**Seriazibilidad en conflictos**

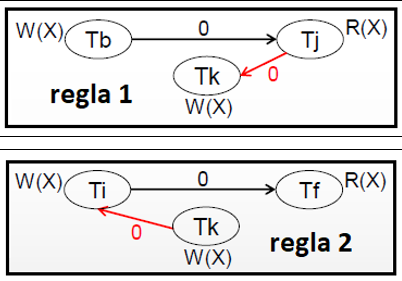
Dada una tabla de transacciones:

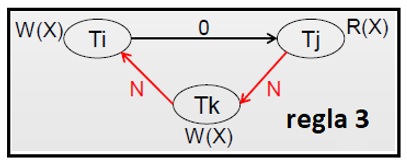
1. Hacemos un nodo por cada transacción
2. Un arco de Ti a Tj si: Ti ejecuta una instrucción I(X) antes que Tj ejecute una instrucción J(X) (sobre el mismo dato X) y alguna de las instrucciones I o J write(X)

**Seriazibilidad en vistas**

Grado de precedencia:

1. Agregamos 2 nodos Tb y Tf, Tb escribe todos los datos al comienzo y Tf lee todos datos al final.
2. Agregamos los nodos de cada transacción
3. Agregamos arcos con etiqueta “0” donde Ti escriba un dato y luego Tj lo lea
4. Luego en el siguiente orden aplicamos las siguientes reglas:





(Para la regla 3 para se agrega un N nuevo cada vez que se aplique la regla)

**5.Eliminamos un arco de cada arco con N>0, si igualmente al final sigue habiendo ciclos no es serializable.**

Protocolo de Validación

Tj valida si para toda transacción Ti tal que V(Ti) < V(Tj) si:

* F(i) < S(J)
* (WS(Ti) ∩ RS(Tj)= Ø) y Start(Tj) < Finish(Ti) < Validation(Tj)

**Protocolo 2 fases**

**Para cada transacción**

Antes de escribir W(X) o leer R(X) un dato X se debe solicitar y obtener el bloqueo

respetar 2 fases: **Fase de crecimiento** y **fase de decrecimiento**

1. Solicitar bloqueo mínimo permitido para poder ejecutar la instrucción
2. Solicitar los bloqueos lo más tarde posible en la planificación
3. Liberar los bloqueos lo antes posible (siempre respetando 2 fases)
4. **Factos:  
   Una vez unlockeado un dato no se puede usar para ninguna otra cosa, ni leer, ni escribir.**
5. **Upgrade y Downgrade se utilizan en la misma transición**

**Protocolo de Arbol**

1. El único **bloqueo** permitido es el **exclusivo**: L-X(A)
2. **El primer bloqueo de cada transacción puede ser sobre cualquier dato.**
3. **Después de su primer bloqueo una transacción puede bloquear un dato Q si tiene bloqueado al padre de Q en el árbol**
   1. El 1er dato R que bloquea una transacción restringe los datos que podrá bloquear a lo largo de su ejecución. Solo podrá bloquear a los datos descendientes en el sub-árbol que tenga el dato R como raíz, avanzando de padres a hijos.
4. Cuando **una transacción desbloquea un dato Q (Un(Q)) no puede volver a bloquear a Q.**
5. **Los datos pueden desbloquearse en cualquier momento**, es decir**, puede no respetar 2 fases.**

**Protocolo de estampillas**

Si Ti ejecuta Read(Q) y ts(Ti) < W-ts(Q) entonces Ti retrocede

Si Ti ejecuta Write(Q):

* Si ts(Ti) < R-ts(Q) entonces Ti retrocede
* Si ts(Ti) < W-ts(Q) entonces Ti retrocede pero con regla Thomas se omite

Si instrucción se realiza con éxito y se actualiza la estampilla correspondiente del dato Q según la instrucción:

* + Read(Q) => R-ts(Q) ← Mayor ( R-ts(Q), ts(Ti) )
  + Write(Q) => W-ts(Q) ← Mayor ( W-ts(Q), ts(Ti) )

**Protocolo multiversion**

– Si Ti ejecuta Read(Q) entonces R-ts(Qk) se le asigna Mayor ( ts(Ti), R-ts(Qk) )

– Si Ti ejecuta Write(Q)

* Si ts(Ti) < R-ts(Qk) entonces Ti retrocede (la escritura “llego tarde”)
* Si ts(Ti) = W-ts(Qk) entonces solo se actualiza el valor de Qk (es la versión que creo Ti)
* En otro caso se crea una nueva versión Qi con R-ts(Qi)= W-ts(Qi)= ts(Ti)

Seguir las reglas y recordar que **cuando ejecutamos un read/write** hay que ver con que versión nos quedamos, **nos quedamos con la que mayor W-ts tenga que sea menor que W-ts(Ti).**

* El read no cambia la versión QK
* El write si es que no retrocede entonces actualiza QK

Recordar: El read no modifica el W-ts

En protocolo estampilla y multiversion, si **retrocede** una transaccion, chequeamos el **retroceso en cascada**, **vemos que escribe la transacción que retrocede y si alguien lo lee entonces también retrocede**, esto se repite.