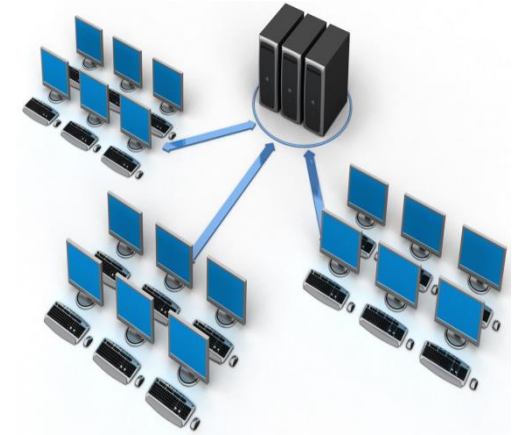


Tema 5

Transacciones y Concurrencia en Bases de Datos

Parte 2/2

**Grado en
Ingeniería
Informática**



**Bases de
Datos**

2020/21

Departamento de Tecnologías de la Información
Universidad de Huelva

Objetivos

- ❑ Conocer la problemática asociada a la concurrencia de transacciones en los sistemas de bases de datos
- ❑ Comprender diversas técnicas para el control de la concurrencia empleadas por los SGBD

Contenidos

...

5. Concurrency en Bases de Datos: Introducción
6. Técnicas de Control de la Concurrency
 - 6.1. Protocolos basados en bloqueos (reservas)
 - 6.2. Protocolo de bloqueo en dos fases
 - 6.3. Protocolos basados en grafos
 - 6.4. Protocolos basados en marcas temporales
7. El problema del interbloqueo: Temporización y detección
8. El problema del bloqueo Indefinido

5. Concurrency en BD: Introducción

- Los sistemas de bases de datos, **según el número de usuarios** que pueden utilizarlos de forma concurrente, se clasifican en sistemas **monousuario** y **multiusuario**
- Varios usuarios pueden usar un mismo equipo a la vez gracias a la **multiprogramación**: el computador puede **procesar al mismo tiempo varias transacciones**
 - Si el equipo tiene **varias CPU**, es posible el **procesamiento simultáneo** (paralelo) de transacciones
 - Si sólo hay **una CPU**, el **SO de multiprogramación** reparte el tiempo de CPU entre las transacciones:

ejecución concurrente intercalada

▲ modelo que supondremos

- **Varias transacciones** introducidas por usuarios, que se ejecutan de manera **concurrente**, pueden **leer/modificar** los **mismos** elementos almacenados en la base de datos
- Razones para permitir la concurrencia:
 - Aumentar la productividad: número de transacciones ejecutadas por minuto
 - Aumentar la utilización de la CPU (menos tiempo ociosa) y Control del disco
 - Reducir el tiempo medio de respuesta de transacciones (las 'pequeñas' no esperan a las 'grandes')

¿Por qué es necesario el control de la concurrencia?

- ... porque pueden surgir **problemas** si las **transacciones concurrentes** se ejecutan de manera **no controlada**
- Una Propiedad importante de las transacciones es el **aislamiento**
- Las técnicas de control de concurrencia sirven para garantizar la no interferencia de las transacciones que se ejecutan **concurrentemente**
 - Estas técnicas aseguran planes serializables → emplean **protocolos**: conjuntos de reglas

¿Por qué es necesario el control de la concurrencia?

Transacción T1

```
leer_elemento(X);  
X:= X-N;  
escribir_elemento(X);  
leer_elemento(Y);  
Y:=Y+N;  
escribir_elemento(Y);
```

Transacción T2

```
leer_elemento(X);  
X:= X+M;  
escribir_elemento(X);
```


- Aunque las transacciones pueden ser perfectamente correctas en sí mismas, la ejecución concurrente de T1 y T2 puede producir un **resultado incorrecto, debido a la intercalación de sus operaciones**, poniendo en cuestión la integridad y la coherencia de la base de datos
- Los problemas potenciales que pueden surgir son:
 - Actualización perdida
 - Lectura no confirmada (lectura 'sucia')
 - Resumen incorrecto (análisis inconsistente)
 - Lectura no repetible

- **Objetivo** de un protocolo de control de concurrencia:
 - “ **Planificar las transacciones de forma que no ocurran interferencias entre ellas, y así evitar la aparición de los problemas mencionados** ”
- Solución obvia: **no** permitir **intercalación** de operaciones de varias transacciones
- Pero el objetivo de un SGBD multiusuario también es maximizar el grado de concurrencia del sistema
- Si **se permite la intercalación** de operaciones, existen muchos órdenes posibles de ejecución de las transacciones → **Teoría de la Serializabilidad**

6. Técnicas de Control de Concurrency

- Métodos **basados en la teoría de la serializabilidad**, que definen un **protocolo** (o conjunto de **reglas**) tal que:
 - Todas las transacciones las cumplen, o
 - El subsistema de control de concurrency del SGBD las impone (automáticamente)

“se asegura la serializabilidad de toda planificación de transacciones”

- Las **Técnicas de Control de Concurrency** se pueden clasificar en:
 - Protocolos basados en **bloqueo** 
 - Reservas
 - Bloqueo en dos fases: básico y estricto
 - Protocolos basados en **marcas de tiempo**
 - Protocolos basados en **grafos**

Protocolos basados en bloqueos

- Uso de **bloqueos** para controlar el acceso concurrente a los elementos de datos almacenados en la base de datos
- **Reglas** básicas del **bloqueo**:
 - **Bloqueo compartido**: si una transacción tiene un bloqueo compartido sobre un elemento de datos, **puede leer el elemento, pero no actualizarlo** (escribir)
 - Varias transacciones pueden mantener a la vez bloqueos compartidos sobre el mismo elemento
 - **Bloqueo exclusivo**: si una transacción tiene un bloqueo exclusivo sobre un elemento de datos, **puede leer y actualizar** (escribir) **el elemento**
 - Un bloqueo exclusivo proporciona acceso exclusivo al elemento

Protocolos basados en bloqueos

- El acceso a los elementos se hace en **exclusión mutua**:
 - Mientras una transacción accede a un dato, ninguna otra transacción puede modificar dicho elemento
- Se implementan mediante **candados** :
 - Variables asociadas a un elemento de información de la BD
- En general hay un candado por cada elemento de información de la BD
- Los candados se usan para sincronizar el acceso a los elementos de la BD por transacciones concurrentes

6.1 Protocolos basados en bloqueos (reservas)

- De las distintas técnicas de control de concurrencia, la que más se utiliza en los sistemas comerciales es la de las **reservas**.
- El resto de técnicas se han difundido menos

Idea básica:

- Una transacción tiene que obtener la **reserva** de un elemento antes de poder operar sobre él.
- Utilizan protocolos . Cada transacción debe **pedir** una reserva antes de operar sobre el elemento, y **liberarla** una vez que termine

Tipos de candados

- Candados binarios
- Candados compartidos y exclusivos.

Candados Binarios

- Estados

- **Bloqueado**: ninguna otra transacción puede tener acceso al elemento.
- **Desbloqueado**: se puede tener acceso al elemento cuando se solicite.

❑ Reglas.

- T debe **bloquear(X)** antes de cualquier operación de **lectura** o **escritura** sobre X.
- T debe **desbloquear(X)** después de terminar todas sus operaciones de lectura y escritura sobre X.
- T no emitirá **bloquear(X)** si ya tiene un bloqueo sobre ese elemento.
- T no emitirá **desbloquear(X)** si no tiene un bloqueo sobre ese elemento.

- ❑ **Problema**: demasiado excluyente. Cualquier transacción que encuentre un candado bloqueado se queda esperando a que éste pase a un estado desbloqueado

Candados Compartidos y Exclusivos

- Permiten que varias operaciones tengan acceso al mismo elemento si lo hacen para leer (candado compartido).
- Si se va a escribir se necesita un candado exclusivo.

❑ Reglas:

- T debe **bloquear_lectura (X)** antes de **leer** el elemento.
- T debe **bloquear_escritura(X)** antes de **escribir** sobre el elemento.
- T debe **desbloquear(X)** al final de todas las **lecturas** o **escrituras**.
- T no debe **bloquear_lectura(X)** si ya posee ese tipo de candado para ese elemento.
- T no debe **bloquear_escritura(X)** si ya posee ese tipo de candado para ese elemento.
- T no debe **desbloquear(x)** si no tiene un bloqueo sobre ese elemento

6. Técnicas de Control de Concurrency

Notación

- **Bloquear_lectura** (A) = BL (A): Bloqueo de A en modo **compartido**
- **Bloquear_escritura**(A) = BE (A): Bloqueo de A en modo **exclusivo**
- **Desbloquear** (A) = DB (A)

- Cuando una transacción **T solicita un bloqueo...**
 - **Si el elemento no ha sido ya bloqueado** por otra transacción, se le **concede** el bloqueo
 - **Si el elemento sí está bloqueado**, el SGBD determina si la solicitud **es compatible** con el bloqueo existente:
 - o Si se pide un **bloqueo compartido sobre** un elemento que ya tiene un **bloqueo compartido**, el bloqueo será **concedido** a T
 - o En **otro caso**, T **debe esperar** hasta que se libere el bloqueo existente
- Una transacción que obtiene un bloqueo lo mantiene **hasta que lo libera explícitamente o termina** (commit o rollback)
 - Sólo cuando se libera un bloqueo exclusivo los efectos de la escritura serán visibles para las demás transacciones

6. Técnicas de Control de Concurrency

- Algunos sistemas permiten la **mejora** (o promoción) y la **reducción** (o degradación) de bloqueos
 - Aumenta el nivel de concurrency del sistema
- Si T emitió bloquear_lectura(X), más tarde puede **mejorarlo a bloqueo exclusivo** emitiendo bloquear_escritura(X)
 - Si T es la **única** que tiene un bloqueo compartido sobre X, se le concede la solicitud
 - En otro caso, T debe esperar
- Si T emitió bloquear_escritura(X), más tarde puede **reducirlo a un bloqueo compartido** emitiendo bloquear_lectura(X)
 - Así permite que otras transacciones lean X

6. Técnicas de Control de Concurrency

Compatibilidad:

	BL	BE
BL	si	no
BE	no	no

Notación

- Bloquear_lectura(A) = BL (A): Bloqueo de A en modo **compartido**
- Bloquear_escritura(A) = BE (A): Bloqueo de A en modo **exclusivo**

- Hay que intentar evitar la “**inanición**”. Una transacción sufre inanición cuando es seleccionada para ser abortada sucesivamente, por lo que nunca termina su ejecución.

Por ejemplo:

- T2 bloquea en modo **compartido** el elemento X.
- T1 solicita un bloqueo **exclusivo** sobre X. No se le concede hasta que T2 libere X.
- Mientras tanto, T3 solicita un bloqueo **compartido** sobre X. Se le concede.
- Si T2 libera ahora X, T1 debe seguir esperando puesto que T3 lo ha bloqueado.
- Si esto sucede más veces, se dice que T1 padece de **inanición**

La solución es asignar **prioridades** más altas a las transacciones que padezcan inanición repetidamente para evitar que sean siempre las víctimas

6. Técnicas de Control de Concurrency

El uso de **bloqueos** para la programación de transacciones **no garantiza** la **serializabilidad** de las planificaciones

Transacción T1

```
bloquear_lectura(Y);
leer_elemento(Y);
desbloquear(Y);
bloquear_escritura(X);
leer_elemento(X);
X:=X+Y;
escribir_elemento(X);
desbloquear(X);
```

Transacción T2

```
bloquear_lectura(X);
leer_elemento(X);
desbloquear(X);
bloquear_escritura(Y);
leer_elemento(Y);
Y:=X+Y;
escribir_elemento(Y);
desbloquear(Y);
```

Valores iniciales: X=20, Y=30

Resultados de las **planificaciones serie**:

T1→T2: X=50, Y=80

T2→T1: X=70, Y=50

T1
bloquear_lectura(Y);
leer_elemento(Y);
desbloquear(Y);

bloquear_escritura(X);
leer_elemento(X);
X:=X+Y;
escribir_elemento(X);
desbloquear(X);

T2

bloquear_lectura(X);
leer_elemento(X);
desbloquear(X);
bloquear_escritura(Y);
leer_elemento(Y);
Y:=X+Y;
escribir_elemento(Y);
desbloquear(Y);

Planificación

6.2. Protocolo de bloqueo en dos fases (PB2F)

- Si las transacciones reservan justo antes de operar y liberan justo después, el protocolo de bloqueo por sí mismo no hace nada.
- Es necesario seguir un **protocolo adicional** que indique **dónde colocar las operaciones de bloqueo y desbloqueo** dentro de las transacciones
- El más conocido es el Protocolo de **Bloqueo en Dos Fases** (PB2F)
- Una transacción T sigue el protocolo de bloqueo en dos fases si **todas las operaciones de bloqueo preceden a la primera operación de desbloqueo**

garantiza la **serializabilidad**
- De este modo, podemos ver a T dividida en **dos fases**:
 - **Fase de expansión** (o crecimiento)
 - T puede **adquirir bloqueos**
 - T **no** puede **liberar** ningún bloqueo
 - **Fase de contracción** (o decrecimiento)
 - T puede **liberar bloqueos** existentes
 - T **no** puede **adquirir** ningún bloqueo

Resumen (PB2F):

- **Fase de crecimiento:** Una transacción puede obtener bloqueos pero no puede liberarlos.
- **Fase de decrecimiento:** Una transacción puede liberar bloqueos pero no puede obtener ninguno nuevo.
- **Punto de bloqueo:** punto en el cual una transacción obtiene el bloqueo final.
- El mismo **SGBD** genera las operaciones de petición y liberación de reservas, de manera transparente, sin que el programador tenga que preocuparse

6. Técnicas de Control de Concurrency

T1: R(X) W(X)

R(Y) abort

PB2F: Lectura no confirmada

T2: R(X) W(X)

T1	T2
Bloquear_lectura (X)	
R(X)	
Bloquear_escritura(X)	
W(X)	
	Bloquear_lectura(X)
Bloquear_lectura(Y)	esperar
R(Y)	esperar
abort	R(X)

6. Técnicas de Control de Concurrency

T1: R(X) W(X) commit

PB2F: Lectura no repetible

T2: R(X) R(X) commit

T1	T2
Bloquear_lectura (X)	
R(X)	
	Bloquear_lectura(X)
	R(X)
Bloquear_escritura(X)	
esperar	R(X)
esperar	Desbloquear(*)
esperar	commit
W(X).....	

6. Técnicas de Control de Concurrency

PB2F: Resumen Incorrecto

T2: $R(A)W(s) \dots R(X)W(s)R(Y)W(s) \dots R(Z);W(s) \text{ commit}$

T1: $R(X)W(X) \dots R(Y)W(Y) \text{ commit}$

6. Técnicas de Control de Concurrency

T1

T2

PB2F: Resumen Incorrecto

Bloquear_lectura(X)

R(X) $X := X - N$

Bloquear_escritura(X)

W(X)

Bloquear_lectura(Y)

R(Y) $Y := Y + N$

Bloquear_escritura(Y)

W(Y)

Desbloquear(X)

Desbloquear(Y)

Suma:=0

Bloquear_lectura(A)

R(A)

Suma:=suma+A

Bloquear_lectura(X)

esperar

esperar

esperar

esperar

esperar

R(X)

Suma:=suma+x

Bloquear_lectura(Y)

R(Y)

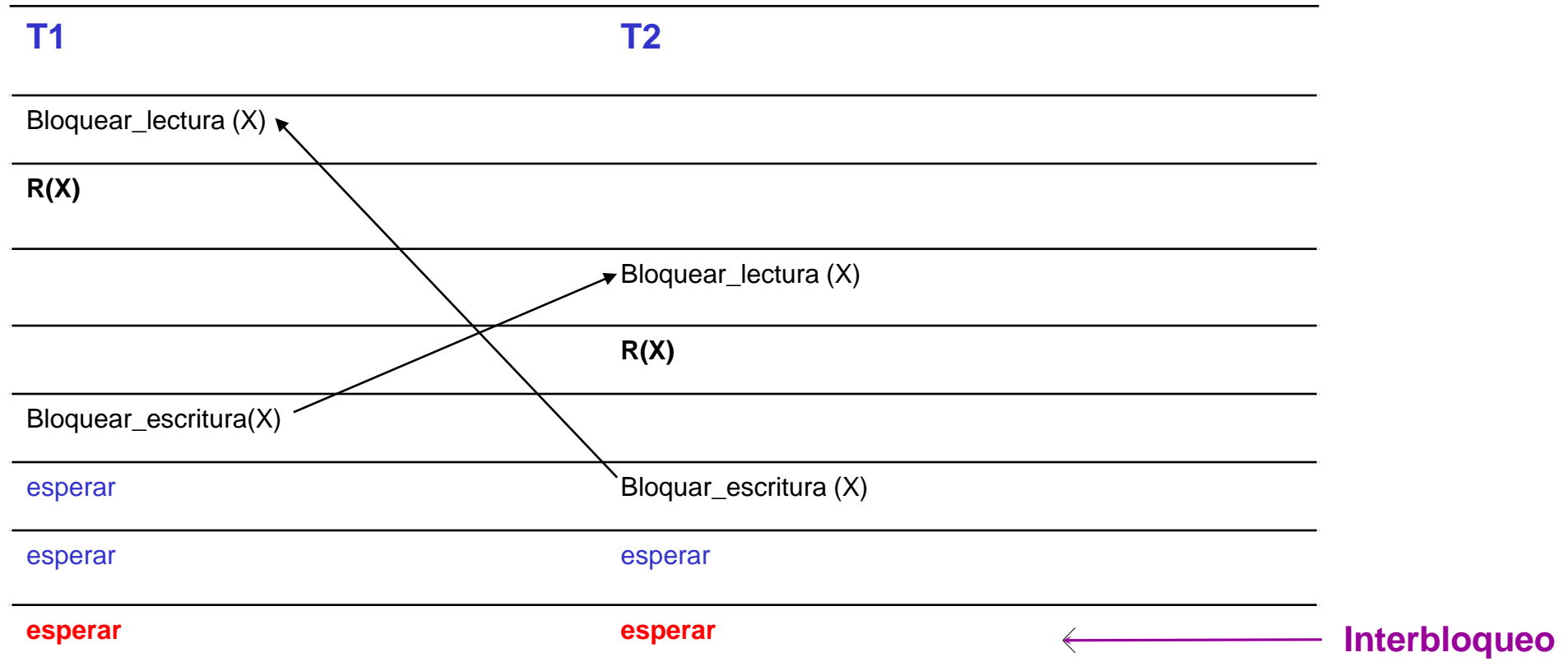
Suma:=suma+Y

Desbloquear(*)

6. Técnicas de Control de Concurrency

T1: R(X) W(X) R(Y) W(Y) commit
 T2: R(X) W(X) commit

PB2F: Actualización perdida



6. Técnicas de Control de Concurrency

Caso 1

T1:	W(A)	ABORT
T2:	W(A)	

Problema del PB2F

T1:	BE(A) W(A)	DB(A)	ABORT
T2:	BE(A)	esperar	W(A)

No se evita esta
actualización perdida

Caso 2

T1:	R(A)	
T2:	W(A)	ABORT

No se evita esta
lectura no confirmada

T1:	BL(A)	esperar	R(A)
T2:	BE(A) W(A)	DB(A)	ABORT

Notación

BE (A) → Bloquear_Escritura(A)
BL (A) → Bloquear_Lectura(A)
DB (A) → Desbloquear(A)

PB2F básico no evita todas las interferencias.

Sólo garantiza la **seriabilidad**, no la **recuperabilidad**

6. Técnicas de Control de Concurrency

PB2F estricto: PB2F básico y además, no libera ninguna reserva hasta que acaba (con COMMIT o ABORT).

→ Garantiza la **recuperabilidad** y la **seriabilidad** ←

PB2F Estricto

T1:	BE(A) W(A)	DB(A)	ABORT
T2:	BE(A)	esperar	W(A)

PB2F Básico

T1:	BE(A) W(A)	ABORT (DB(A))	
T2:	BE(A)	esperar	W(A)

PB2F Estricto

T1:	BL(A)	esperar	R(A)
T2:	BE(A)W(A)	DB(A)	ABORT

PB2F Básico

T1:	BL(A)	esperar	R(A)
T2:	BE(A)W(A)	ABORT (DB(A))	

PB2F Estricto

6. Técnicas de Control de Concurrency

Protocolo de bloqueo en dos fases (PB2F)

Si toda transacción de una planificación **sigue** el **protocolo** de **bloqueo en dos fases** entonces **la planificación** es **serializable**

Ventaja

- **No es necesario comprobar la serializabilidad** de las planificaciones

Inconvenientes

- El PB2F puede **limitar el grado de concurrencia** en una planificación
- El empleo de protocolos de bloqueo puede provocar **problemas** de:
 - **Interbloqueo** (bloqueo mortal o abrazo mortal)
 - **Bloqueo indefinido** (o espera indefinida)

6.3. Protocolos basados en grafos (PBG)

- Es necesario tener información adicional.
- Varios modelos: más simple: orden en el cual se accede a los elementos de la BD.
- Sea $D = \{d_1, d_2, \dots, d_n\}$ el orden de todos los elementos de datos. Si $d_i \rightarrow d_j$ entonces toda transacción que acceda tanto a d_i como a d_j debe acceder a d_i antes que a d_j .
- Este orden implica que el conjunto D se puede ver como un grafo acíclico con raíz (árbol)
- El protocolo en árbol sólo permite la instrucción de bloqueo bloquear_X

6.3. Protocolos basados en grafos (PBG)

Reglas:

- El primer bloqueo de T_i puede ser sobre cualquier elemento de datos.
- Posteriormente, T_i puede bloquear un elemento de datos Q sólo si T_i está bloqueando actualmente al padre de Q .
- Los elementos de datos bloqueados se pueden desbloquear en cualquier momento.
- T_i no puede bloquear de nuevo un elemento de datos que ya haya bloqueado y desbloqueado anteriormente.

6.3. Protocolos basados en grafos (PBG)

- **Ventajas:**

- Los desbloques se pueden dar antes → menores tiempos de espera y aumento de la concurrencia.
- Se evitan interbloqueos.

- **Inconvenientes:**

- A veces una transacción tendrá que bloquear elementos a los que no va a acceder → aumento del coste de los bloqueos, tiempos de espera adicional y descenso de la concurrencia.
- Bloqueo de la raíz del árbol → reducir la concurrencia.
- Hay planificaciones que no se pueden obtener por este protocolo pero si por el dos fases y viceversa.

6.4. Protocolos basados en marcas temporales (PBMT)

- A toda transacción se le asocia una única marca temporal $MT(T_i)$
 - Usar reloj del sistema
 - Contador lógico
- Si $MT(t_i) < MT(t_j)$ t_i es más antigua que T_j .
- A cada elemento de datos Q se le asocian dos valores:
 - $MT_escritura(Q)$. Mayor marca temporal de todas las transacciones que ejecutan con éxito $W(Q)$.
 - $MT_lectura(Q)$. Mayor marca temporal de todas las transacciones que ejecutan con éxito $R(Q)$.

6.4. Protocolos basados en marcas temporales (PBMT)

- Estas marcas se actualizan con cada R y W
- Antes de R o W el algoritmo compara las MT de T con las MT de lectura o escritura para Q.
- Si una transacción viola el orden de ejecución marcado por el algoritmo el sistema aborta la transacción.

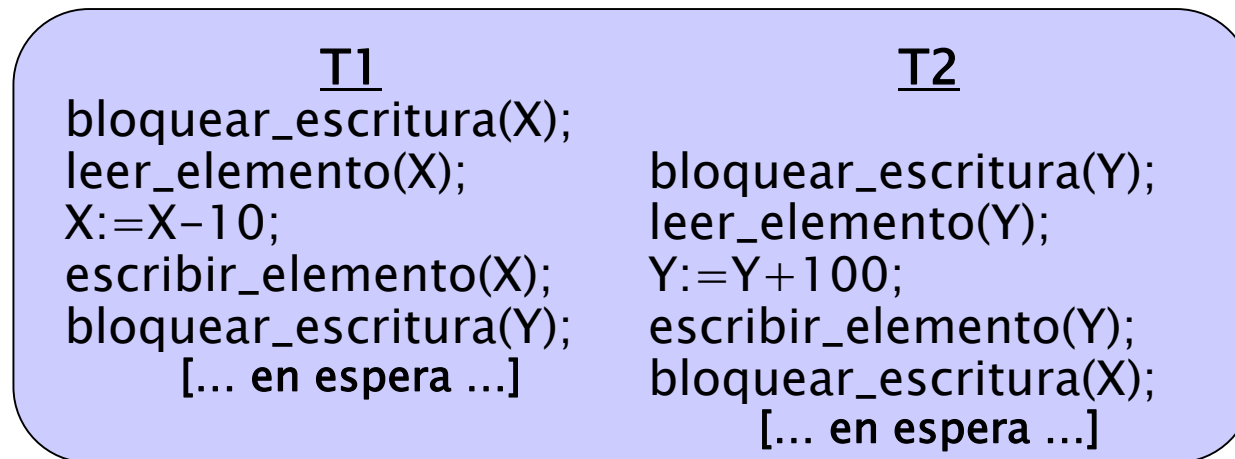
6.4. Protocolos basados en marcas temporales (PBMT)

Reglas:

- Si T emite $W(X)$
 - Si $MT_lectura(X) > MT(T)$ o $MT_escritura(X) > MT(T)$ ➡ rechazar la operación y abortar T.
 - En caso contrario ejecutar $W(X)$ y asignar a $MT_escritura(X)$ el valor de $MT(T)$.
- Si T emite $R(X)$
 - Si $MT_Escritura(X) > MT(T)$ ➡ abortar T y rechazar la operación.
 - En caso contrario se ejecuta la operación $R(X)$ y se le asigna a $MT_lectura(X)$ el mayor de los dos valores : $MT(T)$ o $MT_lectura(X)$ actual.

7. El problema del Interbloqueo

- Situación en la que cada una de dos (o más) transacciones está esperando a que se libere un bloqueo establecido por la otra transacción



- El SGBD ha de reconocer un interbloqueo y romperlo:
 - * **Abortar** una o más transacciones
 - Se deshacen sus escrituras y se liberan sus bloqueos
 - Así, el resto de transacciones podrá continuar su ejecución
 - * **Reiniciar** automáticamente las transacciones abortadas

7. El problema del Interbloqueo

- Las principales técnicas para gestionar los interbloqueos son:
 - **Temporizaciones** de bloqueos
 - **Detección** de interbloqueos
 - **Prevención** de interbloqueos
 - Conviene **detectar** interbloqueos cuando se sabe que hay **poca interferencia** entre transacciones, es decir si:
 - Las transacciones son **cortas** y **bloquean pocos elementos**, o
 - La **carga** de transacciones es **pequeña**
 - En otro caso, conviene usar **temporizaciones** o técnicas de **prevención**
- Es más difícil **prevenir** que utilizar **temporizaciones** o que **detectarlos**, por lo que en la práctica los sistemas no suelen emplear las técnicas de prevención

7. El problema del Interbloqueo

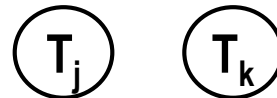
Temporizaciones de bloqueos

- Una transacción que solicita un bloqueo **sólo esperará durante** un período de **tiempo predefinido** por el sistema
- **Si no se concede el bloqueo** durante ese tiempo, se producirá un ‘fin de temporización’: el SGBD **asumirá** que la transacción **está interbloqueada** (aunque puede que no), la **abortará** y la **reiniciará** automáticamente
 - * Es una solución muy sencilla y práctica
 - * Pero puede hacer que sean abortadas y reiniciadas transacciones que en realidad no están en un interbloqueo

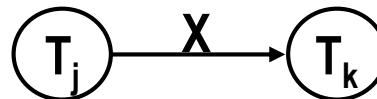
7. El problema del Interbloqueo

Detección de Interbloqueos

- **Verificación periódica del estado del sistema**
¿está en un bloqueo mortal?
- Creación de un **grafo de espera** que muestra las dependencias entre transacciones
 - Crear un **nodo** por cada transacción en ejecución, etiquetado con el identificador de la transacción, T



- Si T_j espera para bloquear el elemento X , ya bloqueado por T_k , crear una **arista dirigida** desde T_j a T_k



- Cuando T_k libera el candado sobre X , **borrar** la arista correspondiente
- Si existe un **ciclo** en el grafo de espera, entonces se ha detectado un **interbloqueo** entre las transacciones

Detección de Interbloqueos

- Pero... ¿**cuándo** hay que **verificar** el estado del sistema (ejecutar el algoritmo que genera el grafo de espera)?
 1. A **intervalos uniformes** de tiempo, o
 2. A **intervalos** de tiempo **desiguales** :
 - Iniciar algoritmo de detección con un tamaño de intervalo inicial
 - Cada vez que no se detecta interbloqueo, incrementar el intervalo
 - Por ejemplo, al doble del anterior
 - Cada vez que se detecta interbloqueo, reducir el intervalo
 - Por ejemplo a la mitad
 - Existirán límites superior e inferior del tamaño del intervalo

Detección de Interbloqueos

- Si el sistema está en un estado de **interbloqueo**, el **SGBD necesita abortar** algunas **transacciones...**
 - ¿**Cuáles**? ⇒ **Selección de víctimas**
 - Es mejor abortar transacciones que lleven **poco tiempo en ejecución**
 - Es mejor abortar una transacción que haya hecho **pocos cambios** en la base de datos
 - Es mejor abortar una transacción que **todavía debe hacer muchos cambios** en la base de datos
 - Puede que el SGBD no conozca esta información
- Se trata de abortar las transacciones que supongan el mínimo coste
- Es necesario evitar la **inanición**

7. El problema del Interbloqueo

Detección de Interbloqueos

- Una transacción sufre **inanición** cuando es seleccionada para ser abortada (**víctima**) sucesivamente: **nunca termina su ejecución**
 - Es similar al bloqueo indefinido
- La solución es asignar **prioridades** más altas a las transacciones abortadas varias veces, para no ser siempre las víctimas

7. El problema del Interbloqueo

Recuperación del interbloqueo

- **Seleccionar una víctima**
 - Coste mínimo.
 - Cálculo realizado y lo que queda.
 - N° de elementos que haya utilizado.
 - Cantidad de elementos que usará hasta que termine.
 - N° de transacciones que se verán involucradas en el retroceso.
- **Retroceso**
 - Retroceder sólo hasta el momento del interbloqueo. (el sistema mantiene información sobre el estado de la transacción).
- **Inanición**
 - Es posible que siempre se elija a la misma víctima
 - Asegurar que una transacción solo puede elegirse como víctima un número determinado de veces.

8. El problema del Bloqueo Indefinido

El protocolo de control de concurrencia **nunca selecciona a una transacción que está esperando para establecer un bloqueo**, mientras otras transacciones continúan ejecutándose con normalidad.

Ocurre si el **esquema de espera** da más prioridad a unas transacciones que a otras ➡ **esquema de espera injusto**

- Existen 2 algoritmos de prevención de bloqueo indefinido:
 - Ambos consiguen un **esquema de espera justo**

El primero que llega, es el primero en ser atendido

Las transacciones pueden bloquear el elemento X en el orden en que solicitaron su bloqueo

Aumento de prioridad en la espera

Cuanto más espera T, mayor es su prioridad

Cuando T tiene la prioridad más alta de todas, obtiene el bloqueo y continúa su ejecución