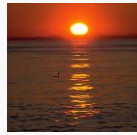
Capítulo 16 : Control de concurrencia

Fundamentos de Bases de datos, 5ª Edición.

Capítulo 16: Control de la concurrencia

- Protocolos basados en el bloqueo
- Protocolos basados en marcas temporales
- Protocolos basados en validación
- Granularidad múltiple
- Esquemas multiversión
- Tratamiento de interbloqueos
- Operaciones para insertar y borrar
- Concurrencia en los índices**



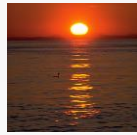
Protocolos basados en bloqueos

- Un bloqueo es un mecanismo para controlar el acceso concurrente a un elemento de datos
- Los elementos de datos se puden bloquear de dos maneras:

  1. *modo exclusivo (X)*. El elemento de datos además de leerse se puede escribir. Un bloqueo de este tipo se solicita con la instrucción **bloquear-X**.

  2. *modo compartido (S)*. Los elementos de datos sólo se pueden leer. Un bloqueo de este tipo se solicita con la instrucción **bloquear-S**.

- Las solicitudes de bloqueo se dirigen al gestor de control de concurrencia. La transacción puede realizar la operación sólo después de que se conceda la solicitud.



Protocolos basados en bloqueos (cont.)

- Matriz de compatibilidad de bloqueos

	C	X
C	cierto	falso
X	falso	falso

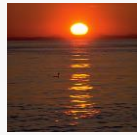
- A una transacción se le puede garantizar un bloqueo en un elemento si el bloqueo solicitado es compatible con los bloqueos que ya tengan otras transacciones sobre ese mismo elemento
- Cualquier número de transacciones puede tener bloqueos compartidos sobre un elemento,
  - pero si una de ellas tiene uno exclusivo sobre un determinado elemento, ninguna otra puede tener ningún otro bloqueo sobre dicho elemento.
- Si no se puede garantizar un bloqueo, la transacción que lo solicita tiene que esperar hasta que los bloqueos incompatibles que tienen otras transacciones se hayan liberado. A continuación se autoriza el bloqueo.

Protocolos basados en bloqueos (cont.)

- Ejemplo de una transacción que realiza un bloqueo:

  T_2: **bloquear-S**(*A*)
  leer *(A)*;
  desbloquear(*A*)
  bloquear-S(*B*)
  leer *(B)*;
  desbloquear(*B*)
  visualizar*(A+B)*

- El bloqueo anterior no es suficiente para garantizar la secuencialidad — si se actualizasen *A* y *B* entre la lectura de *A* y *B*, la suma mostrada sería errónea.
- Un **protocolo de bloqueo** es un conjunto de reglas que siguen todas las transacciones cuando se solicitan o se liberan bloqueos. Los protocolos de bloqueo restringen el número de planificaciones posibles.



Fallos de los protocolos basados en bloqueos

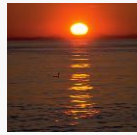
- Considerar la planificación parcial

T_3	T_4
bloquear-X(B)	
leer(B)	
$B := B - 50$	
escribir(B)	
	bloquear-C(A)
	leer(A)
	bloquear-C(B)
bloquear-X(A)	

- Ni T_3 ni T_4 pueden progresar — la ejecución de **bloquear-S**(*B*) ocasiona que T_4 espere a que T_3 libere su bloqueo sobre *B*, mientras que la ejecución de **bloquear-X**(*A*) ocasiona que T_3 espera a que T_4 libere su bloqueo sobre *A*.
- Esta situación se denomina **interbloqueo**.
  - Para manejar un interbloqueo uno de los dos, T_3 o T_4, debe retroceder y liberar sus bloqueos.

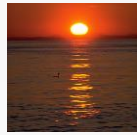
Fallos de los protocolos basados en bloqueos (cont.)

- En la mayoría de los protocolos de bloqueo se pude producir un interbloqueo potencial. Los interbloqueos son un diablo necesario.
- También es posible que se produzca la **inanición** si el gestor de control de concurrencia se ha diseñado defectuosamente. Por ejemplo:
  - Puede que una transacción esté esperando un bloqueo exclusivo sobre un elemento determinado mientra que una secuencia de transacciones diferentes solicitan y obtienen la autorización de bloqueo compartido sobre el mismo elemento.
  - La misma transacción retrocede repetidamente debido a los interbloqueos.
- El gestor de control de concurrencia se puede diseñar para impedir que se produzca la inanición.



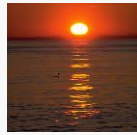
El protocolo de bloqueo de dos fases

- Éste es un protocolo que asegura planificaciones secuenciables en cuanto a conflictos.
- Fase 1: Fase de crecimiento
  - las transacciones pueden conseguir bloqueos
  - las transacciones no pueden liberar bloqueos
- Fase 2: Fase de decrecimiento
  - las transacciones pueden liberar bloqueos
  - las transacciones no pueden conseguir bloqueos
- El protocolo asegura la secuencialidad. Se puede probar que las transacciones se pueden secuenciar en el orden de sus puntos de bloqueo (es decir, el punto de la planificación en el cual la transacción obtiene su bloqueo final).



El protocolo de bloqueo de dos fases (cont.)

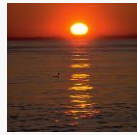
- El protocolo de bloqueo de dos fases *no* asegura la ausencia de interbloqueos
- El retroceso en cascada puede ocurrir en el bloqueo de dos fases. Se pueden evitar por medio de una modificación del protocolo denominado **protocolo de bloqueo estricto de dos fases**. En él una transacción debe mantener todos sus bloqueos exclusivos hasta que se complete o aborte.
- El **protocolo de bloqueo riguroso de dos fases** es incluso más estricto: en él *todos* los bloqueos se mantienen hasta que se complete o aborte. En este protocolo las transacciones se pueden secuenciar en el orden en que se comprometen.



El protocolo de bloqueo de dos fases (cont.)

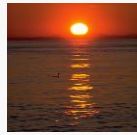
- Pueden existir planificaciones secuenciable en cuanto a conflictos que no se pueden obtener si se utiliza el bloqueo en dos fases.
- Sin embargo, si no hay información adicional (p. e., el orden de acceso a los datos), el bloqueo en dos fases es necesario para la secuencialidad en cuanto a conflictos en el sentido siguiente:

  Dada una transacción T_i que no sigue el bloqueo en dos fases, se puede encontrar una transacción T_j que si lo utilice, y una planificación para T_i y T_j que no sea secuenciable en cuanto a conflictos.



Conversiones de bloqueo

- Bloqueo de dos fases con conversiones de bloqueo:
  - Primera fase:
    - puede obtener bloquear-S sobre un elemento
    - puede obtener bloquear-X sobre un elemento
    - puede convertir bloquear-S en bloquear-X (subir)
  - Segunda fase:
    - puede liberar bloqueo-S
    - puede liberar bloqueo-X
    - puede convertir bloquear-X en bloquear-S (bajar)
- Este protocolo asegura la secuencialidad. Pero aún recae sobre el programador el insertar las diversas instrucciones de bloqueo.



Adquisición automática de bloqueos

- Una transacción T_i genera la instrucción estándar de lectura/escritura, sin llamadas explícitas a bloqueos.
- La operación **leer**(D) se procesa como sigue:

  if T_i tiene un bloqueo en D

  then

  leer(D)

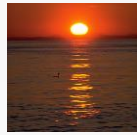
  else begin

  si es necesario, esperar hasta que ninguna otra transacción tenga un **bloquear-X** sobre D

  autorizar a T_i un **bloquear-S** sobre D;

  leer(D)

  end

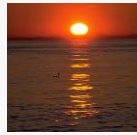


Adquisición automática de bloqueos (cont.)

- **escribir**(*D*) se procesa como sigue:

```
if Ti tiene un bloquear-X en D
        then
          escribir(D)
        else begin
            si es necesario, esperar hasta que ninguna otra transacción
                    tenga un bloqueo sobre D,
            if Ti tiene un bloquear-S en D
                then
                    subir el bloqueo sobre D a bloquear-X
                else
                    autorizar a Ti un bloquear-X sobre D
                escribir(D)
        end;
```

- Todos los bloqueos se liberan después de que se complete o aborte



Implementación de bloqueos

- Un **gestor de bloqueos** se puede implementar como un proceso independiente, al que las transacciones envían los bloqueos y desbloqueos requeridos
- El gestor de bloqueos responde a un bloqueo solicitado, enviando un mensaje de concesión de bloqueo (o un mensaje preguntando por la transacción a retroceder, en el caso de un interbloqueo)
- La transacción solicitada espera hasta que se atiende su solicitud
- El gestor de bloqueos mantiene una estructura de datos, denominada **tabla de bloqueo**, para registrar los bloqueos concedidos y las solicitudes pendientes
- La tabla de bloqueos generalmente se implementa como una tabla asociativa en memoria, indexada sobre el nombre del elemento de datos que se está bloqueando

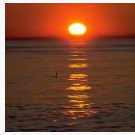
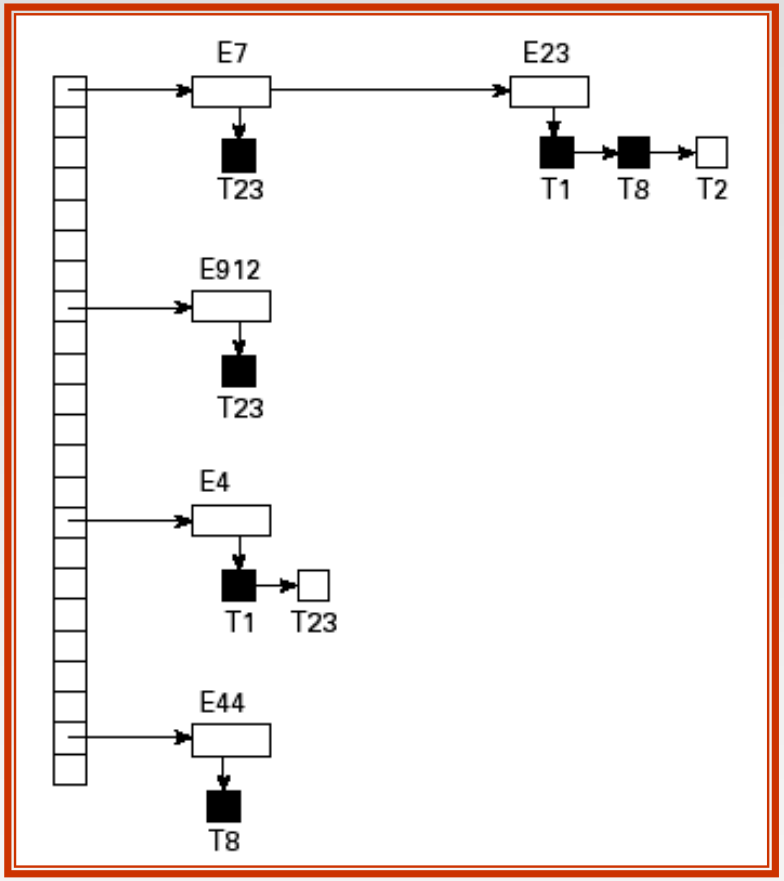


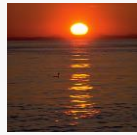
Tabla de bloqueo



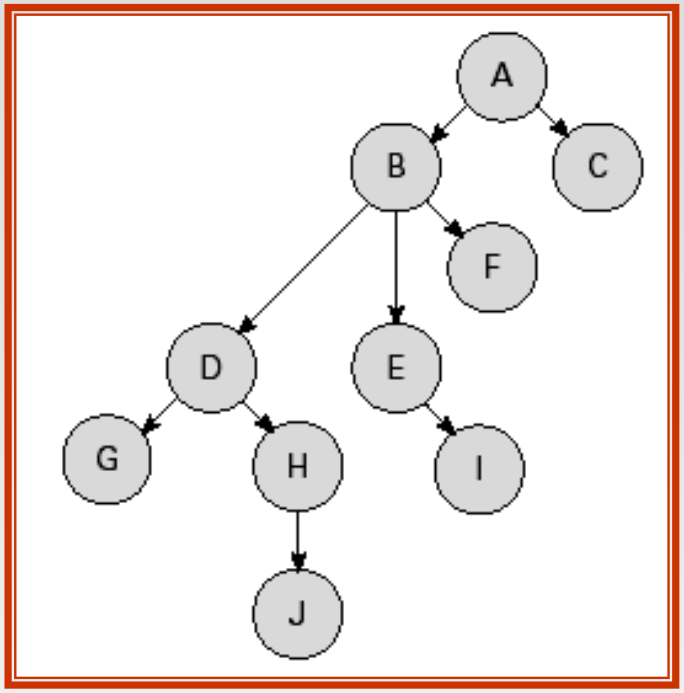
- Los rectángulos negros indican bloqueos concedidos, los blancos indican las solicitudes en espera
- La tabla de bloqueos también registra el tipo de bloqueo concedido o solicitado
- Una nueva solicitud se añade al final de la cola de solicitudes para el elemento de datos, y se concede si es compatible con todos los bloqueos anteriores
- Los desbloqueos solicitados dan lugar a que la solicitud sea borrada y después se comprueban las solicitudes para ver si ahora se pueden conceder
- Si la transacción aborta se borran todas las solicitudes de la transacción, en espera o concedidas
  - el gestor de bloqueos, para implementar esto eficientemente, puede mantener una lista de los bloqueos almacenados por cada transacción

Protocolos basados en grafos

- Los protocolos basados en grafos constituyen una alternativa a los bloqueos de dos fases
- Se impone un orden parcial $\rightarrow$ sobre el conjunto **D** = {d_1, d_2, ... , d_n} 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 de acceder a d_j.
  - Implica que el conjunto **D** se pueda ver como un grafo dirigido acíclico denominado *grafo de la base de datos*.
- El *protocolo de árbol* es un tipo simple de protocolo de grafo.



Protocolo de árbol



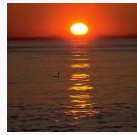
- Sólo se permiten los bloqueos exclusivos.
- El primer bloqueo de T_i puede ser sobre cualquier elemento de datos. Posteriormente, T_i puede bloquear un elemento de datos Q sólo si T_i está bloqueando actualmente al padre de Q.
- Los elementos de datos se pueden desbloquear en cualquier momento.

Protocolos basados en grafos (cont.)

- El protocolo de árbol no sólo asegura la secuencialidad en cuanto a conflictos, sino que también asegura la ausencia de interbloqueos.
- Los desbloqueos se pueden producir antes en el protocolo de bloqueo de árbol que en el protocolo de bloqueo de dos fases.
  - tiempos de espera menores y a un aumento de la concurrencia
  - el protocolo está libre de interbloqueos, así que no se necesitan retrocesos
- Desventajas
  - El protocolo no garantiza la recuperabilidad y la ausencia de cascadas
    - Se necesita introducir dependencias de compromiso para asegurar la recuperabilidad
  - Una transacción puede que tenga que bloquear elementos de datos a los que no accede.
    - aumento del coste de los bloqueos, y la posibilidad de tiempos de espera adicionales
    - descenso potencial de la concurrencia
- Existen planificaciones que no son posibles por medio del protocolo de bloqueo de dos fases que son posibles por medio del protocolo de árbol y viceversa.

Protocolos basados en marcas temporales

- Cada transacción genera una marca temporal cuando se introduce en el sistema. Si a la transacción T_i se le ha asignado la marca temporal MT(T_i) y una nueva transacción T_j entra en el sistema, entonces MT(T_i) < MT(T_j).
- El protocolo maneja la ejecución concurrente de modo que las marcas temporales de las transacciones determinan el orden de secuencia.
- Para asegurar dicho comportamiento, el protocolo mantiene por cada elemento de datos Q dos valores de marca temporal:
  - **marca_temporal-E**(Q) denota la mayor marca temporal de todas las transacciones que ejecutan con éxito **escribir**(Q).
  - **marca_temporal-L**(Q) denota la mayor marca temporal de todas las transacciones que ejecutan con éxito **leer**(Q).

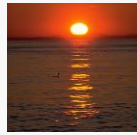


Protocolos basados en marcas temporales (cont.)

- El protocolo de ordenación por marcas temporales asegura que todas las operaciones **leer** y **escribir** conflictivas se ejecutan en el orden de las marcas temporales.
- Supóngase que la transacción T_i ejecuta **leer**(Q)
  1. Si MT(T_i) $\leq$ marca_temporal-**E**(Q) entonces T_i necesita leer un valor de Q que ya se ha sobrescrito.
     - Por tanto se rechaza la operación **leer** y T_i se retrocede.
  2. Si MT(T_i) $\geq$ marca_temporal-**E**(Q) entonces la operación **leer** se ejecuta, y marca_temporal-**L**(Q) se fija al máximo de marca_temporal-**L**(Q) y MT(T_i).

Protocolos basados en marcas temporales (cont.)

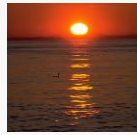
- Supóngase que la transacción T_i ejecuta **escribir**(Q).
  1. Si MT(T_i) < marca_temporal-**L**(Q) entonces el valor de Q que produce T_i se necesita previamente y el sistema asume que dicho valor no se puede producir nunca.
     - Por tanto, se rechaza la operación **escribir** y T_i se retrocede.
  2. Si MT(T_i) < marca_temporal-E(Q) entonces T_i está intentando escribir un valor de Q obsoleto.
     - Por tanto, se rechaza la operación **escribir** y T_i se retrocede.
  3. En otro caso se ejecuta la operación **escribir** y MT(T_i) se asigna a marca_temporal-E(Q).



Ejemplo de uso del protocolo

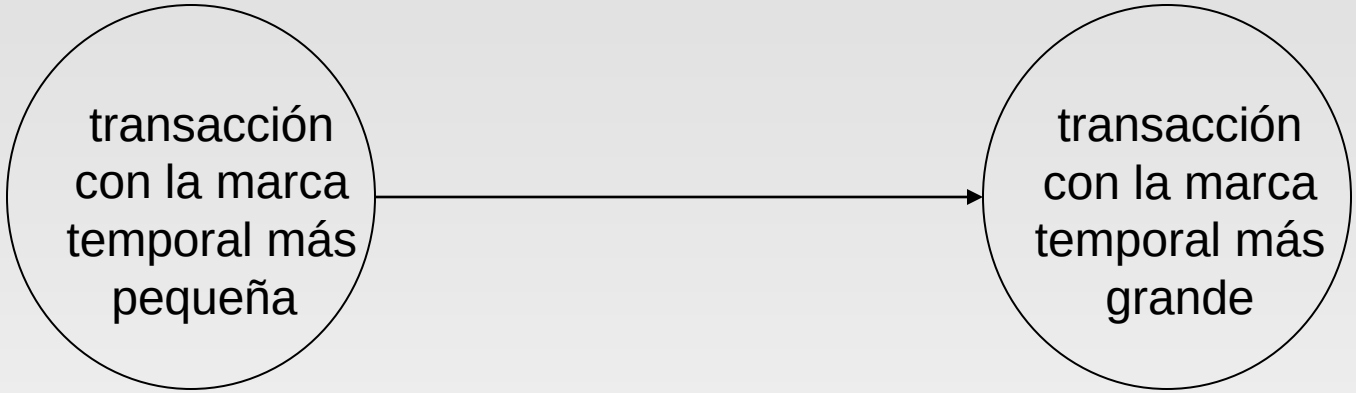
Una planificación parcial de varios elementos de datos para transacciones con marcas temporales 1, 2, 3, 4, 5

T_1	T_2	T_3	T_4	T_5
				leer(X)
leer(Y)	leer(Y)			
		escribir(Y)		
		escribir(Z)		
				leer(Z)
	leer(X)			
	abortar			
leer(X)				
		escribir(Z)		
		abortar		
				escribir(Y)
				escribir(Z)



Corrección del protocolo de ordenación por marcas temporales

- El protocolo de ordenación por marcas temporales garantiza la secuencialidad ya que todos los arcos del grafo de precedencia son del tipo:

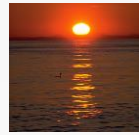


Por lo tanto, no se producirán ciclos en el grafo de precedencia

- El protocolo asegura la ausencia de interbloqueos, ya que ninguna transacción tiene que esperar.
- Pero puede que la planificación no esté libre de cascadas e, incluso, es posible que no sea recuperable.

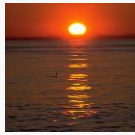
Recuperabilidad y ausencia de cascadas

- Problema del protocolo de ordenación por marcas temporales:
  - Supóngase que T_i aborta, pero T_j ha leído un elemento de datos escrito por T_i
  - A continuación T_j debe abortar; si a T_j se le ha permitido completar antes, la planificación no es recuperable.
  - Además, cualquier transacción que haya leído un elemento de datos escrito por T_j debe abortar
  - Esto puede llevar a un retroceso en cascada --- es decir, a una cadena de retrocesos
- Solución 1:
  - Una transacción se estructura de modo que todas sus escrituras se llevan a cabo al final de su procesamiento
  - Todas las escrituras de una transacción forman una acción atómica, es decir, no se puede ejecutar ninguna transacción mientras que se está escribiendo una transacción
  - Una transacción que aborta se reinicia con una marca temporal nueva
- Solución 2: Forma limitada de bloqueo: las lecturas de elementos no comprometidos se posponen hasta que se actualice el elemento comprometido
- Solución 3: Usar dependencias de compromiso para asegurar la recuperabilidad



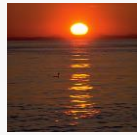
Regla de escritura de Thomas

- Versión modificada del protocolo de ordenación por marcas temporales en el que las operaciones **escribir** obsoletas se puden ignorar bajo determinadas circunstancias.
- Cuando T_i intenta escribir el elemento de datos Q, si MT(T_i) < marca_temporal-E(Q), entonces T_i está intentando escribir un valor obsoleto de {Q}. Por lo tanto, en vez de retroceder T_i como habría hecho el protocolo de ordenación por marcas temporales, esta operación {**escribir**} se puede ignorar.
- En cualquier otro caso este protocolo se comporta igual que el protocolo de ordenación por marcas temporales.
- La regla de escritura de Thomas permite una mayor concurrencia potencial.
  - Permite algunas planificaciones secuenciables en cuanto a vistas que no son secuenciables en cuanto a conflictos.



Protocolos basados en validación

- La ejecución de la transacción T_i se hace en tres fases.
  1. **Fase de lectura**: La transacción T_i sólo escribe en variables locales temporales
  2. **Fase de validación**: La transacción T_i lleva a cabo una ``prueba de validación'' para determinar si se puede escribir en las variables locales sin violar la secuencialidad.
  3. **Fase de escritura**: Si se valida T_i, las actualizaciones se aplican a las bases de datos; en otro caso, T_i se retrocede.
- Las tres fases de las transacciones que se ejecutan concurrentemente se pueden intercalar, pero cada transacción debe seguir las tres fases en ese orden.
  - Suponga por simplicidad que la fase de validación y la de escritura se producen a la ves, atómica y serialmente.
    - Es decir, sólo se ejecuta una transacción de validación/escritura a la vez.
- También se denomina **control de concurrencia optimista** ya que la transacción se ejecuta por completo con la esperanza de que todo irá bien durante la validación



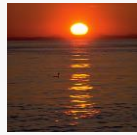
Protocolos basados en validación (cont.)

- Cada transacción T_i tiene 3 marcas temporales
  - **Inicio**(T_i) : momento en el cual T_i comienza su ejecución
  - **Validación**(T_i): momento en el cual T_i comienza su fase de validación
  - **Fin**(T_i) : momento en el cual T_i termina su fase de escritura
- Se determina el orden de secuencialidad a través de las marcas temporales dadas en el momento de la validación, para aumentar la concurrencia.
  - Por lo tanto a $MT(T_i)$ se le da el valor de **Validación**(T_i).
- Este protocolo es útil y proporciona un alto grado de concurrencia si la probabilidad de conflictos es baja.
  - se debe a que el orden de secuencialidad no está predecidido y,
  - relativamente, un número inferior de transacciones tendrá que retroceder.

Prueba de validación para transacciones T_j

- Si para todos los T_i con MT (T_i) < MT (T_j) se mantiene una de las dos condiciones siguientes:
  - **fin**(T_i) < **inicio**(T_j)
  - **inicio**(T_j) < **fin**(T_i) < **validación**(T_j) **y** el conjunto de elementos de datos escritos por T_i no tiene intersección con el conjunto de elementos de datos leído por T_j.

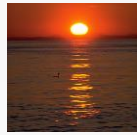
  entonces la validación tiene éxito y T_j se puede completar. En cualquier otro caso, la validación falla y T_j se aborta.
- *Justificación*: Se satisface la primera condición y no se solapa la ejecución, o se satisface la segunda condición y
  - las escrituras de T_j no afectan a las lecturas de T_i ya que se producen después de que T_i haya terminado sus lecturas.
  - las escrituras de T_i no afectan a las lecturas de T_j ya que T_j no lee ningún elemento escrito por T_i.



Planificación producida por validación

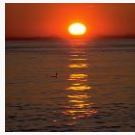
- Ejemplo de planificación producida utilizando validación

T_{14}	T_{15}
leer(*B*)	
	leer(*B*) *B:- B-50* **leer**(*A*) *A:- A+50*
leer(*A*) (*validar*) **visualizar** (*A+B*)	
	(*validar*) **escribir** (*B*) **escribir**(*A*)

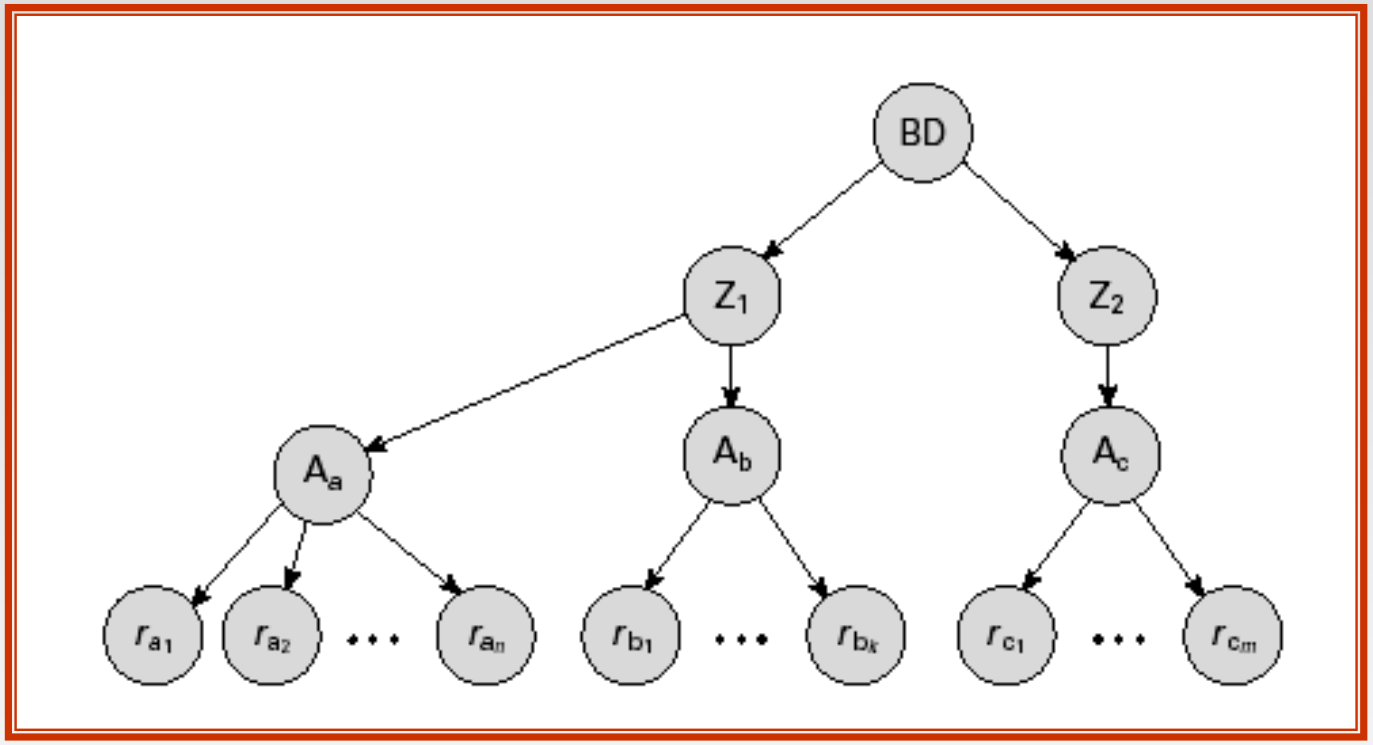


Granularidad múltiple

- Permite que los elementos de datos sean de diferentes tamaños y definir una jerarquía de granularidad de datos, donde las granularidades pequeñas están anidadas en otras más grandes
- Se puede representar gráficamente como un árbol (aunque no debe confundirse con el protocolo de bloqueo de árbol)
- Cuando una transacción bloquea un nodo del árbol *explícitamente*, *implícitamente* bloquea todos los descendientes del nodo de la misma manera.
- Granularidad de bloqueo (nivel del árbol en el que se realiza el bloqueo):
  - **granularidad fina** (niveles inferiores del árbol): alta concurrencia, alta sobrecarga de bloqueo
  - **granularidad gruesa** (niveles superiores del árbol): baja sobrecarga de bloqueo, baja concurrencia



Ejemplo de jerarquía de granularidad

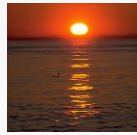


El nivel comenzando por el de mayor granularidad son:

- *base de datos*
- *área*
- *archivo*
- *registro*

Modos de bloqueo intencional

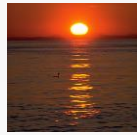
- Además de los modos de bloqueo C y X, existen tres modos de bloqueo adicionales con granularidad múltiple:
  - ***intencional-compartido*** (IC): indica un bloqueo explícito en un nivel inferior del árbol, pero sólo con bloqueos en modo compartido.
  - ***intencional-exclusivo*** (IX): indica un bloqueo explícito en un nivel inferior del árbol, pero sólo con bloqueos en modo exclusivo o compartido
  - ***intencional-exclusivo y compartido*** (IXC): el subárbol cuya raíz es ese nodo se bloquea explícitamente en modo compartido, y dicho bloqueo explícito se produce en un nivel inferior con bloqueos en modo exclusivo.
- los bloqueos intencionales permiten que se bloquee un nodo de nivel superior en modo C o X sin tener que comprobar todos los nodos descendientes.



Matriz de compatibilidad con los modos de bloqueo intencional

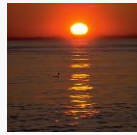
- La matriz de compatibilidad de todos los modos de bloqueo es:

	IC	IX	C	C IX	X
IC	✓	✓	✓	✓	×
IX	✓	✓	×	×	×
C	✓	×	✓	×	×
C IX	✓	×	×	×	×
X	×	×	×	×	×



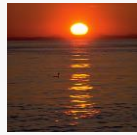
Esquema de bloqueos de granularidad múltiple

- La transacción T_i puede bloquear un nodo Q, utilizando las reglas siguientes:
  1. Debe observar la matriz de compatibilidad de bloqueos.
  2. Debe bloquear la raíz del árbol en primer lugar y puede bloquearla en cualquier modo.
  3. Puede bloquear un nodo Q por T_i en modo C o IC sólo si está bloqueando actualmente al padre de Q por T_i en modo IX o IC.
  4. Puede bloquear un nodo Q por T_i en modo X, IXC o IX sólo si está bloqueando actualmente al padre de Q por T_i en modo IX o IXC.
  5. T_i puede bloquear un nodo sólo si no ha desbloqueado previamente ningún nodo (es decir, T_i es de dos fases).
  6. T_i puede desbloquear un nodo Q sólo si T_i no ha bloqueado a ninguno de los hijos de Q.
- Obsérvese que en el protocolo de granularidad múltiple es necesario que se adquieran los bloqueos en orden descendente (de la raíz a las hojas), y que se liberen en orden ascendente (de las hojas a la raíz).



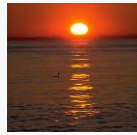
Esquemas multiversión

- Los esquemas multiversión mantienen las versiones anteriores de los elementos de datos para aumentar la concurrencia.
  - Ordenación por marcas temporales multiversión
  - Bloqueo de dos fases multiversión
- Cada **escribir** con éxito tiene como resultado la creación de una nueva versión de los elementos de datos escritos.
- Utilícense marcas temporales para etiquetar las extensiones.
- Cuando se genera una operación **leer**(Q), selecciónese la versión apropiada de Q basándose en la marca temporal de la transacción, y devuélvase el valor de la versión elegida.
- Las operaciones **leer** no tienen que esperar nunca ya que se devuelve la versión apropiada inmediatamente.



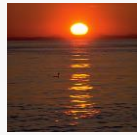
Ordenación por marcas temporales multiversión

- Cada elemento de datos Q tiene una secuencia de versiones $<Q_1, Q_2, ..., Q_m>$. Cada versión Q_k contiene tres campos de datos:
  - **contenido** -- es el valor de la versión Q_k.
  - **marca_temporal-E**(Q_k) -- es la marca temporal de la transacción que haya creado (escrito) la versión Q_k
  - **marca_temporal-L**(Q_k) es la mayor marca temporal de todas las transacciones que hayan leído con éxito la versión Q_k
- cuando una transacción T_i crea una versión nueva Q_k de Q,, la marca_temporal-E y la marca_temporal-L de Q_k se inician con el valor MT(T_i).
- marca_temporal-L de Q_k se actualiza cada vez que una transacción T_j lee Q_k, y MT(T_j) > marca_temporal-L (Q_k).



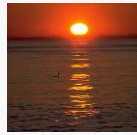
Ordenación por marcas temporales multiversión (cont.)

- Supóngase que la transacción T_i genera una operación **leer**(Q) o **escribir**(Q). Permítase que Q_k indique la versión de Q cuya marca temporal de escritura es la mayor marca temporal menor o igual que MT(T_i).
  1. Si la transacción T_i ejecuta **leer**(Q) entonces el valor que se devuelve es el contenido de la versión Q_k.
  2. Si la transacción T_i ejecuta **escribir**(Q)
     1. si MT(T_i) < marca_temporal-L(Q_k) entonces la transacción T_i se retrocede.
     2. si MT(T_i) = marca_temporal-E(Q_k) se sobrescribe el contenido de Q_k,
     3. en otro caso se crea una nueva versión de Q.
- Obsérvese que:
  - Las lecturas siempre tienen éxito
  - Se rechaza una escritura de T_i si alguna otra transacción T_j que (en el orden de secuencialidad definido por los valores de la marca temporal) debiera leer la escritura de T_i ya ha leído una versión creada por una transacción anterior a T_i.
- El protocolo asegura la secuencialidad.



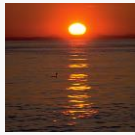
Bloqueo de dos fases multiversión

- Diferencia entre transacciones de sólo lectura y transacciones de actualización
- Las *transacciones de actualización* adquieren bloqueos de lectura y escritura, y mantienen todos los bloqueos hasta el final de la transacción. Es decir, las transacciones de actualización siguen un bloqueo de dos fases riguroso.
  - Cada **escribir** con éxito tiene como resultado la creación de una nueva versión de los elementos de datos escritos.
  - cada versión de un elemento de datos tiene una marca temporal simple cuyo valor se obtiene de un contador **contador-mt** que se incrementa durante el procesamiento del compromiso.
- A las *transacciones de sólo lectura* se les asigna una marca temporal leyendo el valor actual de **contador-mt** antes de que inicien la ejecución; siguen el protocolo de ordenación por marcas temporales multiversión para llevar a cabo las lecturas.



Bloqueo de dos fases multiversión (cont.)

- Cuando una transacción de actualización desee leer un elemento de datos:
  - obtendrá un bloqueo compartido en él, y leerá la última versión.
- Cuando desee escribir un elemento
  - obtiene un bloqueo exclusivo en él y, a continuación, crea una versión nueva del elemento y fija la marca temporal de esta versión a ∞.
- Cuando se complete la transacción de actualización T_i, se produce el procesamiento del compromiso:
  - T_i fija la marca temporal de las versiones que ha creado a **contador-mt** + 1
  - T_i incrementa **contador-mt** en 1
- Las transacciones de sólo lectura que comiencen después de que T_i incremente **contador-mt** verán los valores que T_i ha actualizado
- Las transacciones de sólo lectura que comiencen antes de que T_i incremente **contador-mt** verán los valores anteriores a que T_i se haya actualizado
- Sólo se producen planificaciones secuenciables.



Tratamiento de interbloqueos

- Considérense las dos transacciones siguientes:

  T_1: escribir(X) T_2: escribir(Y)

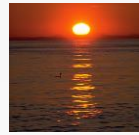
  escribir(Y) escribir(X)

- Planificación con interbloqueos

T_1	T_2
bloquear-X sobre X	
escribir (X)	
	bloquear-X sobre Y
	escribir (X)
	esperar a **bloquear-X** sobre X
esperar a **bloquear-X** sobre Y	

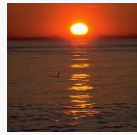
Tratamiento de interbloqueos

- Un sistema está interbloqueado si existe un conjunto de transacciones tales que cada una de ellas está esperando a otra del mismo conjunto.
- Los protocolos de ***prevención de interbloqueos*** aseguran que el sistema *nunca* llega a un estado de interbloqueo. Algunas estrategias de prevención son:
  - Pedir que cada transacción bloquee todos sus elementos de datos antes de que comience a ejecutarse (predeclaración).
  - Imponer el ordenamiento parcial de todos los elementos de datos y pedir que una transacción pueda bloquear elementos de datos solamente en el orden especificado en el ordenamiento parcial (protocolo basado en grafos).



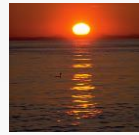
Más estrategias de prevención de interbloqueos

- Los siguientes esquemas utilizan marcas temporales de transacción sólo para la prevención de interbloqueos.
- esquema **esperar-morir** — sin expropiación
  - la transacción más antigua puede esperar a la nueva para liberar el elemento de datos. Las transacciones más nuevas nunca esperan por las más antiguas, en vez de ellos se retroceden.
  - una transacción puede morir varias veces antes de que adquiera el elemento de datos necesario
- esquema **herir-esperar** — con expropiación
  - la transacción más antigua *hiere* (fuerza el retroceso) de la nueva en vez de esperarla. Las transacciones más nuevas esperan por las más antiguas.
  - puede que se produzcan menos retrocesos que en el esquema *esperar-morir*.



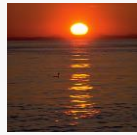
Prevención de interbloqueos (Cont.)

- Tanto en el esquema *esperar-morir* como en el esquema *herir-esperar*, las transacciones retrocedidas se reinician con su marca temporal original. De este modo, las transacciones más antiguas tienen precedencia sobre las más nuevas y, por lo tanto, se impide que se produzca la inanición.
- Esquemas basados en límite de tiempo:
  - una transacción espera un bloqueo sólo durante un período de tiempo especificado. Después, la espera expira y se retrocede la transacción.
  - por lo que los interbloqueos no son posibles
  - fácil de implementar; pero existe la posibilidad de que aparezca la inanición. También es complicado determinar un buen valor para el intervalo de límite de tiempo.

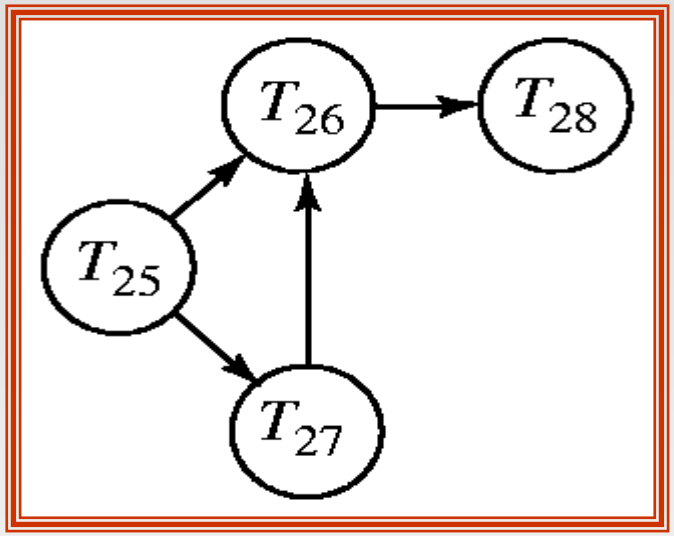


Detección de interbloqueos

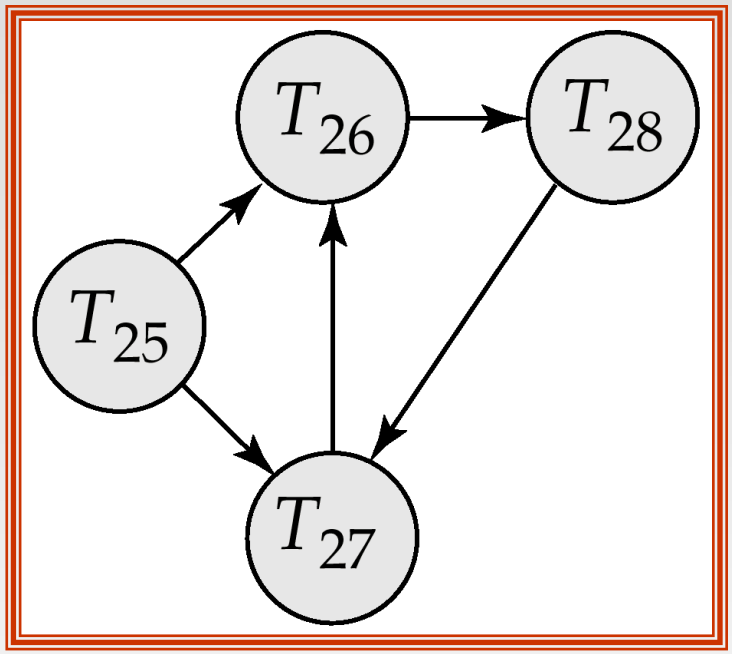
- Los interbloqueos se pueden describir como un *grafo de espera*, que consta de un par $G = (V,A)$,
  - V es un conjunto de vértices (todas las transacción del sistema)
  - A es el conjunto de arcos ; cada elemento es un par ordenado $T_i \rightarrow T_j$.
- Si $T_i \rightarrow T_j$ pertenece a A, entonces hay un arco dirigido de la transacción T_i a T_j, lo cual implica que la transacción T_i está esperando a que la transacción T_j libere un elemento de datos que necesita.
- Cuando la transacción T_i solicita un elemento de datos que posee actualmente la transacción T_j, entonces se inserta el arco $T_i \rightarrow T_j$ en el grafo de espera. Este arco sólo se borra cuando la transacción T_j deja de poseer un elemento de datos que necesite la transacción T_i.
- Existe un interbloqueo en el sistema si y sólo si el grafo de espera contiene un ciclo. Se debe invocar periódicamente a un algoritmo de detección de interbloqueos que busque un ciclo en el grafo.



Detección de interbloqueos (Cont.)



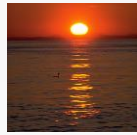
Grafo de espera sin ciclo



Grafo de espera con un ciclo

Recuperación de interbloqueos

- Cuando se detecta un interbloqueo:
  - Se tendrán que retroceder algunas transacciones (selección de una víctima) para romper el interbloqueo. Se debe seleccionar como víctima aquella transacción que incurra en un coste mínimo.
  - Retroceso -- determinar hasta dónde se retrocederá dicha transacción
    - Retroceso total: Se aborta la transacción y luego vuelve a comenzar.
    - Sin embargo, es más efectivo retroceder la transacción sólo lo necesario para romper el interbloqueo.
  - Se produce la inanición cuando siempre se elige a la misma transacción como víctima. Para impedir la inanición se debe incluir en el factor de coste el número de retrocesos.

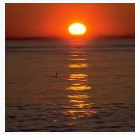


Operaciones insertar y borrar

- Si se utiliza el bloqueo de dos fases:
  - Una operación **borrar** sólo se puede llevar a cabo si la transacción que borra la tupla tiene un bloqueo exclusivo sobre la tupla que se está borrando.
  - A una transacción que inserta una tupla nueva en la base de datos se le proporciona el bloqueo exclusivo de dicha tupla
- Las inserciones y borrados pueden llevar al **fenómeno fantasma**.
  - Una transacción que rastree una relación (p. e., encontrar todas las cuentas en Navacerrada) y una transacción que inserte una tupla en la relación (p. e., insertar una cuenta nueva en Navacerrada) pueden estar en conflicto a pesar de no acceder a ninguna tupla común.
  - Si sólo de utilizan bloqueos de tuplas, las planificaciones no secuenciables pueden tener como resultado: que la transacción de rastreo no pueda ver la cuenta nueva, todavía se está secuencializando antes de insertar la transacción.

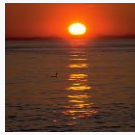
Operaciones insertar y borrar (cont.)

- La transacción que rastrea la relación lee la información que indica qué tuplas contiene la relación, al mismo tiempo que la transacción que inserta una tupla actualiza la misma información.
  - La información debería bloquearse.
- Una solución:
  - Asociar un elemento de datos con la relación, para representar la información acerca de qué tuplas contiene la relación.
  - Las transacciones que rastrean la relación adquieren un bloqueo compartido en el elemento de datos,
  - Las transacciones que insertan o borran una tupla adquieren un bloqueo exclusivo en el elemento de datos. (Nota: los bloqueos en los elementos de datos no entran en conflicto con los bloqueas en tuplas individuales.)
- El protocolo anterior proporciona concurrencia baja para las inserciones/borrados.
- Los protocolos de bloqueo de índice proporcionan una concurrencia superior al tiempo que evitan el fenómeno fantasma, al pedir bloqueos de determinados índices de la relación.



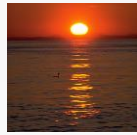
Protocolo de bloqueo de índices

- Cada relación debe tener, al menos, un índice. El acceso a una relación sólo debe hacerse a través de uno de los índices de la relación.
- Una transacción T_i que lleva a cabo una consulta debe bloquear todos los índices de la relación a los que tiene acceso en modo C.
- Una transacción T_i no puede insertar una tupla t_i en una relación r sin actualizar todos los índices a r.
- T_i debe llevar a cabo una consulta en cada índice para encontrar todos los índices de la relación que tengan posibilidades de contener un puntero a la tupla t_i, si ya existía, y obtener los bloqueos en modo exclusivo de todos los índices de la relación. T_i también debe obtener los bloqueos en modo exclusivo de todos los índices de la relación que modifique.
- Se deben observar las reglas del protocolo de bloqueo en dos fases.
  - Garantiza que el fenómeno fantasma no ocurre



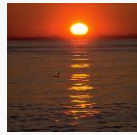
Niveles débiles de consistencia

- **Consistencia de grado dos** difiere del bloqueo de dos fases en que los bloquear-S se pueden liberar en cualquier momento y los bloqueos se pueden adquirir en cualquier momento
  - los bloqueos X deben conservarse hasta el final de la transacción
  - la secuencialidad no está garantizada, los programadores deben asegurar que no se puede producir ningún estado de base de datos errónea
- **Estabilidad del cursor**:
  - Para lecturas, cada tupla está bloqueada; leer y el bloqueo se libera inmediatamente
  - Los bloqueos X se conservan hasta el final de la transacción
  - Caso especial de consistencia de grado dos



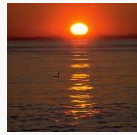
Niveles débiles de consistencia en SQL

- SQL permite ejecuciones no secuenciales
  - **Secuenciable:** es el valor por defecto
  - **Lectura repetible**: permite solamente registros comprometidos a ser leídos y al repetir una lectura debería devolver el mismo valor (así, deberían mantenerse bloqueos de lectura)
    - Sin embargo, no es necesario impedir el fenómeno fantasma
      - T1 puede ver algunos registros, pero otros no, insertados por T2
  - **Con compromiso de lectura**: igual que el grado de consistencia dos, pero la mayoría de los sistemas lo implementan como estabilidad del cursor
  - **Sin compromiso de lectura**: permite incluso que datos sin comprometer sean leídos



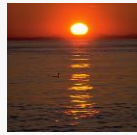
Concurrencia en las estructuras de índices

- Los índices son diferentes de otros elementos de las bases de datos en las que su único trabajo es ayudar a acceder a los datos.
- A las estructuras de índice generalmente se accede muy a menudo, mucho más que a otros elementos de las bases de datos.
- El tratar las estructuras de índices como a otros elementos de las bases de datos conduce a una baja concurrencia. El bloqueo en dos fases de un índice puede tener como resultado que las transacciones se ejecuten prácticamente una cada vez.
- Es aceptable no tener acceso concurrente no secuenciable a un índice en tanto en cuanto se mantenga la precisión del índice.
- En particular, los valores exactos leídos en un nodo interno de un árbol B^+ son irrelevantes siempre y cuando vayamos a parar al nodo de la hoja correcto.
- Existen protocolos de concurrencia de índice en los que los bloqueos en los nodos internos se liberan con antelación y no en dos fases.



Concurrencia en las estructuras de índices (cont.)

- Ejemplo de protocolo de concurrencia de índice:
- Utilícese la técnica denominada del **cangrejo** en vez del bloqueo en dos fases en los nodos del árbol B^+, como sigue. Durante la búsqueda/inserción/borrado:
  - Bloquear primero el nodo raíz en modo compartido.
  - Después de bloquear todos los hijos necesarios de un nodo en nodo compartido, liberar el bloqueo del nodo.
  - Durante la inserción/borrado, subir los bloqueos del nodo hoja al modo exclusivo.
  - Cuando la división o combinación requiera cambios en un padre, bloquear éste en modo exclusivo.
- El protocolo anterior puede ocasionar excesivos interbloqueos. Se dispone de mejores protocolos; consúltese la Sección 16.9 sobre uno de dichos protocolos, el denominado de árboles B enlazados



Fin del capítulo

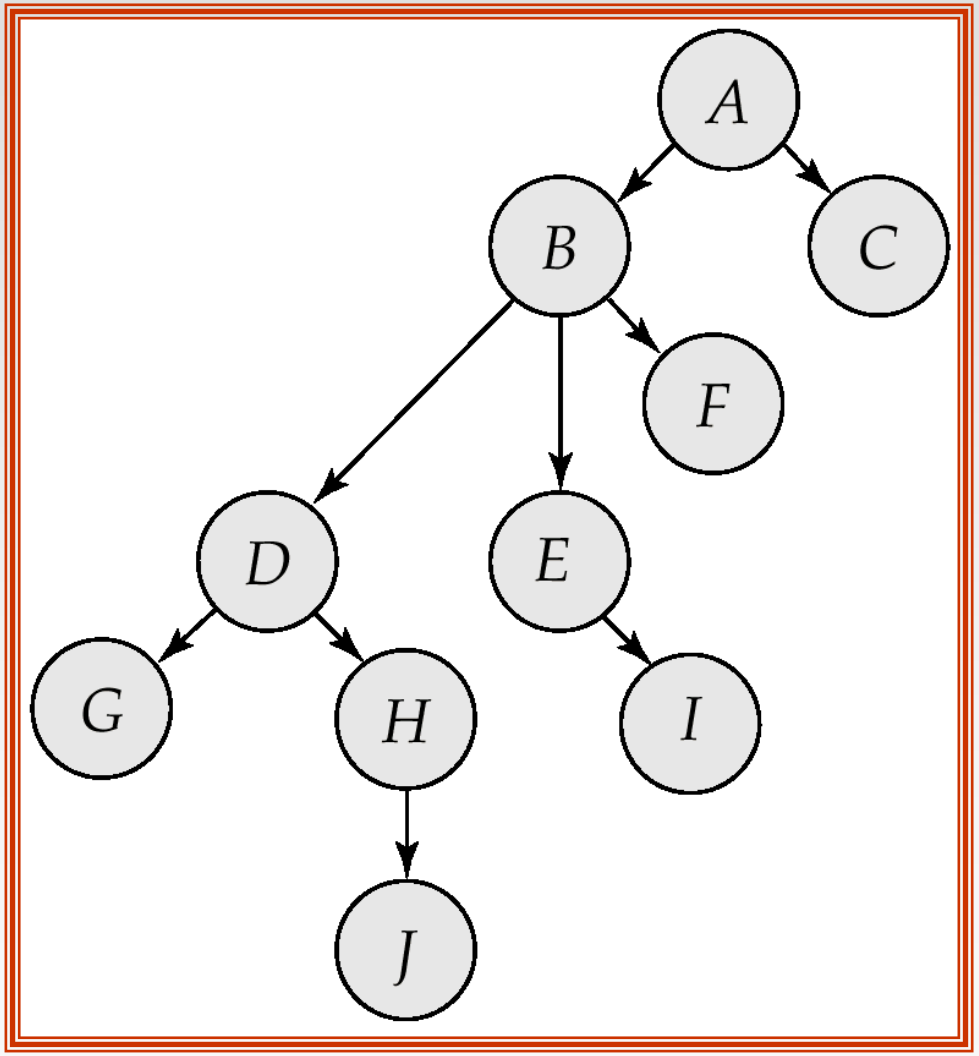
Planificación parcial bajo bloqueo de dos fases

T_5	T_6	T_7
bloquear-X(A)		
leer(A)		
bloquear-C(B)		
leer(B)		
escribir(A)		
desbloquear(A)		
	bloquear-X(A)	
	leer(A)	
	escribir(A)	
	desbloquear(A)	
		bloquear-C(A)
		leer(A)

Planificación incompleta con una conversión de bloqueo

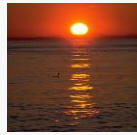
T_8	T_9
bloquear-C(a_1)	
	bloquear-C(a_1)
bloquear-C(a_2)	
	bloquear-C(a_2)
bloquear-C(a_3)	
bloquear-C(a_4)	
	desbloquear(a_1)
	desbloquear(a_2)
bloquear-C(a_n)	
subir(a_1)	

Grafo de la base de datos estructurada en árbol



Planificación de la secuencialidad bajo el protocolo de árbol

T_{10}	T_{11}	T_{12}	T_{13}
bloquear-X(B)			
	bloquear-X(D)		
	bloquear-X(H)		
	desbloquear(D)		
bloquear-X(E)			
bloquear-X(D)			
desbloquear(B)			
desbloquear(E)			
		bloquear-X(B)	
		bloquear-X(E)	
	desbloquear(H)		
bloquear-X(G)			
desbloquear(D)			
			bloquear-X(D)
			bloquear-X(H)
			desbloquear(D)
			desbloquear(H)
		desbloquear(E)	
		desbloquear(B)	
desbloquear(G)			



Planificación 3

T_{14}	T_{15}
leer(B)	
	leer(B)
	$B := B - 50$
	escribir(B)
leer(A)	
	leer(A)
visualizar($A + B$)	
	$A := A + 50$
	escribir(A)
	visualizar($A + B$)

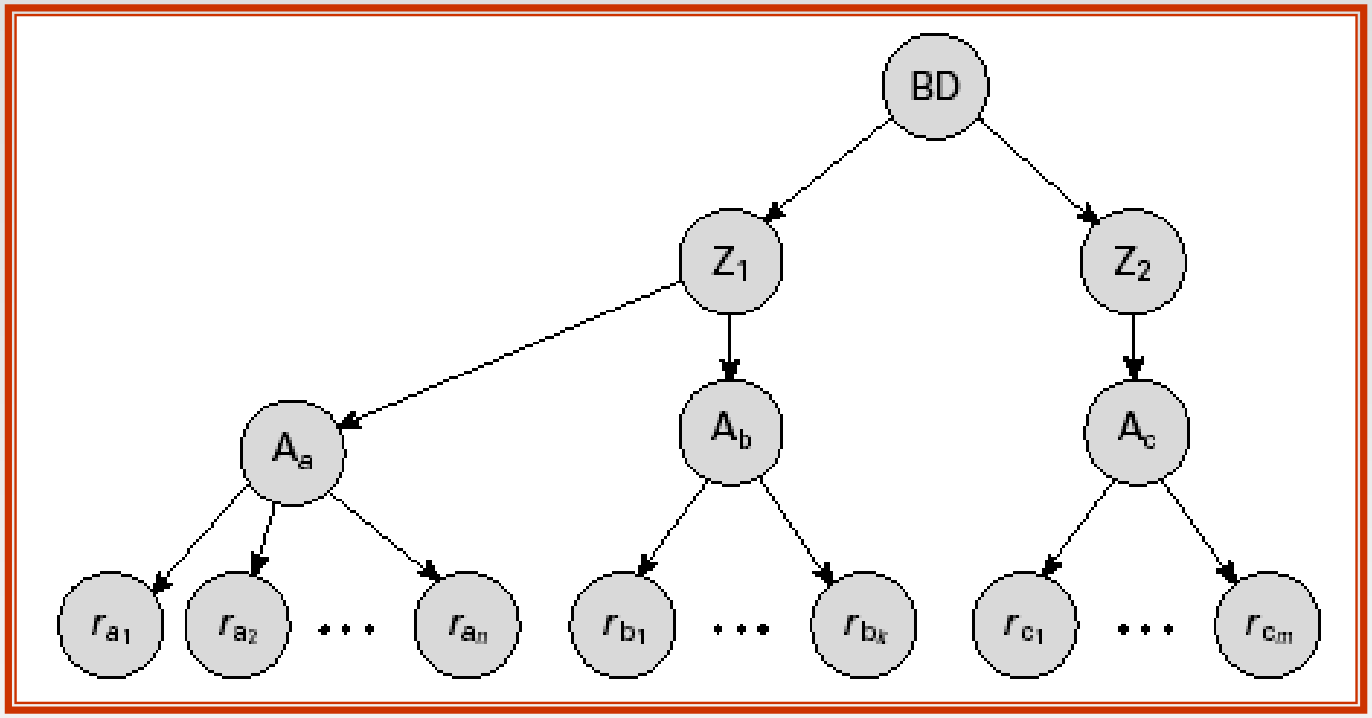
Planificación 4

T_{16}	T_{17}
leer(Q)	
	escribir(Q)
escribir(Q)	

Planificación 5, una planificación producida mediante el empleo de validaciones

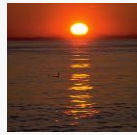
T_{14}	T_{15}
leer(B)	
	leer(B)
	$B := B - 50$
	leer(A)
	$A := A + 50$
leer(A)	
$\langle validar \rangle$	
visualizar($A + B$)	
	$\langle validar \rangle$
	escribir(B)
	escribir(A)

Jerarquía de granularidad



Matriz de compatibilidad

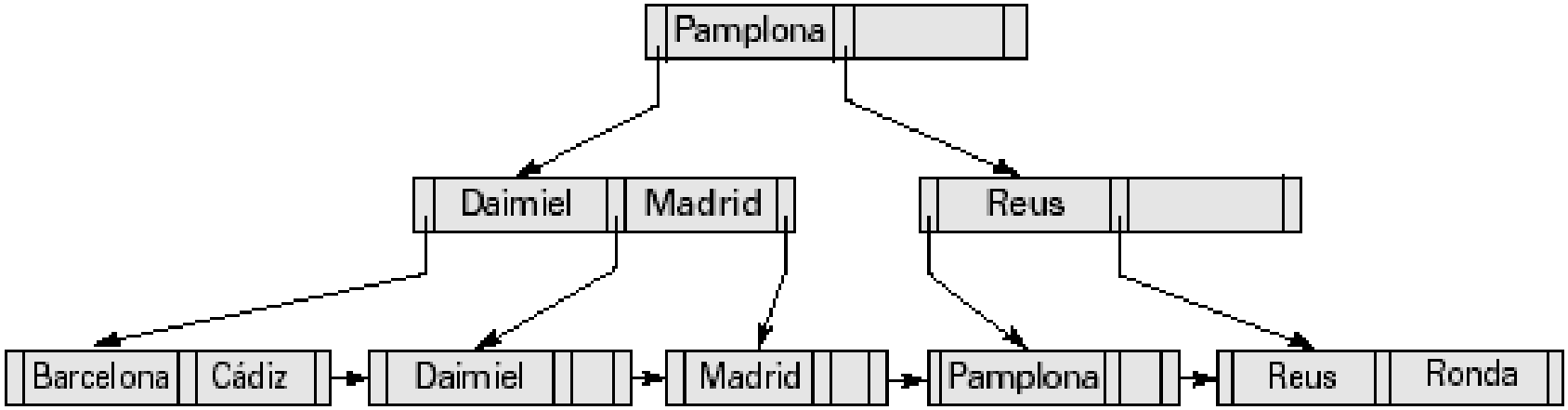
	IC	IX	C	IXC	X
IC	cierto	cierto	cierto	cierto	falso
IX	cierto	cierto	falso	falso	falso
C	cierto	falso	cierto	falso	falso
IXC	cierto	falso	falso	falso	falso
X	falso	falso	falso	falso	falso



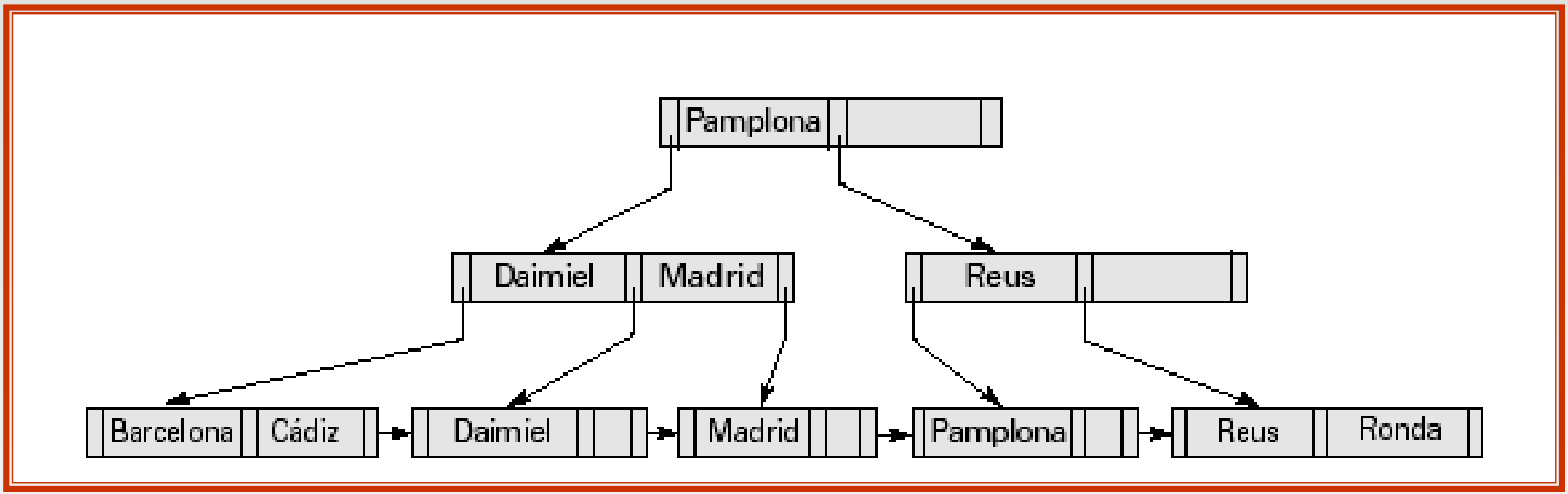
Planificación no secuenciable con consistencia de grado dos

T_3	T_4
bloquear-C(Q)	
leer(Q)	
desbloquear(Q)	
	bloquear-X(Q)
	leer(Q)
	escribir(Q)
	desbloquear(Q)
bloquear-C(Q)	
leer(Q)	
desbloquear(Q)	

Árbol B^+ para el archivo *cuenta* con $n = 3$.



Inserción de “clearview“ en el árbol B+ de la figura 16.21



Matriz de compatibilidad de bloqueos

	C	X	I
C	cierto	falso	falso
X	falso	falso	falso
I	falso	falso	cierto