



Transacciones



¿Qué es una transacción?

Es una unidad lógica y atómica de procesamiento de la base que datos que puede incluir una o más operaciones.

Transacción T1

```
INSERT INTO T1 VALUES (1, 'Juan')  
DELETE FROM T2 WHERE id=1
```

Transacción T2

```
UPDATE T1 SET Precio=0 where id=1
```



Control de una transacción

El SGBD debe controlar las transacciones para no alterar la consistencia de la base de datos, ya que las transacciones puede utilizar los mismos recursos.

Saldo Inicial: 0 pesos

T1: Deposita Dinero

```
INSERT INTO MOVIMIENTO  
(NroCuenta, TipoMov, Monto)  
VALUES (1, 'D', 10000)
```

```
UPDATE SALDO  
SET MSaldo=MSaldo + 10000  
Where NroCuenta=1
```

T2: Extrae Dinero

```
INSERT INTO MOVIMIENTO  
(NroCuenta, TipoMov, Monto)  
VALUES (1, 'E', 10000)
```

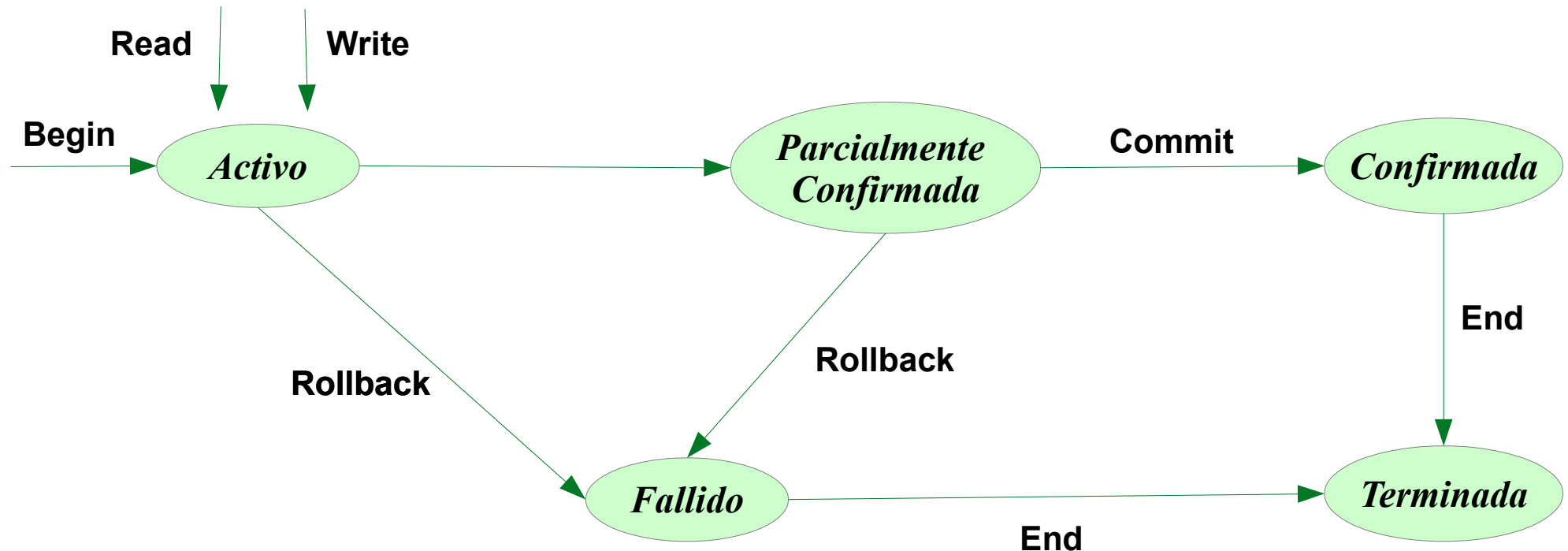
```
UPDATE SALDO  
SET MSaldo=MSaldo - 10000  
Where NroCuenta=1
```

T3: Consulta el Dinero

```
Select MSaldo  
From SALDO  
Where NroCuenta=1
```



Estados de una transacción



- Begin Transaction
- End Transaction
- Commit Transaction
- Rollback Transaction
- Read(x) / Write(x)

Iniciar una transacción




Transacción T1

```
BEGIN TRANSACTION
```

```
INSERT INTO T1 VALUES (1, 'Juan')
```

```
DELETE FROM T2 WHERE id=1
```

```
COMMIT TRANSACTION
```



**Finalizó sin errores y
se confirmó**


Transacción T2

```
BEGIN TRANSACTION
```

```
INSERT INTO T1 VALUES (1, 'Ana')
```

```
DELETE FROM T2 WHERE id=1
```

```
ROLLBACK TRANSACTION
```



**Finalizó con errores y
se deshizo**

Propiedades de una transacción



ACID (Atomicidad – Consistencia – Aislamiento – Durabilidad)

Atomicidad: una transacción es la unidad atómica de procesamiento, se realiza por completo o bien se aborta.

Conservación de consistencia: una transacción conserva la consistencia si su ejecución completa lleva la base de datos de un estado consistente a otro. Una transacción no puede violar la integridad de la base de datos.

Aislamiento: una transacción debería parecer que se está ejecutando en forma aislada de las demás transacciones. Es decir, la ejecución de una transacción no debe interferir con la ejecución de otra transacción, debe dar el mismo resultado que si se ejecuta en serie. Ningún valor será revelado al resto de las transacciones hasta que la transacción haya sido finalizada correctamente.

Durabilidad: los cambios aplicados en la base de datos por una transacción confirmada deben perdurar en la base de datos. Estos cambios no deben perderse por un fallo posterior.



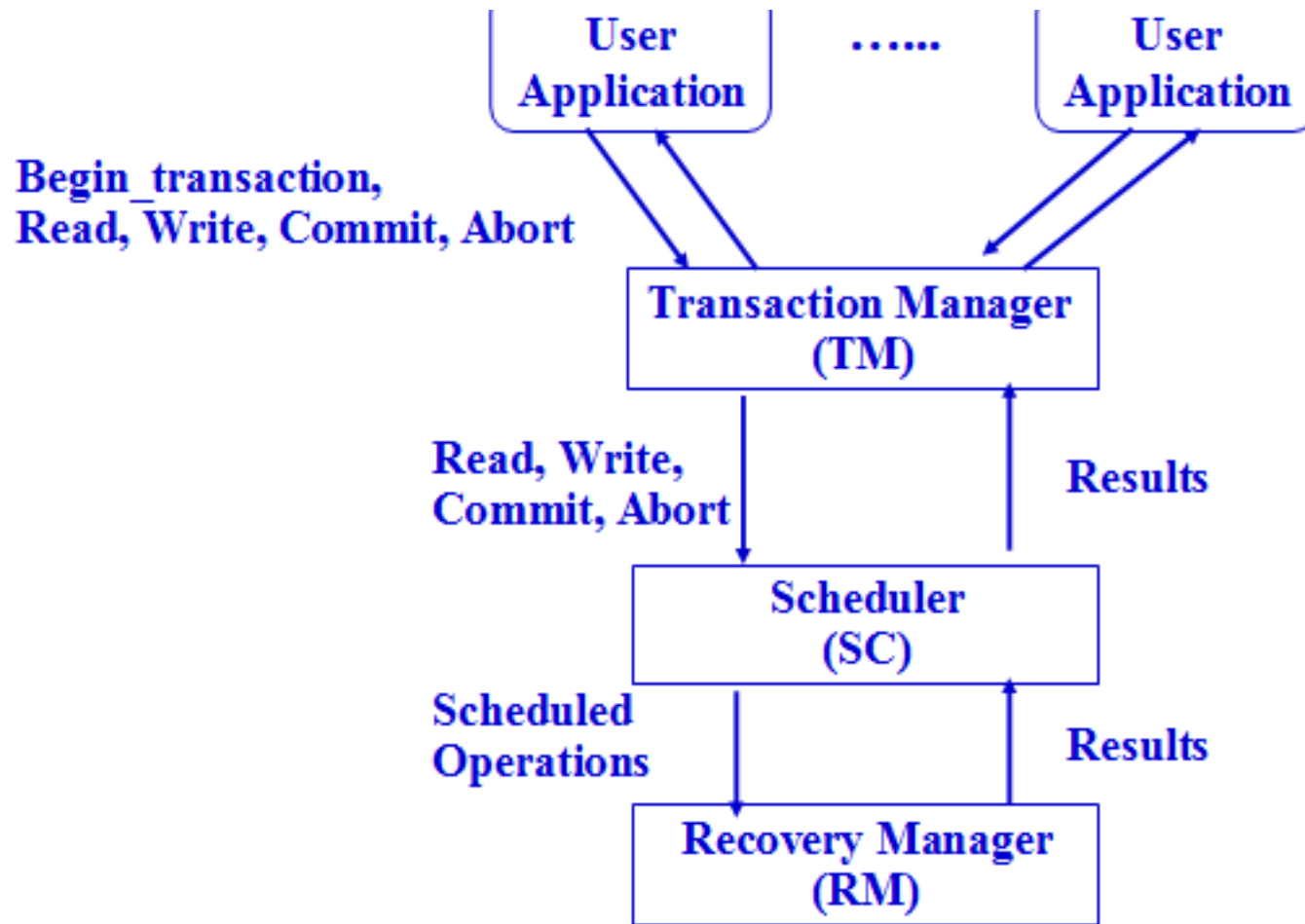
Funciones de del SGBD

El SGBD es responsable de realizar:

- **Control de Concurrencia (Transaction Manager)**
- **Control de recuperación (Recovery Manager)**



Funciones de del SGBD





Control de concurrencia

Dado que tendremos múltiples usuarios accediendo a la misma información y recursos, es necesario considerar la concurrencia para tener consistencia en los datos leídos y escritos.

Problemas de la concurrencia



Problema de la actualización Perdida (Lost Update):

T1	T2
Leer (R1,x)	
X=x-N	
	Leer (R1,x)
	X=x+M
Escribir (R1,x)	
Commit;	
	Escribir (x)
	Commit;

X=10
N=1
M=2



T2 no debería haber podido leer el valor de X si T1 lo estaba modificando

Problemas de la concurrencia



Problema de la actualización sucia (Dirty Reads):

T1	T2
Leer (R1,x)	
X=x-N	
Escribir (R1,x)	
	Leer (R1,x)
Rollback;	
	X=x+M
	Escribir (R1,x)

X=10
N=1
M=2



T2 no debería haber leído el valor de X si T1 aún no había confirmado su valor

Problemas de la concurrencia



Problema del resumen incorrecto (NonRepeatable Reads):

T1	T2
Leer(R1,x)	
	Leer(R1, x)
	x=x-100
	Escribir(R1,x)
	Leer(R2, y)
	y=y+1000
	Escribir(R2,y)
	Commit;
Leer(R1,x)	
Leer(R2,y)	
Leer(R3,z)	
sum=x+y+z	
Commit;	



T1 tomó un valor en X anterior a la actualización y un valor Y posterior.



Seriabilidad de los planes

Si los planes se ejecutaran en forma serial, siempre serían correctos. Es decir, esperar a que un plan termine para comenzar el otro. Pero el problema q esto tiene es que se desaprovecharía tiempo de procesamiento esperando a que otra transacción finalice. Por ello se introduce el concepto de seriabilidad de los planes para identificar qué planes son correctos, a pesar que posean operaciones en forma intercalada y no serial.

- **Planes en serie:** *si las transacciones se ejecutan en forma consecutiva.*
- **Planes no en serie:** *si las operaciones de cada transacción se ejecutan en forma intercalada.*
- **Plan serializable:** *un plan P de n transacciones es serializable si es equivalente a algún plan en serie. Es decir, todo plan serializable es un plan correcto. Es decir, si el plan es equivalente a un plan en serie se considera que es correcto.*



Equivalencia de planes

*Se dice que dos planes son **equivalentes por resultados** si producen el mismo estado al final del plan. Pero esto puede ocasionar que se hagan muchas operaciones diferentes y que el resultado sea el mismo. Por lo cual, no es muy confiable.*

*Por lo tanto, existe el concepto de **equivalentes por conflictos** si el orden de dos operaciones cualquiera en conflicto es el mismo en ambos planes. Se considera en conflicto, si utilizan el mismo elemento, pero pertenecen a diferentes transacciones.*

Conflictos



Se dice que existe un conflicto entre dos transacciones, cuándo utilizan el mismo recurso y al menos, alguna de las dos quiere escribir el elemento.

- *T1: Leer(R1,x) / T2: Escribir(R1,x)*
- *T1: Escribir(R1,x) / T2: Leer(R1,x)*
- *T1: Escribir(R1,x) / T2: Escribir(R1,x)*

Prueba de seriabilidad por Conflictos



El siguiente algoritmo puede comprobar si un plan es serializable por conflictos, es decir, si es equivalente a algún plan en Serie. Para ello se utilizará un grafo de precedencia. Si al finalizar de dibujar ese grafo, encontramos ciclos significará que el plan no es serializable, por lo cual no es correcto.

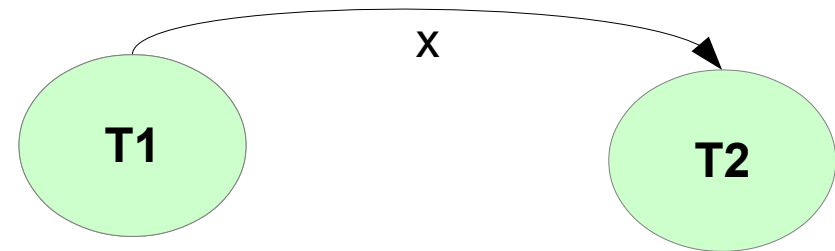
Algoritmo:

- 1. Para cada transacción T , crear un nodo T_i en el grafo de precedencia.*
- 2. LEER/ESCRIBIR: Para cada paso en que una transacción ejecuta un LEER(X) después que otra transacción realizó un ESCRIBIR(X), entonces generar un arco de $T_i \rightarrow T_j$ (en donde i y j es el orden de ejecución de los pasos).*
- 3. ESCRIBIR/LEER: Para cada paso en que una transacción ejecuta un ESCRIBIR(X) y después otra transacción ejecuta una operación LEER(X), generar un arco $T_i \rightarrow T_j$.*
- 4. ESCRIBIR/ESCRIBIR: Para cada paso en que una transacción ejecuta un ESCRIBIR(X) y después otra transacción ejecuta una operación ESCRIBIR(X), generar un arco $T_i \rightarrow T_j$.*

Algoritmo: Ejemplo I



T1	T2
Read(X)	
X=x+1	
Write(X)	
Read(Y)	
Y=Y+2	
Write(Y)	
	Read(X)
	X=x+3
	Write(X)



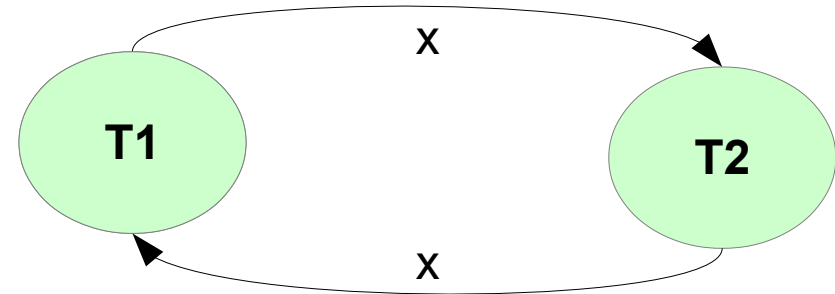
ES SERIALIZABLE

T1 → T2 es un plan en serie equivalente



Algoritmo: Ejemplo II

T1	T2
Read(X)	
X=x+1	
	Read(X)
	X=x+3
Write(X)	
Read(Y)	
	Write(X)
Y=Y+2	
Write(Y)	



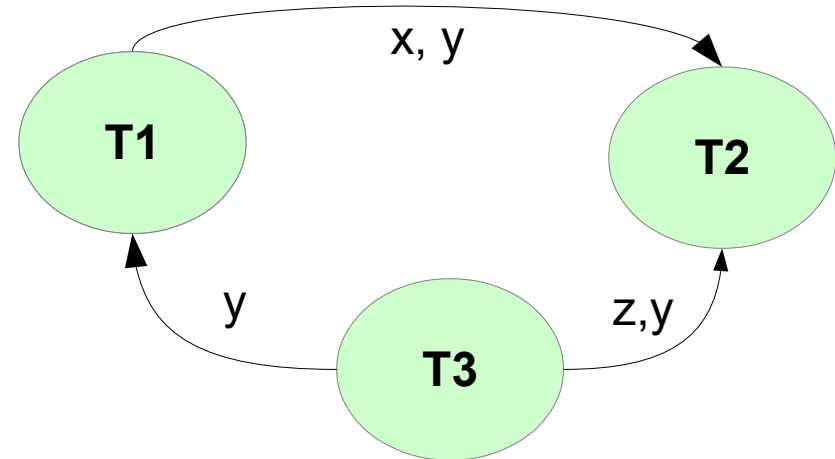
ES NOSERIALIZABLE

No hay un plan en serie equivalente



Algoritmo: Ejemplo III

T1	T2	T3
		Read(Y)
		Read(Z)
Write(X)		
Write(X)		
		Write(Y)
		Write(Z)
	Read(Z)	
Read(Y)		
Write(Y)		
	Read(Y)	
	Write(Y)	
	Read(X)	
	Write(X)	



ES SERIALIZABLE

T3 → T1 → T2 es un plan en serie equivalente

Lockeos



Es un mecanismo que permite que las transacciones puedan ejecutar en forma serializable, aunque no lo sean, basado en controlar el uso de recursos para que las transacciones que necesitan el mismo elemento puedan aguardar a obtenerlo si otra transacción lo tiene en uso.

Los principios del lockeo son:

- *Obtener un lockeo antes de acceder al recurso*
- *Luego de usarlo, liberar el recurso*



Protocolo de Lockeos

Bloqueo Binario: un bloqueo binario puede tener dos estados, bloqueado y desbloqueado (0/1). Las operaciones son `bloquear_elemento(X)` y `desbloquear_elemento(X)`. Esta operación limita a que los bloqueos no puedan ser tomados en un mismo elemento por más de una transacción.

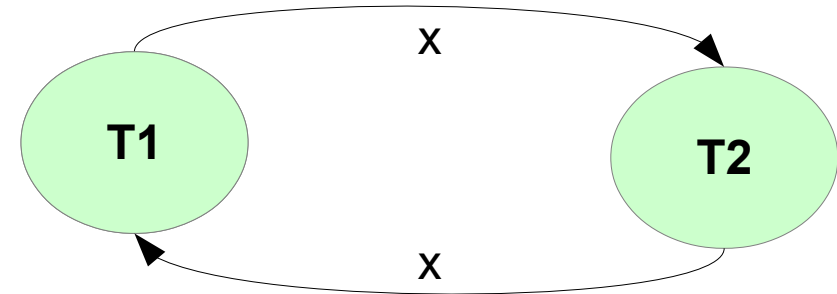
Un bloqueo binario cumple las siguientes reglas:

- Una transacción *T* debe emitir la operación `bloquear_elemento(X)` antes de que se realice cualquier operación `leer_elemento(X)` ó `escribir_elemento(X)` en *T*.
- Una transacción *T* debe emitir una operación `desbloquear_elemento(X)` después de haber completado todas las operaciones `leer_elemento(X)` y `escribir_elemento(X)` en *T*.
- Una transacción *T* no emitirá una operación `bloquear_elemento(X)` si ya posee el bloqueo del elemento *X*.
- Una transacción *T* no emitirá una operación `desbloquear_elemento(X)` a menos que ya posea el bloqueo del elemento *X*.



Lockeos: Ejemplo

T1	T2
Lock(X)	
Read(X)	
X=x+1	
	Lock(X)
	Read(X)
	X=x+3
Write(X)	
Unlock (X)	
	Write(X)
	Unlock(x)
Lock(Y)	
Read(Y)	
Y=Y+2	
Write(Y)	
Unlock(Y)	





Protocolo de Lockeos

Bloqueo Compartido/exclusivo: El modelo binario es muy restrictivo. Este modelo ofrece operaciones en donde se pueda convertir un nivel de bloqueo a uno más restrictivo. Las operaciones que se realizan son `bloquear_lectura(X)`, `bloquea_escritura(X)` y `desbloquear(X)`.

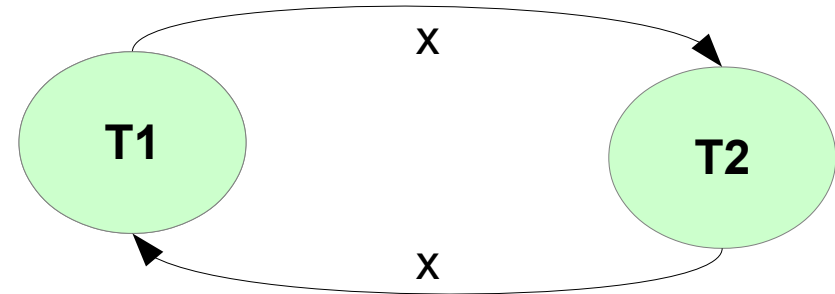
Las reglas que cumple este bloqueo son:

- Una transacción *T* debe emitir la operación `bloquear_lectura(X)` o `bloquear_escritura(X)` antes de que se realice cualquier operación `leer_elemento(X)` de *T*.
- Una transacción *T* debe emitir la operación `bloquear_escritura(X)` antes de que se realice cualquier operación `escribir_Elemento(X)` de *T*.
- Una transacción *T* debe emitir la operación `desbloquear(X)` una vez que se hayan completado todas las operaciones `leer_Elemento(X)` y `escribir_Elemento(X)` de *T*.
- Una transacción *T* no emitirá una operación `bloquear_lectura(X)` si ya posee un bloqueo de lectura (compartido) o escritura para el elemento *X*. Esta regla puede emitir excepciones.
- Una transacción *T* no emitirá una operación `bloquear_escritura(X)` si ya posee un bloqueo de lectura o escritura sobre el elemento *X*.
- Una transacción *T* no emitirá una operación `desbloquear(X)` a menos que ya posea un bloqueo de lectura o escritura para el elemento *X*.



Lockeos: Ejemplo

T1	T2
Lock-X(X)	
Read(X)	
X=x+1	
	Lock-X(X)
	Read(X)
	X=x+3
Write(X)	
Unlock (X)	
	Write(X)
	Unlock(x)
Lock-S(Y)	
Read(Y)	
Unlock(Y)	





Compatibilidad de Lockeos

Tipos de Lockeos:

- *Lock-S: Lockeo Shared o de Sólo Lectura*
- *Lock-X: Lockeo Exclusive ó de Escritura*

Reglas:

- *Si una transacción necesita leer un objeto, primero tendrá que solicitar un lockeo de Shared.*
- *Si una transacción necesita modificar un objeto, primero tendrá que solicitar un lockeo de X exclusive.*
- *Sólo se podrá tener un solo lockeo X por objeto, pero múltiples S por objeto.*

	Shared	Exclusive
Shared	S	N
Exclusive	N	N



Protocolo de 2 fases

Se dice que sigue el protocolo de dos fases si todas las operaciones de bloqueo (bloquear_lectura, bloquear_escritura) preceden a la primera operación de desbloqueo de la transacción. Es decir, la transacción no puede realizar un bloqueo una vez que ha liberado algún ítem que había bloqueado.

Se podría demostrar que si las transacciones siguen el protocolo de dos fases siempre serán serializables.

El protocolo tiene dos fases:

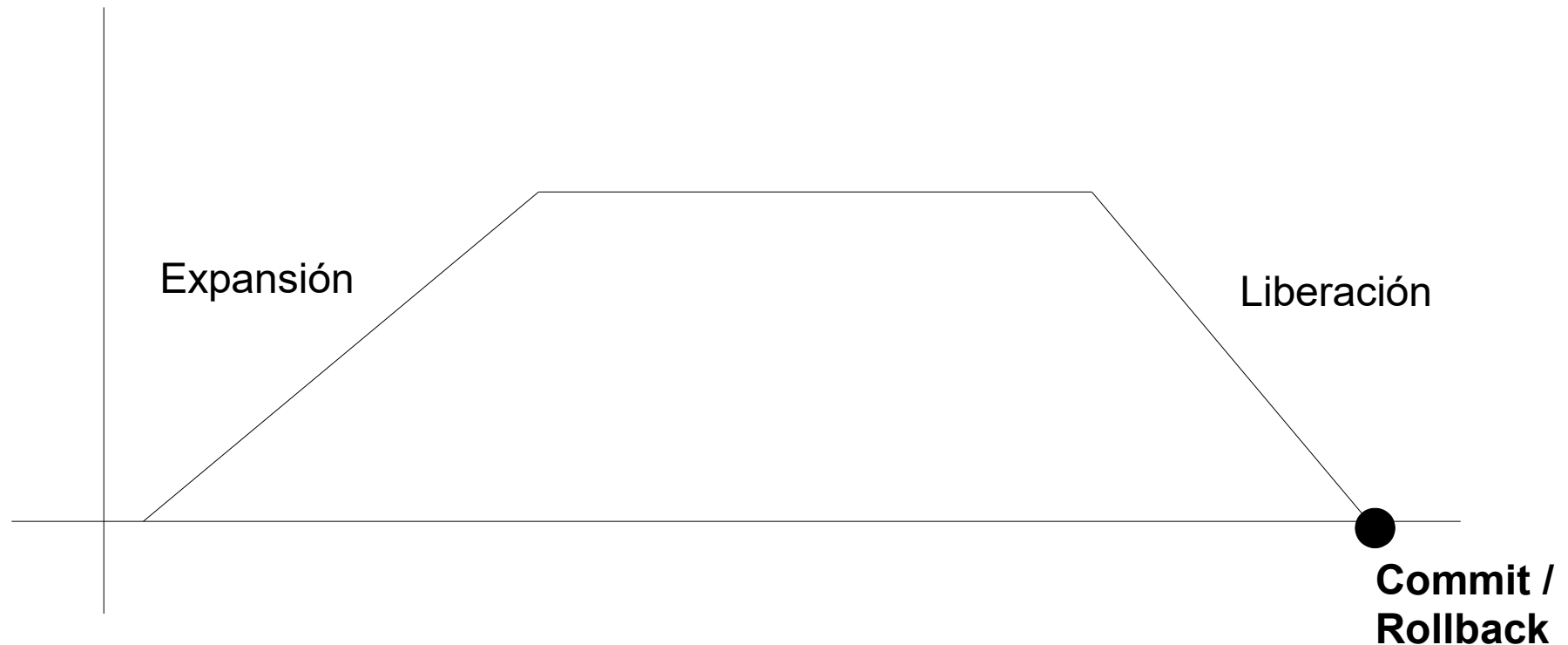
Fase de expansión: *se adquieren los bloqueos en los elementos, pero no se liberan. Aquí también se pueden convertir los bloqueos, de lectura a escritura.*

Fase de contracción: *se pueden liberar bloqueos, pero no se pueden adquirir nuevos. Aquí también se pueden convertir los bloqueos de degradación, de escritura a lectura.*

Protocolo de 2 fases: Simple



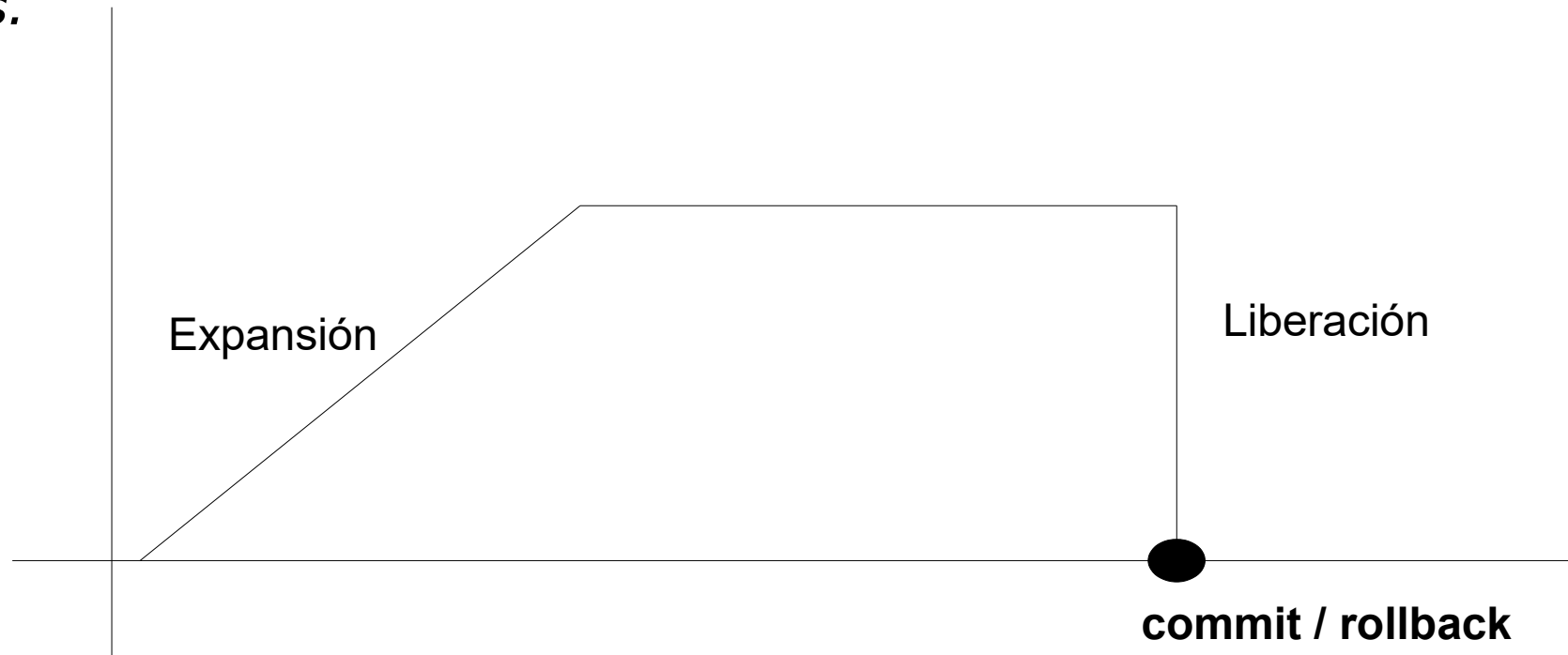
Es aquel en donde se adquieren los lockeos en la fase de expansión y luego se van liberando en la fase de contracción. El problema que aborda son los aborts en cascadas, ya que liberó los lockeos antes de commitear.



Protocolo de 2 fases: Estricto



Es aquel que también adquiere los lockeos en la fase de expansión pero una transacción no libera ninguno de sus bloqueos exclusivos hasta el commit o rollback. Esto previene uno de los problemas del protocolo de 2 fases, en donde una transacción al realizar rollback volvería atrás los valores y si ya había liberado el lockeo otra transacción pudo haber tomado un valor incorrecto. Este es el protocolo utilizado por la mayor parte de los motores de bases de datos.





Lockeos en SGBD

Los SGBD realizan automáticamente los tipos de lockeos. Generalmente, estos son los tipos de lockeos que establecen según la operación realizada y según el nivel de aislamiento establecido:

- *select → no realiza lockeos*
- *insert/update/delete → xlock /row exclusive*
- *select . . . for update → slock /row share*
- *commit / rollback → libera todos los lockeos*



¿Siempre se establecen este tipo de lockeos? ¿De qué depende?



Niveles de Aislamiento

Pueden ocurrir 3 tipos de violaciones en los datos:

- *Lectura sucia (dirty reads): es cuando una transacción lee un dato, luego otra lee el dato actualizado, pero la primera aborta, por lo cual la segunda transacción leyó un dato que no fue aplicado a la base.*
- *Lectura no repetible (repeatable read): es cuando una transacción lee un valor, luego otra transacción actualiza ese valor y si la primera transacción repitiera la operación, ya no tendría el mismo valor original.*
- *Lectura fantasma (phantom reads): es cuando una transacción a través de una condición opera con un conjunto de registros a través de una cláusula where. Otra transacción en ese momento puede estar actualizando datos que cumplan con esa misma condición, por lo cual si se volviera a ejecutar podría no darnos el mismo resultado. Se verán filas fantasmas.*





Niveles de Aislamiento

SET TRANSACTION ISOLATION LEVEL <Nivel de Aislamiento>

	Dirty Reads	Repeatable Read	Phantom Reads
Read Uncommitted	SI	SI	SI
Read Committed	NO	SI	SI
Repeatable Read	NO	NO	SI
Serializable	NO	NO	NO



Lockeos: Granularidad

Además del tipo de lockeo que podrá ser Shared(S) ó Exclusive(X), el SGBD establecerá qué debe lockear, lo que llamaremos la granularidad del lockeo:

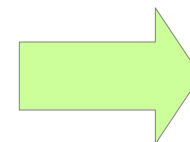
- *Database - Base de datos*
- *Table – Tabla*
- *Page - Página/s ó Bloque/s*
- *Row – Fila*



Lockeos: Deadlock

Se produce cuando cada transacción T en un conjunto de dos o más transacciones está esperando a algún elemento que está bloqueado por alguna otra transacción T' de dicho conjunto.

T1	T2
Lock-X(X)	
Read(X)	
Write(X)	
	Lock-X(Y)
	Read(Y)
	Write(Y)
Lock-X(Y)	
Read(Y)	
Write(Y)	
	Lock-X(X)
	Write(X)



DEADLOCK



Lockeos: Deadlock

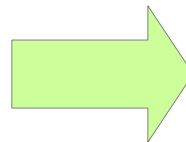
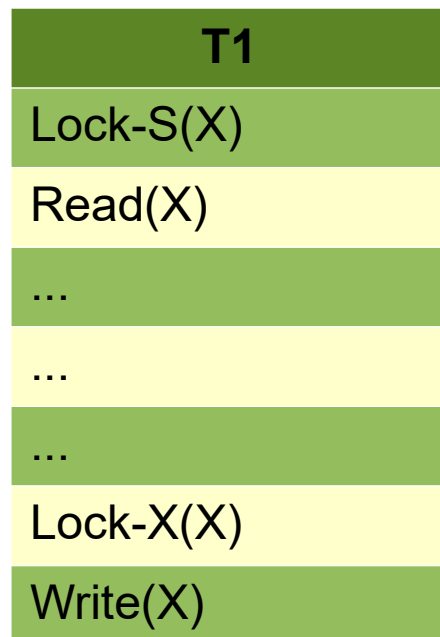
¿Qué hacer con los deadlocks?

- **Ignorarlos:** Dejar que los programadores lo manejen.
- **Prevenirlo:** Garantizar que los deadlocks jamás puedan ocurrir y esto se podría realizar lockeando todos los recursos antes de usarlo, también se podría realizar los lockeos en un orden estricto.
- **Evitarse:** Detectar los potenciales deadlocks en forma adelantada y tomar una acción sobre las transacciones.
- **Detectar y recuperar:** permitir los deadlocks, detener y recuperar las transacciones.



Lockeos: Lock Escalation

Se llama Lock Escalation cuando la transacción T debe aumentar el nivel de lockeo establecido previamente, ya sea en tipo de lockeo o en granularidad de lockeo. Por ejemplo, tenía un lockeo de shared y necesita aumentar a exclusive, tenía un lockeo a nivel row y necesita aumentar a un lockeo a nivel table.



Lock Escalation



Lockeos: Timeout

En los SGBD se puede establecer el tiempo en que un recurso puede aguardar por el lockeo. Si este tiempo se excede, automáticamente aborta la transacción provocando un evento de Lock-Timeout.

Si el parámetro se encuentra en -1 significa que esperará en forma indefinida, de lo contrario tendrá establecido un número que indicará la cantidad de milisegundos que esperará. Si el parámetro se encuentra en 0, no esperará a que un lockeo se libere, sino que si está siendo usado automáticamente devolverá el evento de timeout.

```
SET LOCK_TIMEOUT timeout_period
```



Funciones de del SGBD

- *Control de Concurrency (Transaction Manager)*
 - *Problemas de la Concurrency*
 - *Serialización de los planes*
 - *Protocolos de Locking*
 - *Tipo de Locks*
 - *Granularidad de Locks*
 - *Niveles de Aislamiento*
 - *Deadlock*
 - *Lock-Timeout*
 - *Lock-Escalation*
- **Control de recuperación (Recovery Manager)**



Control de recuperación

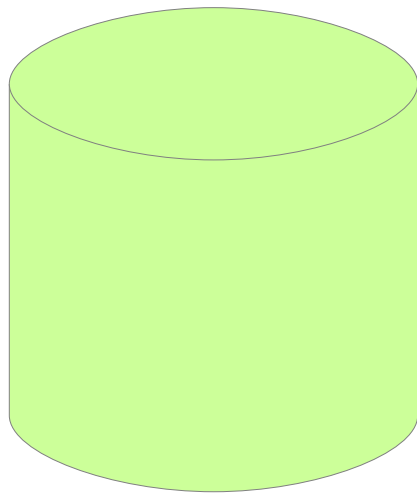
Dado que esas transacciones que están ejecutándose pueden cancelarse por un factor intencional o un factor externo, el SGBD debe garantizar la consistencia de los datos que estén siendo utilizados en el momento de la interrupción. Para ello posee un módulo de Recovery Manager encargado de dicha gestión.



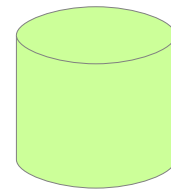
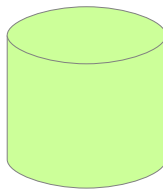
Log de Transacciones

Los SGBD poseen una bitácora de todas las operaciones que se van realizando en el motor. Esta bitácora es uno o más archivos físicos llamados archivos de LOG en donde guardan las operaciones.

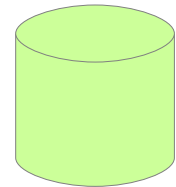
Cada vez que se inicia una transacción que realiza cambios, dichos cambios se van almacenando en el archivo de LOG.



Datafiles



...



Logfiles



Log de Transacciones

¿Qué se almacena?

- *LSN (Log Sequential Number)*
- *LSN Previo*
- *XID (Indicador de la transacción)*
- *Tipo (commit, abort, end, checkpoint, compensation log (para undo))*
- *PáginaID*
- *Longitud*
- *Offset*
- *Image Page*



Escritura del Log

Log Output	Write
$\langle T_0, \text{begin} \rangle$	
$\langle T_0, A, 1000, 900 \rangle$	
$\langle T_0, B, 2000, 1900 \rangle$	
	A=900
	B=1900
$\langle T_0, \text{commit} \rangle$	
$\langle T_1, \text{begin} \rangle$	
$\langle T_1, C, 3000, 2900 \rangle$	
	C=2900
$\langle T_1, \text{commit} \rangle$	



Ante una falla

Cuando se produce una falla y el SGBD se reinicia, es necesario verificar las transacciones para reconstruir el estado al momento de la falla, sin generar inconsistencia de datos. Toda esta información se obtiene verificando el Log de Transacciones:

- La transacción T_i tiene que deshacer todo lo que había realizado, si la transacción había sido abortada < T_i rollback>
- La transacción T_i tiene que aplicar todos los cambios realizados si la transacción había sido commiteada y aún no se habían escrito los bloques cambiados < T_i commit>
- La transacción T_i tiene que deshacer todo lo realizado si no se ha encontrado un fin de esa transacción, ya sea por commit o rollback.



Proceso de Commit

1. Genera una entrada en el log de commit
2. Actualiza todas las páginas que hayan sido utilizadas en la transacción al log en disco
3. Retorna el commit
4. Fin de la transacción en el log



Proceso de Rollback

1. Genera una entrada en el log de abort
2. Obtener el último **lastLSN** que haya generado la transacción.
3. Irá recorriendo cada una de las páginas afectadas, tomando el previo LSN de cada una de las entradas.
4. Generar un **CLR** (Compensation Log Record) por cada una de las operaciones de actualización.
5. Deshacer la operación, utilizando la imagen anterior.
6. Fin de la transacción en el log

Checkpoint

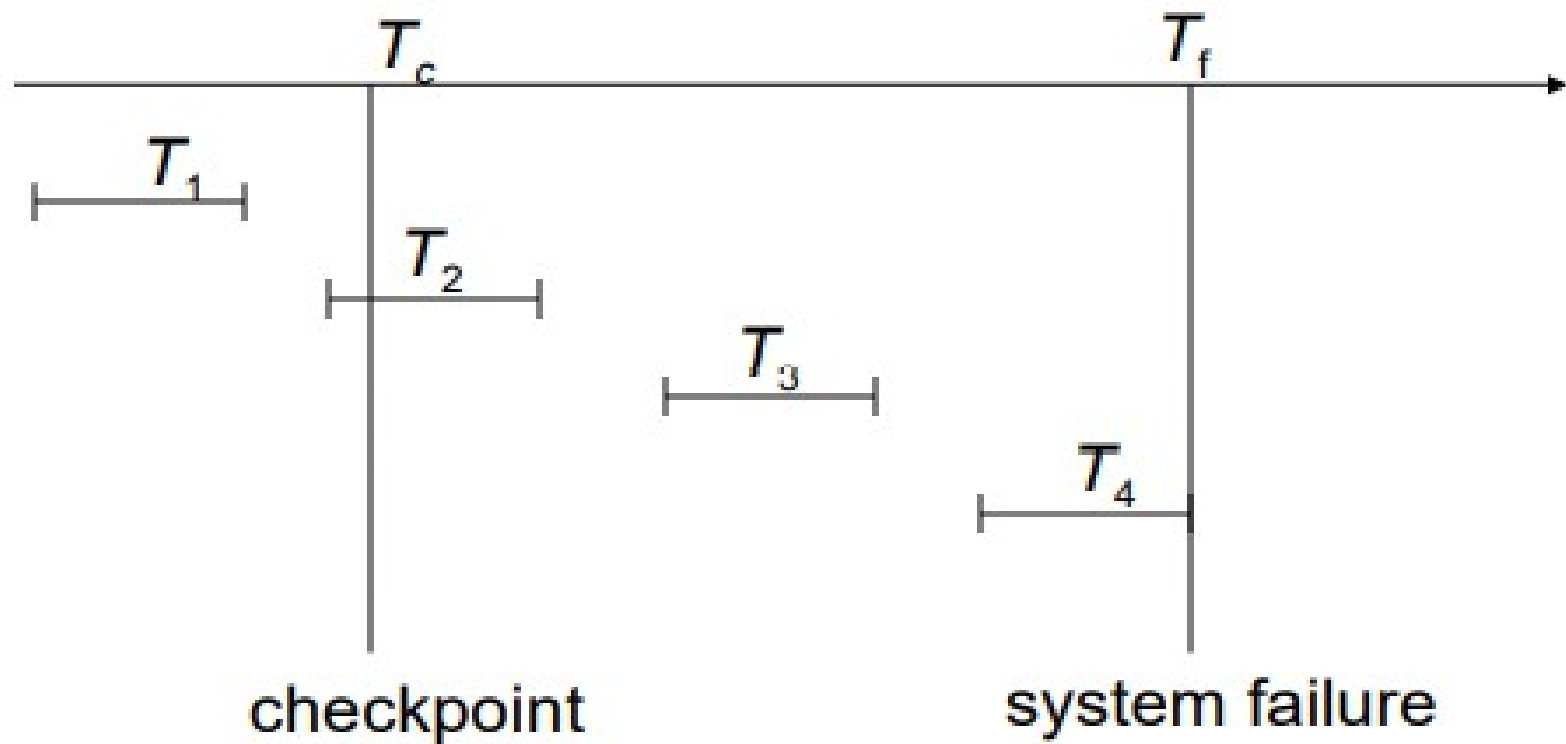


Cuando cambiamos los bloques de la base de datos, esos bloques no se envían a disco, hasta que no se produce un evento de CheckPoint. El checkpoint es el evento por el cual los bloques modificados son realmente llevados a disco.

En los SGBD se puede establecer el tiempo en el cual el checkpoint se provoca. Cuanto más bloques tenga sin actualizar en disco, mayor será el tiempo de recuperación de la base de datos, en el caso de una falla.

Cuando el CheckPoint se produce, se podrá liberar esa sección del log que ya no se necesitará porque los bloques ya están actualizados.

Proceso de Recovery

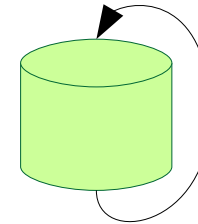


Logs



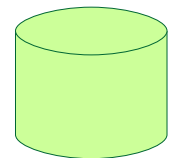
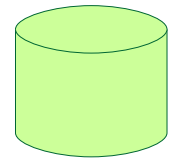
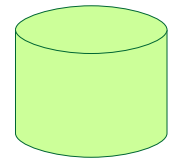
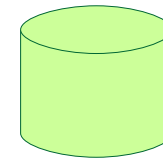
- Circulares

Active Logs



Archived Logs

- Secuenciales



Consultas



Gracias.