

Transacciones



Universidad de Oviedo

Repositorios de Información

Ingeniería Informática

2020-2021

Introducción

Propiedades de las transacciones

Concurrencia, Planificación y Anomalías

Datos Anómalos/Erróneos, si no tenemos cuidado

Introducción

Rendimiento en el almacenamiento

Disco Vs. Memoria

Peter Norvig's blog norvig.com

Latency numbers every engineer should know

Ballpark timings on typical PC:

execute typical instruction	1/1,000,000,000 sec = 1 nanosec
fetch from L1 cache memory	0.5 nanosec
fetch from L2 cache memory	7 nanosec
Mutex lock/unlock	25 nanosec
fetch from main memory	100 nanosec
send 2K bytes over 1Gbps network	20,000 nanosec
read 1MB sequentially from memory	250,000 nanosec
fetch from new disk location (seek)	8,000,000 nanosec
read 1MB sequentially from disk	20,000,000 nanosec
send packet US to Europe and back	150 milliseconds = 150,000,000 nanosec



Srigha
@srigha

"Latency Numbers Every Programmer Should Know"

It is hard for humans to get the picture until you translate it to "human numbers":

1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access	120 ns	6 min
Solid-state disk I/O	50-150 μ s	2-6 days
Rotational disk I/O	1-10 ms	1-12 months
Internet: SF to NYC	40 ms	4 years
Internet: SF to UK	81 ms	8 years
Internet: SF to Australia	183 ms	19 years
OS virtualization reboot	4 s	423 years
SCSI command time-out	30 s	3000 years
Hardware virtualization reboot	40 s	4000 years
Physical system reboot	5 m	32 millenia

Rendimiento en el almacenamiento

Disco Vs. Memoria


- **Memoria:** Rápido pero con capacidad limitada y volátil.
- **Disk:** Lento pero con gran capacidad y persistencia.

¿Cómo podemos utilizar ambas eficazmente y con garantías?

Modelo con tres tipos de áreas en memoria

1. **Local:** Cada proceso en un SGBD tiene su propia area de memoria local que sólo él puede "**ver**".
2. **Global or Shared:** Cada proceso puede leer o escribir datos compartidos en memoria global.
3. **Disk:** La memoria global puede ser leída/volcada a disco.
4. **Log:** Asumiendo que el almacenamiento en disco es estable, puede abarcar desde la memoria principal, el disco ...

	Local	Global
Main Memory (RAM)	1	2 4
Disk		3

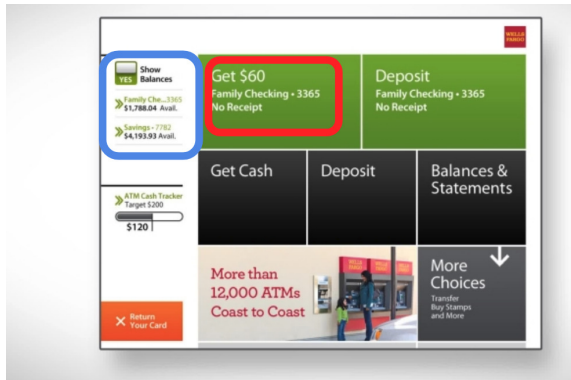


Log secuencia
de Memoria →
Disco

Volcado a disco = Escribir de
Memoria a Disco

Ejemplo

Transacción en cajeros automáticos



Read Balance
Give money
Update Balance

vs

Read Balance
Update Balance
Give money

Transacción: Definición

Una transacción (TXN) es una secuencia de una o más operaciones (lectura/escritura) que refleja una única operación en el mundo real.

En el mundo real una TXN o sucede por completo o no sucede.

```
START TRANSACTION
UPDATE Product
SET Price = Price - 1.99
WHERE pname = 'Gizmo'
COMMIT
```

- Transferencia de dinero entre cuentas.
- Comprar un conjunto de productos.
- Inscribirse en una clase (o a una lista de espera),

Transacciones en SQL

- Todas la instrucciones SQL individuales, con raras excepciones, se encuentran automáticamente en una transacción, ya sea que se establezca explícitamente o no (incluso si insertan, actualizan o eliminan millones de filas).
- En un programa, se pueden agrupar varias consultas en una TXN.

```
START TRANSACTION
  UPDATE Bank SET amount = amount - 100
  WHERE name = 'Bob'
  UPDATE Bank SET amount = amount + 100
  WHERE name = 'Joe'
COMMIT
```

Motivación

1. **Recuperación & Persistencia:** Capacidad de recuperación ante daños (bloqueos, apagados inesperados, ...), persistencia y consistencia de los datos. Si algo va mal, la BD vuelve a una copia consistente. Si la TXN se ejecuta con éxito, los cambios serán perdurables.
2. **Concurrencia:** Para obtener un mejor rendimiento en la ejecución, el SGBD puede ejecutar de forma concurrente varias TXNs. En este caso la consistencia se pone en riesgo.

Lectura complementaria: [1]

Propiedades de las transacciones

ACID

Guía rápida



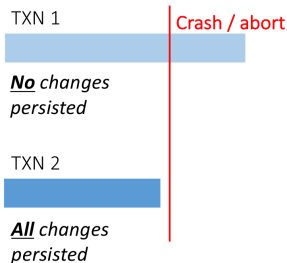
ACID: Atomicity

Las partes de una TXN son atómicas en bloque: **todo o nada**

- Intuitivamente: En el mundo real, una TXN es algo que u ocurre completamente o no.

Hay dos posibles salidas para una TXN:

- **Commit**: se realizan todos los cambios
- **Abort**: No se realiza nada.



Atomicity es una característica de sistemas de **journaling** o **logging**

ACID: Consistency

Las diferentes tablas de la base de datos se crean con una condiciones especificas de integridad.

- Ejemplos:

- ☐ Número de cuenta único
- ☐ El cantidad de un producto no puede ser negativa.
- ☐ La suma de deudas y creditos debe ser 0.

Cómo se logra la consistencia:

- Es el programador el que se encarga de que una TXN vaya de un estado consistente a otro.
- Es el SGBD el que se encarga de que la TXN sea **atómica**

ACID: Isolation

- Una TXN se ejecuta de forma concurrente junto con otras TXNs.
- **Isolation:** El aislamiento proporciona a la TXN la capacidad para que se ejecute sin que se solape con otras.
 - No se deberían poder ver cambios de otras TXNs durante su ejecución.

ACID: Durability

- El efecto de un a TXN debe ser continuado (*persistente*).
 - Y después de que el programa haya terminado.
 - Incluso si hay fallos eléctricos o cualquier otro problema.
- Implica escribir datos en el disco.

Desafios

- Las propiedades ACID deben mantenerse, **a pesar de los fallos**.
- Los usuarios pueden abortar la ejecución del programa, *revirtiendo los cambios*.
 - Es necesario mantener un log de lo ocurrido.
- Muchos usuarios ejecutando concurrentemente.
 - Estableciendo bloqueos.

Y todo esto, manteniendo el **rendimiento**.

Concurrencia, Planificación y Anomalías

Concurrencia: Aislamiento y Consistencia

El SGBD debe mantener la concurrencia para que ...

1. Se mantenga el aislamiento (**Isolation**): Los diferentes usuarios deberían ser capaces de ejecutar las TXN como si fueran el **único usuario** accediendo.
 - El SGBD gestiona los detalles de que *convivan* varias TXNs a la vez.
2. Se mantenga la **Consistencia**: Las TXNs deben poder ejecutarse en un estado **consistente**
 - El SGBD gestiona los detalles para forzar las restricciones de integridad.

La parte más difícil es mantener la consistencia de los datos cuando ocurren bloqueos (crash) del sistema.

Ejemplo – considere dos TXNs

T1: START TRANSACTION

UPDATE Accounts
SET Amt = Amt + 100
WHERE Name = 'A'

UPDATE Accounts
SET Amt = Amt - 100
WHERE Name = 'B'

COMMIT

T1 transfers \$100 from B's account to A's account

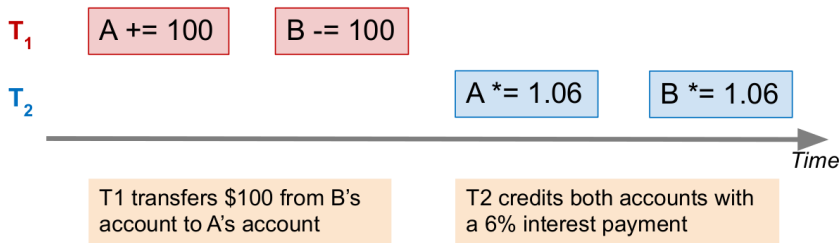
T2: START TRANSACTION

UPDATE Accounts
SET Amt = Amt * 1.06
COMMIT

T2 credits both accounts with a 6% interest payment

Ejemplo

Podemos imaginarnos las TXNs que se ejecutan en **serie** (en una misma línea de tiempo)

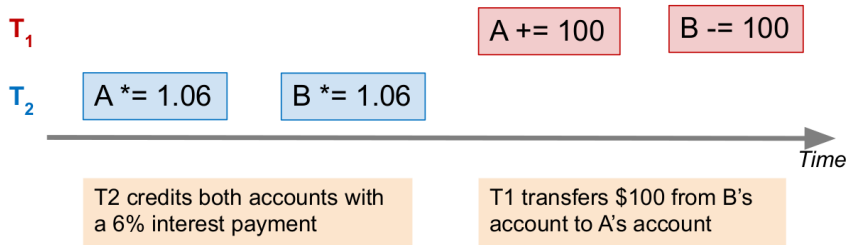


El objetivo a la hora de planificar las TXNs

- Intercalar las TXN para aumentar el rendimiento.
- Que los datos se mantengan consistentes después de un commit y/o abort (ACID).

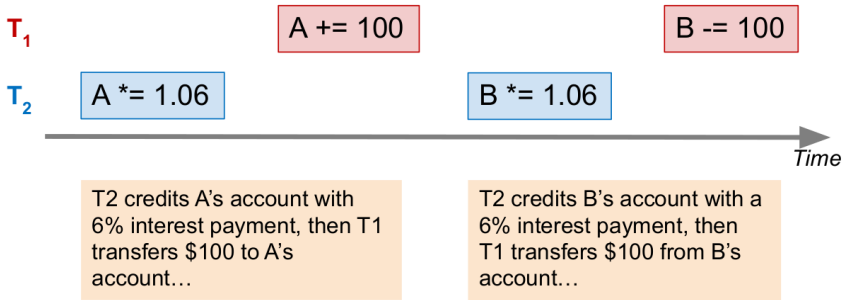
Ejemplo

Las TXNs pueden ejecutarse en cualquier orden ya que el SGBD lo permite.



Ejemplo

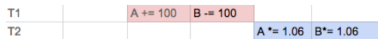
Incluso **intercalarlas**



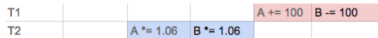
Ejemplo

El intercalado puede ocurrir en cualquier orden.

Serial Schedules

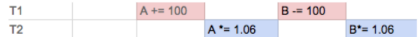


S1

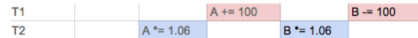


S2

Interleaved Schedules



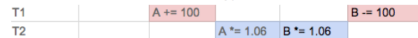
S3



S4



S5



S6

Intercalado & Aislamiento

- El SGBD tiene total libertad para intercalar las TXNs.
- Sin embargo esto lo debe hacer planificandolas de tal manera que se asegure el **aislamiento** y la **consistencia**.
 - Debería parecer como *si* las TXNs se ejecutasen en serie.

*With great power
comes great res-
ponsibility*

ACID

Planificación: Definición

Una planificación representa una secuencia de (potenciales) ejecuciones de Lectura (R), Escritura (W), Anulación (A) o Consolidación (C)

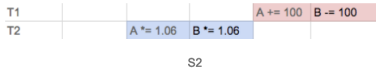
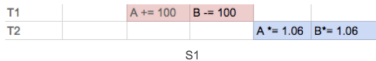
Una planificación en **serie** es aquella que no intercala las acciones de diferentes TXNs.

A y B son planificaciones **equivalentes** si, para cualquier estado de la BD, el efecto de ejecutar A es igual que el de ejecutar B.

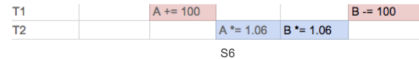
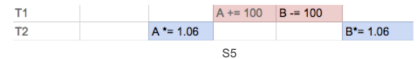
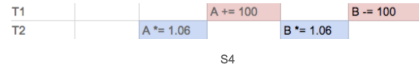
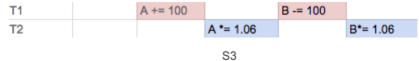
Una planificaciones **serializable** es aquella que es equivalente a alguna de las posibles ejecuciones en serie de varias TXNs.

Ejemplos

Serial Schedules



Interleaved Schedules



Serial Schedules	S1, S2
Serializable Schedules	S3, S4
Equivalent Schedules	<S1, S3> <S2, S4>
Non-serializable (Bad) Schedules	S5, S6

Modelo general de un SGBD: Concurrencia como intercalado de TXNs

Serial Schedule



Interleaved Schedule



Cada acción en una TXNs lee un valor de la memoria y después lo reescribe.

Para nuestro propósito, tener TXNs que se ejecuten de forma concurrente significa intercalar las acciones que las componen (Leer/Escribir)

Una planificación será un orden particular de intercalar sus acciones.

Cómo comprobar si una planificación es serializable

Dada las siguientes TXNs¹ T_1 , T_2 and T_3 y las siguientes planificaciones:

Schedule 1	Schedule 2
$R_1(X)$	$R_1(X)$
$R_3(X)$	$R_2(X)$
$W_1(X)$	$W_1(X)$
$R_2(X)$	$R_3(X)$
$W_3(X)$	$W_3(X)$

Determinar qué planificación es serializable, si la hay, y encontrar la ejecución en serie equivalente.

¹Las operaciones de W escriben un valor diferente al original

Ejercicio

Determinar si la siguiente planificación es serializable:

Schedule 1

$R_1(A)$

$R_1(B)$

$R_2(A)$

$R_2(C)$

$W_1(B)$

$R_3(B)$

$R_3(C)$

$W_3(B)$

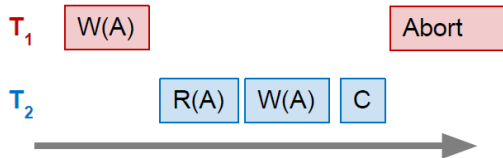
$W_2(A)$

$W_2(C)$

Datos Anómalos/Erróneos, si no tenemos cuidado

Anomalías frecuentes con intercalado en la ejecución

Lectura Sucia / Lectura de datos no confirmados

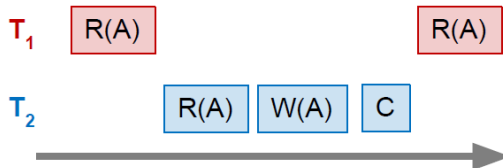


1. T_1 escribe en A
2. T_2 lee de A y escribe en A y confirma la escritura.
3. T_1 aborta - la escritura de T_2 's se hizo sobre datos obsoletos o inconsistentes.

Esto sucede porque hay un **conflicto de W/R**

Anomalías frecuentes con intercalado en la ejecución

Lectura no repetible



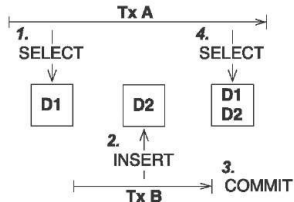
1. T_1 lee de A
2. T_2 escribe en A
3. T_1 lee otra vez de A, obteniendo un valor diferente / inconsistente.

Esto sucede porque hay un **conflicto de R/W**

Anomalías frecuentes con intercalado en la ejecución

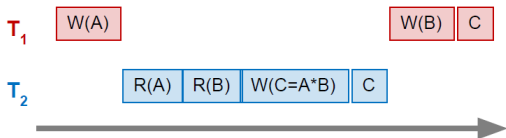
Lectura Fantasma

Sucede cuando en una misma TXN se accede a los mismos registros en momentos diferentes obteniendo resultados distintos.



Anomalías frecuentes con intercalado en la ejecución

Lectura inconsistente / Lectura de datos parcialmente confirmados

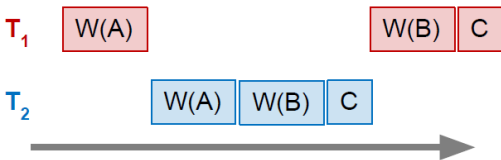


1. T_1 escribe en A.
2. T_2 lee de A y B, y escribe un valor que depende de A y B.
3. T_1 escribe en B - El resultado que se escribe en la T_2 esta basado en datos parcialmente confirmados.

Esto sucede porque hay un **conflicto de W/R**

Anomalías frecuentes con intercalado en la ejecución

Actualización parcialmente perdida



1. T_1 escribe en A
2. T_2 escribe en A y B
3. T_1 escribe en A; el valor de A es el que escribió T_2 y el de B el que escribió T_1 - no es equivalente a una ejecución en serie any serial schedule!

Esto sucede porque hay un **conflicto de W/W**

Ejercicio

Asumimos que el valor de A y B es 1000 en ambos.

Considere que la transacción T_1 transfiere 50 de A a B y T_2 retira el 10 % de A.

A	T_1	T_2	A
i_{11}	R(A)	R(A)	i_{21}
i_{12}	A=A-50	aux=A*0.1	i_{22}
i_{13}	W(A)	A=A-aux	i_{23}
i_{14}	R(B)	W(A)	i_{24}
i_{15}	B=B+50	COMMIT	i_{25}
i_{16}	W(B)	A	A
i_{17}	COMMIT	A	A

If T1 then T2	If T2 then T1
Final values are, A = 855 B = 1050	Final values are, A = 850 B = 1050

Table 1: Final values of A and B if T_1 and T_2 are executed in serial order

Ejercicio

1. Considere el siguiente intercalado $i_{11}, i_{12}, i_{21}, i_{22}, i_{23}, i_{13}, i_{14}, i_{24}, i_{25}, i_{15}, i_{16}, i_{17}$. Identifique la anomalía y el conflicto (RW, WR, WW).
2. Ahora considere el siguiente intercalado $i_{11}, i_{12}, i_{13}, i_{21}, i_{22}, i_{23}, i_{24}, i_{25}$. Mientras que T_2 se esta ejecutando, T_1 hace un **rollback** por alguna razón. Identifique la anomalía y el conflicto (RW, WR, WW).

¿Cómo gestionar los Conflictos de forma adecuada?

- Control de la concurrencia basado en bloqueos (**Lock-based Concurrency Control**)
 - En cada lectura se establece un bloqueo compartido (**shared lock**), mientras que en cada escritura se establece un bloqueo exclusivo (**exclusive lock**).
 - Un bloqueo compartido no permite la escritura pero sí que se acceda a los datos para leer.
 - Un bloqueo exclusivo no permite ni lecturas ni escrituras.
- Control de la concurrencia basado en versiones (**Multi-Versioned Concurrency Control (MVCC)**).

Los bloqueos generalmente no se establecen a nivel de aplicación.
Lo realiza el SGBD a través de los llamados **Niveles de aislamiento**

Niveles de Aislamiento

- Cuando se establecen bloqueos cuando se leen datos, y qué tipos de bloqueos se solicitan.
- La duración de los bloqueos.
- Si una operación de lectura hace referencia a filas modificadas por otra TXN.
 - Bloquear hasta que la fila es liberada.
 - Recuperar la versión confirmada de la fila que existía en el momento en el que empezó la consulta o transacción.
 - Leer datos modificados que no han sido confirmados.

El nivel de aislamiento **no afecta a** a los bloqueos que se establecen para proteger la modificación de los datos.

- Una transacción siempre obtiene un bloqueo exclusivo en cualquier dato que modifique y mantiene ese bloqueo hasta que se complete la transacción.

Niveles de Aislamiento

1. Read Uncommitted (Menos Restrictiva)

- No se establecen bloqueos compartidos cuando se leen datos.
- La escritura en una TXN establece un bloqueo (exclusivo) de escritura sólo *cuando* la TXN está modificando la fila, y lo libera inmediatamente después.
- Los datos corruptos no se leen.

2. Read Committed

- Se establecen bloqueos compartidos en la lectura.
- La escritura en una TXN establece un bloqueo de escritura (en los datos/filas que se escriben) y se mantiene hasta que la TXN termina.
- No permite la lectura de datos no confirmados.

Niveles de Aislamiento

1. Repeatable Read

- Además de los bloqueos anteriores la BD también bloquea los datos después de la lectura y los mantiene bloqueados hasta que termine la TXN.

2. Serializable (Más Restrictiva)

- Impide actualizaciones o inserciones de nuevas filas hasta que se complete la TXN.

Anomalías Vs. Niveles de Aislamiento

Level/Anomaly	Dirty Read	Unrepeatable read	Phantom
Read Uncommitted	Maybe	Maybe	Maybe
Read Committed	No	Maybe	Maybe
Repeatable Read	No	No	Maybe
Serializable	No	No	No

Ejercicios

Consider table $W(\text{name}, \text{pay})$ where name is a key, and two concurrent txns. Assume individual statements $S1$, $S2$, $S3$, and $S4$ always execute atomically. Let Amy's pay be 50 before either txn begins execution.

T1:

Begin Transaction

$S1$: update W set $\text{pay} = 2 * \text{pay}$ where $\text{name} = \text{'Amy'}$

$S2$: update W set $\text{pay} = 3 * \text{pay}$ where $\text{name} = \text{'Amy'}$

Commit

T2:

Begin Transaction

$S3$: update W set $\text{pay} = \text{pay} - 20$ where $\text{name} = \text{'Amy'}$

$S4$: update W set $\text{pay} = \text{pay} - 10$ where $\text{name} = \text{'Amy'}$

Commit

Suppose T1 and T2 execute to completion with different isolation levels. What are the possible values for Amy's final pay if

1. both execute with Serializable?
2. both execute with Read-Committed?.
3. T1 is Read-Committed and T2 executes with Read-Uncommitted?.
4. both execute with Read-Uncommitted?.

Suppose T1 and T2 execute with isolation level Serializable. T1 executes to completion, but T2 rolls back after statement $S3$ and does not re-execute. What are the possible values for Amy's final pay?

Lectura recomendada (1)

- [1] <https://www.codemag.com/Article/0607081/Database-Concurrency-Conflicts-in-the-Real-World>
- [2] <https://www.youtube.com/watch?v=FA85kHsJss4> and <https://www.youtube.com/watch?v=aNCpEC0-VVs>
- [3] <https://www.youtube.com/playlist?list=PLroEs25KGvwzmvIxYHRhoGTz9w8LeXek0> 12-01, 12-02, 12-03
- [4] <https://cs.stanford.edu/~chrismre/>
- [5] <https://cs.stanford.edu/people/widom/>
- [6] <https://es.slideshare.net/brshristov/database-transactions-and-sql-server-concurrency>

Lectura recomendada (2)

- [7] <https://ucbrise.github.io/cs262a-spring2018/notes/07-concurrency.pdf>
- [8] <https://15445.courses.cs.cmu.edu/fall2018/>