



Dpto. Ciencias e Ingeniería de la Computación
 Universidad Nacional del Sur

ELEMENTOS DE BASES DE DATOS

Segundo Cuatrimestre 2014

**Clase 14:
 Control de Concurrencia
 (Parte II)**



Mg. María Mercedes Vitturini
 [mvitturi@uns.edu.ar]

Transacciones




Transacciones - Repaso



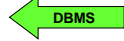

Se denomina **transacción** a una colección de operaciones sobre la base de datos que forman una *unidad lógica de trabajo*.

Características

- Eventualmente puede **acceder y/o actualizar varios ítems de datos**.
- **Accede y deja** a la BD en un estado consistente.
- Puedes suceder que múltiples **transacciones** se ejecuten **concurrentemente**.
- Es el **programador** quién **define los límites** de una transacción.

EBD2014_14 - Mg. Mercedes Vitturini

Propiedades ACID

- ✓ **Atomicidad** (*atomicity*). 
- ✓ **Consistencia** (*consistency*). 
- ✓ **Aislación** (*isolation*). 
- ✓ **Durabilidad** (*durability*). 

EBD2014_14 - Mg. Mercedes Vitturini


Planificaciones

PLANIFICACIÓN se denomina así a la secuencia cronológica en el cual se ejecutan en el sistema las instrucciones de transacciones concurrentes.

$$P_k = \langle I_{i,j} \rangle \quad (\text{con } I_{i,j} \text{ instrucción } j\text{-ésima de } T_i)^1$$

1- Framework centralizado

- Se dice de una planificación que es una **planificación es en serie** si las instrucciones pertenecientes a cada transacción aparecen juntas.
- Una planificación *no en serie* que es una **“planificación serializable”** si el resultado de su ejecución *equivale a alguna* planificación en serie. 😊



EBD2014_14 - Mg. Mercedes Vitturini

Planificaciones Serializables

Estrategias de Testeo

TEST

Equivalencia

Dos planificaciones P_i y P_j son **planificaciones equivalentes**, si a pesar de *no tener la misma secuencia de instrucciones* I_{ip} *generan el mismo resultado final*.

Tests de Equivalencia

- Existen distintos mecanismos para testear o verificar la calidad de serializabilidad de una planificación:
 - Test de **serializabilidad en conflictos** (test con *grafo de conflictos*) ✓
 - Test de **serializabilidad en vistas**.

EBD2014_14 - Mg. Mercedes Vitturini

Conflictos

Sean las instrucciones I_i e I_j referentes a dos transacciones T_i y T_j respectivamente. Se dice que tales instrucciones están en **conflicto** cuando son *instrucciones de transacciones distintas sobre el mismo dato* y *al menos una de ellas es una instrucción Write*.

- Si I_i e I_j **no están en conflicto**, pueden intercambiarse para obtener una planificación S' equivalente a S .
- La forma práctica de verificarlo es construir el grafo de precedencia para conflictos.

EBD2014_14 - Mg. Mercedes Vitturini

Instrucciones Conflictivas

El **conflicto** con instrucciones consecutivas se produce si al menos una de las instrucciones I_i e I_j es una instrucción Write(Q):

- Si $I_i = \text{Read}(Q)$ y $I_j = \text{Write}(Q)$. Si I_i viene antes que I_j entonces T_i no lee el valor de Q que escribe T_j . Si I_j viene antes que I_i entonces T_i lee el valor que escribe T_j . El orden si importa.
- Si $I_i = \text{Write}(Q)$ y $I_j = \text{Read}(Q)$ el caso es análogo al anterior.
- Si $I_i = \text{Write}(Q)$ y $I_j = \text{Write}(Q)$. El orden de las instrucciones afecta a las próximas sentencias Read de la planificación S .

EBD2014_14 - Mg. Mercedes Vitturini

Grafo de Precedencia en Conflicto

Consideremos alguna posible planificación para un conjunto de transacciones T_1, T_2, \dots, T_n

Grafo de Precedencia — $G = (V, A)$ un grafo dirigido donde cada vértice V_i se corresponde con cada transacción T_j .

- Se añade una **arco desde T_i a T_j ($T_i \rightarrow T_j$)** cuando se da alguna de las siguientes condiciones:
 - T_i ejecuta write(Q) antes que T_j ejecute read(Q).
 - T_i ejecuta read(Q) antes que T_j ejecute write(Q).
 - T_i ejecuta write(Q) antes que T_j ejecute write(Q).

Dada una planificación P , se construye el grafo para determinar si es o no serializable en cuanto a conflictos.

EBD2014_14 - Mg. Mercedes Vitturini

Ejemplos

Serializable en Conflictos

P1	T1	T2
1	Read (A)	
2	Write (A)	
3		Read (A)
4		Write (A)
5	Read (B)	
6	Write (B)	
7		Read (B)
8		Write (B)

No serializable en Conflictos

P2	T1	T2
1	Read (A)	
2	Write (A)	
3		Read (A)
4		Write (A)
5		Read (B)
6	Read (B)	
7	Write (B)	
8		Write (B)

EBD2014_14 - Mg. Mercedes Vitturini

Equivalencia en Vistas

Dos planificaciones S y S' son **equivalentes en cuanto a vistas** si cumplen todas las siguientes condiciones para cada ítem de dato Q:

- Si T_i ejecuta **Read(Q)** y lee el valor inicial de Q en S , entonces T_i debe leer el valor inicial de Q en S' .
- Si T_i ejecuta **Read(Q)** en S y ese valor fue producido por T_j (si existe), entonces T_i debe leer en S' el valor producido por T_j .
- La transacción (si existe) que ejecuta **Write(Q)** final en la planificación S debe ejecutar la operación final **Write(Q)** en la planificación S' .

- 1 y 2** aseguran que cada transacción lee los mismos valores en ambas planificaciones y, por lo tanto, realiza el mismo cálculo.
- 3**, junto con **1 y 2**, asegura que ambas planificaciones resultan en el mismo estado final.

EBD2014_14 - Mg. Mercedes Vitturini

Serializabilidad de Vistas

T ₁	T ₂	T ₃
Read(Q) ;		
Write(Q) .	Write(Q) .	
		Write(Q) .

- Es **serializable en vistas** y es equivalente a la planificación en serie $\langle T_1, T_2, T_3 \rangle$.
- No es serializable en conflictos *¿grafo?*
- Las transacciones T₂ y T₃ realizan escrituras sobre Q sin haberlo leído: *escrituras ciegas*. Las planificaciones serializables en vistas (no en conflictos) admiten escrituras ciegas.

EBD2014_14 - Mg. Mercedes Vitturini

Serializabilidad de Vistas

- Una **planificación** es **serializable en vistas** si es **equivalente en vistas** a alguna planificación en serie.
- La equivalencia en vistas es menos rigurosa que la equivalencia en conflictos.
- Toda planificación serializable en conflictos también es serializable en vistas
Serializabilidad en conflictos \Rightarrow **Serializabilidad en vistas**
- Sin embargo, no es cierto que toda planificación serializable en vistas también lo es en conflictos.
Serializabilidad en vistas $\not\Rightarrow$ **Serializabilidad en conflictos**

EBD2014_14 - Mg. Mercedes Vitturini



Serializabilidad y Equivalencias

- La siguiente planificación produce una salida equivalente a la planificación en serie $\langle T_1, T_3 \rangle$. Sin embargo, *no es equivalente en cuanto a vistas* (ie ni en cuanto a conflictos)

T ₁	T ₃
read(A)	
A := A - 50	
write(A)	
	read(B)
	B := B - 10
	write(B)
read(B)	
B := B + 50	
write(B)	
	read(A)
	A := A + 10
	write(A)

EBD2014_14 - Mg. Mercedes Vitturini

Vistas vs. Conflictos

- Los módulos para testear serializabilidad en vistas tienen un costo exponencial relativo al tamaño del grafo de precedencia (*ver anexo*).
- El problema de **chequear si una planificación es serializable en vistas** cae en la clase de problemas **NP-completos**. Así la existencia de un algoritmo eficiente es extremadamente poco probable.

EBD2014_14 - Mg. Mercedes Vitturini

Implementación de Aislamiento

Vamos a estudiar alternativas en **protocolos de control de concurrencia** como mecanismos para **garantizar aislamiento**:

- Proporcionar concurrencia.
- Generando únicamente planificaciones serializables.

RECORDAMOS

- Los entornos concurrentes proveen las ventajas:
 - **Mejor la utilización de disco y procesador**, aumentando la productividad (*throughput*).
 - **Mejoran el tiempos de respuesta**.

EBD2014_14 - Mg. Mercedes Vitturini

Control de Concurrency

- Un **protocolo de control de concurrencia** garantiza que se generen planificaciones *concurrentes*, pero asegurando *serializabilidad*.
 - Los protocolos que vamos a estudiar **no examinan el grafo de precedencia**.
 - **Imponen reglas que evitan planificaciones no serializables (pesimistas)**.
 - Los distintos protocolos varían en cuanto al paradigma, al nivel de concurrencia y el overhead.
- Los **tests de serializabilidad** ayudan a entender por qué un protocolo es correcto.

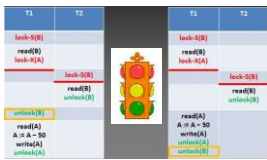
EBD2014_14 - Mg. Mercedes Vitturini

Control de concurrencia

- Los protocolos de control de concurrencia sirven para garantizar *aislamiento*.
- Existen dos estrategias básicas para control de concurrencia:
 - **Basadas en bloqueo** de recursos.
 - **Basadas en estampillas de tiempo** de las transacciones.
- ➔ Distintas implementaciones usan una u otra estrategia o una combinación de ellas.

EBD2014_14 - Mg. Mercedes Vitturini

Mecanismos de Control de Concurrency: Protocolos Basados en Bloqueos



Estrategia: bloqueos de recursos

Protocolos Basados en Bloqueos (PBB)

- Un **bloqueo (Lock)** es un mecanismo para controlar el acceso concurrente a un recurso (item de dato).

Modos de bloqueo:

Compartido (lock-S): si una transacción T solicita y obtiene un bloqueo compartido sobre un dato Q entonces T puede leer el dato pero no escribir Q.

Exclusivo (lock-X): si una transacción T solicita y obtiene un bloqueo exclusivo sobre un dato Q entonces T puede leer y escribir Q.

- Las transacciones envían sus pedidos de bloqueo al *Gestor de Control de Concurrency*.

EBD2014_14 - Mg. Mercedes Vitturini

Gestor de Control de Concurrency

1. Una transacción que requiere un bloqueo (compartido o exclusivo) sobre un ítem de dato Q lo solicita al gestor.
2. El **gestor de control de concurrencia se lo concederá únicamente si su solicitud es compatible** con los bloqueos ya asignados a otras transacciones sobre el mismo dato Q. Caso contrario T deberá esperar.

Tabla de compatibilidad de bloqueos:

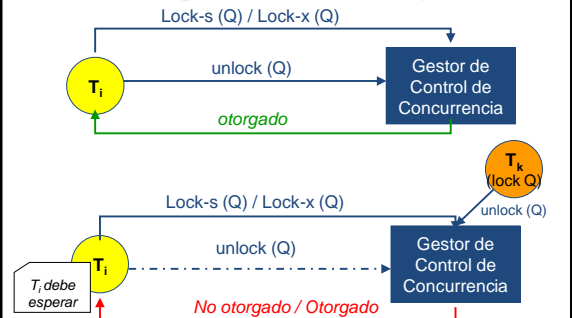
Compatibilidad	Lock-S	Lock-X
Lock-S	Si	No
Lock-X	No	No

Bloqueos previos

Pedido de bloqueo sobre Q

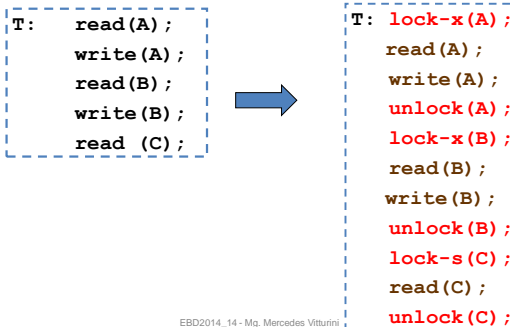
EBD2014_14 - Mg. Mercedes Vitturini

Otorgamiento de bloqueos



EBD2014_14 - Mg. Mercedes Vitturini

Protocolo Basado en Bloqueos (PBB)



EBD2014_14 - Mg. Mercedes Vitturini

Protocolos – Propiedades

- ⇒ **SERIALIZABILIDAD** ¿el protocolo asegura generar sólo planificaciones serializables?
- ⇒ **DEADLOCK** ¿el protocolo genera situaciones en donde un grupo de transacciones no avanza esperando infinitamente recursos que tiene otra?
- ⇒ **INANICIÓN** ¿el protocolo genera situaciones donde una transacción espera infinitamente por algún recurso?
- ⇒ **RETROCESO EN CASCADA** ¿genera situaciones en donde una transacción que aborta arrastra a otra/s transacción/es?
- ⇒ **PLANIFICACIONES NO RECUPERABLES** ¿genera situaciones en donde una transacción que aborta arrastraría a otra/s transacciones cometidas?

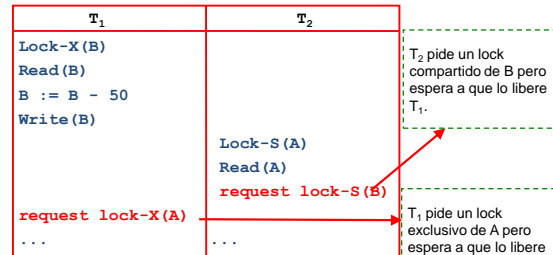
EBD2014_14 - Mg. Mercedes Vitturini

Protocolo Basado en Bloqueos

- **Problema #1:** El mecanismo de solicitud de bloqueos a libertad no garantiza únicamente planificaciones serializables en conflictos.

	T ₁	T ₂
1	Lock-x (A)	
2	Read (A)	
3	A = A - 50	
4	Write (A)	
5	Unlock (A)	
6		Lock-S (B)
7		Read (B)
8		tt = B*0.1
9		Unlock (B)
10		Lock-X (A)
11		Read (A)
12		A = A + tt
13		Write (A)
14		Unlock (A)
15	Lock-x (B)	
16	Read (B)	
17	B = B + 50	
18	Write (B)	
19	Unlock (B)	

PBB – Deadlock



Problema #2: no asegura planificaciones libres de deadlock

El sistema deberá **retroceder** alguna de las transacciones

EBD2014_14 - Mg. Mercedes Vitturini

Problemas del PBB

- ⊗ El PBB no está libre de *deadlock*.
 - **Solución:** usar alguna política de administración de deadlock (detección o prevención).
- ⊗ PBB también puede causar problemas de *inanición* (*starvation*), esto es, transacciones que nunca alcanzan a disponer de los recursos que necesitan, porque son otorgados antes a otras transacciones.
 - **Solución:** establecer políticas de otorgamiento de bloqueos con prioridades.
- ⊗ PBB genera planificaciones no serializables, esto es significa, estados inconsistentes de la BD.
 - No es un protocolo de control de concurrencia que garantice serilizabilidad.

EBD2014_14 - Mg. Mercedes Vitturini

Protocolo de Bloqueo de 2 Fases PB2F

Solución: requerir que cada transacción haga sus solicitudes de bloqueos y desbloqueos en dos fases:

- **Fase de Crecimiento:** Una transacción puede obtener nuevos bloqueos pero *no puede liberar ningún bloqueo*.
- **Fase de Encogimiento:** Una transacción puede liberar bloqueos pero *no puede obtener ningún bloqueo nuevo*.

- El PB2F garantiza planificaciones **serializables en conflictos**.

EBD2014_14 - Mg. Mercedes Vitturini

Ejemplo

$T_1 = \text{read}(A); \text{write}(A);$
 $\text{read}(B); \text{write}(B);$

$T_2 = \text{read}(D);$
 $\text{read}(A); \text{write}(A);$
 $\text{write}(D);$

Serie equivalente:
 $\langle T_1, T_2 \rangle$

	T1	T2
FC	lock-x (A)	
	read (A)	
	write (A)	
		lock-x (D)
		read (D)
FD	lock-x (B)	
	unlock (A)	
	read (B)	
		lock-x (A)
		read (A)
		write (A)
		unlock (A)
	write (B)	
	unlock (B)	
		write (D)
	unlock (D)	

EBD2014_14 - Mg. Mercedes Vitturini

Protocolo de Bloqueo de 2 Fases

Características de PB2F

- El PB2F asegura serializabilidad en conflictos.
- La serie equivalente queda definida según cada *punto de lock* (último lock obtenido).
- No está libre de deadlock.

EBD2014_14 - Mg. Mercedes Vitturini

Definiciones

- Si una transacción falla durante su ejecución, para asegurar *atomicidad*, se deben deshacer sus efectos.
- Así, en un sistema concurrente, si existe una transacción T_j que depende de T_i (T_j leyó un dato escrito por T_i) T_j también deberá ser retrocedida.

Planificación no recuperable: se dice que una planificación es no recuperable si existe T_j que depende de T_i , T_j comete antes que T_i termine y posteriormente ocurre una falla de T_i

Retrocesos en cascada: se dice que una planificación tiene retrocesos en cascada si existe T_k que depende de T_j , T_j que ... que depende de T_i . Si falla T_i , deberán retroceder en cascada T_j , T_j , T_k

Retroceso en Cascada

$T_1 = \text{read}(A); \text{write}(A);$
 $\text{read}(B); \text{write}(B);$

$T_2 = \text{read}(A);$
 $\text{read}(B); C=A+B; \text{write}(C)$

⇒ La falla de T_1 arrastra a T_2

	T1	T2
1	Lock-x (A)	
2	read (A)	
3	write (A)	
4	Lock-x (B)	
5	unlock (A)	
6		Lock-x (A)
7		read (A)
8	read (B)	
9	falla T1	

EBD2014_14 - Mg. Mercedes Vitturini

Retrocesos en Cascada

T1	T2	T3	T4
read (A)			
write (A)			
	read (A)		
	write (B)		
		read(B)	
			read(A)
			write (A)
falla T1			

T2, T3 y T4 Retroceden en cascada

Mercedes Vitturini

Planificación no recuperable

$T_1 = \text{read}(A); \text{write}(A);$
 $\text{read}(B); \text{write}(B);$

$T_2 = \text{read}(A);$
 $\text{read}(B); C=A+B; \text{write}(C)$

⇒ La falla de T_1 debería arrastrar a T_2 que cometi6.

	T1	T2
1		Lock-x (A)
2		read (A)
3		Lock-s (B)
4		read (B)
5		Lock-x (C)
6		Unlock (A)
7		Unlock (B)
8	Lock-x (A)	
9	read (A)	
	write (A)	
	Lock-x (B)	
10	unlock (A)	
11	read (B)	
12	write (B)	
13	unlock (B)	
14	Commit T1	
15		Falla T2

EBD2014_14 - Mg. Mercedes Vitturini

Problemas del PB2F

- El PB2F tiene las siguientes **desventajas**:
 - ⊗ Posibilidad de que alguna planificación caiga en **deadlock**.
 - ⊗ Puede generar **retrocesos en cascada** (*cascading rollback*), esto es, ante la falla de una transacción, se genera que fallen otras transacciones.
 - ⊗ Puede generar **planificaciones no recuperables**, como resultado de un retroceso en cascada.

EBD2014_14 - Mg. Mercedes Vitturini

PB2F Estricto

Variantes a PB2F: **Protocolo de Bloqueo de 2 Fases Estricto**. Similar al PB2F, donde

- Cada transacción debe conservar todos los bloqueos exclusivos que solicitó hasta alcanzar el estado cometida.
- De esta manera se garantiza:
 - Que el dato “escrito” por una transacción T_i no pueda ser “leído” (ni escrito) por otra T_j hasta que T_i cometa.
 - Solución a los problemas de retrocesos en cascada y planificaciones recuperables.

EBD2014_14 - Mg. Mercedes Vitturini

Ejemplo

Dadas las transacciones:

- $T_1 = \text{read}(A); \text{read}(B); \text{write}(A); \text{write}(B);$
- $T_2 = \text{read}(D); \text{read}(A); \text{write}(A);$
- Esta planificación es **equivalente a la serie $\langle T_1, T_2 \rangle$**

T1	T2
lock-x (A)	
read (A)	
lock-x (B)	
read (B)	
	lock-s (D)
	read (D)
write (A)	
write (B)	
unlock (A,B)	
	lock-x (A)
	unlock (D)
	read (A)
	write (A)
	unlock (A)

EBD2014_14 - Mg. Mercedes Vitturini

Variante: PB2F Riguroso

- El **Protocolo de Bloqueo de 2 Fases Riguroso** exige que toda transacción mantenga **todos sus bloqueos** (sean exclusivos o compartidos) hasta que la transacción alcance el estado cometida.
- El orden de serializabilidad es el orden en que se comprometen las transacciones.

EBD2014_14 - Mg. Mercedes Vitturini

🔍 Para analizar

- Los protocolos de PB2F Estricto y PB2F Rigurosos ¿solucionan el problema de deadlock?
- ¿Retrocesos en cascada?
- ¿Planificaciones no recuperables?
- ¿Se le ocurre algún mecanismo de bloqueo para evitar deadlock?

EBD2014_14 - Mg. Mercedes Vitturini

Variante: PB2F Refinado

- Supongamos una transacción T_i :
 $T_i = \text{read}(A_1); \text{read}(A_2); \dots \text{read}(A_n); \text{write}(A_1);$
- El **PB2F Refinado** permite **conversiones de bloqueos**:
 - De **Lock-S a Lock-X**.
 - **Upgrade**: de modo compartido a exclusivo.
 - Se incluyen en la **fase de crecimiento**.
 - De **Lock-X a Lock-S**.
 - **Downgrade**: de modo exclusivo a compartido.
 - Se incluyen en la **fase de decrecimiento**.

EBD2014_14 - Mg. Mercedes Vitturini

PB2F Refinado: Ejemplo

Dadas las transacciones:

- $T_1 = \text{read}(A); \text{read}(B); \text{write}(A); \text{write}(B);$
- $T_2 = \text{read}(D); \text{read}(A); \text{write}(A); \text{write}(D);$

T1	T2
lock-s (A)	
read (A)	
lock-s (B)	
read (B)	
	lock-s (D)
	read (D)
upgrade (A)	
write (A)	
upgrade (B)	
write (B)	
unlock (A, B)	
	lock-s (A)
	read (A)
	upgrade (A)
	write (A)
	upgrade (D)
	write (D)
	unlock (A, D)

EBD2014_14 - Mg. Mercedes Vitturini

Implementación de bloqueos transparente

PB2F Refinado+ Riguroso

- Cuando una transacción T realiza una operación **Read(Q)** se ejecuta:
 - Lock-S(Q); Read(Q)
- Cuando una transacción T realiza una operación **Write(Q)** se ejecuta:
 - Si T tiene un acceso compartido entonces ejecuta:
 - Upgrade(Q); Write(Q)
 - De lo contrario, T ejecuta:
 - Lock-X(Q); Write(Q)
- Todos los bloqueos que tenga una transacción los conserva hasta que dicha transacción se comprometa o aborte.

EBD2014_14 - Mg. Mercedes Vitturini

Temas de la clase de hoy

- Serializabilidad en Vistas
- Control de concurrencia:
 - Protocolos de bloqueos de dos fases: 2 Fases, 2 Fases Estricto, 2 Fases Riguroso, 2 Fases Refinado
- **Bibliografía**
 - Database system Concepts – A. Silberschatz. Capítulos 14 y 15 (ed. 2010).
- **Otras lecturas sugeridas**
 - “Introducción a las Bases de Datos” – Jeffrey Ullman
 - “Databases and Transaction Processing” - Philip Lewis.

EBD2014_14 - Mg. Mercedes Vitturini

Test de Serializabilidad de Vistas

- Para determinar la serializabilidad de vistas en una planificación, se construye un **grafo de precedencia etiquetado**.
- Sea S una planificación que involucra a las transacciones $\{T_1, T_2, \dots, T_n\}$.
- Se agregan dos transacciones **ficticias** T_b y T_f .
 - T_b (transacción inicial) escribe cada dato leído por las transacciones de S.
 - T_f (transacción final) lee cada dato escrito por las transacciones de S.

EBD2014_14 - Mg. Mercedes Vitturini

Pruebas de Serializabilidad de Vistas

1. Se añade una arista $T_i \rightarrow T_j$ si la transacción T_j lee el dato Q escrito por T_i .
2. Por cada dato Q tal que T_j lee el valor de Q escrito por T_i y T_k ejecuta Write(Q) tal que $T_k \neq T_b$ se hace lo siguiente:
 - a. Si $T_i = T_b$ y $T_j \neq T_f$ entonces se inserta en el grafo la arista $T_j^0 \rightarrow T_k$.
 - b. Si $T_i \neq T_b$ y $T_j = T_f$ entonces se inserta en el grafo la arista $T_k^0 \rightarrow T_f$.
 - c. Si $T_i \neq T_b$ y $T_j \neq T_f$ entonces se insertan en el grafo las aristas $T_k^p \rightarrow T_i$ y $T_j^p \rightarrow T_k$, donde p es un número de etiqueta no usada en el grafo etiquetado.

EBD2014_14 - Mg. Mercedes Vitturini

Pruebas de Serializabilidad de Vistas

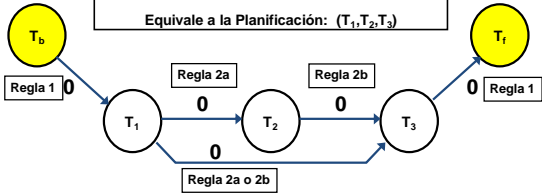
- Las reglas a y b son casos especiales que resultan de que T_b y T_f representan la primera y última transacción respectivamente.
- La regla c dice que si T_i escribe un dato que lee T_j y T_k escribe el mismo dato entonces T_k debe aparecer antes de T_j , o bien después de T_j .
- Al aplicar la regla c no se requiere que T_k esté antes de T_i y después de T_j . Se exige que preceda a T_i , o bien suceda a T_j .
- **Si el grafo de precedencia no contiene ciclos entonces la planificación es serializable en vistas.**
- No obstante, la aparición de un ciclo (cuando existen etiquetas distintas de 0) no implica que la planificación no sea serializable en vistas.

EBD2014_14 - Mg. Mercedes Vitturini

Planificación 1: serializable en vistas

T ₁	T ₂	T ₃
Read(Q) ;	Write(Q) .	
Write(Q) .		Write(Q) .

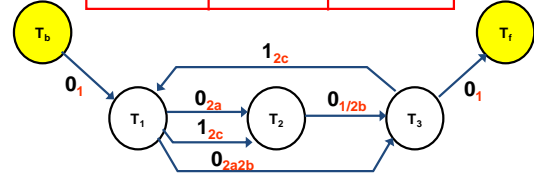
Equivalente a la Planificación: (T₁,T₂,T₃)



EBD2014_14 - Mg. Mercedes Vitturini

Planificación 2: serializable en vistas

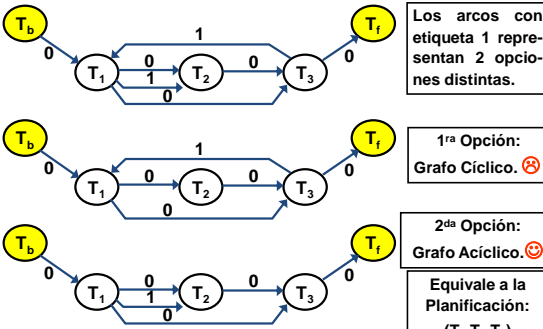
T ₁	T ₂	T ₃
Read(Q) ;	Write(Q) .	
Write(Q) .		Read(Q) ;
		Write(Q) .



Los subíndices en las etiquetas indican las reglas que generan el arco.

EBD2014_14 - Mg. Mercedes Vitturini

Planificación 3: serializable en vistas



Los arcos con etiqueta 1 representan 2 opciones distintas.

1^{ra} Opción:
Grafo Cíclico. ☹️

2^{da} Opción:
Grafo Acíclico. 😊

Equivalente a la Planificación:
(T₁,T₂,T₃)

EBD2014_14 - Mg. Mercedes Vitturini