Teórico 8

Transacciones

Transacciones

- Una **transacción** es una unidad de la ejecución de un programa que accede y posiblemente actualiza varios elementos de datos.
- Una transacción debe ver una base de datos consistente.
- Durante la ejecución de una transacción la base de datos puede quedar inconsistente temporalmente.
- Cuando la transacción se completa con éxito (se compromete), la base de datos debe quedar consistente.
- Después de que una transacción se compromete, los cambios que ha hecho a la base de datos persisten, aun cuando hay fallos del sistema.
- Varias transacciones pueden ejecutarse en paralelo (Concurrentemente).

Considere el Siguiente Caso

- Una transacción que transfiere $500 de la cuenta A a la cuenta B:

1. **read**(*A*)
2. *A* := *A* – 500
3. **write**(*A*)
4. **read**(*B*)
5. *B* := *B* + 500
6. **write**(*B)*

Se Debería Garantizar

- Si la transacción falla después del paso 3 y antes del 6, el sistema debería asegurar que la actualización hecha no se refleje en la base de datos, si no es así, la base de datos quedará inconsistente.

- Que la suma de A y B no cambia por la ejecución de la transacción.

- Que ninguna otra transacción acceda a los datos entre los pasos 3 y 6 porque los registros están actualizados parcialmente en la base de datos y esa transacción verá valores inconsistentes.

- Una vez que el usuario ha sido notificado que la transacción ha sido completada, o sea que se han transferido los $500, las actualizaciones en la base de datos perdurarán en el tiempo, por más que haya alguna falla en el sistema (de hardware o software).

Propiedades ACID

Una **transacción** es una unidad de la ejecución de un programa que accede y posiblemente actualiza elementos datos. Para asegurar la integridad de los datos, un sistema de base de datos debe asegurar las siguientes propiedades:

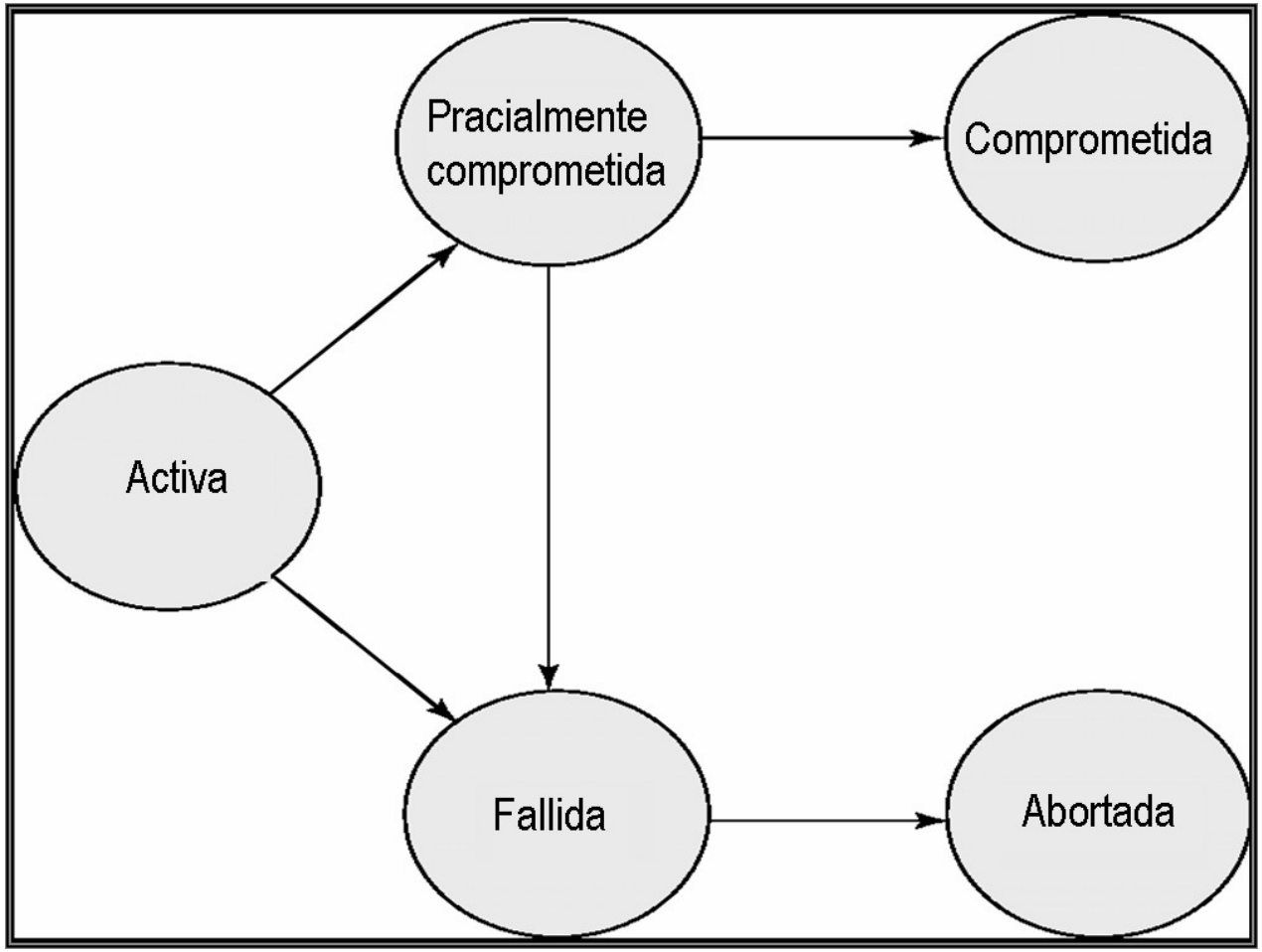
- **Atomicidad:** O todas las operaciones de una transacción se realizan adecuadamente en la base de datos o ninguna.
- **Consistencia:** La ejecución aislada de una transacción (sin que otra transacción se ejecute concurrentemente con esta) preserva la consistencia de la base de datos.
- **Aislamiento:** Aunque se puedan ejecutar varias transacciones concurrentemente, el sistema garantiza que para cada par de transacciones T_i y T_j, se cumple que T_i comienza su ejecución cuando ha terminado de ejecutarse T_j, o T_j comienza cuando ha terminado de ejecutarse T_i . De esta forma cada transacción ignora al resto de las transacciones que se ejecutan concurrentemente en el sistema
- **Durabilidad:** Después que termina con éxito una transacción, los cambios en base de datos hechos por la transacción permanecen incluso si hay fallas en el sistema.

Gráfico Temporal de la Ejecución de una Transacción

Estados de una Transacción

- **Activa:** La transacción está en este estado mientras se está ejecutando.
- **Parcialmente comprometida:** Después que la última instrucción fue ejecutada.
- **Fallida:** Imposible de continuar su ejecución normal.
- **Abortada:** Transacción retrocedida y base de datos restaurada al estado anterior a su ejecución. Se puede reiniciar o cancelar :
  - Se puede reiniciar la transacción, sólo no ha habido algún error lógico interno.
  - Matar la transacción.
- **Comprometida:** Se completó correctamente.

Estados de una transacción (Cont)



Ejecución Concurrente

- Si se permite correr concurrentemente a varias transacciones en el sistema, permitirá:
  - **Incrementar la utilización del procesador y el disco**: mientras una transacción usa la CPU otra puede estar leyendo o escribiendo el disco
  - **Reducir el promedio de tiempo de respuesta de las transacciones**: Por ejemplo no es necesario que transacciones cortas esperen a que transacciones largas terminen.
- Son necesarios mecanismos de control de concurrencia apara asegurar la consistencia de la base de datos

Planificación (Schedule)

- Una planificación define el orden en que las instrucciones de un conjunto de transacciones concurrentes deben ser ejecutadas.
  - Debe preservar el orden original de las instrucciones en la transacción original
  - Debe contener todas las instrucciones de las transacciones.

Ejemplos de Diferentes Planificaciones

Considerar la transacción *T*1 que transfiere $500 de *A* a *B*, y la transacción *T*2 transfiere el 10% del saldo de *A* a *B.*

Planificación 1

- Una planificación serie donde T_1 se ejecuta antes que T_2:

T1	**T2**
Leer(A)	
A:=A-500	
Escribir(A)	
Leer(B)	
B:=B+500	
Escribir(B)	
	Leer(A)
	aux:=A*0.1
	A:=A-aux
	Escribir(A)
	Leer(B)
	B:=B+aux
	Escribir(B)

Planificación 2

- Una planificación serie donde T_2 se ejecuta antes que T_1:

T1	T2
	Leer(A)
	aux:=A*0.1
	A:=A-aux
	Escribir(A)
	Leer(B)
	B:=B+aux
	Escribir(B)
Leer(A)	
A:=A-500	
Escribir(A)	
Leer(B)	
B:=B+500	
Escribir(B)	

Planificación 3

• La siguiente planificación no es serie o secuencial, pero es equivalente a la planificación 2 que es una planificación serie, preserva la suma de A + B.

T1	T2
	Leer(A)
	aux:=A*0.1
	A:=A-aux
	Escribir(A)
Leer(A)	
A:=A-500	
Escribir(A)	
	Leer(B)
	B:=B+aux
	Escribir(B)
Leer(B)	
B:=B+500	
Escribir(B)	

Planificación 4

• La siguiente planificación no es serie, y no es equivalente a ninguna la planificación serie, pues no preserva la suma de A + B.

T1	T2
Leer(A)	
A:=A-500	
	Leer(A)
	aux:=A*0.1
	A:=A-aux
	Escribir(A)
	Leer(B)
Escribir(A)	
Leer(B)	
B:=B+500	
Escribir(B)	
	B:=B+aux
	Escribir(B)

Secuencialidad

- Se asume que:
  - Cada transacción preserva la consistencia de la base de datos.
  - La ejecución en serie de un conjunto de transacciones preservan la consistencia de la base de datos.
- Una (posiblemente concurrente) planificación es secuenciable si es equivalente a una planificación secuencial. Las dos formas de equivalencia son:
  1. Seriabilidad en cuanto a conflicto.
  2. Seriabilidad en cuanto a vistas.

Seriabilidad en Cuanto a Conflictos

- Las instrucciones I_i y I_j de las transacciones T_i y T_j respectivamente están en conflicto si y sólo si existen algún item Q accedido por ambas instrucciones y al menos una de las instrucciones escribe Q.
- Es decir:

1. I_i = **leer**(*Q*), I_j = **leer**(*Q*).	I_i y I_j no tienen conflicto.
2. I_i = **leer**(*Q*), I_j = **escribir**(*Q*).	Tienen conflicto.
3. I_i = **escribir**(*Q*), I_j = **leer**(*Q*).	Tienen conflicto
4. I_i = **escribir**(*Q*), I_j = **escribir**(*Q*).	Tienen conflicto

Secuencialidad en Cuanto a Conflictos (cont)

- Si una planificación P puede ser transformada en una planificación P' por una serie de intercambios en el orden de ejecución de una secuencia de instrucciones que no están en conflicto, se dice que P y P' son **equivalentes en cuanto a conflicto**.

- Se dice que una planificación es **secuenciable en cuanto a conflicto** si es equivalente en cuanto a conflicto a una planificación secuencial.

- Ejemplo de una planificación no secuenciable en cuanto a conflictos:

T_3	T_4
	leer(Q)
leer(Q)	
	escribir(Q)
escribir(Q)	

Esta planificación no es equivalente en cuanto a conflicto a $< T_3, T_4 >$, ni a $< T_4, T_3 >$.

Ejemplo de una Planificación Secuenciable en Cuanto a Conflicto

T1	T2
Leer(A)	
Escribir(A)	
	Leer(A)
	Escribir(A)
Leer(B)	
Escribir(B)	
	Leer(B)
	Escribir(B)

Test de Secuencialidad en Cuanto a Conflicto

- Se construye un Grafo cuyos nodos son las transacciones de la planificación y los arcos se definen de la siguiente forma,

  se agrega un arco al grafo de $T_i \rightarrow T_j$ si:

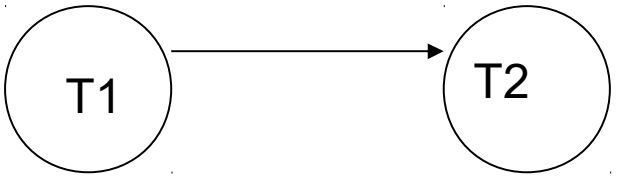
  - *T_i escribe(Q) antes que T_j lee(Q).*
  - *T_i lee(Q) antes que T_j escribe(Q).*
  - *T_i escribe(Q) antes que T_j escribe(Q).*

- *Una vez construido el grafo, si el grafo tiene ciclos la planificación no es secuenciable, en caso contrario si lo es.*

Ejemplo

T1	T2
Leer(A)	
Escribir(A)	
	Leer(A)
	Escribir(A)
Leer(B)	
Escribir(B)	
	Leer(B)
	Escribir(B)

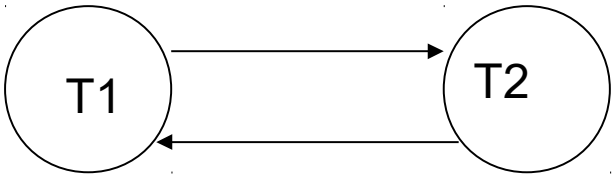
Grafo:



Ejemplo (sigue)

T1	T2
Leer(A)	
	Leer(A)
Escribir(A)	
	Escribir(A)

Grafo:



Secuencialidad en Cuanto a Vista

- Dado dos planificaciones P y P´ con el mismo conjunto de transacciones. P y P´ son equivalente en cuanto a vistas si se cumplen la siguientes condiciones:
  1. Para todo elemento de datos *Q,* si la transacción T_i lee el valor inicial de Q en la Planificación P , entonces la transacción T_i debe leer también el valor inicial de Q en la planificación P'*.*
  1. Para todo elemento de datos *Q,* si la transacción T_i ejecuta **leer**(*Q)* en la planificación P, y el valor lo ha producido la transacción T_j (si existe), entonces la transacción T_i debe leer también el valor de Q que haya producido la transacción T_j en la P'.
  3. Para todo elemento de datos *Q*, la transacción (si existe) que realice la última operación **escribir**(*Q*) en la planificación P, debe realizar la última operación **escribir**(*Q*) en la planificación P´.

Secuencialidad en cuanto a Vista

- Una planificación es **secuenciable en cuanto a vista** si es equivalente en cuanto a vista a una planificación secuencial.
- Si una planificación es secuenciable en cuanto a conflicto es secuenciable en cuanto a vista también.
- La siguiente planificación es secuenciable en cuanto a vista y no en cuanto a conflicto

T1	T2	T3
Leer(A)		
	Escribir(A)	
Escribir(A)		
		Escribir(A)

Recuperabilidad

- En un sistema concurrente, si una transacción falla se debe deshacer los efectos de esta para asegurar la propiedad de Atomicidad. Para esto, también es necesario asegurar que toda transacción que dependa de la que fallo también debe abortarse.

- Para lograr esto es necesario definir un conjunto de restricciones a las planificaciones del sistema.

Planificaciones Recuperables

- Una Planificación es recuperable si para todo par de transacciones T_i T_j tal que T_j lee elementos de datos que a escrito previamente $T_{i,}$,la operación commit de T_i aparece antes que la operación commit de T_j

- Ejemplo*:*

T1	T2
Leer(A)	
Escribir(A)	
	Leer(A)
	Escribir(A)
Leer(B)	

Si T2 hace el commit inmediatamente después de Escribir(A), la planificación no es recuperable.

Planificación sin Cascada

- Una Planificación sin cascada es aquella que para todo par de transacciones *Ti* y *Tj* tal que *Tj* lee elementos de datos que a escrito previamente *Ti,* la operación commit de *Ti* aparece antes que la operación Lectura de *Tj.*
- Ejemplo:

T1	**T2**	**T3**
Leer(A)		
Leer(B)		
Escribir(A)		
	Leer(A)	
	Escribir(A)	
		Leer(A)

- Si el commit de T1 aparece después del Leer de T2 la planificación tiene retroceso en cascada. En cambio si el commit de T1 aparece antes de el Leer de T2 , y si el commit de T2 aparece antes del Leer de T3 la planificación no tiene retroceso en cascada.

Definición de Transacciones en SQL

- Todas las transacciones en SQL terminan con una de las siguientes instrucciones:
  - Commit: Compromete la transacción actual y comienza una nueva.
  - Rollback: Provoca que la transacción aborte (se vuelve al estado anterior al comienzo de la transacción).
- La norma especifica que el sistema debe garantizar la secuencialidad como la ausencia del retroceso en cascada.

Problemas Relacionados con el Nivel de Aislamiento

- Lectura sucia: una transacción T1 puede leer la actualización de T2 que todavía no ha confirmado. Si T2 aborta, T1 habría leído un dato incorrecto.

- Lectura no reproducible: Si una transacción lee dos veces un mismo dato y en medio una transacción lo modifica, verá valores diferentes para el dato.

- Fantasmas: una transacción T1 puede leer un conjunto de filas (que cumplan una condición). Si una transacción T2 inserta una fila que también cumple la condición y T1 se repite verá un fantasma, una fila que previamente no existía.

	Lectura sucia (Dirty Read)	Lectura no repetible (Nonrepeatable Read)	Lectura de registros nuevos (Phantom Read)
Read Uncommitted	Posible	Posible	Posible
Read Committed	No posible	Posible	Posible
Repeatable Read	No posible	No posible	-Posible (pero improbable)
Serializable	No posible	No posible	No posible

Control de Concurrencia

- Una de las propiedades fundamentales de las transacciones es el aislamiento.
- Si se ejecutan varias transacciones concurrentes es posible que se viole la propiedad de aislamiento.
- Es necesario para garantizar el aislamiento implementar esquemas de control de concurrencia.
- Los esquemas que se verán garantizan la secuencialidad de las planificaciones.

Protocolos Basados en Bloqueos

- Un mecanismo de bloqueo es un mecanismo para controlar el accesos concurrente a los datos.

- Los elementos de datos pueden ser bloqueados en 2 modos:
  1. ***Exclusivo (X)*.** El elemento de dato X puede ser leído y escrito. Este bloqueo se realiza utilizando la instrucción **bloquear-X(R)**.
  2. ***Compartido (C)*.** El elemento de datos sólo puede ser leído. Este bloqueo de realiza mediante la instrucción **bloquear-C(R)**.

- Los elementos de datos se desbloquean con la instrucción **desbloquear(R)**.

- Los requerimientos de bloqueos se hacen al gestor de control de concurrencia. La transacción puede seguir su ejecución luego de que el bloqueo fue concedido.

Protocolos Basados en Bloqueos (Cont.)

- Matriz de compatibilidad

	C	X
C	Verdadero	Falso
X	Falso	Falso

- Cualquier número de transacciones puede bloquear en modo compartido un elemento de datos.

- Si una transacción *Ti* quiere bloquear un elemento de datos ya bloqueado en un modo incompatible con el que se quiere adquirir, la transacción *Ti* debe esperar hasta que el gestor de concurrencia le otorgue el bloqueo, cuando la/las transacciones que tenían adquirido el bloqueo del recurso lo libere.

Ejemplo

T2: **bloquear-C***(A)*;
leer *(A)*;
desbloquear*(A)*;
bloquear-C*(B)*;
leer *(B)*;
desbloquear*(B)*;
mostrar*(A+B)*

- Si alguien actualiza A o B entre las lecturas de A y B se estará mostrando valores incorrectos
- La utilización de bloqueos no garantiza la secuencialidad de las planificaciones.

Posibilidad de Dead Lock

- Considere la siguiente planificación con incorporación de bloqueos:

T1	T2
Bloquear-X(B)	
Leer(B)	
B:=B-50	
Escribir(B)	
	Bloquear-C(A)
	Leer(A)
	Bloquear-C(B)
Bloquear-X(A)	

- En estos casos el sistema debe hacer el rollback de alguna de las dos transacciones, y de esta forma se liberan los recursos adquiridos por la transacción

Concesión de Bloqueos

- Para garantizar la secuencialidad las transacciones deben seguir un conjunto de reglas llamado protocolo de bloqueo.

- Se verán los siguientes protocolos de bloqueos:
  - Bloqueo de dos fases.
  - Basado en grafos.

Protocolo de Bloqueo de Dos Fases

- Es un protocolo que asegura planificaciones secuenciables en cuanto a conflicto.

Las dos fases del protocolo son:

- Fase 1: Fase de crecimiento
  - La transacción puede obtener bloqueos.
  - La transacción no puede realizar desbloqueos.
- Fase 2: Fase de decrecimiento
  - La transacción puede realizar desbloqueos.
  - La transacción no puede realizar bloqueos.

Ejemplo

T1	T2	T3
Bloquear-X(A)		
Escribir(A)		
Bloquear-X(B)		
Desbloquear(A)		
	Bloquear-X(A)	
	Leer(A)	
Leer(B)		
	Escribir(A)	
	Desbloquear(A)	
		Bloquear-C(A)
		Leer(A)
Escribir(B)		

Protocolo de dos Fases Riguroso

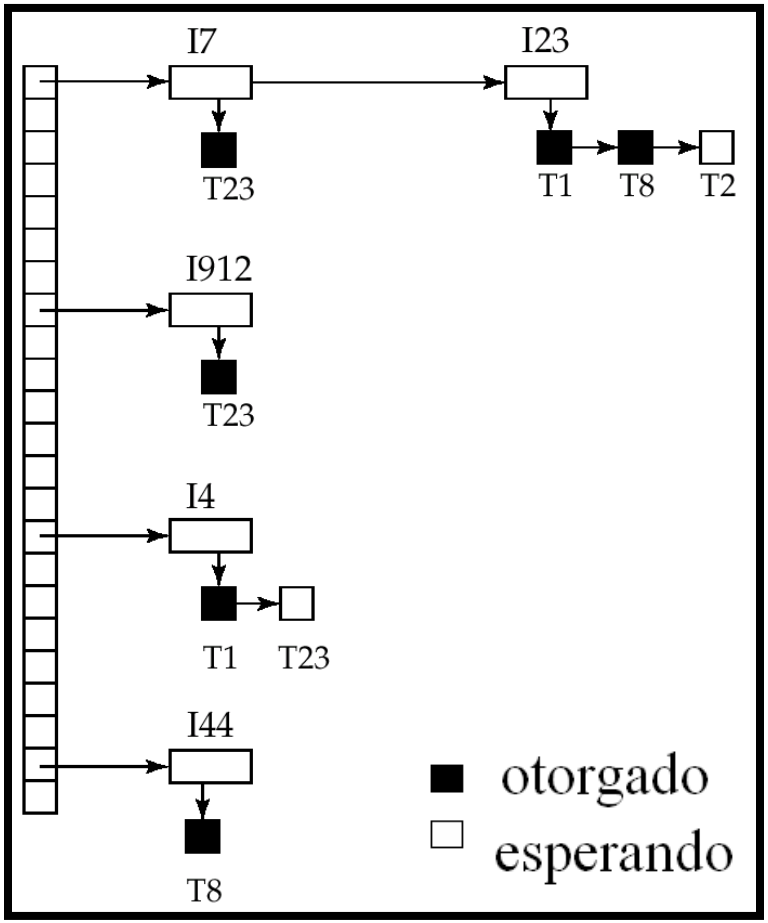
- Con el protocolo de dos fases se puede tener retroceso en cascada.

- Para evitar eso se hace una modificación sobre protocolo, que se denomina protocolo de dos fases riguroso, el cual exige que se posean todos los bloqueos hasta que la transacción haga el commit o el rollback.

Conversión de Bloqueos

- Para permitir un grado mayor de concurrencia se puede hacer una variante al protocolo de dos fases básico.
- Se permite hacer conversiones de bloqueos, estas conversiones son:
  - Subir: sube de modo compartido a exclusivo
  - Bajar: Baja de modo exclusivo a compartidos:
- La operación de **subir** sólo se puede hacer en la fase de crecimiento.
- La operación de **bajar** sólo se puede hacer en la fase de decrecimiento.
- Si los bloqueos se mantienen hasta el final las planificaciones son sin cascada.

Implementación de los bloqueos



Protocolo Basado en Grafos

- Los protocolos basados en grafos son una alternativa al protocolo de dos Fases.

- Impone un orden parcial $\rightarrow$ sobre un conjunto **D** = {d_1, d_2 ,..., d_h} de todos los ítem de datos.
  - Si $d_i \rightarrow d_j$ entonces cualquier transacción que accede a ambos datos, d_i y d_j debe bloquear d_i antes que a d_j.
  - Esto implica que el conjunto **D** puede ser visto como un grafo dirigido acíclico, este grafo se denomina grafo de la base de datos.

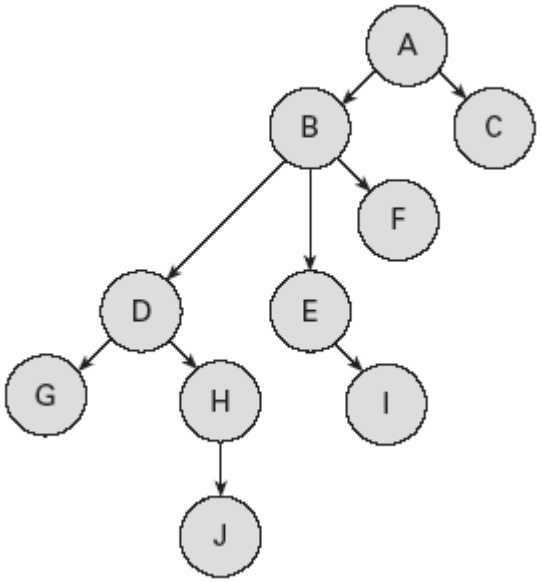
- Un de los protocolos basados en grafos es el protocolo de bloqueo de árbol.

Protocolo de Árbol

- El protocolo sólo permite bloqueos en modos exclusivos.
- El protocolo sigue las siguientes reglas:
  - El primer bloqueo de T_i puede ser sobre cualquier elemento de datos.
  - Después, 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.
  - T_i no puede bloquear un elemento de datos que ya haya bloqueado y desbloqueado anteriormente.

Ejemplo

- Dado el siguiente Árbol



Ejemplo (Cont.)

- La siguiente planificación está hecha con transacciones que siguen el protocolo de árbol para la realización de los bloqueos, siguiendo orden impuesto en el árbol de la transparencia anterior.

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)			

Protocolos de Granularidad Múltiple

- Permite que los elementos de datos sean de tamaños variables y define una jerarquía de granularidad de datos, donde la granularidad más chica esta incluidas en las mas grandes

- La jerarquía puede ser representada por un árbol.

- Cuando una transacción bloquea explícitamente un nodo en el árbol, implícitamente bloquea todos los descendientes en el mismo modo.

- Granularidad del bloqueo (es el nivel en el árbol donde se hace el bloqueo):
  - **Granularidad fina** (próximo a las hojas del árbol): alta concurrencia, alto overhead de bloqueo.
  - **Granularidad gruesa** (próximo a la raíz del árbol): bajo overhead de bloqueo, baja concurrencia.

Tipos de bloqueo

- Se agrega un nuevo modo de bloqueo que se denomina **modo de bloqueo intencional**.

- Si un nodo se bloquea en modo intencional se está haciendo un bloqueo explícito en un nivel inferior del árbol (es decir, en una granularidad más fina). Los bloqueos intencionales se colocan en todos los ascendientes de un nodo antes de bloquearlo explícitamente.

- Cuando una transacción quiere bloquear un recurso Q, debe recorrer el camino desde la raíz al recurso bloqueando en modo intencional.

- No es necesario que una transacción busque en todo el árbol para determinar si puede bloquear un nodo con éxito.

- Además de los modos de bloqueos C y X, se introducen 3 modos nuevos :
  - ***Intencional-compartido*** (IC): se realiza un bloqueo explícito en un nivel inferior del árbol, pero sólo con bloqueos en modo compartido.
  - ***Intencional-exclusivo***(IX): entonces el bloqueo explícito se hace en un nivel inferior con bloqueos en modo exclusivo o en modo compartido.
  - ***Intencional-exclusivo y compartido*** (IXC): el subárbol cuya raíz es ese nodo se bloquea explícitamente en modo compartido, y el bloqueo explícito se produce en un nivel inferior con bloqueos en modo exclusivo.

Función de Compatibilidad de Bloqueos

	IC	IX	C	IXC	X
IC	Verdadero	Verdadero	Verdadero	verdadero	Falso
IX	Verdadero	Verdadero	Falso	Falso	Falso
C	Verdadero	Falso	Verdadero	Falso	Falso
IXC	Verdadero	Falso	Falso	Falso	Falso
X	Falso	Falso	Falso	Falso	Falso

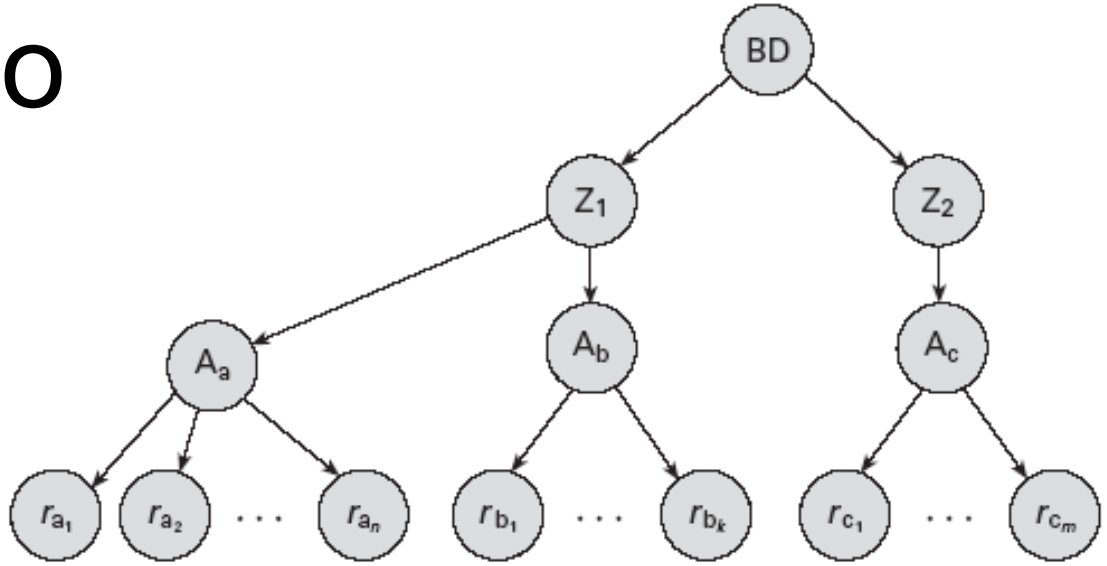
Protocolo

- El **protocolo de bloqueo de granularidad múltiple** siguiente asegura la secuencialidad. Cada transacción *Ti* puede bloquear un nodo *Q* usando las reglas siguientes:

  1. Debe observar la función de compatibilidad de bloqueos de la transparencia anterior.
  2. Debe bloquear la raíz del árbol en primer lugar y puede bloquearla en cualquier modo.
  3. Puede bloquear un nodo *Q* en modo C o IC sólo si está bloqueando actualmente al padre de *Q* en modo IX o IC.
  4. Puede bloquear un nodo *Q* en modo X, IXC o IX sólo si está bloqueando actualmente al padre de *Q* en modo IX o IXC.
  5. Puede bloquear un nodo sólo si no ha desbloqueado previamente ningún nodo (es decir, *Ti* es de dos fases).
  6. Puede desbloquear un nodo *Q* sólo si no ha bloqueado a ninguno de los hijos de *Q*.

- 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).

Ejemplo



1)Supóngase que la transacción *T*1 lee el registro *ra2* del archivo *A*a. Entonces *T*1 necesita bloquear la base de datos, la zona *Z*1 y *A*a en modo IC (y en este orden) y finalmente debe bloquear *ra2* en modo C.

2) Supóngase que la transacción *T2* modifica el registro *ra9* del archivo *A*a. Entonces *T*2 necesita bloquear la base de datos, la zona *Z*1 y el archivo *A*a en modo IX y finalmente debe bloquear *ra9* en modo X.

3) Supóngase que la transacción *T3* lee todos los registros del archivo *A*a. Entonces, *T3* necesita bloquear la base de datos y la zona *Z*1 (y ello en este orden) en modo IC y finalmente debe bloquear *A*a en modo C.

Protocolos Basados en Marcas Temporales

- Los protocolos de bloqueo que se han descrito anteriormente el orden entre dos transacciones conflictivas se determina en tiempo de ejecución a través del primer bloqueo que soliciten ambas que traiga consigo modos incompatibles.

- Otro método para determinar el orden de secuencialidad es seleccionar previamente un orden entre las transacciones.

- El método más común para hacer esto es utilizar un esquema de *ordenación por marcas temporales*.

... Marcas Temporales (Cont.)

- A toda transacción *Ti* del sistema se le asocia una única marca temporal fijada, denotada por MT(*Ti*).
- El sistema de base de datos asigna esta marca temporal antes de que comience la ejecución de *Ti*.
- Si a la transacción *Ti* se le ha asignado la marca temporal MT(*Ti*) y una nueva transacción *Tj* entra en el sistema, entonces MT(*Ti*) < MT(*Tj*).
- Para asignar esta marca temporal se utiliza el reloj del sistema o un contador lógico.

... Marcas Temporales (Cont.)

- Si MT(*Ti*) < MT(*Tj*) entonces el sistema debe asegurar que toda planificación que produzca es equivalente a una planificación secuencial en la cual la transacción *Ti* aparece antes que la transacción *Tj*.

- Para implementar este esquema se asocia a 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*).

Protocolo de Ordenamiento por Marcas Temporales

- 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. Este protocolo sigue la siguiente política.

1. Supóngase que la transacción *Ti* ejecuta leer(*Q*).

a) Si MT(*Ti*) < marca_temporal-E(*Q*) entonces *Ti* necesita leer un valor de *Q* que ya se ha sobrescrito. Por tanto se rechaza la operación leer y *Ti* se retrocede.

b) Si MT(*Ti*) ≥ marca_temporal-E(*Q*) entonces se ejecuta la operación leer y marca_temporal-L(*Q*) se asigna al máximo de marca_temporal-L(*Q*) y de MT(*Ti*).

Protocolo de Ordenamiento por Marcas Temporales (Cont.)

2. Supóngase que la transacción *Ti* ejecuta escribir(*Q*).

a) Si MT(*Ti*) < marca_temporal-L(*Q*) entonces el valor de Q que produce *Ti* se necesita previamente y el sistema asume que dicho valor no se puede producir nunca. Por tanto, se rechaza la operación escribir y *Ti* se retrocede.

b) Si MT(*Ti*) < marca_temporal-E(*Q*) entonces *Ti* está intentando escribir un valor de Q obsoleto. Por tanto, se rechaza la operación escribir y *Ti* se retrocede.

c) En otro caso se ejecuta la operación escribir y MT(*Ti*) se asigna a marca_temporal-E(*Q*).

Ejemplo

Paso	T1	T2	T3
1	L(A)		
2		L(A)	
3		E(A)	
4			L(B)
5	E(A)		
...	.....	...	....

MT-E(A) = ~~0~~, 2

MT-L(A) = ~~1~~, 2

En el paso 5 la instrucción E(A) cumple la condición 2a) y 2b) por lo tanto T1 se retrocede

- Este protocolo:
  - Garantiza planificaciones secuenciables.
  - Puede generar planificaciones no recuperables.
  - Se lo puede extender para que genere planificaciones recuperables.

Esquemas Multiversión

- Los esquemas multiversión llevan varias versiones de cada ítem de datos para incrementar la concurrencia, puede ser:
  - Ordenación de marcas temporales multiversión.
  - Bloqueo de dos fases multiversión.

- Cada operación satisfactoria de escritura resulta en la creación de una nueva versión del ítem de datos escrito.

- Cuando se quiere hacer una operación **leer**(*Q*), el gestor de control de concurrencia elige una de las versiones de *Q* que se va a leer.

- Las lecturas nunca esperan ya que la versión correcta se devuelve inmediatamente.

Ordenación de Marcas Temporales Multiversión

- Cada ítem 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 *Qk*.
  - **marca_temporal-E**(*Qk*) es la marca temporal de la transacción que haya creado la versión *Qk*.
  - **marca_temporal-L**(*Qk*) es la mayor marca temporal de todas las transacciones que hayan leído con éxito la versión *Qk*.

- Cuando una transacción *Ti* crea una nueva versión *Qk* del elemento de datos *Q* realizando la operación escribir(*Q*), el campo contenido de la versión tiene el valor que ha escrito *Ti*. El sistema inicializa la marca_temporal-E y marca_temporal-L con MT(*Ti*).

- El valor marca_temporal-L se actualiza cada vez que una transacción *Tj* lee el contenido de *Qk* y además marca_temporal-L(*Qk*) < MT(*Tj*).

Protocolo

El esquema opera como sigue:

Supóngase que la transacción *Ti* realiza una operación leer(*Q*) o escribir(*Q*). Sea *Qk* la versión de *Q* cuya marca temporal de escritura es la mayor marca temporal menor o igual que MT(*Ti*). Entonces

1. Si la transacción *Ti* ejecuta leer(*Q*), entonces el valor que se devuelve es el contenido de la versión *Qk*.

2. Si la transacción *Ti* ejecuta escribir(*Q*) y si MT(*Ti*) < marca_temporal-L(*Qk*), entonces la transacción *Ti* se retrocede. Si no, si MT(*Ti*) = marca_temporal-E(*Qk*) se sobrescribe el contenido de *Qk*, y en otro caso se crea una nueva versión de *Q*.

- La justificación de la regla 1 es clara. Una transacción lee la versión más reciente que viene antes de ella en el tiempo.

- La regla 2 fuerza a que se aborte una transacción que realice una escritura demasiado tarde.

Ejemplo

Paso	T0	T1	T2	T3
0	E(A)			
1		L(A)		
2			L(A)	
3			E(A)	
4				L(A)
5		E(A)		
...		.....	...	....

Q_{n-1} --- MT-E(Q_{n-1}) = 0
MT-L(Q_{n-1}) = ~~0~~, ~~1~~, 2.

Q_n --- MT-E(Q_n) = 2
MT-L(Q_n) = ~~2~~, 3

En el paso 5, T1 no puede escribir porque MT(*T1*) < MT-L(Q_{n-1}), por lo tanto T1 debe abortarse

Operaciones de Insertar y Borrar

- Hasta el momento se centró la atención en las operaciones de leer y escribir.

- Algunas transacciones necesitan insertar o borrar elementos de datos.

Fenómeno Fantasma

- Considere la siguiente situación:

Una transacción T1 ejecuta

Select min(precio) from provee where nart = 2

Y una transacción T2 ejecuta

Insert into provee (nart,nprov,precio) values(2,4,1.23);

- T1 y T2 están en conflicto en una tupla fantasma. Si se realiza el control de concurrencia con granularidad de tupla, no se detecta dicho conflicto.

- Este problema recibe el nombre de **fenómeno fantasma**.

- Para evitar el fenómeno fantasma se permite que *T1* impida a otras transacciones crear nuevas tuplas en la relación provee con *nart* = 2.

Protocolo de Bloqueo de Índices

- Para evitar este fenómeno se utiliza el **bloqueo de índice**.

- Toda transacción que inserte una tupla en una relación debe insertar información en cada uno de los índices que se mantengan en la relación.

- Se elimina el fenómeno fantasma al imponer un protocolo para los índices.

Niveles de Consistencia en SQL 92

Hay casos que se permite que las transacciones no se conviertan en forma secuencial, por ejemplo transacciones largas para poder realizar estadísticas, en esto casos no es necesario que el resultado sea tan preciso.

- Los niveles de consistencia son:
  - Serializable(Secuenciable), es el nivel predeterminado
  - Repeatable read: Solo lee registros que se han comprometido, siempre lee el mismo registro varias veces siempre leerá el mismo valor.
  - Read Committed: Solo lee registros que se han comprometido, pero diferentes lectura del mismo registro puede resultar en valores distintos
  - Read Uncomitted: Permite leer registros que no se han comprometido

Comando de Cambio de Nivel de Aislamiento

- Para cambiar el nivel de aislamiento se utiliza la sentencias se utiliza el comando:

MySQL:

SET TRANSACTION ISOLATION LEVEL <nivelAilamiento>;

PostgreSQL:

SET DEFAULT_TRANSACTION_ISOLATION = <nivelAilamiento>;

Control de Concurrencia en Postgres

- Implementa MVCC (Multiversion Control Concurrency).
- Las lecturas no bloquean las escrituras y viceversa.
- A cada tabla creada por los usuarios se le agregan atributos del sistema, estos atributos son xmax y xmin.
- xmin representa el identificador de la transacción creadora.
- xmax representa el identificador de la transacción que sustituye o elimina esa versión.

Control de Concurrencia en Postgres(Cont.)

- Postgres ofrece varios modos de bloqueo.
- Unos modos de bloqueo los adquiere Postgres automáticamente antes de la ejecución de una instrucción.
- Otros son proporcionados para ser usados explícitamente por las aplicaciones.
- Todos los modos de bloqueo (excepto para AccessShare) adquiridos en una transacción se mantienen hasta la duración de la transacción.
- Los bloqueos se pueden tomar explícitamente con el comando

LOCK [TABLE] [ONLY] *name* [, ...] [IN *lockmode* MODE] [NOWAIT]

Tipos de bloqueo

- ACCESS SHARE
  - Un modo de bloqueo adquirido automáticamente sobre tablas que están siendo consultadas. Postgres libera estos bloqueos después de que se haya ejecutado una declaración. Lo adquiere el comando **SELECT.**
- ROW SHARE
  - Adquirido por **SELECT FOR UPDATE y SELECT FOR SHARE**
- ROW EXCLUSIVE
  - Lo adquieren **UPDATE**, **DELETE**, **INSERT**
- SHARE UPDATE EXCLUSIVE
  - Lo adquiere el comando **VACCUM**
- SHARE
  - Lo adquiere **CREATE INDEX**.
- SHARE ROW EXCLUSIVE
  - No lo adquiere ningún comando automáticamente.
- EXCLUSIVE
  - No lo adquiere ningún comando automáticamente.
- ACCESS EXCLUSIVE
  - Lo toman **ALTER TABLE, DROP TABLE, TRUNCATE, REINDEX, VACUUM**.

Matriz de Compatibilidades

	Modo de bloqueo corriente							
Modo de bloqueo requerido	**ACCESS SHARE**	**ROW SHARE**	**ROW EXCLUSIVE**	**SHARE UPDATE EXCLUSIVE**	**SHARE**	**SHARE ROW EXCLUSIVE**	**EXCLUSIVE**	**ACCESS EXCLUSIVE**
ACCESS SHARE								X
ROW SHARE							X	X
ROW EXCLUSIVE					X	X	X	X
SHARE UPDATE EXCLUSIVE				X	X	X	X	X
SHARE			X	X	X	X	X	X
SHARE ROW EXCLUSIVE			X	X	X	X	X	X
EXCLUSIVE		X	X	X	X	X	X	X
ACCESS EXCLUSIVE	X	X	X	X	X	X	X	X

Niveles de Aislamiento

- Postgres implementa dos niveles:
  - Read committed.
  - Serializable.

Control de Concurrencia en MySQL

- En el siguiente link del manual de mysql se puede ver el manejo de concurrencia en tablas innodb de mysql:

http://dev.mysql.com/doc/refman/5.0/es/innodb-transaction-model.html

Niveles de Aislamiento

- MySQL implementa los 4 niveles:
  - Read uncommitted.
  - Read committed.
  - Repeatable Read.
  - Serializable.

Sistemas OLTP

- **Procesamiento de Transacciones En Línea** (**O**n**L**ine **T**ransaction **P**rocessing) es un tipo de sistemas que facilitan y administran aplicaciones transaccionales

- Gran cantidad de transacciones concurrentes.

- Esta orientado a transacciones cortas.

- Un cajero automático de un banco es un ejemplo clásico de una aplicación OLTP.