Teórico 10

Transacciones

Considere el Siguiente Caso

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

1. **Leer**(*A*)
2. *A* := *A* – 500
3. **Escribir**(*A*)
4. **Leer**(*B*)
5. *B* := *B* + 500
6. **Escribir**(*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 bancaria 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).

Transacciones de Bases de Datos

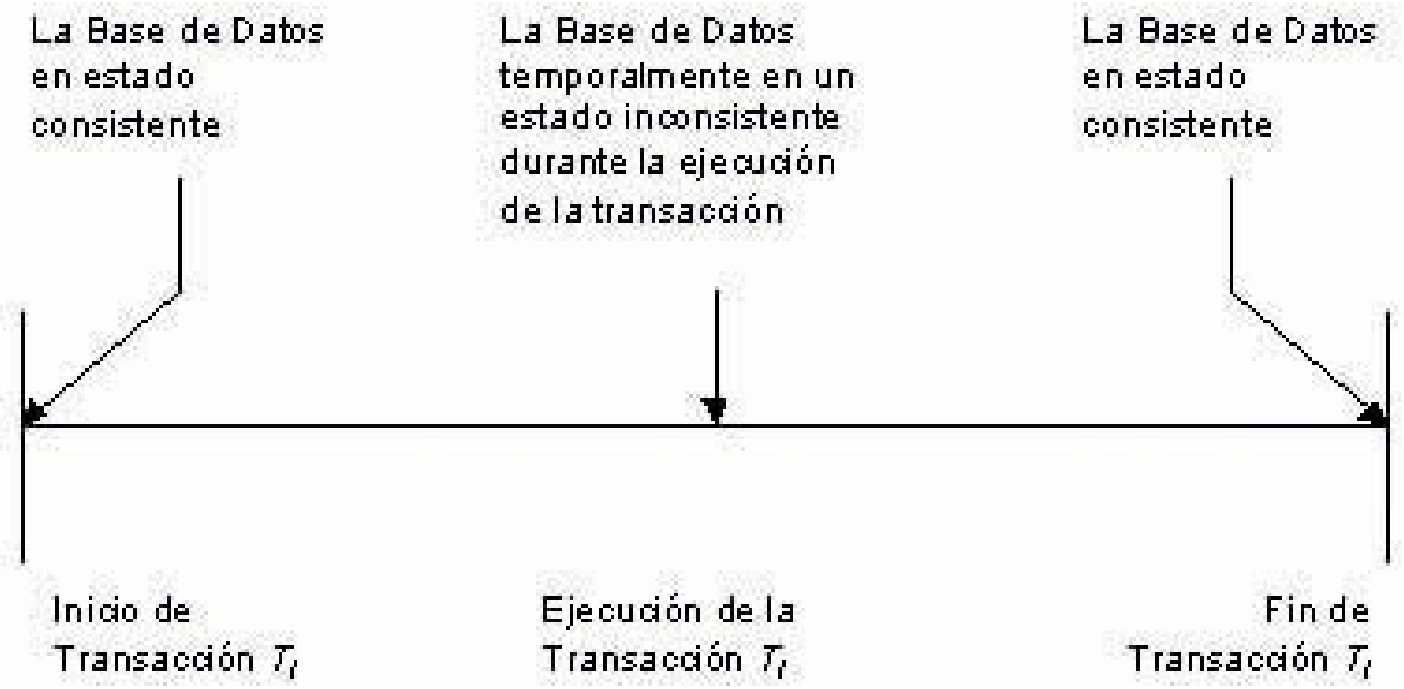
- Una **transacción** es una unidad de la ejecución de un programa que accede y posiblemente actualiza varios elementos de datos.
- Una transacción debe ver una base de datos consistente.
- Durante la ejecución de una transacción la base de datos puede quedar inconsistente temporalmente.
- Cuando la 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).

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.

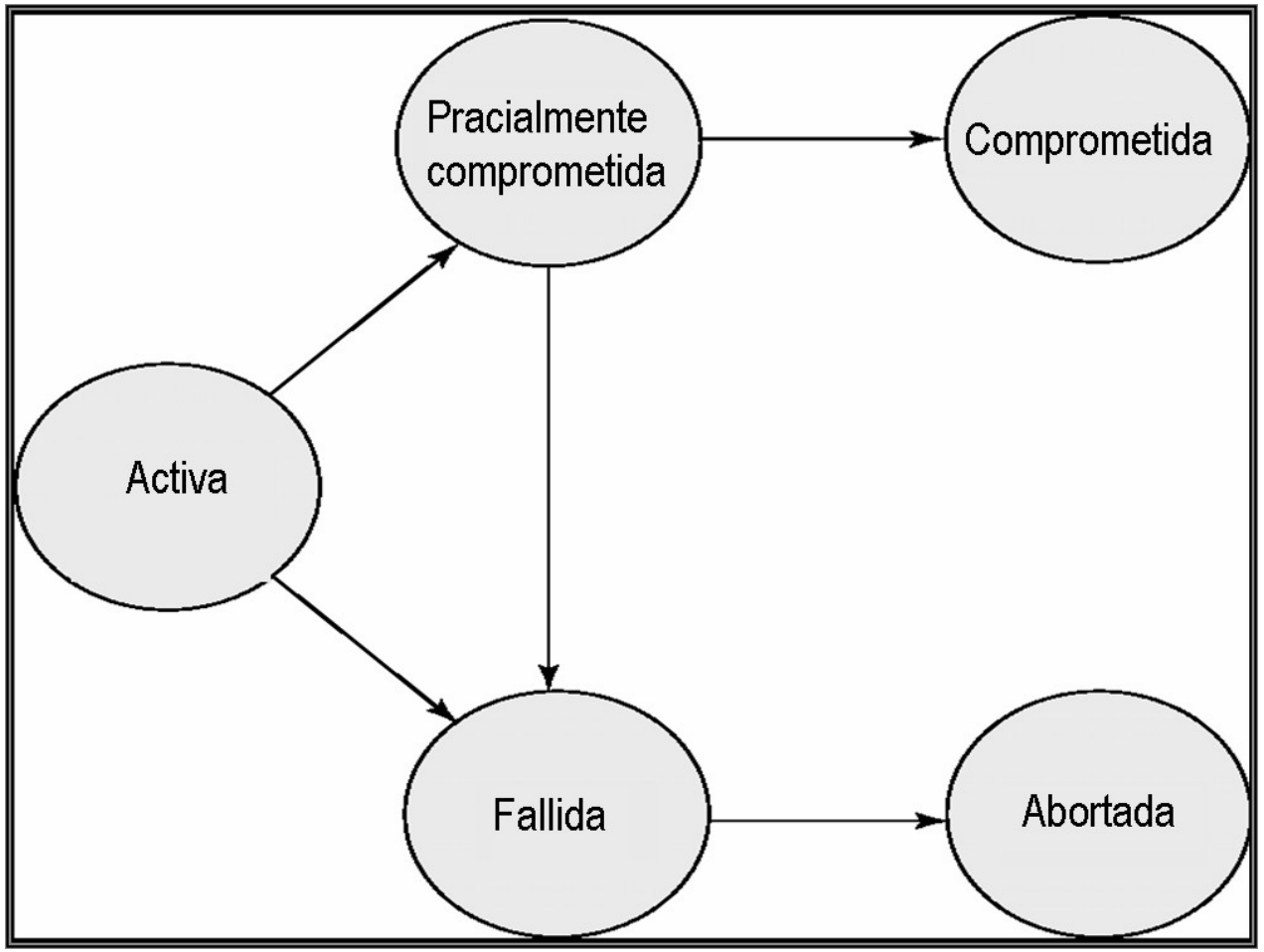
Gráfico Temporal de la Ejecución de una Transacción



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 si no ha habido algún error lógico interno.
  - Matar la transacción.
- **Comprometida:** Se completó correctamente.

Estados de una Transacción (Cont)



Ejecución Concurrente

- Se permite correr concurrentemente a varias transacciones en el sistema. Esto permite:
  - **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 para 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 que 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 o Seriabilidad

- 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 ítem 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).*

Ejemplo

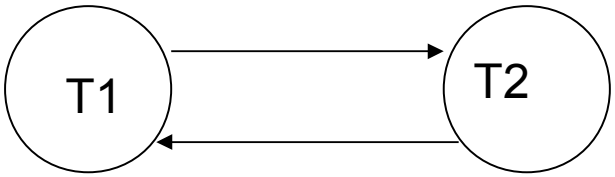
T1	T2
Leer(A)	
Escribir(A)	
	Leer(A)
	Escribir(A)
Leer(B)	
Escribir(B)	
	Leer(B)
	Escribir(B)

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'.
  2. 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.

Niveles de Consistencia en SQL

- Hay casos que se permite que las transacciones no se conviertan en forma secuencial, por ejemplo transacciones largas para poder realizar estadísticas, en estos 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

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.

- **Lecturas 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

Ejemplos de Niveles de Aislamiento en MySQL

- En los siguientes ejemplos se trabaja sobre la tabla artículos, y el valor inicial de la cantidad del artículo número 100 es 8.

Ejemplos de Niveles de Aislamiento

Conexión 1	Read Uncommited	Read commited	Repeatable read	serializable
leer(A)				
A:=A+10;				
	leer(A) (a=10)	leer(A) (a=10)	leer(A) (a=10)	leer(A) (a=10)
Escirbe(A)				
	leer(A) (a=20)	leer(A) (a=10)	leer(A) (a=10)	leer(A) (a=10)
commit				
	leer(A) (a=20)	leer(A) (a=20)	leer(A) (a=10)	leer(A) (a=10)

Nivel de Aislamiento Serializable

Conexión 1	Conexión 2
Set autocommit = 0;	Set autocommit = 0;
	set transaction isolation level serializable;
	Select * from articulos where nart=100; // cant = 8
Select * from articulos where nart=100;	
update articulos set cant = cant - 2 where nart=100; El update se bloquea	
	Commit;
El update se desbloquea	
	Select * from articulos where nart=100; se bloquea
Select * from articulos where nart=100 // cant = 6	
Commit;	
	Desbloqueo // cant = 6

Nivel de Aislamiento Repeatable read

Conexión 1	Conexión 2
Set autocommit = 0;	Set autocommit = 0;
	set transaction isolation level **repeatable read**;
update articulos set cant = cant-2 where nart=100;	
	Select * from articulos where nart=100 Resultado cant = 8
Select * from articulos where nart=100; // Resultado cant = 6	
Commit;	
	Select * from articulos where nart=100; // Resultado cant = 8
	Commit;
	Select * from articulos where nart=100; //Resultado cant = 6

Nivel de Aislamiento Read Committed

Conexión 1	Conexión 2
Set autocommit = 0;	Set autocommit = 0;
	set transaction isolation level **read committed**;
	Select * from articulos Resultado cant = 8
update articulos set cant = cant - 2 where nart=100;	
Select * from articulos where nart=100; //Resultado cant = 6	
	Select * from articulos Resultado cant = 8
Commit;	
	Select * from articulos Resultado cant = 6
	Commit;

Nivel de Aislamiento Read uncommitted

Conexión 1	Conexión 2
Set autocommit = 0;	Set autocommit = 0;
	set transaction isolation level **read uncommitted**;
update articulos set cant=cant-2 where nart=100;	
Select * from articulos where nart=100 Resultado cant = 6	
	Select * from articulos where nart=100 Resultado cant = 6
	Commit;
Commit;	