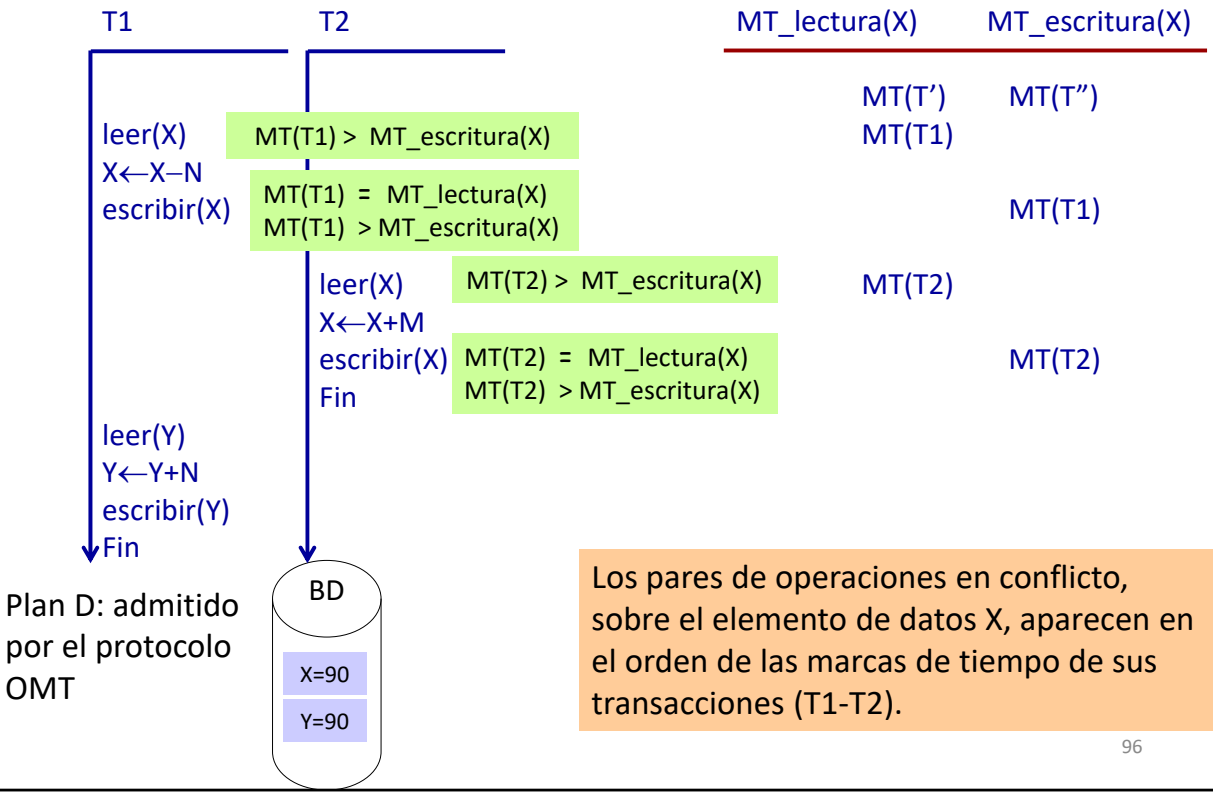
Ante cada nueva operación de lectura (respectivamente escritura) de una transacción, el protocolo OMT **compara** la marca de tiempo de la transacción que solicita la operación con la marca de tiempo de escritura (respectivamente lectura y escritura) del elemento de datos, si estas marcas de tiempo no guardan entre sí el orden cronológico, la transacción solicitante es rechazada.

En la transparencia, se muestra la información guardada por el protocolo, durante la ejecución del plan, y las comparaciones (controles) que el protocolo hace para aplicar su estrategia.

La transacción T1 es rechazada e introducida de nuevo en el sistema, con una marca de tiempo posterior a la de T2. T2 puede proseguir.

5 Protocolos de ordenamiento por marcas de tiempo.

Constantes: M=2 N=3 MT(T'')<MT(T') < MT(T1) < MT(T2)



Plan D: admitido por el protocolo OMT

Los pares de operaciones en conflicto, sobre el elemento de datos X, aparecen en el orden de las marcas de tiempo de sus transacciones (T1-T2).

96

Ante cada nueva operación de lectura (resp. escritura) de una transacción, el protocolo OMT **compara** la marca de tiempo de la transacción que solicita la operación con la marca de tiempo de escritura (resp. lectura o escritura) del elemento de datos, si estas marcas de tiempo no guardan entre sí el orden cronológico, la transacción solicitante es rechazada.

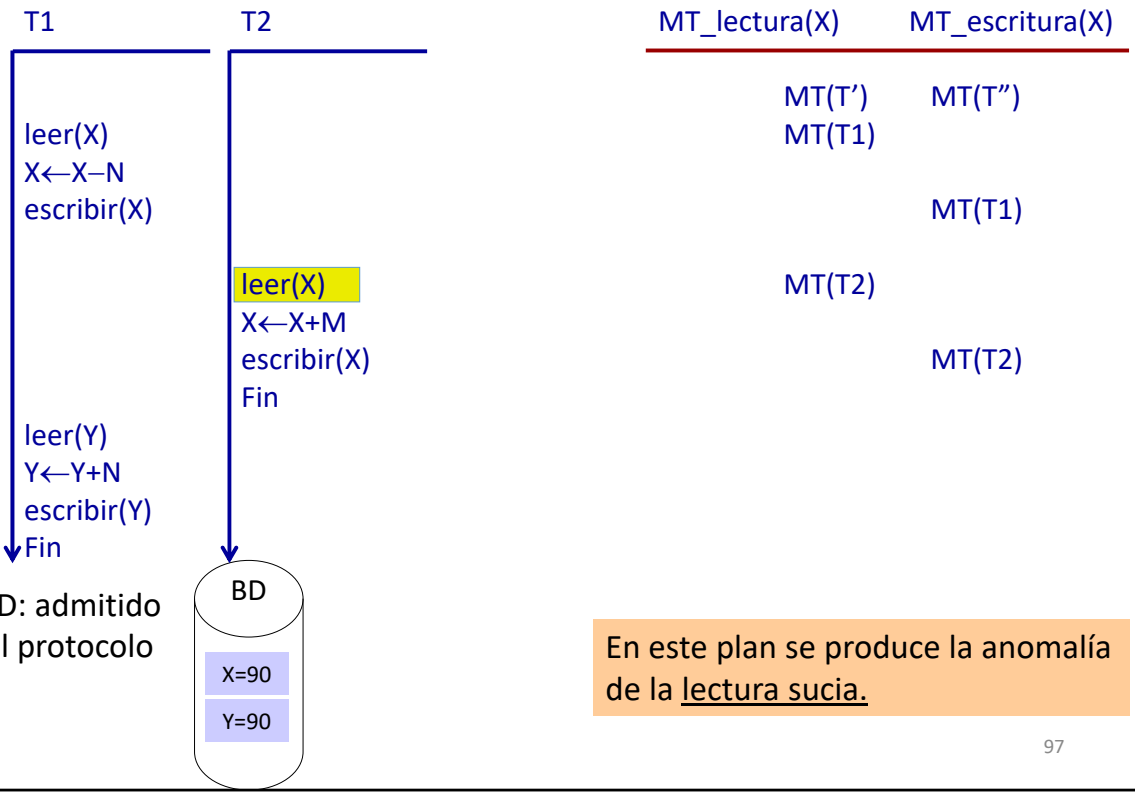
En la transparencia, se muestra la información guardada por el protocolo, durante la ejecución del plan, y las comparaciones (controles) que el protocolo hace para aplicar su estrategia.

En este caso, el plan concurrente D es admitido por el protocolo OMT. El plan es seriabilizable por conflictos.

Nota: Dado que sobre el elemento Y no hay operaciones en conflicto, en la transparencia no se han incluido los controles sobre las marcas de tiempo de Y.

5 Protocolos de ordenamiento por marcas de tiempo.

Constantes: M=2 N=3 MT(T'')<MT(T') < MT(T1) < MT(T2)



Plan D: admitido por el protocolo OMT

En este plan se produce la anomalía de la lectura sucia.

97

En este plan concurrente, admitido por el protocolo OMT, se produce la anomalía de la "lectura sucia". Esta seudoanomalía se estudiará en el punto 7 del tema.

5 Protocolos de ordenamiento por marcas de tiempo.

Algoritmo de ordenamiento por marcas de tiempo (OMT):

- ✓ El algoritmo OMT detecta operaciones en conflicto que ocurren en orden incorrecto (con respecto al orden cronológico de las transacciones) y rechaza la operación más reciente de las dos (aborta la transacción que la emitió).
- ✓ El algoritmo OMT puede provocar la espera indefinida si una transacción se aborta y se reinicia continuamente.
- ✓ El algoritmo OMT no evita la anomalía de la lectura sucia (se estudiará posteriormente).

98

El protocolo OMT permite que en el plan aparezcan pares de operaciones en conflicto, siempre que sucedan en el orden de las marcas de tiempo de sus transacciones.

En OMT, cuando no se puede atender la petición de una transacción, ésta se aborta y se vuelve a considerar más tarde.

El protocolo OMT admite menos planes concurrentes que el protocolo B2F, ya que sólo son aceptados los planes serializables por conflictos equivalentes al plan en serie cronológico.

En el protocolo OMT, se aborta la transacción que entra en conflicto, aunque sea la más antigua.

El protocolo OMT no evita la anomalía de la lectura sucia, por lo que se puede producir el fenómeno de la anulación en cascada (punto 7 del tema).

5 Protocolos de ordenamiento por marcas de tiempo.

Algoritmo de ordenamiento por marcas de tiempo (OMT):

La transacción T emite una operación **leer(X):**

a) Si MT_escritura (X) > MT(T), entonces **abortar** y revertir* T.

b) Si MT_escritura (X) <= MT(T), entonces ejecutar la operación leer(X) y asignar a MT_lectura(X) el **mayor** valor entre los valores MT(T) y MT_lectura(X).

c) Si el elemento de datos X no existe, entonces la operación no se puede ejecutar.

* T se vuelve a introducir en el sistema con una nueva marca de tiempo

99

Significado de los casos:

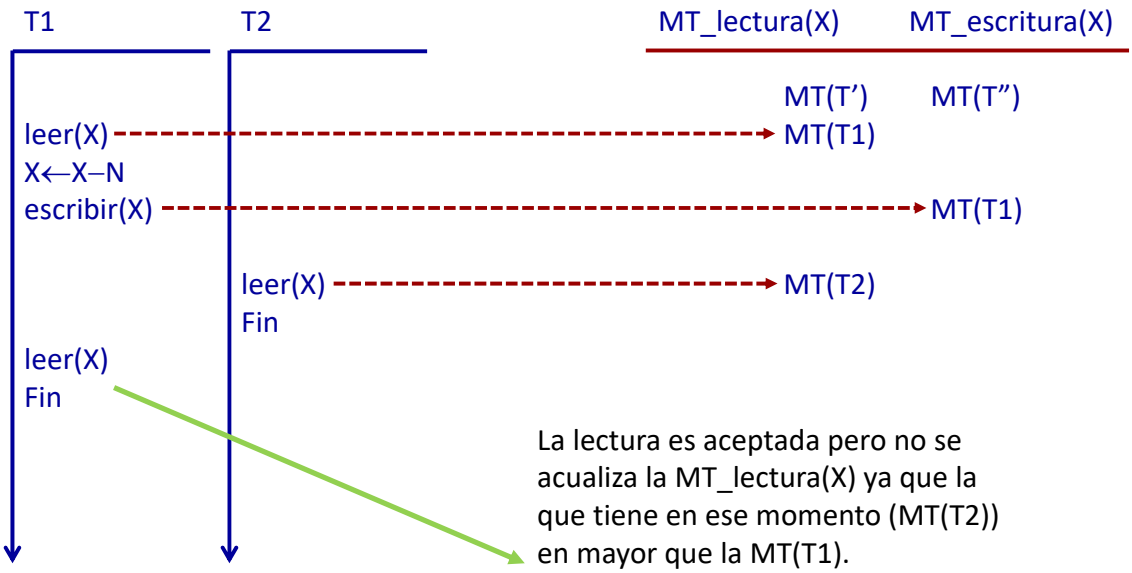
a) La última operación que ha escrito el elemento X pertenece a una transacción **posterior (más joven)** a T . Si se permitiese la operación de lectura de T, aparecerían en el plan dos operaciones en conflicto, en orden inverso de las marcas de tiempo de sus transacciones. El plan no sería equivalente por conflictos al plan en serie cronológico.

b) La última operación que ha escrito el elemento X pertenece a una transacción **anterior (más vieja)** a T o es la propia T, por lo tanto se puede permitir la operación de lectura de T porque en este caso las operaciones en conflicto con ella aparecen en el plan en el orden de las marcas de tiempo de sus transacciones. Hay que recordar que la transacción T puede no ser la más joven que ha leído X por eso se escoge el mayor de los dos valores entre MT(T) y MT_lectura(X).

c) Cuando el elemento de datos no existe, la operación de lectura no se puede ejecutar. Si tiene marca de escritura y no tiene marca de lectura significa que el elemento de datos existe, pero no ha sido leído, la operación está permitida.

5 Protocolos de ordenamiento por marcas de tiempo.

MT(T'')<MT(T') < MT(T1) < MT(T2)



La lectura es aceptada pero no se acualiza la MT_lectura(X) ya que la que tiene en ese momento (MT(T2)) en mayor que la MT(T1).
La marca de tiempo de lectura de X siempre es la marca de la transacción ***<u>más joven</u>*** *que ha leído X*.
En este caso, esa transacción es T2.

100

5 Protocolos de ordenamiento por marcas de tiempo.

Algoritmo de ordenamiento por marcas de tiempo (OMT):

La transacción T emite una operación **escribir(X)**:

a) Si MT_lectura(X) > MT(T) o MT_escritura(X) > MT(T), entonces **abortar** y revertir* T.

b) Si MT_lectura(X) <= MT(T) y MT_escritura(X) <= MT(T), entonces ejecutar la operación escribir(X) y asignar a MT_escritura (X) el valor MT(T).

c) Si el elemento de datos X no existe, entonces ejecutar la operación escribir(X) y asignar a MT_escritura (X) el valor MT(T).

* T se vuelve a introducir en el sistema con una nueva marca de tiempo

101

Significado de los casos:

a) La última operación que ha leído o que ha escrito el elemento de datos X pertenece a una transacción **posterior (más joven)** a T . Si se permitiese la operación de escritura de T, aparecerían en el plan dos operaciones en conflicto, en orden inverso de las marcas de tiempo de sus transacciones. El plan no sería equivalente por conflictos al plan en serie cronológico.

b) La última operación que ha leído y la última operación que ha escrito el elemento de datos X pertenecen a transacciones **anteriores (más viejas)** a T o es la propia T, por lo tanto se puede permitir la operación de escritura de T porque en este caso las operaciones en conflicto con ella aparecen en el plan, en el orden de las marcas de tiempo de sus transacciones.

c) ¿Puede un elemento de datos no tener marca de escritura?: Sí, cuando el elemento de datos no existe. En este caso la operación de lectura no tiene sentido, y la operación de escritura generaría el elemento. El caso (c) tiene sentido porque las condiciones de los casos (a) y (b) no pueden evaluarse si el elemento no existe.

Control de la concurrencia.

1 Ejecución concurrente de transacciones: anomalías.

2 Control de la concurrencia en SQL.

3 Planes serializables por conflictos.

4 Protocolos de bloqueo.

5 Protocolos de ordenamiento por marcas de tiempo.

6 Protocolos multiversión.

7 Concurrencia y recuperación.

6 Protocolos multiversión.

PROTOCOLOS

- Bloqueo de elementos de datos
- Ordenamiento por marcas de tiempo
- Técnicas multiversión

Protocolos multiversión (MV):

Los protocolos multiversión se basan en la idea de mantener, durante la ejecución del plan, varias versiones de los elementos de datos actualizados, con el objetivo de flexibilizar la concurrencia en las operaciones de lectura.

6 Protocolos multiversión.

Protocolos multiversión (MV):

Plan en serie cronológico para un plan concurrente: Plan en serie en el que las transacciones se procesan siguiendo el orden de sus marcas de tiempo en el plan concurrente.

Cuando el protocolo multiversión controla un plan concurrente, el plan ejecutado* es el **plan en serie cronológico**.

Cuando una transacción solicita la lectura de un elemento de datos se elige una versión del elemento que asegure que se está ejecutando el plan en serie cronológico: se flexibiliza la concurrencia para las operaciones de lectura.

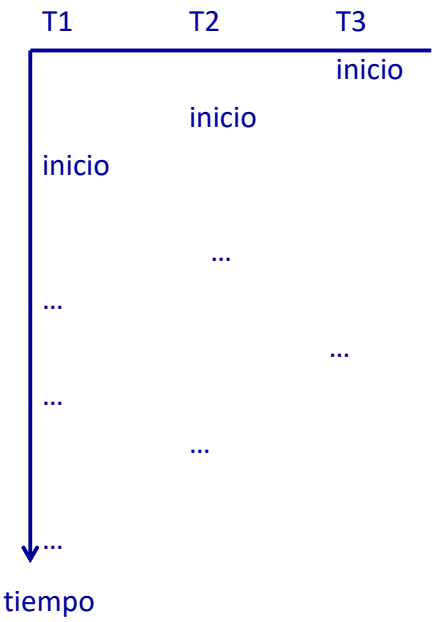
* El uso de las versiones en las operaciones de lectura, hace que el plan ejecutado no sea el plan concurrente original, sino el plan cronológico.

105

Para alcanzar este objetivo, los protocolos mantienen varias versiones de los elementos de datos accedidos en el plan. Cuando una transacción solicita la lectura de un elemento de datos, se le muestra la versión (valor) del elemento que vería si las transacciones se ejecutasen en orden cronológico.

6 Protocolos multiversión.

…<MT(T3) < MT(T2) < MT(T1)<…



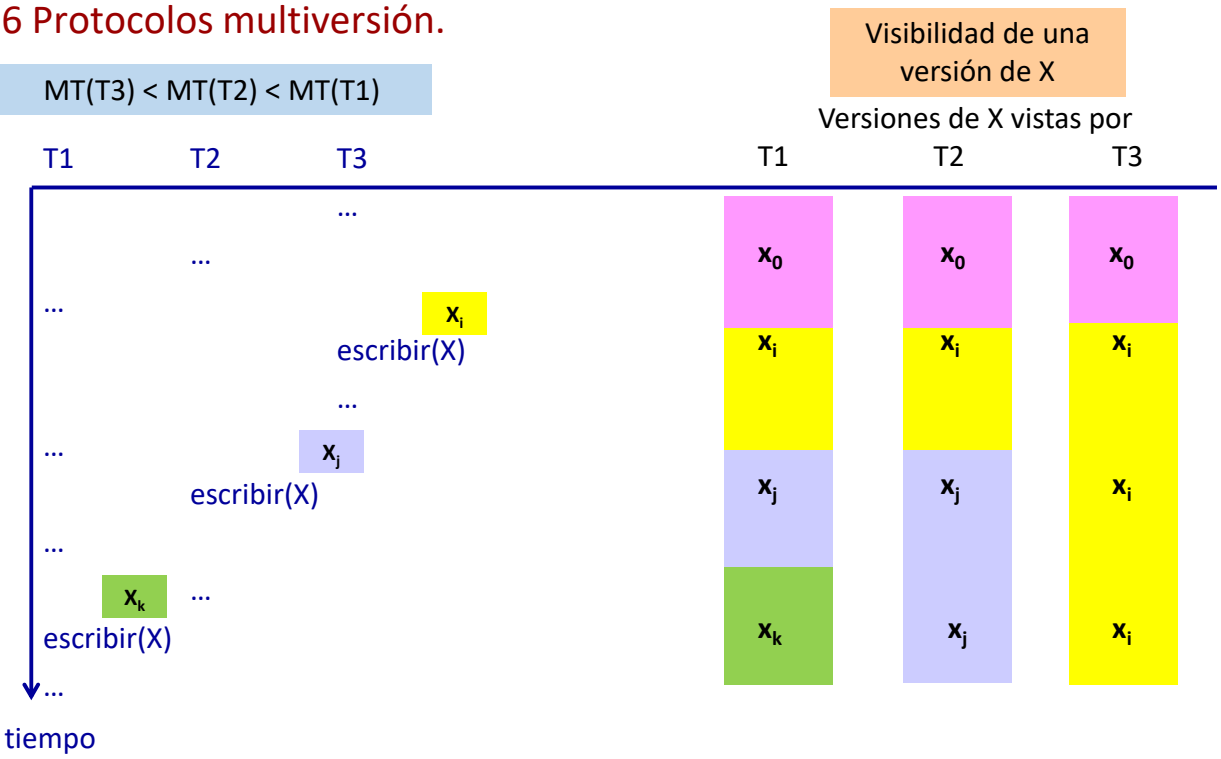
Plan en serie cronológico:
…anteriores ->T3->T2->T1->…posteriores

Cualquier elemento escrito por una transacción podrá ser leído por transacciones más jóvenes pero no por transacciones más viejas

- T3 podrá leer cualquier elemento escrito por transacciones más viejas que ella (…anteriores) pero no por lás más jóvenes (T2, T1, …posteriores)
- T2 podrá leer cualquier elemento escrito por transacciones más viejas que ella (…anteriores, T3) pero no por lás más jóvenes (T1, …posteriores)
- T1 podrá leer cualquier elemento escrito por transacciones más viejas que ella (…anteriores, T3, T2) pero no por lás más jóvenes (…posteriores)

6 Protocolos multiversión.

MT(T3) < MT(T2) < MT(T1)



Después de las tres operaciones de escritura la versión X_i será leída por T3, la versión X_j será leída por T2 y la versión X_k será leída por T1: equivalencia al plan en serie cronológico

107

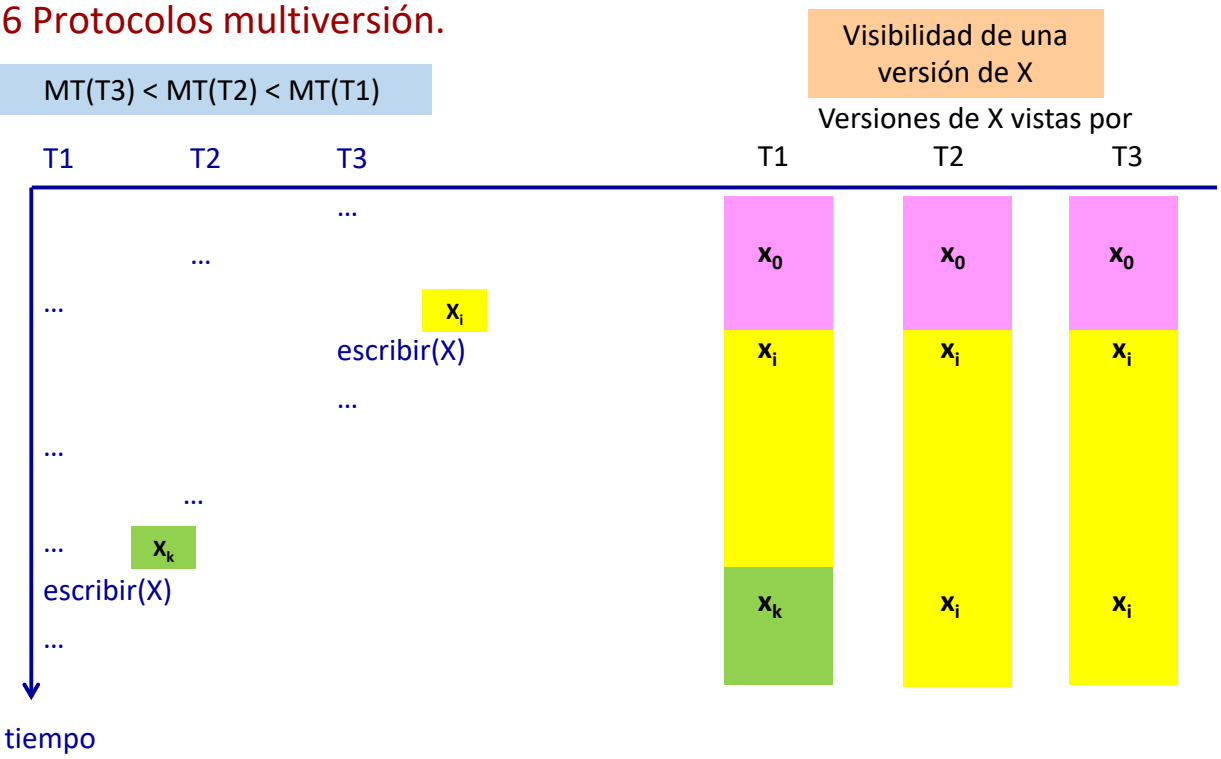
Las transacciones crean versiones de los elementos de datos al realizar operaciones de escritura. Más adelante, se verá que no siempre va a estar permitido que una transacción actualice (escriba) un elemento de datos y, en consecuencia, cree una nueva versión (reglas del protocolo).

En las transparencia, se muestra la visibilidad de cada versión del elemento de datos X, es decir, el valor que cada transacción leería después de las escrituras.

En el ejemplo, se asume que el elemento de datos X tiene un valor inicial (antes del plan) X_0 , o que no existe (NULO).

6 Protocolos multiversión.

MT(T3) < MT(T2) < MT(T1)

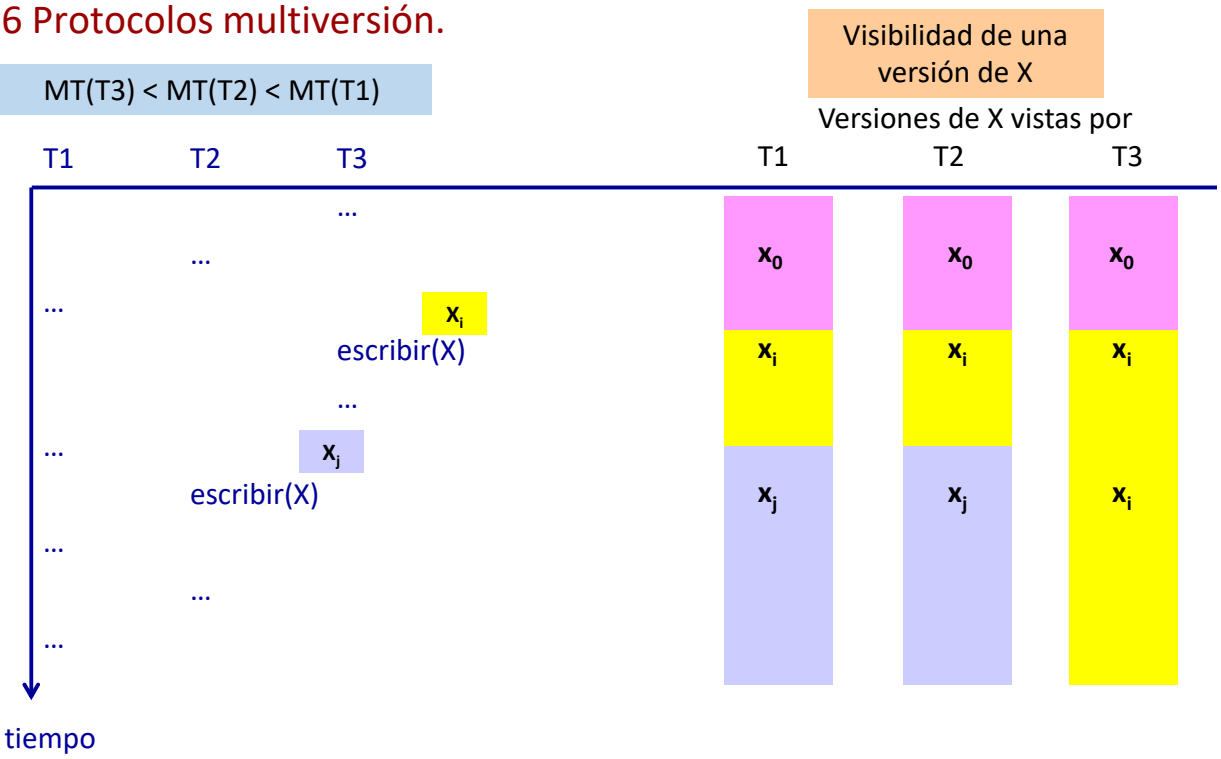


Después de las dos operaciones de escritura la versión X_i será leída por T3 y por T2 y la versión X_k será leída por T1: equivalencia al plan en serie cronológico

108

6 Protocolos multiversión.

MT(T3) < MT(T2) < MT(T1)

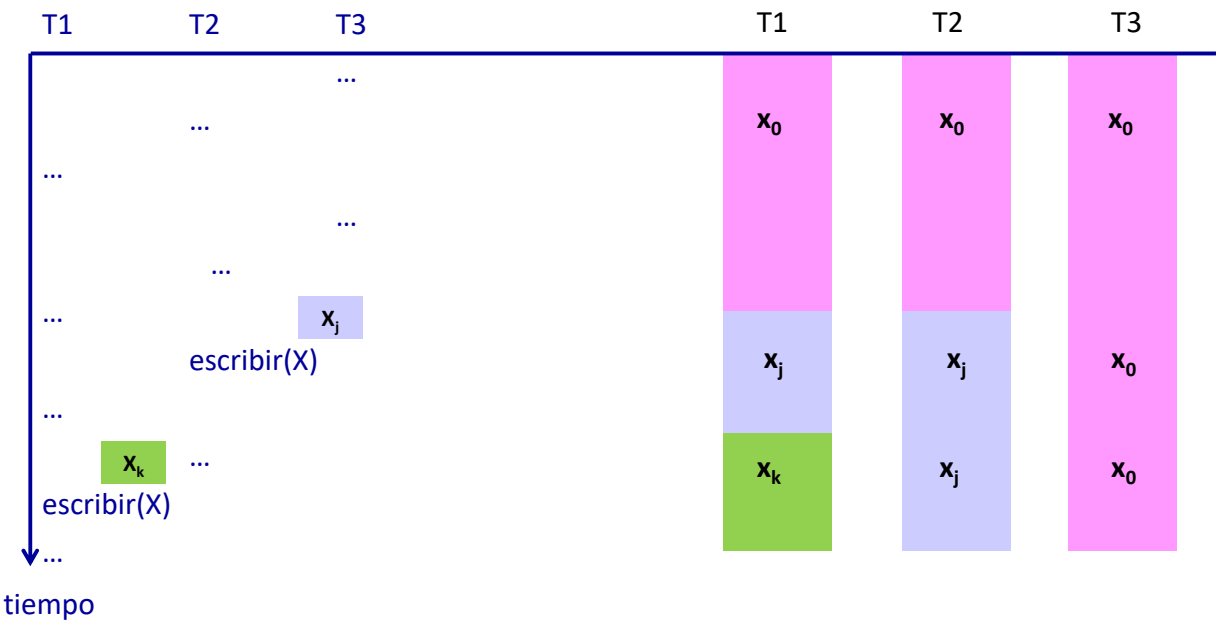


Después de las dos operaciones de escritura la versión X_i será leída por T3 y la versión X_j será leída por T2 y por T1: equivalencia al plan en serie cronológico

109

6 Protocolos multiversión.

MT(T3) < MT(T2) < MT(T1)

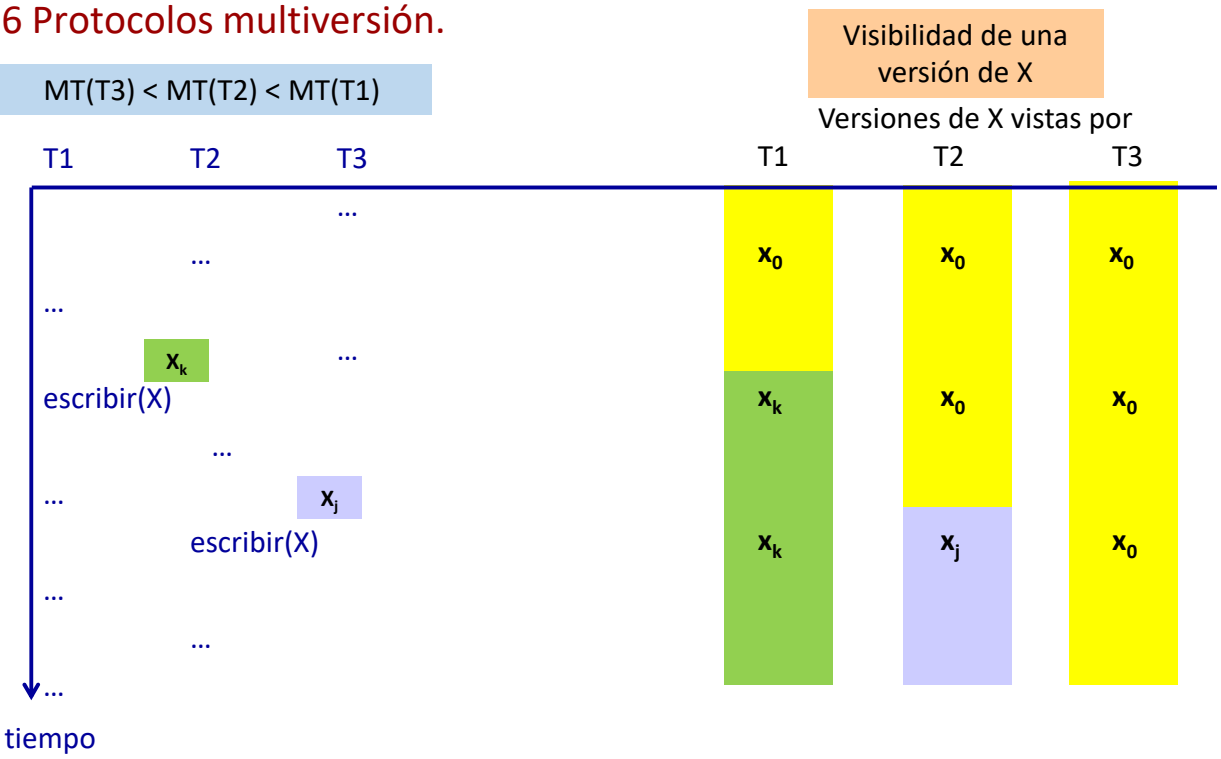


Después de las dos operaciones de escritura la versión X_j será leída por T2, la versión X_k será leída por T1. T3 leerá el valor original (o el valor nulo): equivalencia al plan en serie cronológico

110

6 Protocolos multiversión.

MT(T3) < MT(T2) < MT(T1)



Las operaciones de escritura (creación de versiones) de las transacciones se pueden hacer en cualquier orden, siempre que se cumplan ciertas reglas.

111

6 Protocolos multiversión.

Protocolos multiversión basados en marcas de tiempo (MV): información mantenida por el sistema

El sistema conserva para cada elemento de datos X varias versiones: X_1, X_2, X_3, …, X_k. Para cada versión X_i se conserva el valor de la versión y las marcas de tiempo:

puede cambiar con las lecturas

- ✓ **MT_lectura (X_i):** la marca de tiempo **mayor** entre todas las marcas de tiempo de las transacciones que han leído con éxito la versión X_i: marca de tiempo de la transacción más joven que ha leído con éxito la versión X_i.
- ✓ **MT_escritura(X_i)**: es la marca de tiempo de la transacción que escribió el valor X_i (de la transacción que creó la versión).

no cambia nunca

112

Para poder aplicar las ideas del protocolo MV, es decir, permitir crear una nueva versión (actualización), y mostrar la versión adecuada (lectura) de un elemento de datos, el protocolo necesita guardar información durante la ejecución del plan. **¿Qué información?**

Los protocolos multiversión, **basados en marcas de tiempo**, se apoyan en la marca de tiempo de las transacciones que acceden a los elementos de datos (lectura o escritura), para controlar el orden de ejecución de las operaciones en conflicto. El plan concurrente debe ser equivalente por conflictos al plan en serie cronológico.

La información registrada en el sistema es la misma que en el protocolo OMT, pero, en este caso, para cada versión de los elementos de datos accedidos: **la marca de tiempo de lectura** y la **marca de tiempo de escritura**. Estas marcas de tiempo tienen el mismo significado que en el protocolo OMT.

6 Protocolos multiversión.

Protocolo multiversión (MV):

a) Cuando una transacción T ejecuta una operación de lectura del dato X, el protocolo MV busca la versión de X, sea X_i, que tiene mayor marca de tiempo de escritura entre todas las versiones que cumplen **MT_escritura(X_i) <=MT(T)** y proporciona esa versión a T. *Busca la versión de X más reciente creada por una transacción cronológicamente anterior a T.*

   - Una operación de lectura siempre se satisface si existe el elemento a leer.

Protocolo multiversión (MV):

b) Cuando una transacción T ejecuta una operación de escritura del dato X, el protocolo MV busca la versión de X, sea X_i, que tiene mayor marca de tiempo de escritura entre todas las versiones que cumplen **$MT\_escritura(X_i) <= MT(T)$**. Sea T' la transacción más joven que leyó esa versión de X ($MT\_lectura(X_i) = MT(T')$):

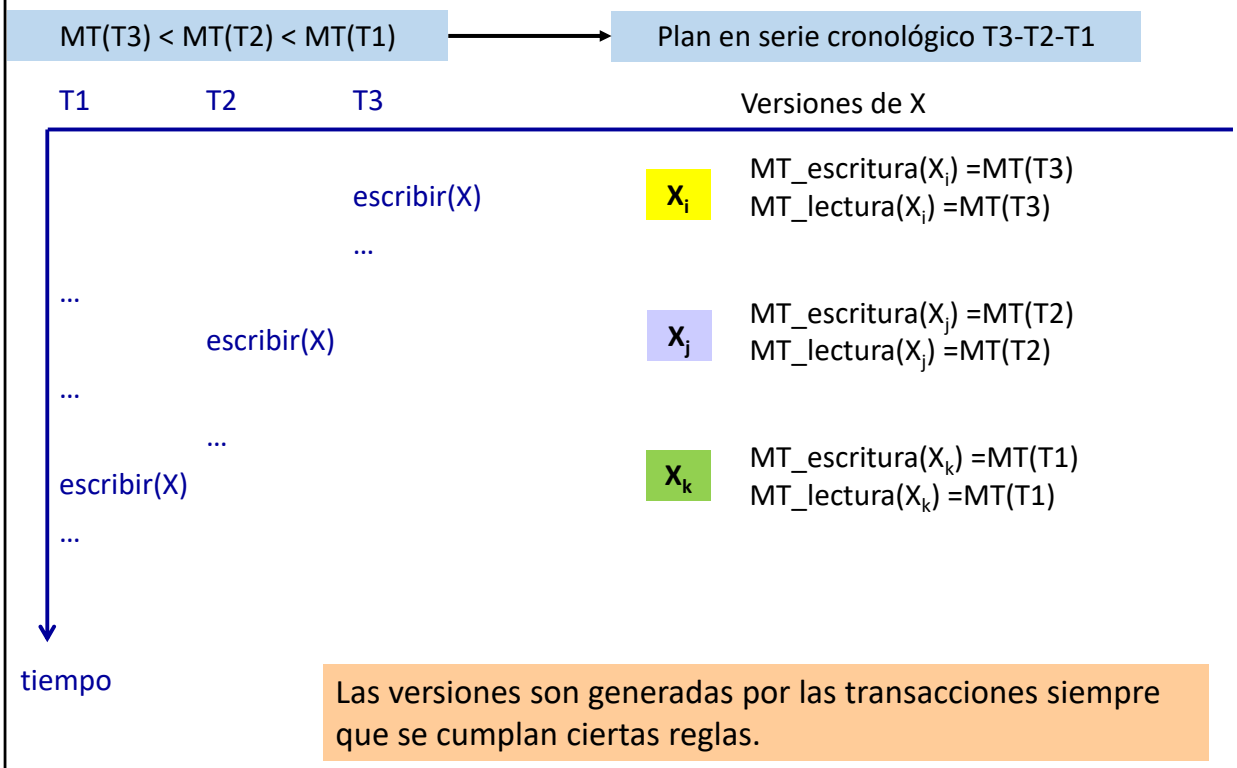
1. Si **$MT(T) < MT\_lectura(X_i)$** $\rightarrow$ $MT(T) < MT(T')$
   - T es cronológicamente anterior a T'.
   - La operación de escritura de T va a entrar en conflicto con la operación de lectura de T'.
   - El orden de ese par de operaciones en conflicto es el inverso al orden cronológico (la escritura de X por T es posterior a la lectura de X por T').
   - La escritura de X es rechazada y la transacción T es anulada por el SGBD.

Protocolo multiversión (MV):

b) Cuando una transacción T ejecuta una operación de escritura del dato X, el protocolo MV busca la versión de X, sea X_i, que tiene mayor marca de tiempo de escritura entre todas las versiones que cumplen **MT_escritura(X_i) <=MT(T)**. Sea T' la transacción más joven que leyó esa versión de X (MT_lectura(X_i) =MT(T')):

2. Si **MT(T) > MT_lectura(X_i)** → MT(T) > MT(T')
   - T es cronológicamente posterior a T'.
   - La operación de escritura de T va a entrar en conflicto con la operación de lectura de T'.
   - El orden de ese par de operaciones en conflicto es el mismo que en el orden cronológico (la escritura de X por T es posterior a la lectura de X por T').
   - La escritura de X es aceptada.
3. Si **MT(T) = MT_lectura(X_i)** → MT(T) = MT(T'): T y T' son la misma transacción. La escritura es aceptada.

6 Protocolos multiversión.

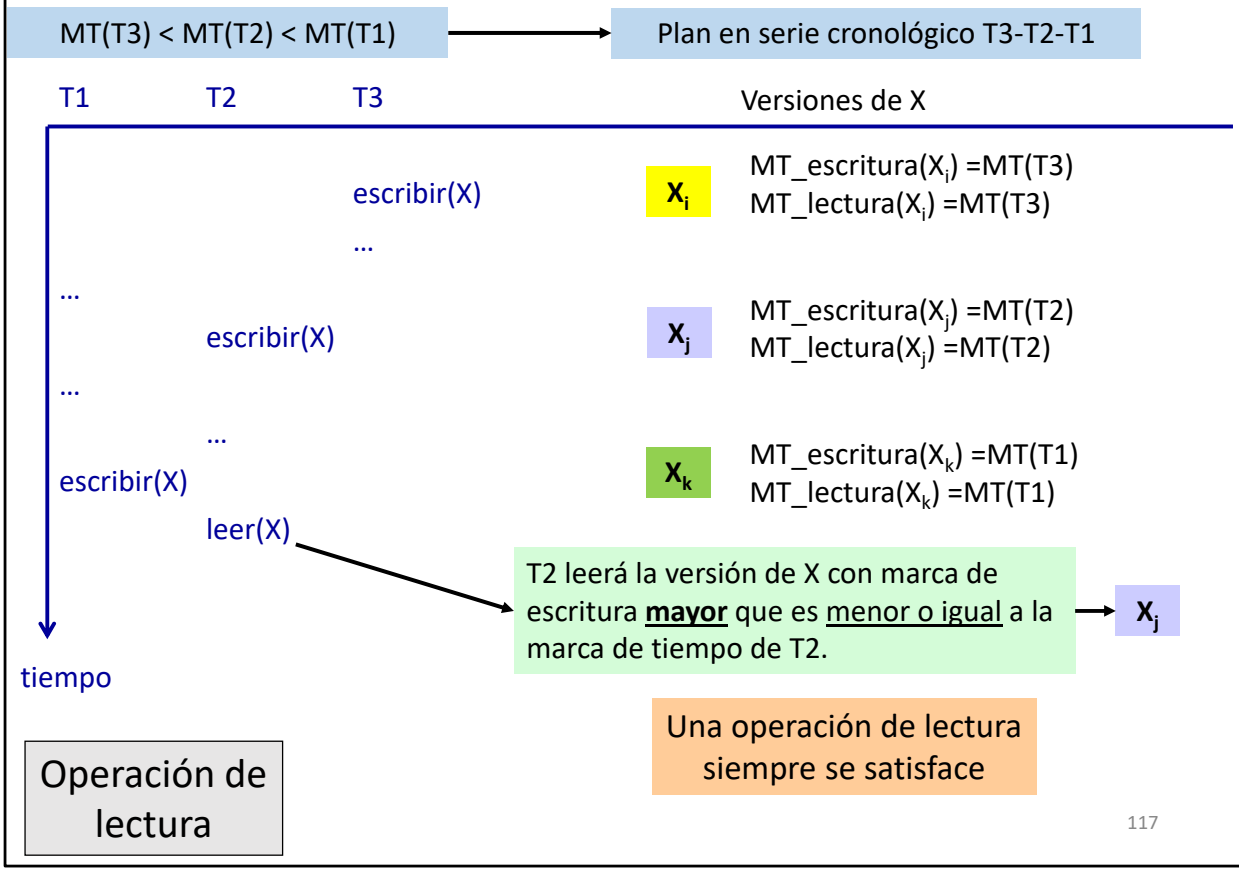


Las versiones son generadas por las transacciones siempre que se cumplan ciertas reglas.

116

Hasta que no aparecen operaciones de lectura, cualquier transacción puede generar nuevas versiones de los elementos de datos. El conflicto puede aparecer cuando se han producido ya operaciones de lectura.

6 Protocolos multiversión.

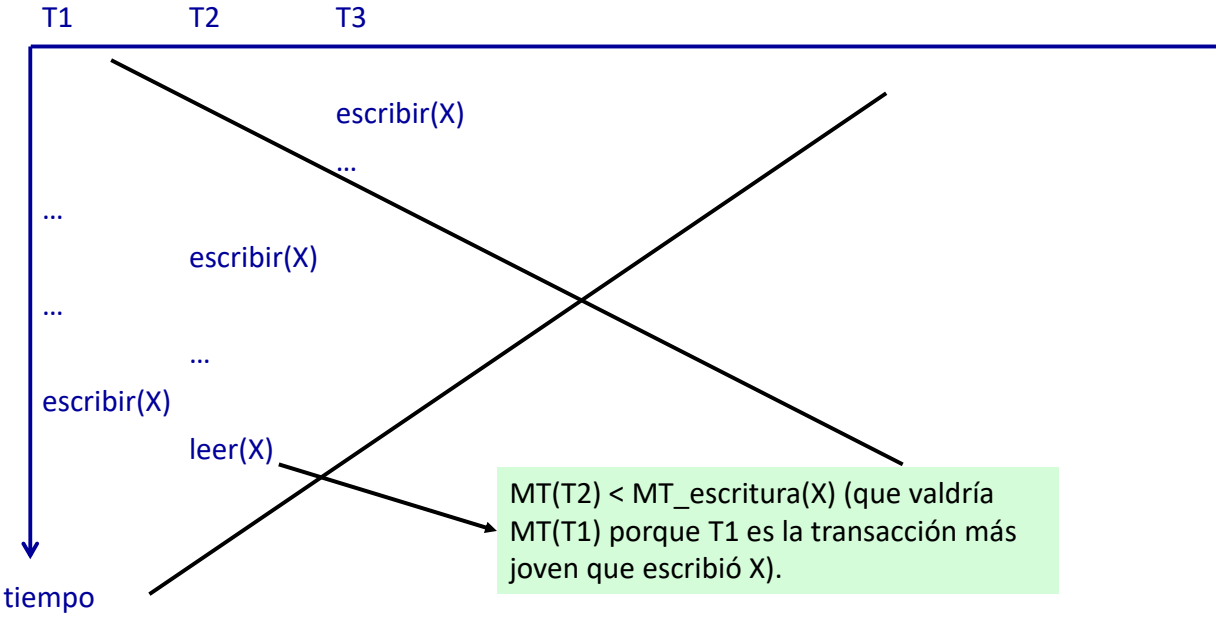


En el protocolo MV, una operación de lectura **siempre** se satisface, el problema es determinar qué versión del elemento se enseña a la transacción solicitante.

En la transparencia, se muestra la información que el protocolo guarda durante la ejecución del plan: versiones de los elementos de datos y sus marcas de tiempo de lectura y de escritura, y las comparaciones (controles) que el protocolo debe hacer para responder a la operación de lectura de la transacción T2.

6 Protocolos multiversión.

MT(T3) < MT(T2) < MT(T1) → Plan en serie cronológico T3-T2-T1



MT(T2) < MT_escritura(X) (que valdría MT(T1) porque T1 es la transacción más joven que escribió X).

Operación de lectura

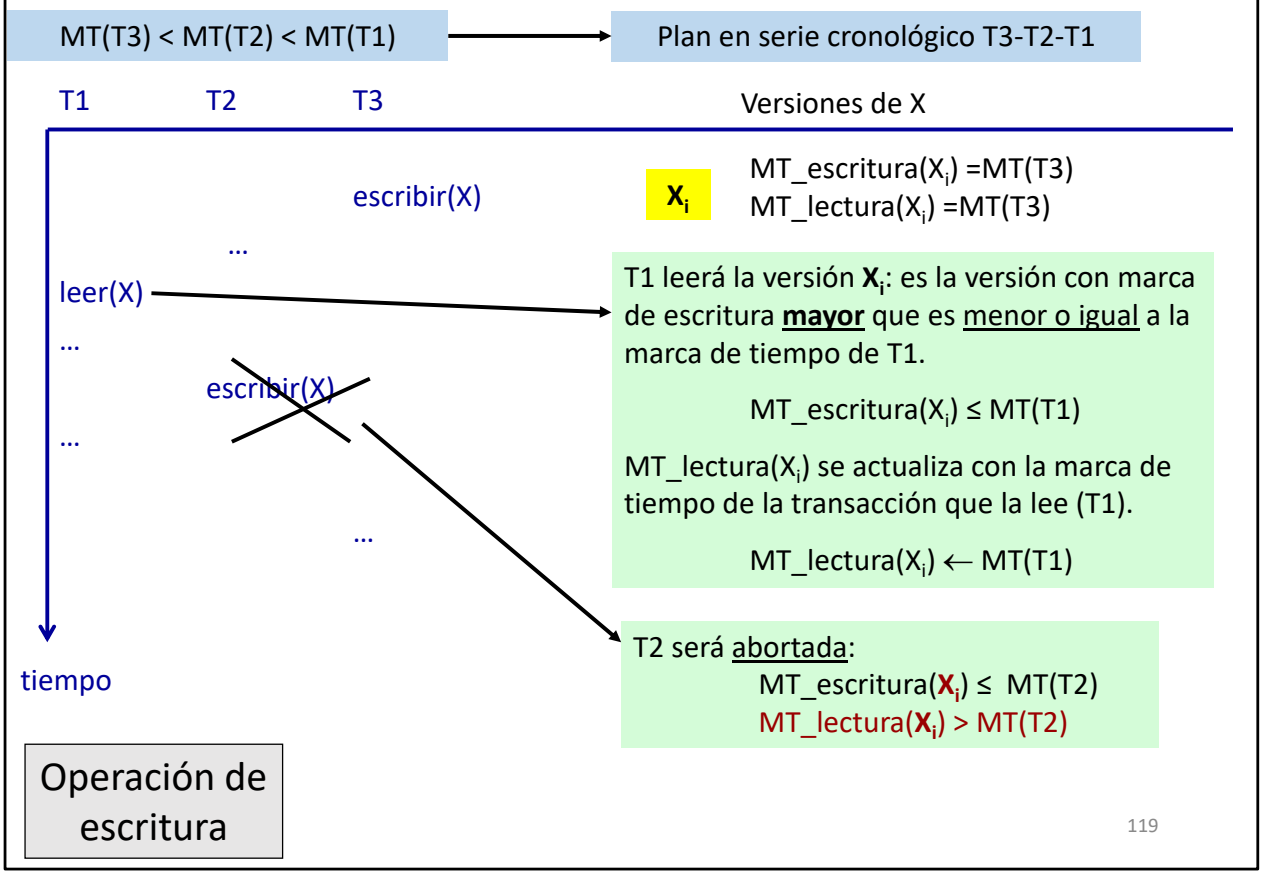
En el protocolo de **ordenamiento por marcas de tiempo** este plan no sería posible: la transacción T2 sería abortada.

118

En el protocolo OMT, la operación de lectura de T2 no está permitida porque en el plan aparecerían dos operaciones en conflicto en orden incorrecto: $w_1(X)$ y $r_2(X)$.

En el protocolo multiversión, la lectura de T2 está permitida porque no se le muestra el valor actualizado por T1 (X_k), se le muestra el valor anterior (X_j) creado por T2.

6 Protocolos multiversión.



Una operación de escritura no está siempre permitida, el protocolo debe asegurar la equivalencia del plan concurrente con su plan en serie cronológico (T3-T2-T1).

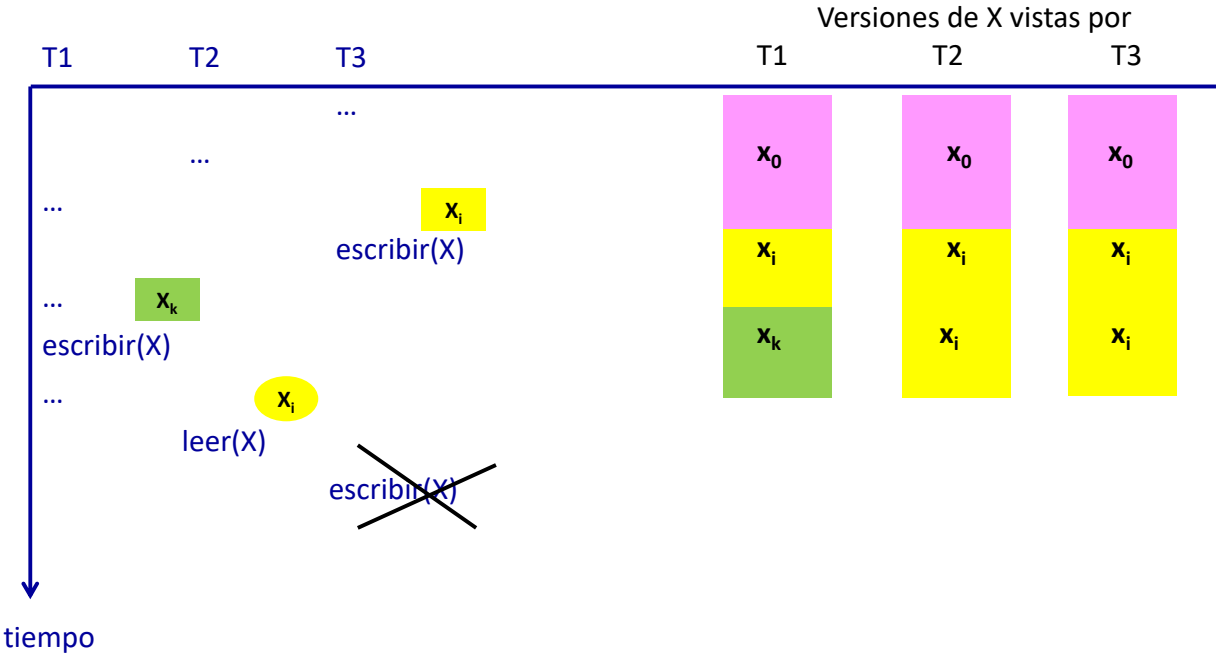
T2 es abortada porque T1 (más joven que T2) ya ha leído X_i. Si se admitiese la operación de escritura de T2, T1 debería haber leído la versión que hubiera creado T2, y esto ya no es posible.

El protocolo multiversión enseña a una transacción (lectura) la última versión actualizada por la transacción cronológicamente anterior, o bien una versión creada por ella misma. El plan en serie equivalente es el plan cronológico.

No siempre se aborta la transacción más reciente (más joven).

6 Protocolos multiversión.

MT(T3) < MT(T2) < MT(T1) → Plan en serie cronológico T3-T2-T1



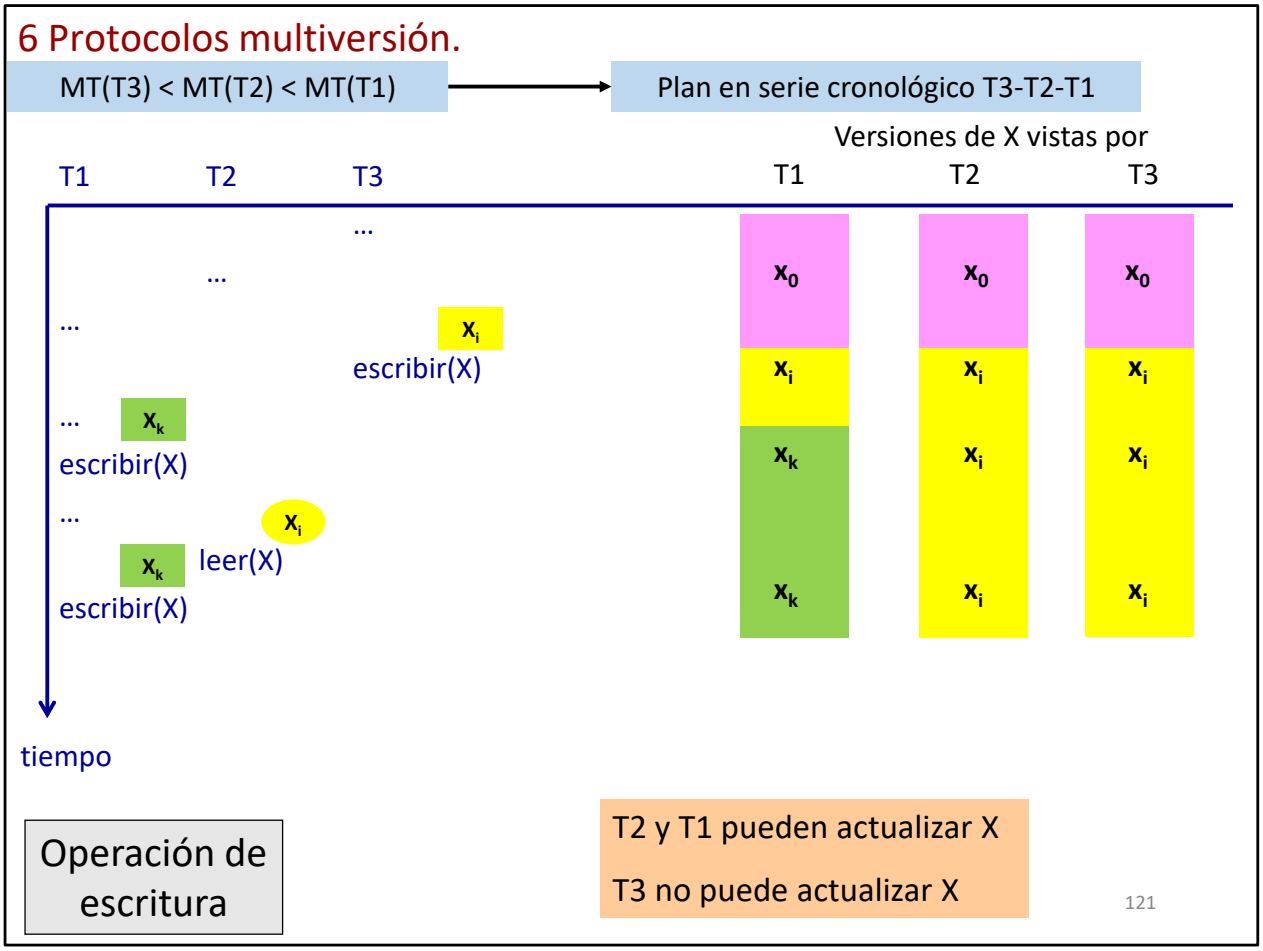
Operación de escritura

T2 y T1 pueden actualizar X

T3 no puede actualizar X

120

En esta transparencia y las siguientes, se muestran ejemplos de operaciones de escritura en distintas situaciones.

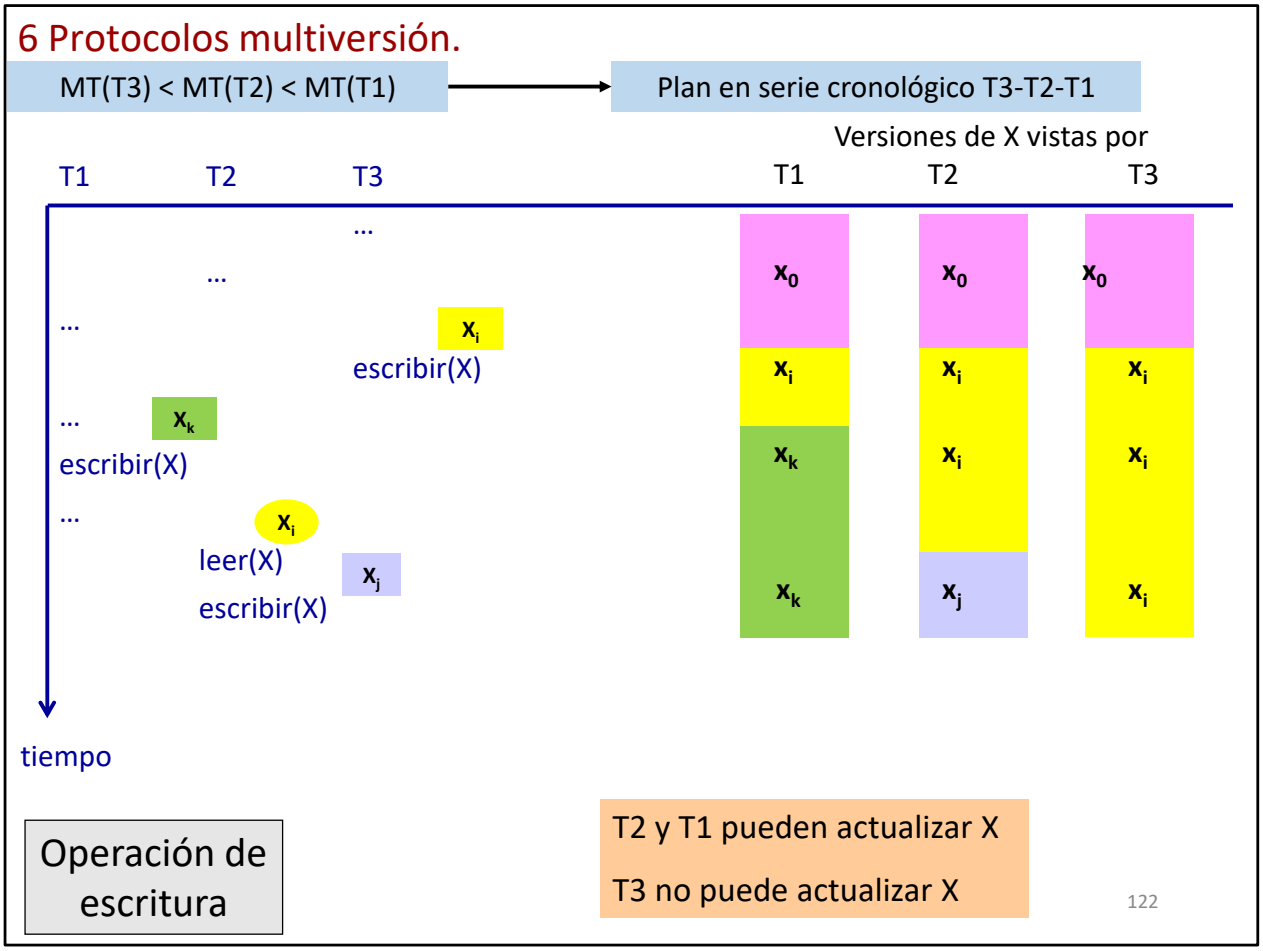


T2 siempre leerá la <u>última versión creada por T3, en el momento de la lectura</u>, en el ejemplo X_i, hasta que cree sus propias versiones. La lectura de T2 <u>impide</u> a T3 crear nuevas versiones.

T2 puede crear nuevas versiones de X que sólo serán leídas por T2, estas versiones no tienen ningún efecto sobre las transacciones posteriores: T1 ya ha creado sus propias versiones.

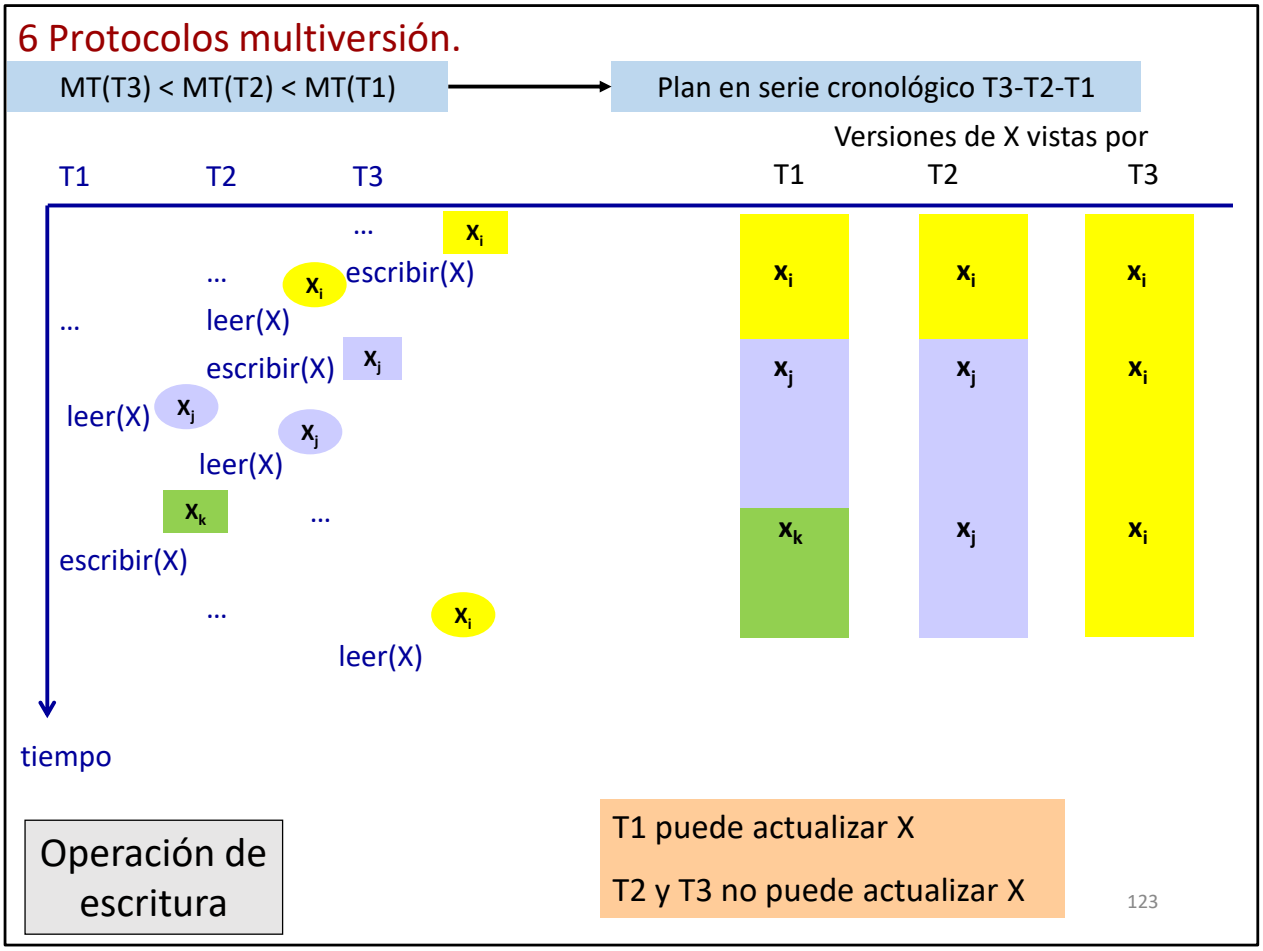
La segunda escritura de la transacción T1 se permite porque ninguna transacción más joven ha leído x_k; el efecto sería sobreescribir el valor de la primera escritura.

6 Protocolos multiversión.



Es importante observar que T2 crea una versión del elemento X intermedia entre las versiones X_i y X_k.

T2 puede escribir porque la versión anterior X_i no ha sido leída por ninguna transacción más joven que T_2. A partir de ese momento T1, T2 y T3 leen sus propias versiones.



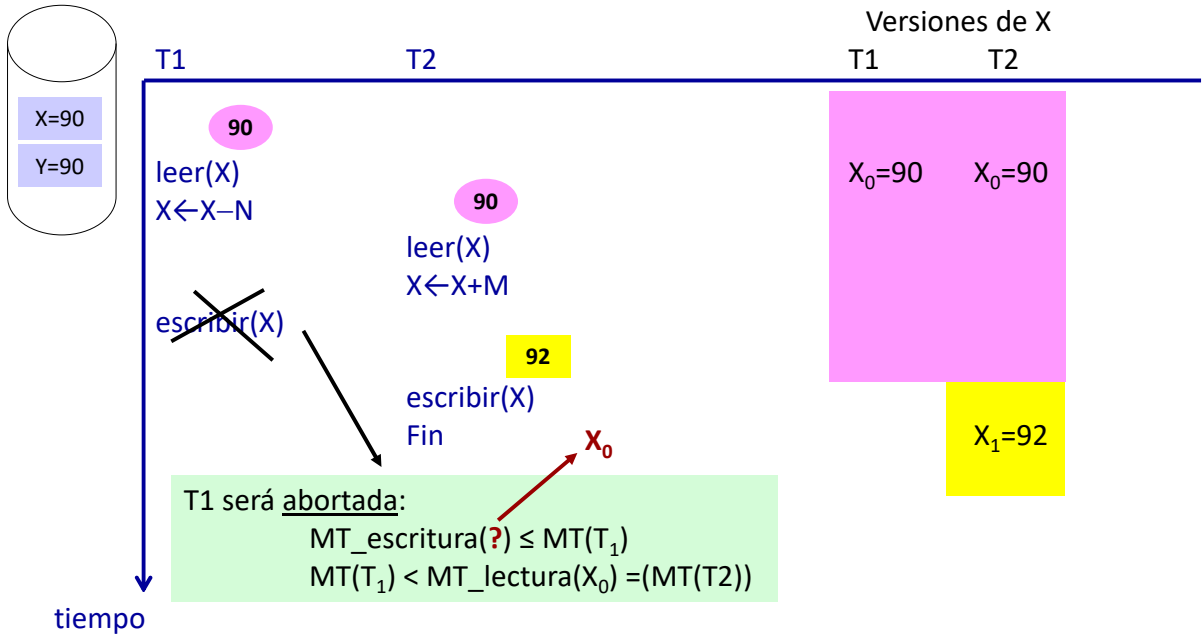
T3 y T2 ya no pueden actualizar X, la versión creada por T3 ha sido leída por T2, y la versión creada por T2 ha sido leída por T1.

6 Protocolos multiversión.

Plan C: no es admitido por el protocolo

Ejemplo 1

Constantes: M=2 N=3

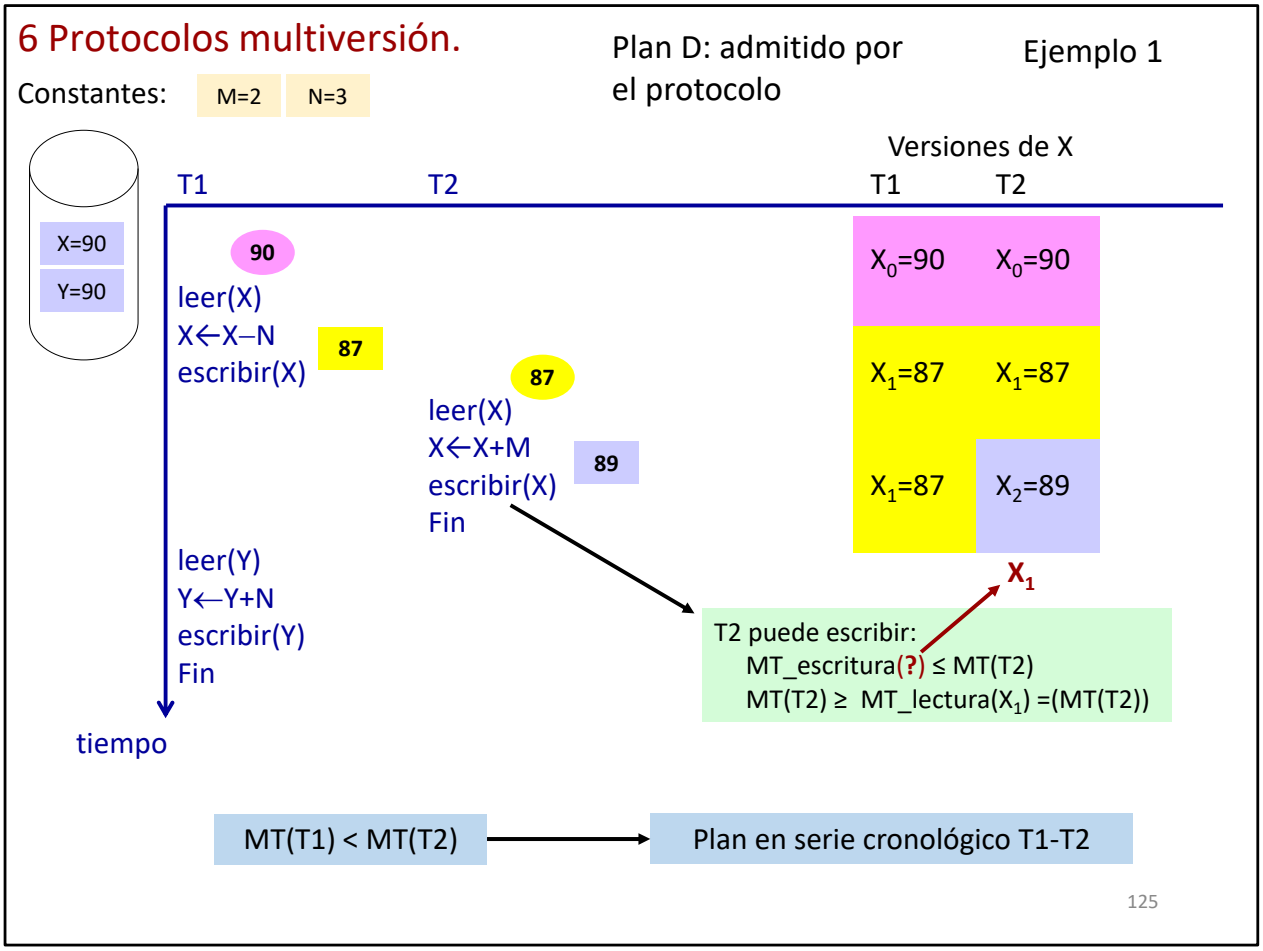


MT(T1) < MT(T2) → Plan en serie cronológico T1-T2

124

En la transparencia se muestra el plan C del Ejemplo 1 introducido al principio del tema.

En este plan, se produciría la anomalía de pérdida de actualizaciones si no fuera controlado por el protocolo MV.



En la transparencia, se muestra el plan concurrente D del ejemplo 1, introducido al principio del tema.

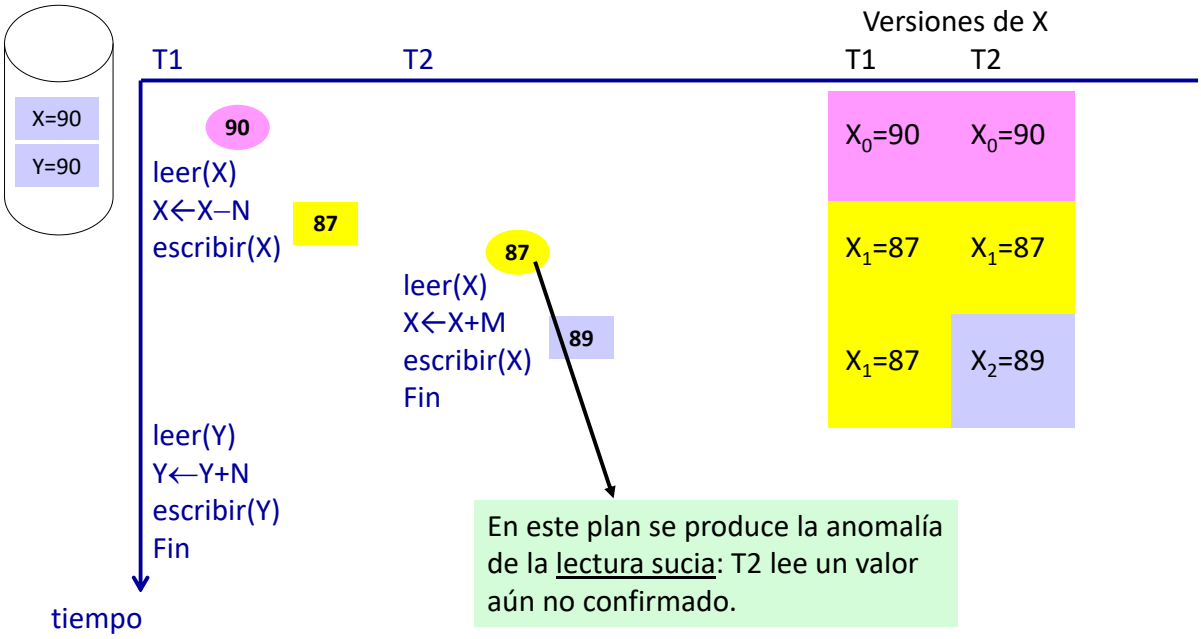
En este plan concurrente, no se produce la anomalía de la “pérdida de actualizaciones”, pero se podría haber producido la anomalía de la “lectura sucia” (punto 7 del tema).

6 Protocolos multiversión.

Plan D: admitido por el protocolo

Ejemplo 1

Constantes: M=2 N=3



MT(T1) < MT(T2) → Plan en serie cronológico T1-T2

126

6 Protocolos multiversión.

Protocolo multiversión basados en marcas de tiempo (MV):

- ✓ Siempre se satisfacen las operaciones de lectura.
- ✓ El plan en serie equivalente es el plan determinado por las marcas de tiempo de las transacciones (cronológico).
- ✓ Puede provocar el problema de la lectura sucia.
- ✓ Exige más espacio de almacenamiento que los otros protocolos.

127

El protocolo B2F sólo permite que una transacción lea un dato actualizado por otra transacción cuando que esta última lo desbloquea.

El protocolo OMT permite que una transacción lea un dato actualizado por otra transacción, siempre que esta última sea más antigua que la primera.

El protocolo MV siempre permite que una transacción lea un elemento de datos.

El protocolo B2F no permite que dos transacciones actualicen concurrentemente un elemento de datos.

El protocolo OMT permite que dos transacciones actualicen un elemento de datos, si lo hacen en el orden cronológico de sus transacciones.

El protocolo MV permite que las transacciones actualicen elementos de datos en cualquier orden, siempre que el orden de las lecturas ya realizadas lo permita (plan en serie cronológico).

6 Protocolos multiversión.

Algoritmo multiversión (MV):

La transacción T emite una operación **leer(X)**:

a) **Buscar** la versión X_i que tiene la marca de tiempo de escritura mayor entre todas las versiones de X que cumplen que MT_escritura(X_i) <= MT(T), **devolver** a T el valor de X_i y **asignar** a MT_lectura(X_i) el **mayor** valor entre los valores MT(T) y MT_lectura(X_i).

b) Si el elemento de datos X no existe, entonces la operación no puede ejecutarse.

Una operación de lectura siempre se satisface*

* Si el dato a leer existe.

128

En lectura, a cada transacción se le proporciona el valor del elemento de datos que correspondería si las transacciones se ejecutasen serialmente en su orden cronológico.

6 Protocolos multiversión.

Algoritmo multiversión (MV):

La transacción T emite una operación **escribir(X)**:

a) Si la versión X_i es la versión que tiene la marca de tiempo de escritura mayor entre todas las versiones de X tales que MT_escritura(X_i) <= MT(T) y MT_lectura (X_i) > MT(T), entonces **abortar** T;

b) Si no se cumple la condición de (a), entonces **crear** una nueva versión X_j y asignar a MT_lectura (X_j) y a MT_escritura (X_j) el valor MT(T).

La transacción T sólo puede generar una nueva versiónde un dato, si la versión anterior de ese dato no ha sido leída por una transacción posterior a T.

129

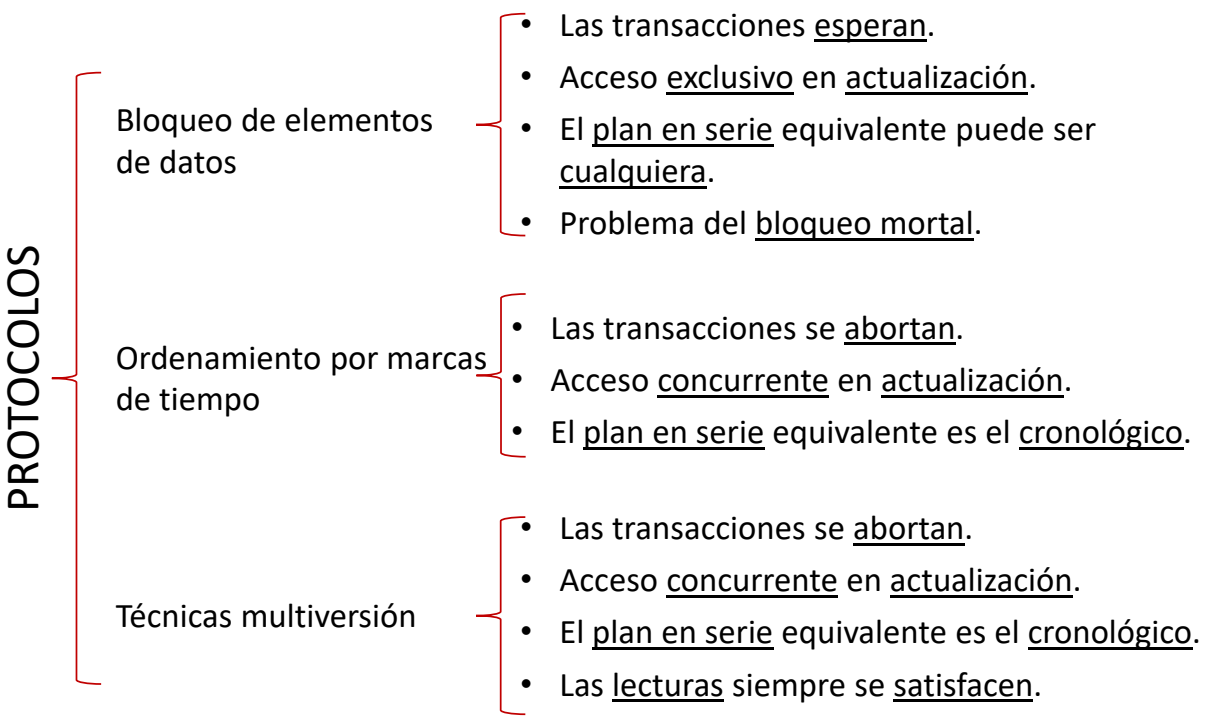
La transacción T sólo puede generar una nueva versión, si la versión anterior no ha sido leída por una transacción posterior a T.

T puede crear una nueva versión incluso cuando pueden existir versiones de X generadas por transacciones posteriores a T.

No se aborta necesariamente la transacción más reciente (más joven).

Si un elemento de datos existe, siempre tiene una versión con marca de lectura y marca de escritura.

Control de la concurrencia.



El protocolo multiversión es el más flexible, el que admite más concurrencia.

Control de la concurrencia.

1 Ejecución concurrente de transacciones: anomalías.

2 Control de la concurrencia en SQL.

3 Planes serializables por conflictos.

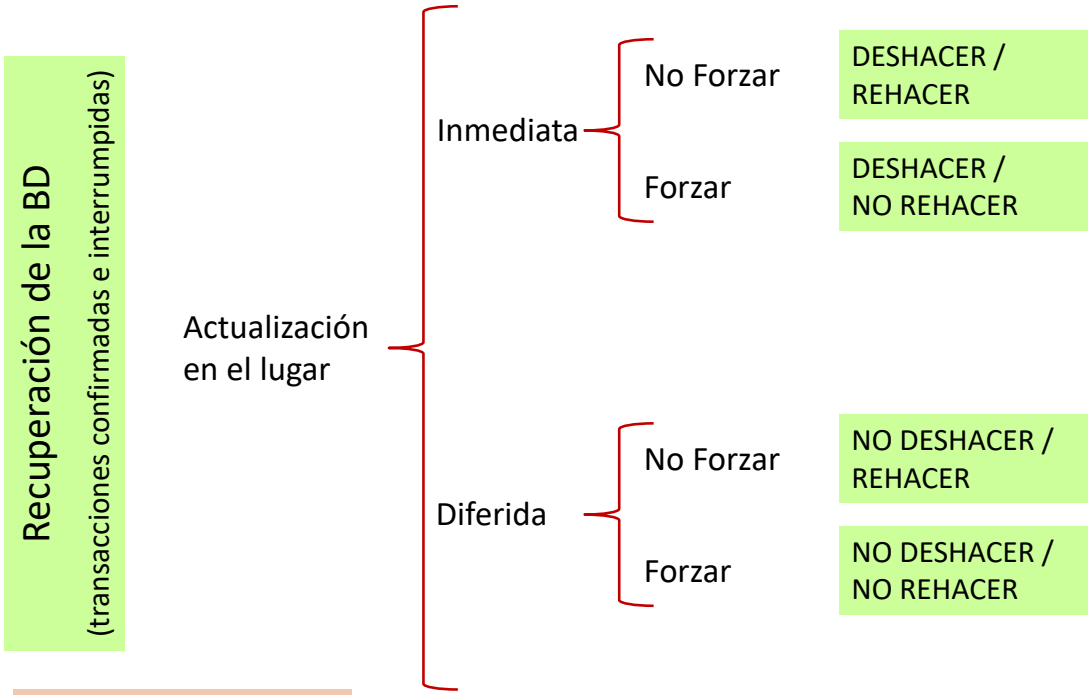
4 Protocolos de bloqueo.

5 Protocolos de ordenamiento por marcas de tiempo.

6 Protocolos multiversión.

7 Concurrencia y recuperación.

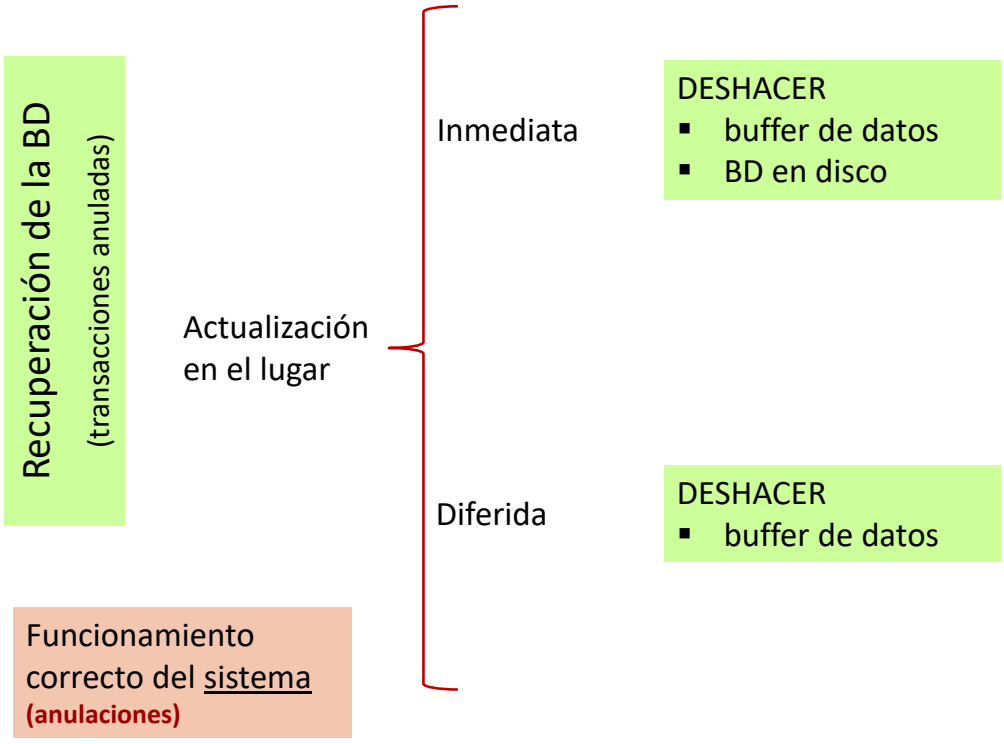
7 Concurrencia y recuperación.



Fallo del sistema
(pérdida de MP)

132

7 Concurrencia y recuperación.



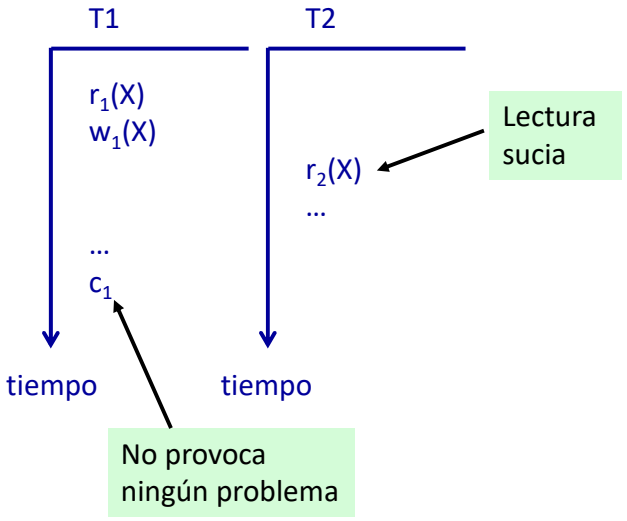
133

Recuperación de transacciones falladas: anuladas o interrumpidas

- ✓ Cuando una transacción T es **anulada** (usuario o SGBD) o **interrumpida** por algún motivo, el SGBD debe **deshacer** sus efectos (actualizaciones) de forma que el resto de transacciones se sigan ejecutando como si T no hubiese ocurrido.
- ✓ El procedimiento para realizar la recuperación dependerá de la estrategia de actualización de la BD seguida por el SGBD (inmediata, diferida) y del protocolo para el control de la concurrencia (control de la **lectura sucia**).

7 Concurrencia y recuperación.

Lectura sucia



Anomalía de la lectura sucia: T2 lee un dato actualizado por T1 que todavía no ha sido confirmada

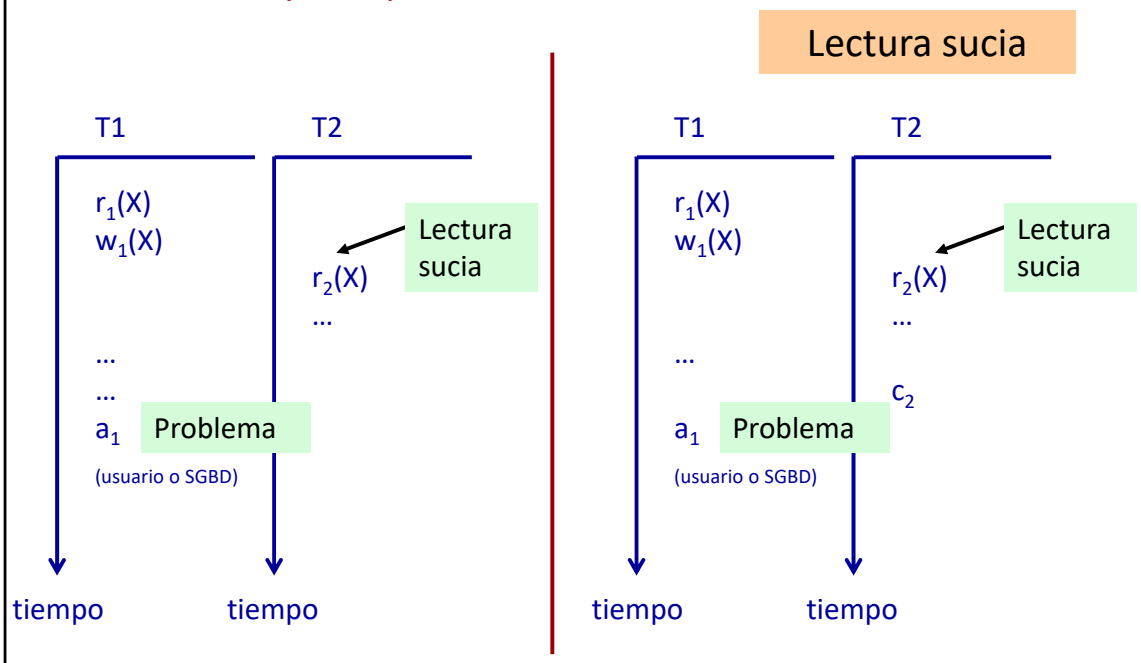
135

En el plan, se presenta la anomalía de la “lectura sucia”.

Sin embargo, ésta lectura sucia no provoca ningún problema, el dato (actualizado por T1) y leído por T2 (lectura sucia) finalmente ha sido confirmado por T1.

Como se ha dicho anteriormente, esta anomalía es realmente una seudoanomalía, la ejecución concurrente es correcta, el plan es serializable. La lectura sucia se convierte en un problema cuando la transacción de la que se ha hecho la lectura sucia (T1) es anulada o interrumpida por un fallo.

7 Concurrencia y recuperación.



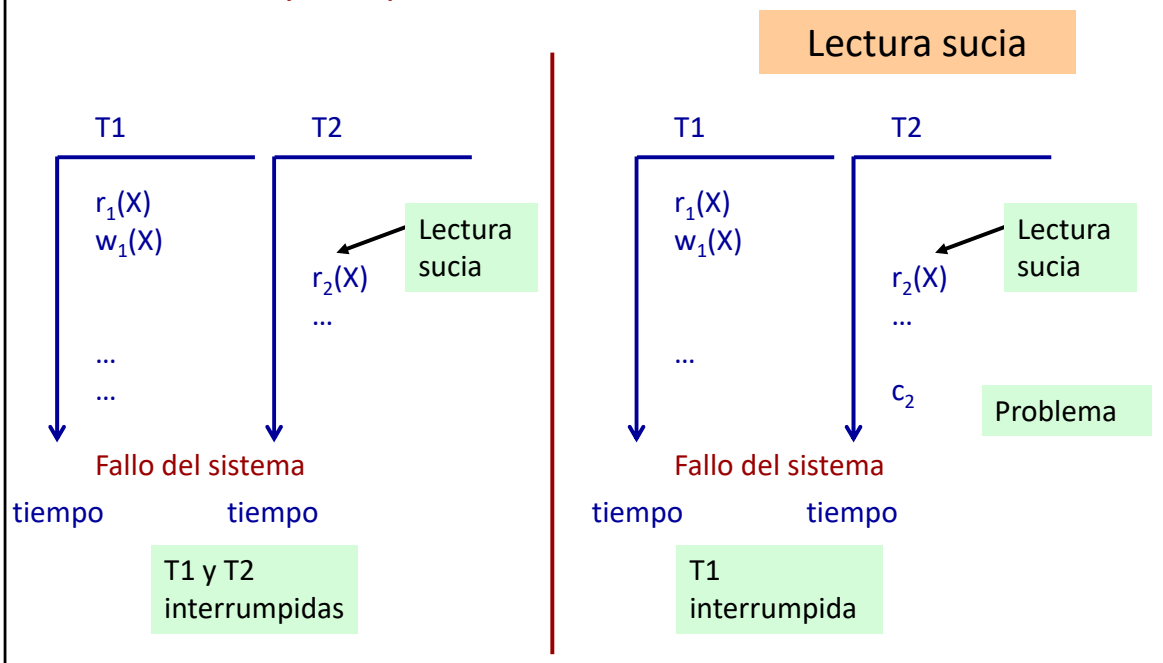
Anomalía de la lectura sucia: T2 lee un dato actualizado por T1 que todavía no ha sido confirmada

136

En estos dos ejemplos, la lectura sucia de T2 se convierte en problema porque finalmente la transacción T1 es anulada.

El problema es mayor en el plan de la derecha, ya que la transacción T2 ya ha sido confirmada por el usuario. Se ha confirmado una transacción que ha utilizado un dato falso (ha sido anulado).

7 Concurrencia y recuperación.



Anomalía de la lectura sucia: T2 lee un dato actualizado por T1 que todavía no ha sido confirmada

137

En estos dos ejemplos, hay la lectura sucia ya que T2 lee un dato modificado por la transacción T1 que aún no ha sido confirmada.

- En el plan de la izquierda no se presenta ningún problema ya que ambas transacciones se han interrumpido y por lo tanto sus efectos deberán deshacerse.
- En el plan de la derecha sí que aparece un problema ya que la transacción T2 ya ha sido confirmada por el usuario. Se ha confirmado una transacción que ha utilizado un dato falso (ha sido anulado).

Recuperación de transacciones falladas: anuladas o interrumpidas

Cuando una transacción T es anulada (o interrumpida), si el protocolo de control de la concurrencia no evita la anomalía de la **lectura sucia**, deberán **anularse en cascada** todas aquellas transacciones que hayan hecho una lectura sucia de T:

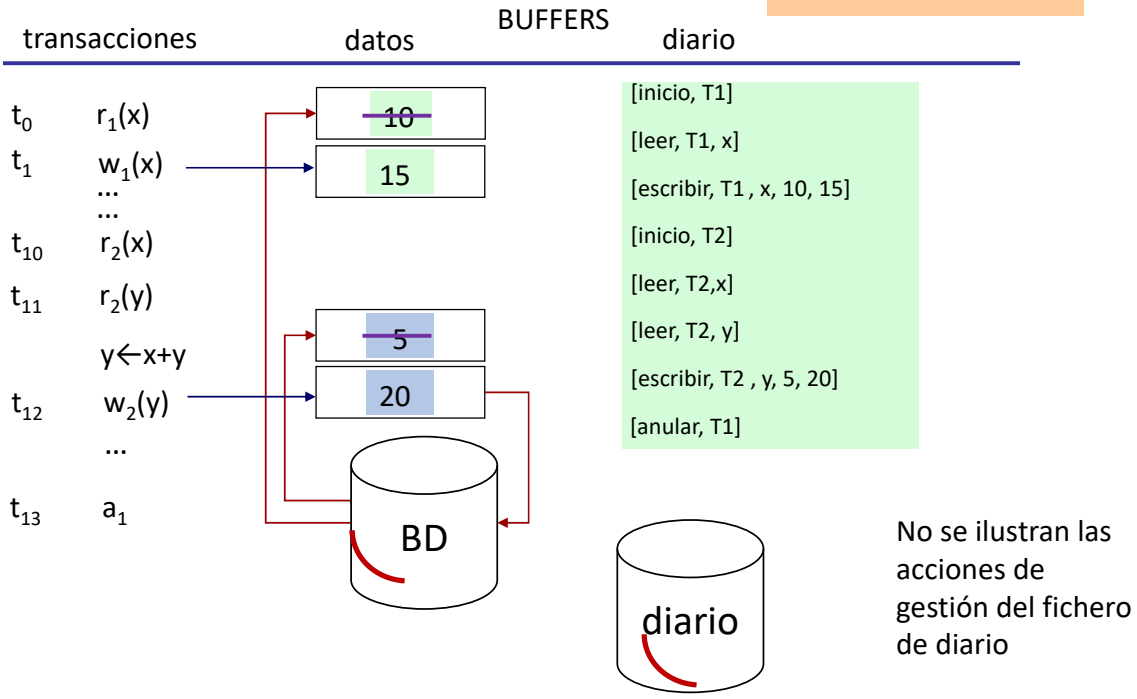
⬇

Anulación en cascada: Una transacción debe ser anulada porque ha leído (lectura sucia) un elemento de datos de una transacción que falla, ya sea por anulación o interrupción (en este último caso la anulación se realizará en la recuperación).

138

7 Concurrencia y recuperación.

Lectura sucia



No se ilustran las acciones de gestión del fichero de diario

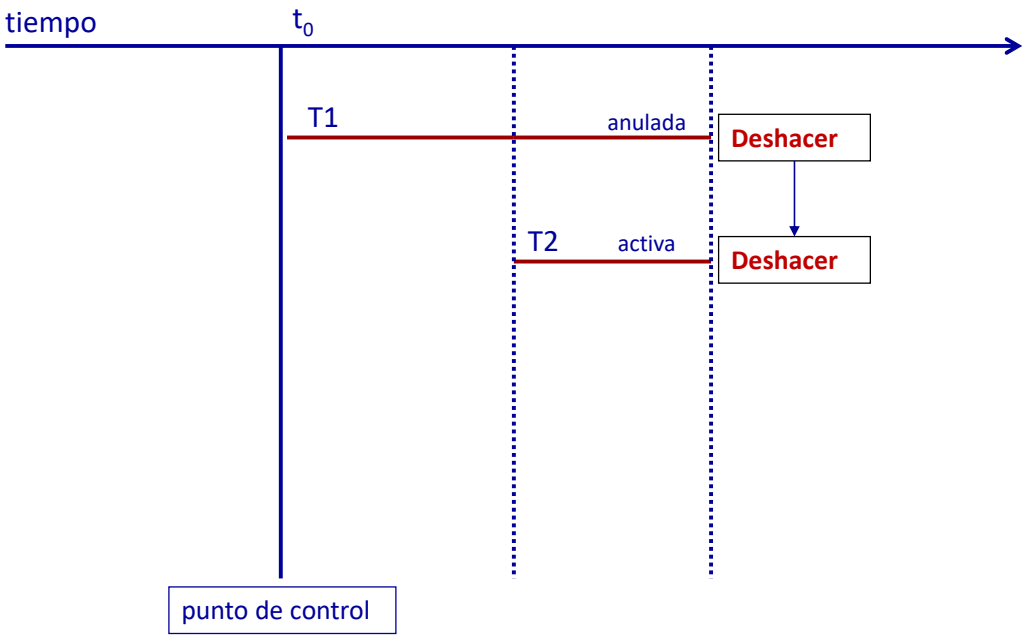
Actualización de la BD: en el lugar + inmediata + no forzar

139

En el ejemplo, se presenta la ejecución concurrente de dos transacciones (T1 y T2). Como se puede observar, la transacción T2 hace una lectura sucia de T1, y finalmente T1 es anulada por el usuario, mientras T2 está activa.

La anulación de T1 provocará la anulación en cascada de T2 que todavía no ha finalizado, como se muestra en la transparencia siguiente.

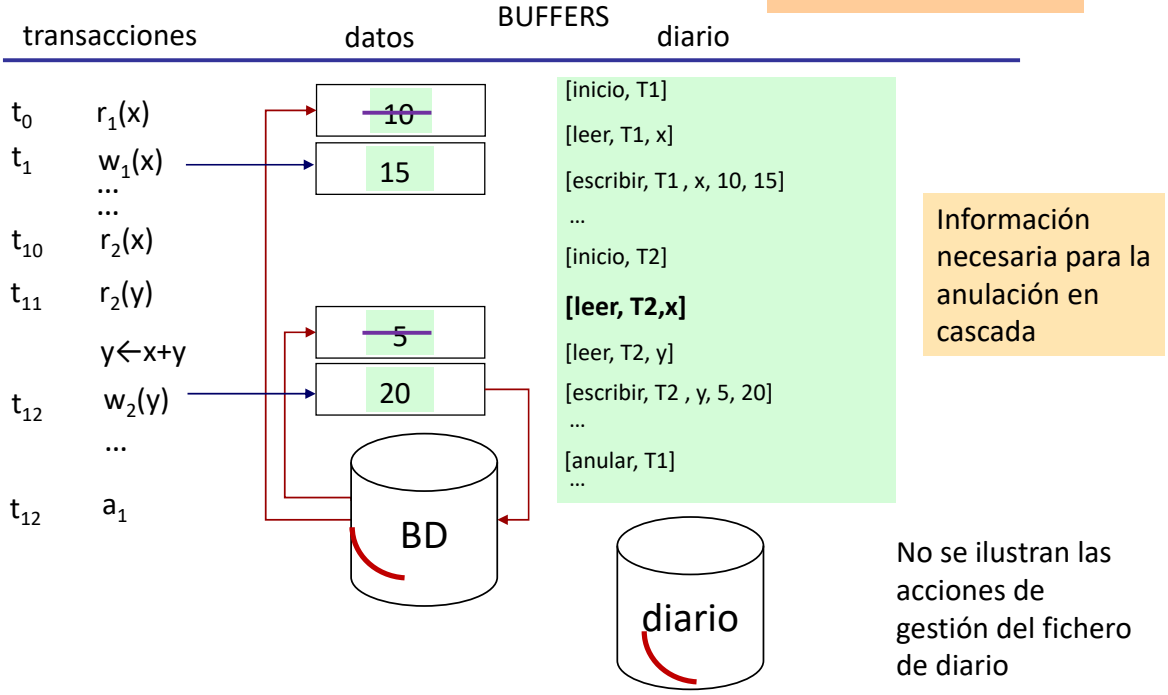
Anulación en cascada



T2 ha leído (**lectura sucia**) un elemento de datos actualizado por una transacción que ha sido anulada: T2 debe ser anulada sin haber finalizado.

7 Concurrencia y recuperación.

Lectura sucia



Información necesaria para la anulación en cascada

No se ilustran las acciones de gestión del fichero de diario

Actualización de la BD: en el lugar + inmediata + no forzar

141

Para realizar la anulación en cascada de una transacción, son necesarias las entradas del diario de tipo [leer, T, X], que no habían sido utilizadas hasta ahora para la recuperación de transacciones.

En el ejemplo, el algoritmo de recuperación, frente la anulación de T1, necesita saber qué transacciones han hecho una lectura sucia de T1. Esto lo puede conocer usando la información almacenada en el diario.

Recuperación de transacciones falladas: anuladas o interrumpidas

Cuando una transacción es anulada (o interrumpida), si el protocolo de control de la concurrencia controla (**evita**) la lectura sucia, no puede producirse el problema de la anulación en cascada:

Las entradas del diario de tipo [leer, T, X] no son necesarias.

142

7 Concurrencia y recuperación.

Planes y recuperabilidad:

- ✓ Las transacciones pueden ser **anuladas** por el usuario o por el SGBD (comprobación de RI), o pueden ser **interrumpidas** por un fallo.
- ✓ Los protocolos de control de la concurrencia pueden **anular** (y revertir en algunos protocolos) transacciones para asegurar la seriabilidad de los planes.
- ✓ Si el protocolo de control de la concurrencia **no evita** la anomalía de la **lectura sucia**, la anulación de una transacción puede provocar **anulaciones en cascada** de otras transacciones.

143

Los protocolos que anulan transacciones son:

- Protocolo B2F: puede anular transacciones para resolver el bloqueo mortal.
- Protocolo OMT: anula transacciones para asegurar la seriabilidad del plan y revierte posteriormente la transacción anulada.
- Protocolo MV: anula transacciones para asegurar la seriabilidad del plan. No las revierte posteriormente.

En los tres protocolos (B2F, OMT, MV), existen variantes (planes estrictos) que evitan la lectura sucia y, por lo tanto, la anulación en cascada.

7 Concurrencia y recuperación.

Planes y recuperabilidad:

Objetivo: caracterizar planes de ejecución concurrentes para flexibilizar la recuperación de transacciones falladas (anuladas o interrumpidas).

- ✓ Planes recuperables.
- ✓ Planes sin anulación en cascada.
- ✓ Planes estrictos.

144

Como la lectura sucia puede provocar la anulación en cascada, se trata de extender los protocolos de control de la concurrencia, para controlar los efectos de dicha anomalía, de forma que la anulación de una transacción no tenga efectos colaterales, evitando que tenga consecuencias muy graves y que sea costosa de ejecutar.

Estas extensiones de los protocolos de control de la concurrencia evitan la anomalía de la lectura sucia (planes sin anulación en cascada) o la admiten, pero evitan que en el fenómeno de la anulación en cascada se llegue a anular una transacción que ya ha sido confirmada (planes recuperables).

Planes y recuperabilidad: planes recuperables

Plan recuperable: en un plan recuperable, una transacción T no se puede confirmar antes de que se confirmen todas las transacciones que han actualizado (escrito) un elemento de datos leído por T.

↓

En un plan recuperable una transacción confirmada nunca se deshace.

↓

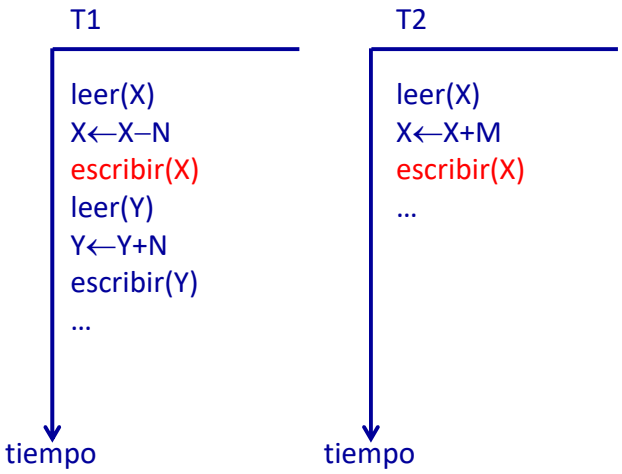
No se evita la anomalía de la **lectura sucia**.

145

Los planes para el control de la concurrencia, que admiten sólo planes recuperables, aseguran que una transacción confirmada nunca será anulada.

El protocolo para el control de la concurrencia de cualquier SGBD debe asegurar, como mínimo, planes recuperables.

Ejemplo 1

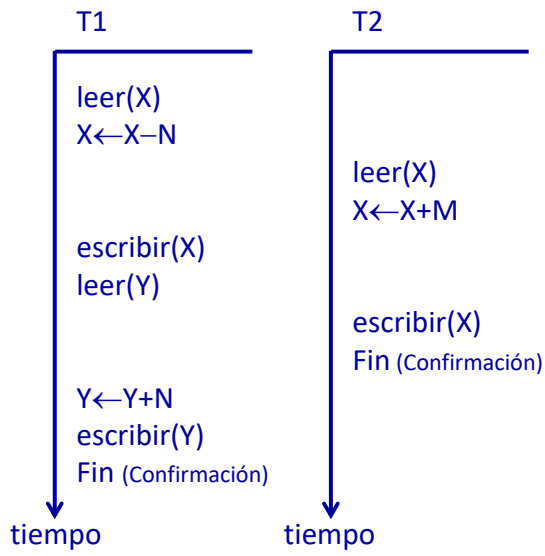


146

Vamos a analizar ejecuciones concurrentes de estas dos transacciones, cambiando la forma de finalizar cada una de ellas, para ilustrar la idea de plan recuperable.

7 Concurrencia y recuperación.

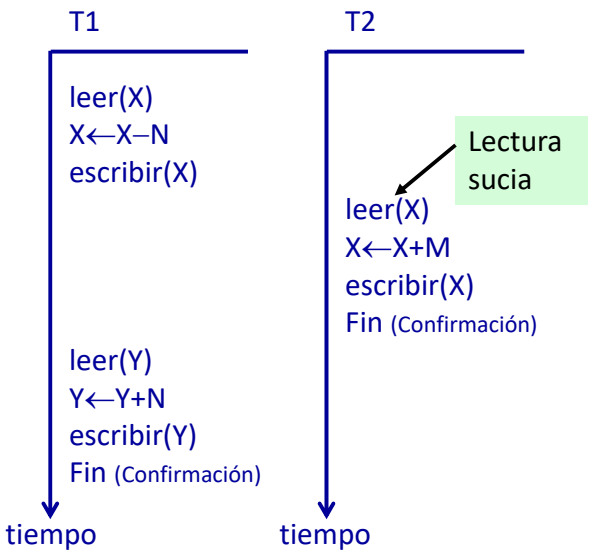
Ejemplo 1



Pérdida de actualizaciones

C: Plan recuperable

Ninguna transacción lee un dato actualizado por la otra



D: Plan NO recuperable

T2 lee un dato actualizado por T1 y T2 y se confirma antes que T1

147

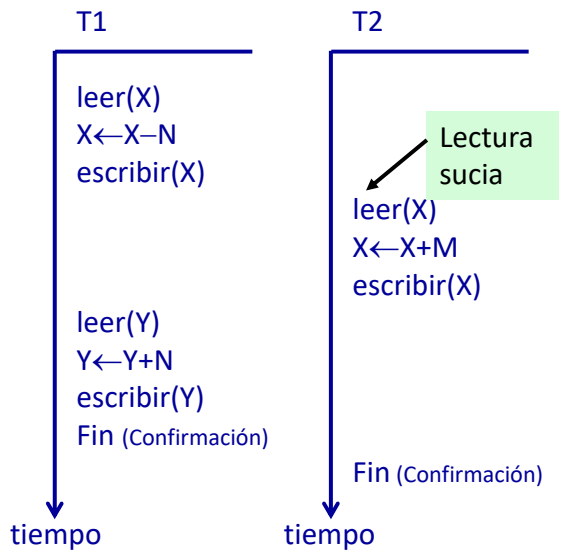
C no es un plan serializable.

D es un plan serializable.

Aunque D no da problemas, no es recuperable: si T1 fuese anulada habría que anular en cascada T2 que ya ha sido confirmada.

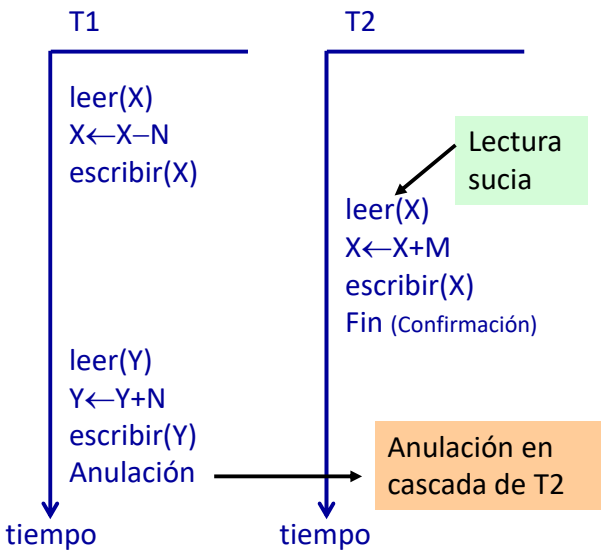
7 Concurrencia y recuperación.

Ejemplo 1



E: Plan recuperable

T2 lee un dato actualizado por T1, pero es confirmada después de que se confirme T1



F: Plan NO recuperable

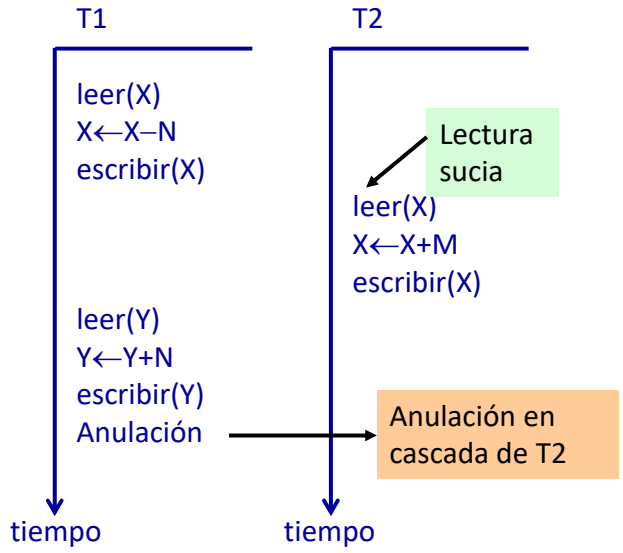
T2 lee un dato actualizado por T1 y es confirmada antes de que se confirme T1

148

Los planes E y F son serializables, aunque se produzca la anomalía de la lectura sucia.

7 Concurrencia y recuperación.

Ejemplo 1



G: Plan recuperable

T2 lee un dato actualizado por T1, pero no se confirma antes de que se confirme T1

149

G es un plan serializable y recuperable, con lectura sucia.

Se produciría la anulación en cascada.

Planes y recuperabilidad: planes recuperables

En un plan recuperable una transacción confirmada nunca debe deshacerse pero es posible que aparezca la lectura sucia y el problema de la anulación en cascada (plan G).

Anulación en cascada en planes recuperables: Una transacción **que todavía no ha finalizado** debe ser anulada porque ha leído un elemento de datos de una transacción que ha fallado.

La anulación en cascada puede consumir muchos recursos porque la anulación de una transacción puede provocar muchas anulaciones en cadena.

150

En los protocolos que aseguran planes concurrentes recuperables, puede seguir produciéndose la lectura sucia y el fenómeno de la anulación en cascada.

Si se quiere evitar el fenómeno de la anulación en cascada, hay que evitar la lectura sucia.

Esto es lo que hacen los protocolos que aseguran planes sin anulación en cascada.

Planes y recuperabilidad: planes “sin anulación en cascada”

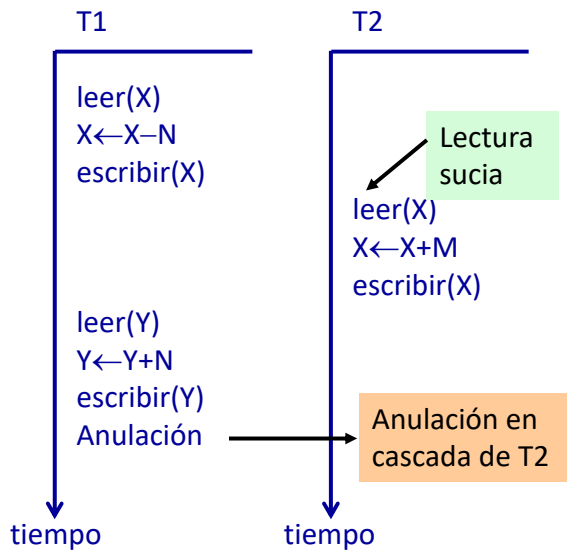
Plan “sin anulación en cascada”: las transacciones del plan **sólo pueden leer** elementos de datos actualizados (escritos) por transacciones confirmadas.

Se **evita** la anomalía de la **lectura sucia**.

Si queremos evitar la anulación en cascada tenemos que prohibir la lectura sucia.

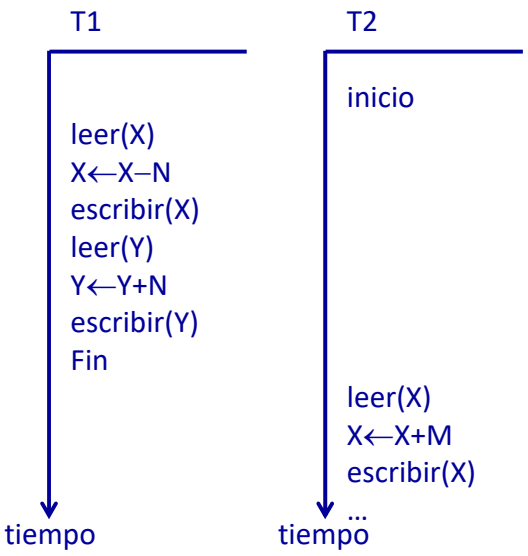
En un protocolo que controle planes “sin anulación en cascada”, la lectura de T2 del elemento X, se quedaría en espera hasta que T1 finalizase: si T1 se anula se deshace su actualización y T2 leerá el valor original de X, si T1 se confirma T2 podrá leer el valor actualizado de X.

Ejemplo 1



G: Plan con anulación en cascada

T2 lee un dato actualizado por T1, pero no se confirma antes de que se confirme T1



H: Plan "sin anulació en cascada"

T2 lee un dato actualizado por T1, pero no se confirma antes de que se confirme T1

152

G es un plan serializable y recuperable, con lectura sucia.

Se produciría la anulación en cascada.

El plan H es una plan sin anulación en cascada, en este caso se ha añadido una operación inicial a T2 al ejemplo original para evitar que sea un plan en serie y dé lugar a pensar que para ser un plan sin anulación en cascada, se tiene que ser un plan en seri.

Planes y recuperabilidad: planes “sin anulación en cascada”

Plan “sin anulación en cascada” ➡ No hay lectura sucia

Plan “sin anulación en cascada” ➡ Plan “recuperable”

Planes y recuperabilidad: planes "sin anulación en cascada"

P: … $w_1(x)$, …, $w_2(x)$, $\mathbf{a_1}$ …

P es **"sin anulación en cascada"**.

- ✓ Aunque no se da la lectura sucia (y por lo tanto tampoco la anulación en cascada), si T1 se anula, la recuperación es más compleja: si se aplicara el *valor_antes* de la actualización de T1 se perdería la actualización de T2.
- ✓ La recuperación obliga al SGBD a analizar si ha habido actualizaciones posteriores del mismo elemento de datos.

154

En este ejemplo, T1 se anula y no desencadena la anulación en cascada de T2. T2 no ha hecho una lectura sucia de T1.

Como es sabido, la anulación de T1 implica deshacer sus actualizaciones, es decir, aplicar a X el *valor_antes* de la actualización $w_1(X)$. Para ejecutar correctamente esta anulación, el SGBD debe asegurarse de que X no ha sido actualizada posteriormente por otras transacciones, como sucede en el ejemplo.

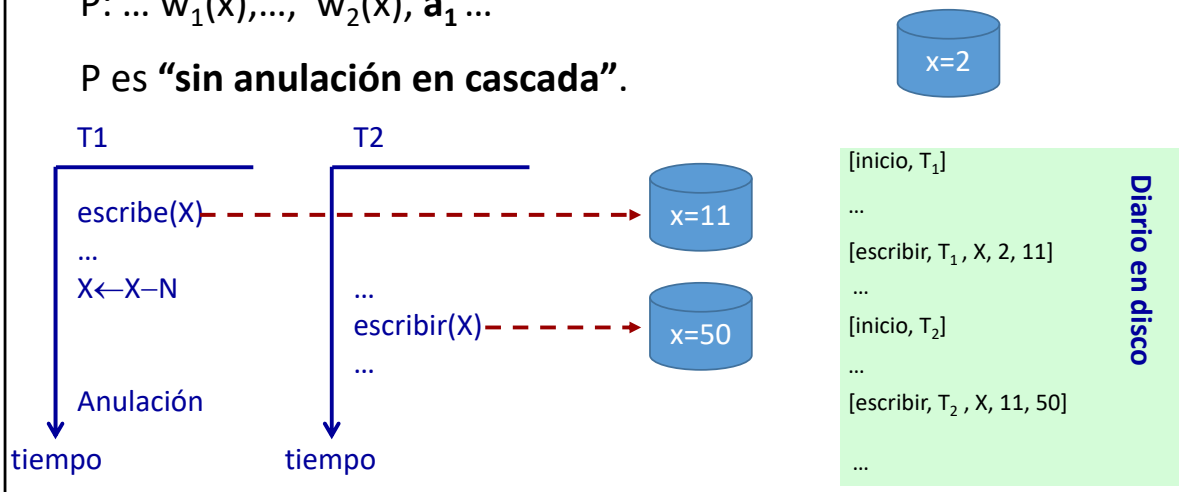
Estas comprobaciones adicionales complican la ejecución del proceso de anulación. Para evitar esta situación, se definen los planes estrictos.

3 Planes serializables por conflictos.

Planes y recuperabilidad: planes "sin anulación en cascada"

P: … $w_1(x)$,…, $w_2(x)$, $\mathbf{a_1}$ …

P es **"sin anulación en cascada"**.



- ✓ Si al anularse T1 se pone en X el valor 2 (*valor_antes* de la escritura de T1) se pierde el valor 50 escrito por T2.
- ✓ No hay lectura sucia porque T2 no lee lo que ha escrito T1

155

En la transparencia se muestra como si el valor de X se escribiera en disco aunque podría ser que la modificación sólo estuviera en memoria principal.

Planes y recuperabilidad: planes estrictos

Plan estricto: las transacciones del plan **no pueden leer ni escribir** un elemento de datos X hasta que no finalice (confirmación o anulación) la última transacción que escribió X.

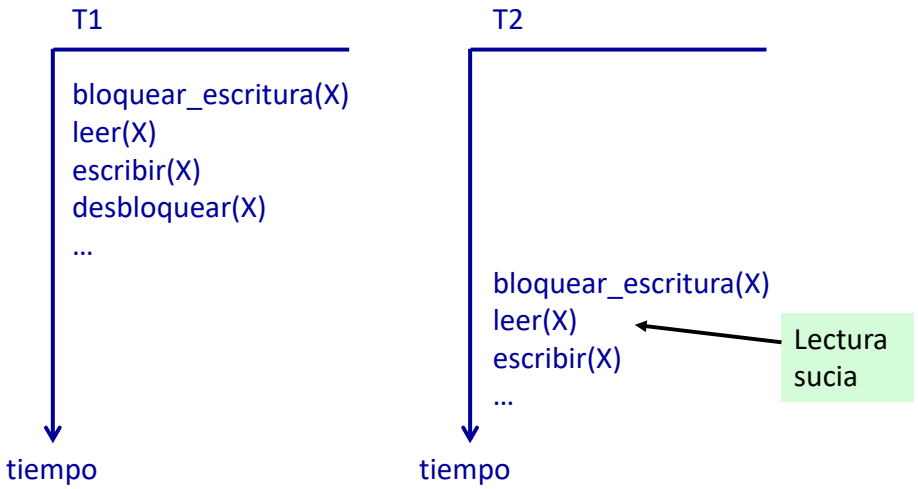
Los planes estrictos simplifican el proceso de recuperación: deshacer una operación escribir(X) de una transacción fallada significa restaurar (exclusivamente) el valor anterior (*valor_antes*) del elemento X.

156

Si el protocolo de control de la concurrencia asegura planes estrictos, los algoritmos de recuperación (Tema 9) son válidos: aplican el procedimiento DESHACER a todas las transacciones interrumpidas por el fallo (en un entorno concurrente puede haber varias transacciones activas).

Análisis del tratamiento de la *lectura sucia* en los protocolos:

Protocolo de bloqueo en dos fases explícito (B2F)



157

Con el protocolo B2F explícito puede aparecer el problema de la "lectura sucia", que obliga a la anulación en cascada, como se observa en el ejemplo.

Análisis del tratamiento de la *lectura sucia* en los protocolos:

Bloqueo en dos fases **estricto**: una transacción no libera ninguno de sus bloqueos exclusivos hasta que finaliza la transacción.

⬇

Ninguna transacción puede leer ni escribir un elemento de datos escrito por otra transacción hasta que esta última finalice.

B2F Implícito ➡ B2F Estricto

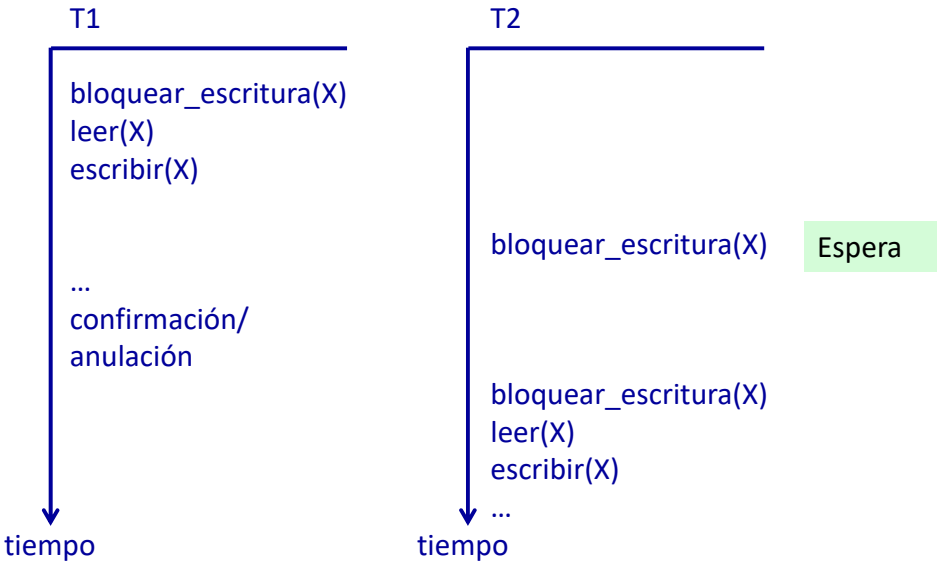
158

El protocolo B2F estricto significa que cuando una transacción bloquea para escritura un elemento de datos X, es porque ninguna otra transacción activa ha accedido a él y además ninguna otra transacción podrá hacerlo hasta que T finalice. Esto significa que sólo se permite concurrencia en lectura.

El bloqueo/desbloqueo implícito implica B2F estricto.

Análisis del tratamiento de la *lectura sucia* en los protocolos:

Protocolo de bloqueo en dos fases **estricto** (B2F)

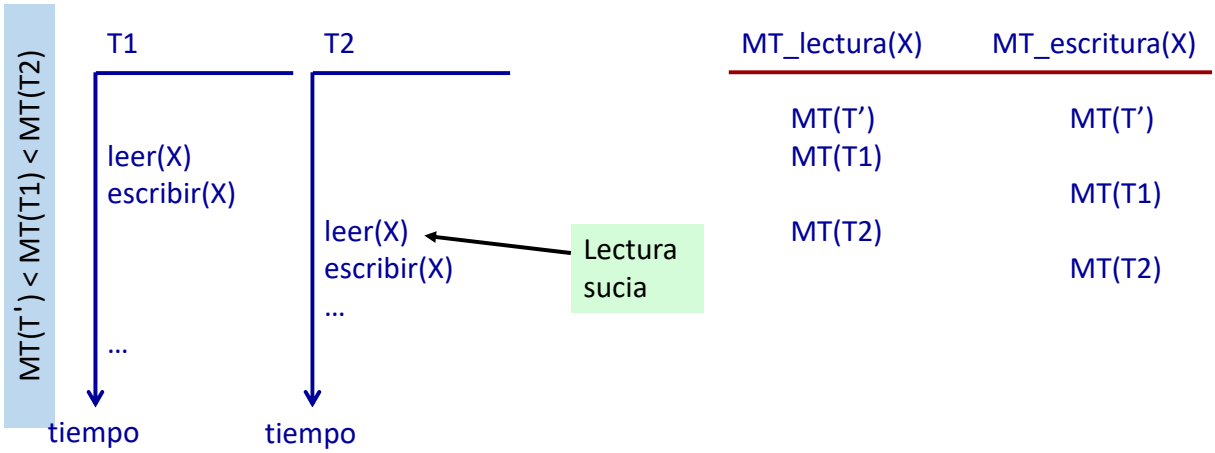


159

En este plan concurrente no se produce la anomalía de la “lectura sucia”.

Análisis del tratamiento de la *lectura sucia* en los protocolos:

Protocolo de ordenamiento por marcas de tiempo (OMT)



160

En este plan concurrente se produce la anomalía de la “lectura sucia”.

Análisis del tratamiento de la *lectura sucia* en los protocolos:

Protocolo de ordenamiento OMT **estricto**: Una transacción T que emite una operación leer(X) o escribir(X) tal que MT(T) > MT_escritura(X), sufre un retraso de su operación de lectura o escritura hasta que la transacción T' que escribió el valor de X (MT_escritura(X) = MT(T')) finalice.

Ninguna transacción puede leer ni escribir un elemento de datos escrito por una transacción hasta que esta última finalice.

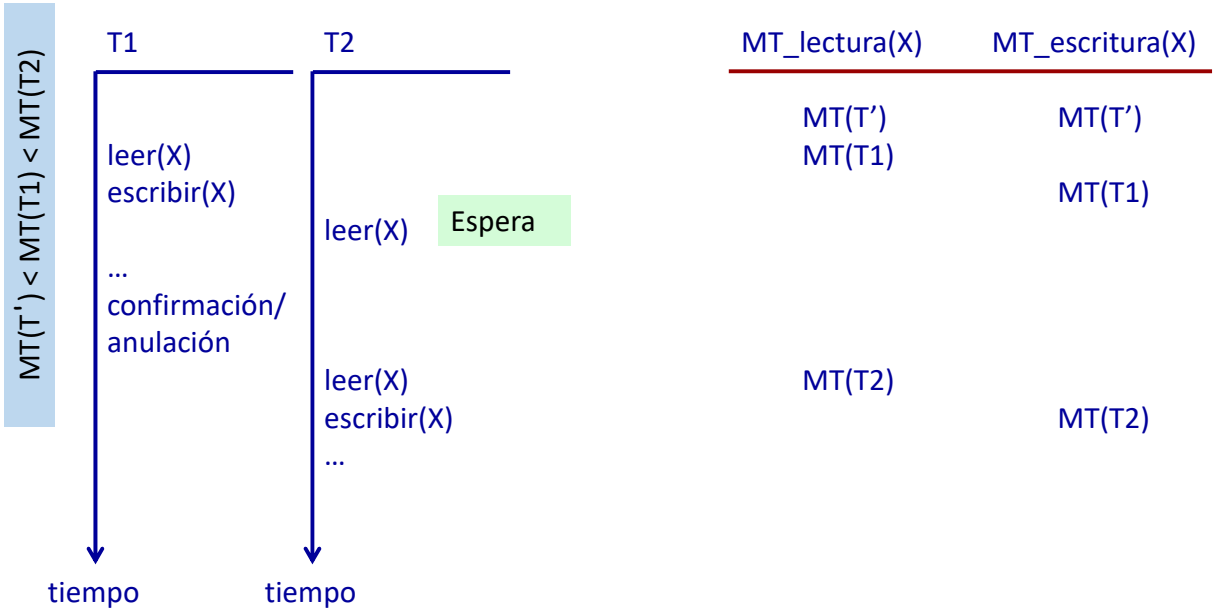
La operación de T debería ser aceptada según las reglas de OMT, pero se retrasa hasta asegurar que la actualización de X es confirmada.

El algoritmo de ordenamiento estricto evita la anomalía de la "lectura sucia" y por lo tanto el fenómeno de la "anulación en cascada".

El protocolo de ordenamiento estricto actúa como B2F estricto, cuando una transacción accede en escritura a un elemento de datos, ninguna otra transacción puede acceder a él (ni en lectura ni en escritura). Esto quiere decir que sólo se admite la concurrencia en lectura.

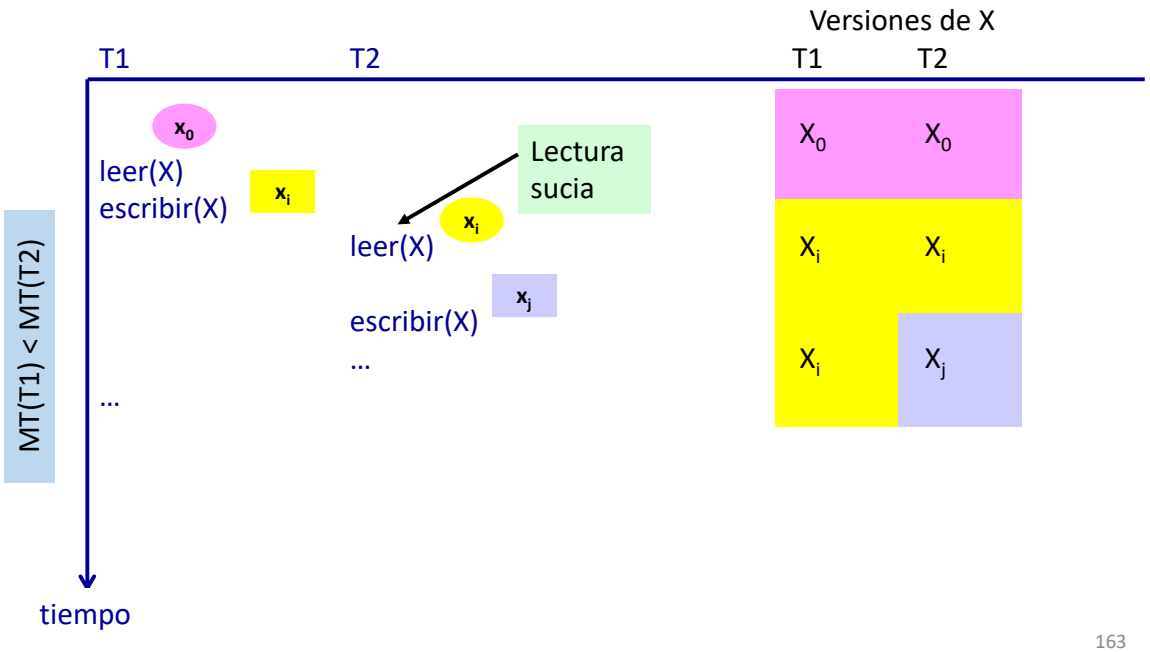
Análisis del tratamiento de la *lectura sucia* en los protocolos:

Protocolo de ordenamiento por marcas de tiempo **estricto** (OMT)



Análisis del tratamiento de la *lectura sucia* en los protocolos:

Protocolo multiversión MV



163

En este plan concurrente se produce la anomalía de la “lectura sucia”.

Análisis del tratamiento de la *lectura sucia* en los protocolos:

Protocolo de multiversión **estricto**: Una transacción T que emite una operación leer(X) tal que MT(T) > MT_escritura(X), sufre un retraso de su operación de lectura hasta que la transacción T' que escribió el valor de X (MT_escritura(X) = MT(T')) finalice.

Ninguna transacción puede leer un elemento de datos escrito por una transacción hasta que esta última finalice.

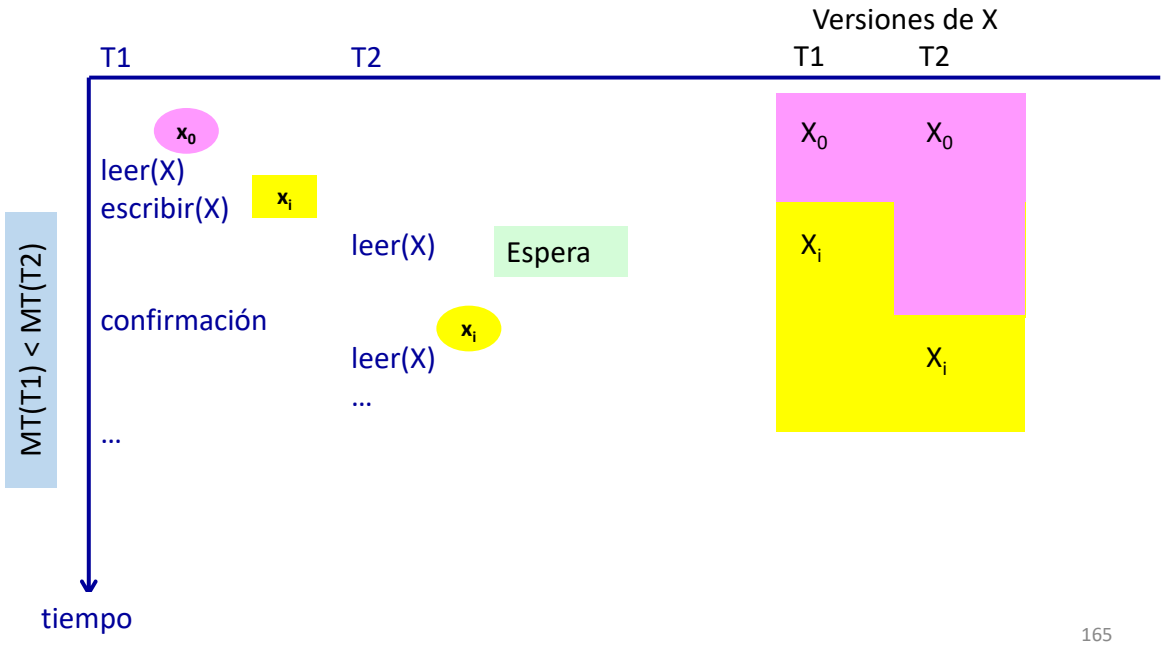
La operación de T debería ser aceptada según las reglas de MV, pero se retrasa hasta asegurar que la actualización de X es confirmada.

El algoritmo de MV estricto evita la anomalía de la "lectura sucia" y por lo tanto el fenómeno de la "anulación en cascada".

El protocolo de MV estricto actúa como B2F estricto, cuando una transacción accede en escritura a un elemento de datos, ninguna otra transacción puede acceder a él (en lectura).

Análisis del tratamiento de la *lectura sucia* en los protocolos:

Protocolo multiversión **estricto** MV

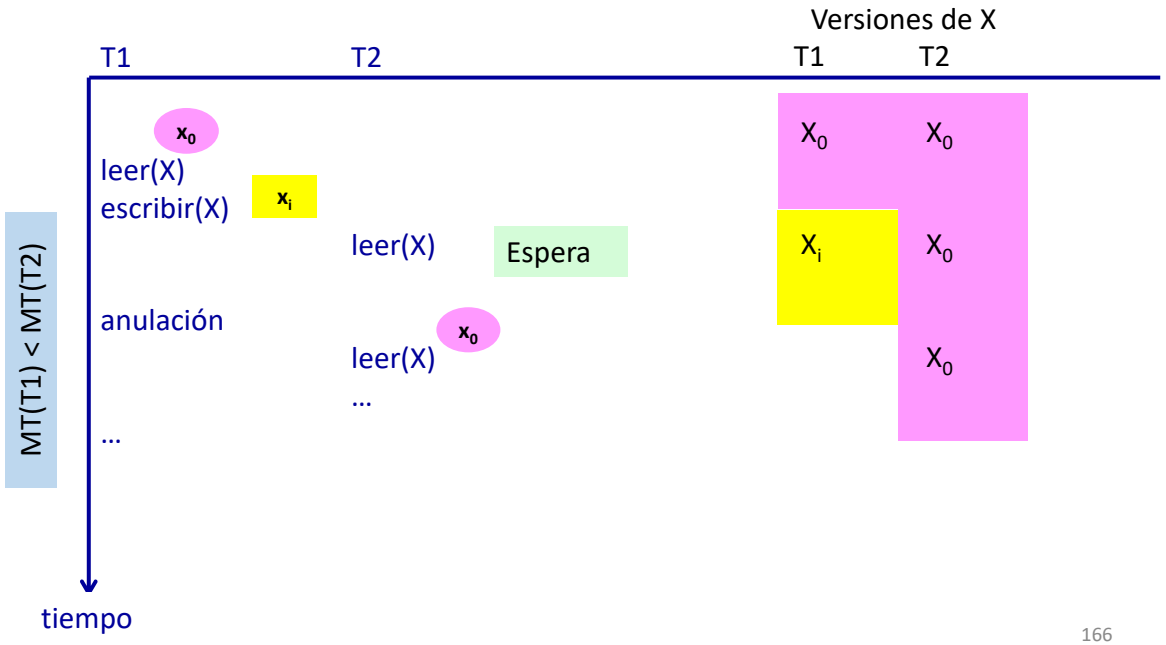


165

En este plan concurrente no se produce la anomalía de la “lectura sucia”.

Análisis del tratamiento de la *lectura sucia* en los protocolos:

Protocolo multiversión **estricto** MV



166

En este plan concurrente no se produce la anomalía de la “lectura sucia”.