BASES DE DADES

Grau en Enginyeria en Informàtica

Universitat de Barcelona

Curs 2017/18

TEMA 6
TRANSACCIONS I CONCURRÈNCIA

Objectius

1) **Conèixer concepte de transacció**
2) Conèixer propietats ACID de les transaccions
3) Conèixer problemes importants relacionats amb la concurrència
4) Conèixer concepte de serialització i planificació en sèrie i no en sèrie
5) Conèixer tècnica de control de concurrència basada en bloqueig
6) Conèixer protocol 2PL de bloqueig / desbloqueig

Transaccions

- Una transacció és una **unitat lògica** de treball
- És una acció, o un conjunt d'accions, que es realitza en el moment de llegir o modificar (escriure) dades a una base de dades
  - Per exemple, actualitzar el salari d'un empleat a la taula STAFF
- Una transacció consisteix habitualment en
  - dues operacions (lectura i escriptura) – i combinacions
  - operació que no es pròpia de la base de dades (per ex. incrementar salari)

Transaccions

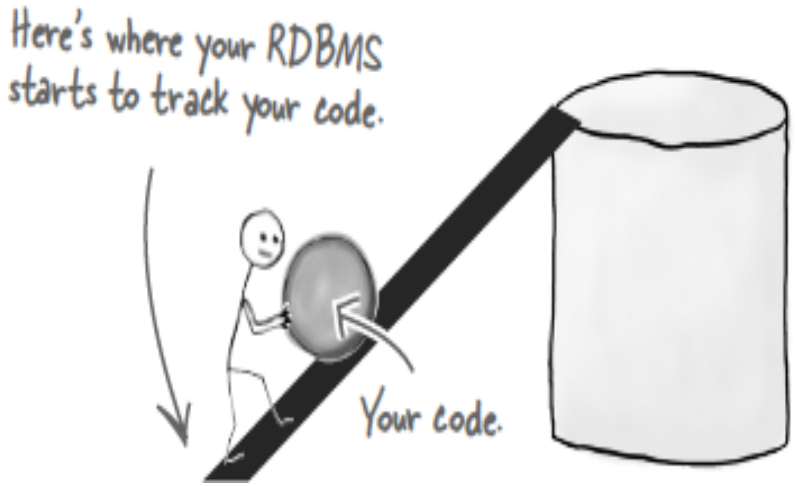
- Una transacció sempre ha de deixar una base de dades en un **estat consistent**
  - Quan s'està executant la transacció, la base de dades pot estar en un estat inconsistent, però mai quan finalitza
- Estat consistent
  - Segueix les restriccions d'integritat (veure Tema 2)
- Si una transacció s'ha realitzat **satisfactòriament**, ha fet **COMMIT.** En cas contrari, es realitza **ROLLBACK**
  - I la base de dades torna al seu estat anterior

Transaccions

```
START TRANSACTION;
```

This tracks what your SQL's doing.

START TRANSACTION ***keeps track of all the SQL*** that follows until you enter either `COMMIT` or `ROLLBACK`.



Capítol 11. Head First SQL

Transaccions

```
COMMIT;
```

This commits all your code once you're happy with it.

If you've got all your statements in place and everything looks good, type **COMMIT** to ***make it permanent***.

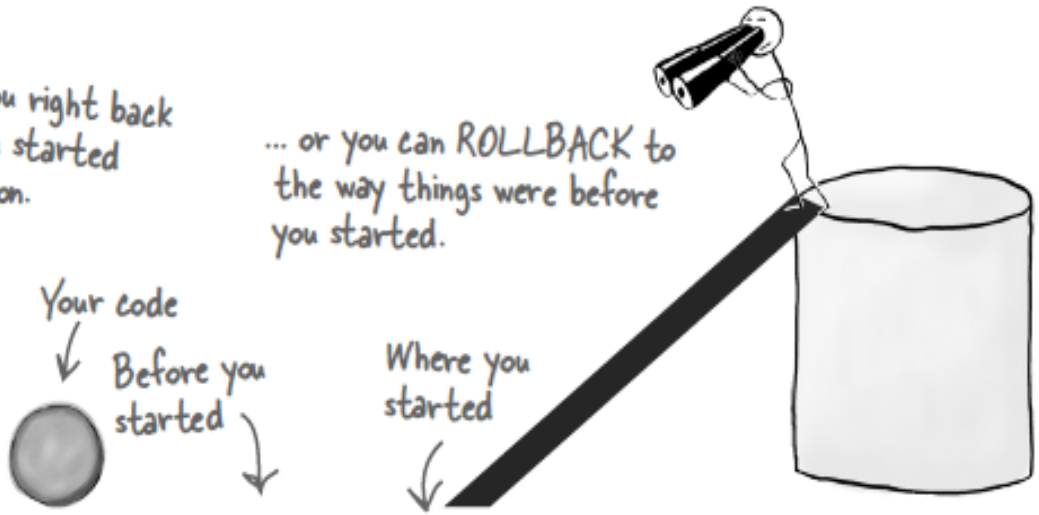
Capítol 11. Head First SQL

Transaccions

```
ROLLBACK;
```

This takes you right back to before you started the transaction.

If something isn't quite right, **ROLLBACK** ***reverses everything*** to the way it was ***before*** you typed `START TRANSACTION`.



Capítol 11. Head First SQL

Transaccions

Fill in the piggy_bank contents after these transactions. Here's how it looks now:

piggy_bank

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'Q' where coin = 'P'
AND coin_year < 1970;
COMMIT;
```

id	coin	coin_year
1		
2		
3		
4		

Capítol 11. Head First SQL

Transaccions

piggy_bank

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'Q' where coin = 'P'
AND coin_year < 1970;
COMMIT;
```

No matches, so no change.

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999

Capítol 11. Head First SQL

Transaccions

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'N' where coin = 'Q';
ROLLBACK;
```

id	coin	coin_year
1		
2		
3		
4		

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'Q' where coin = 'N'
AND coin_year > 1950;
ROLLBACK;
```

id	coin	coin_year
1		
2		
3		
4		

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'D' where coin = 'Q'
AND coin_year > 1980;
COMMIT;
```

id	coin	coin_year
1		
2		
3		
4		

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'P' where coin = 'N'
AND coin_year > 1970;
COMMIT;
```

id	coin	coin_year
1		
2		
3		
4		

Capítol 11. Head First SQL

Transaccions

piggy_bank

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'N' where coin = 'Q';
ROLLBACK;
```

Rollback, no change.

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'Q' where coin = 'N'
AND coin_year > 1950;
ROLLBACK;
```

Rollback, no change

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'D' where coin = 'Q'
AND coin_year > 1980;
COMMIT;
```

This row is affected.

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	D	1999

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'P' where coin = 'N'
AND coin_year > 1970;
COMMIT;
```

This row is affected

id	coin	coin_year
1	Q	1950
2	P	1972
3	P	2005
4	Q	1999

Capítol 11. Head First SQL

TEMA 6
TRANSACCIONS I CONCURRÈNCIA

Objectius

1) Conèixer concepte de transacció
2) **Conèixer propietats ACID de les transaccions**
3) Conèixer problemes importants relacionats amb la concurrència
4) Conèixer concepte de serialització i planificació en sèrie i no en sèrie
5) Conèixer tècnica de control de concurrència basada en bloqueig
6) Conèixer protocol 2PL de bloqueig / desbloqueig

Propietats ACID

- **Atomicitat**. Una transacció és una unitat indivisible que s'executa totalment o no s'executa
- **Consistència**. Una transacció ha de transformar una base de dades d'un estat consistent a un altre estat consistent
- **Aïllament**. Les transaccions s'executen de manera independent. Podem tenir N transaccions a la vegada, però el resultat final és com si s'haguessin realitzar en sèrie, una després d'una altra
- **Durabilitat**. Els efectes de les transaccions que s'han realitzar correctament es guarden a la base de dades

TEMA 6
TRANSACCIONS I CONCURRÈNCIA

Objectius

1) Conèixer concepte de transacció
2) Conèixer propietats ACID de les transaccions
3) **Conèixer problemes importants relacionats amb la concurrència**
4) Conèixer concepte de serialització i planificació en sèrie i no en sèrie
5) Conèixer tècnica de control de concurrència basada en bloqueig
6) Conèixer protocol 2PL de bloqueig / desbloqueig

Control de concurrència

- És el procés de controlar operacions que s'executen de manera simultània de manera que no s'interfereixin les unes a les altres

- I això és important? Un dels objectius és permetre que varis (molts) usuaris accedeixen a les dades, que són compartides, de manera concurrent.

- Si tenim operacions de lectura, no tenim molts problemes. Però si uns usuaris llegeixen i altres escriuen a la vegada…llavors podem tenir inconsistències

concurrència: 3 problemes típics

1. El problema de l'actualització perduda (*WW conflict*)
2. El problema de la lectura corrupta (*dirty read, WR conflict*)
3. El problema de l'anàlisi inconsistent (*RW conflict*)

En estos ejemplos, que los verás a continuación, hay que pensar que las transacciones tienen el valor de bal_x cuando lo leen, y las operaciones de suma y resta no modifican el valor de la variable en la base de datos hasta que no se realiza la operación de escritura.

Problema de l'actualització perduda (WW)

Tiempo	T_1	T_2	bal_x
t_1		BEGIN	100
t_2	BEGIN	read (bal_x)	100
t_3	read (bal_x)	$bal_x = bal_x + 100$	100
t_4	$bal_x = bal_x - 10$	write (bal_x)	200
t_5	write (bal_x)	COMMIT	90
t_6	COMMIT		90

És un problema de sobreescriptura

Problema de l'actualització perduda (WW)

Tiempo	T_1	T_2	bal_x
t_1		BEGIN	100
t_2	BEGIN	read (bal_x)	100
t_3	read (bal_x)	$bal_x = bal_x + 100$	100
t_4	$bal_x = bal_x - 10$	write (bal_x)	200
t_5	write (bal_x)	COMMIT	90
t_6	COMMIT		90

Si las dos transacciones (T1y T2) se ejecutan en serie, una después de la otra, el resultado final es 190. Pero no ocurre así en la Figura 11. Las dos transacciones leen el valor de bal_x, que es 100. El problema es que la escritura que realiza T_1 sobrescribe la actualización que ha hecho T_2. T1 resta 10 a su copia de bal_x **El valor de bal_x para T1 es de 100 (y no 200) – que es el valor cuando lo leyó**. Este problema se puede solucionar impidiendo que T1 lea el valor de bal_x hasta que T2 no lo haya actualizado.

Problema de la lectura corrupta (WR)

Tiempo	T_1	T_2	bal_x
t_1		BEGIN	100
t_2		read (bal_x)	100
t_3		$bal_x = bal_x + 100$	100
t_4	BEGIN	write (bal_x)	200
t_5	read (bal_x)	...	200
t_6	$bal_x = bal_x - 10$	ROLLBACK	100
t_7	write (bal_x)		190
t_8	COMMIT		190

Problema de la lectura corrupta (WR)

Una transacció pot veure els resultats d'una altra abans que hagi fet COMMIT (o ROLLBACK)

Tiempo	T_1	T_2	bal_x
t_1		BEGIN	100
t_2		read (bal_x)	100
t_3		$bal_x = bal_x + 100$	100
t_4	BEGIN	write (bal_x)	200
t_5	read (bal_x)	...	200
t_6	$bal_x = bal_x - 10$	ROLLBACK	100
t_7	write (bal_x)		190
t_8	COMMIT		190

Problema de la lectura corrupta (WR)

Tiempo	T_1	T_2	bal_x
t_1		BEGIN	100
t_2		read (bal_x)	100
t_3		$bal_x = bal_x + 100$	100
t_4	BEGIN	write (bal_x)	200
t_5	read (bal_x)	...	200
t_6	$bal_x = bal_x - 10$	ROLLBACK	100
t_7	write (bal_x)		190
t_8	COMMIT		190

En este ejemplo, T2 incrementa en 100 el valor de bal_x. Sin embargo, T2 decide abortar la operación una vez que ha realizado la operación de escritura. Para cuando aborta, T1 ya he empezado y ha leído de la base de datos el valor de bal_x que es 200, resultado de la actualización de T2. Entonces realiza la operación de resta, siendo el resultado final de bal_x 190 y no 90, si tenemos en cuenta que T2 ha deshecho los cambios.

Realizar el *rollback* en T2 no es un error. **El problema es que T1 no debería acceder al valor de bal_x hasta que T2 no haya abortado y hecho *commit*.**

Problema anàlisi inconsistent (RW)

- Una transacció llegeix valors de la base de dades però una segona transacció actualitza les mateixes dades al mateix temps

- No és un problema de sobreescriptura (el primer cas)
- Tampoc és un problema de que una transacció accedeixi a dades abans que una segona transacció que treballa amb les mateixes dades hagi acabat (el segon cas)

- Aquest és un problema diferent

Problema anàlisi inconsistent (RW)

Tiempo	T_1	T_2	bal_x	bal_y	bal_z	sum
t_1		BEGIN				
t_2	BEGIN	sum=0	100	50	25	0
t_3	read(bal_x)	read(bal_x)	100	50	25	0
t_4	bal_x = balx-10	sum = sum + bal_x	100	50	25	100
t_5	write (bal_x)	read(bal_y)	90	50	25	100
t_6	read (bal_z)	sum = sum + bal_y	90	50	25	150
t_7	bal_z = bal_z + 10		90	50	25	150
t_8	write (bal_z)		90	50	35	150
t_9	COMMIT	read (bal_z)	90	50	35	150
t_{10}		sum = sum + bal_z	90	50	35	185
t_{11}		COMMIT	90	50	35	185

Problema anàlisi inconsistent (RW)

Tiempo	T_1	T_2	bal_x	bal_y	bal_z	sum
t_1		BEGIN				
t_2	BEGIN	sum=0	100	50	25	0
t_3	read(bal_x)	read(bal_x)	100	50	25	0
t_4	bal_x = balx-10	sum = sum + bal_x	100	50	25	100
t_5	write (bal_x)	read(bal_y)	90	50	25	100
t_6	read (bal_z)	sum = sum + bal_y	90	50	25	150
t_7	bal_z = bal_z + 10		90	50	25	150
t_8	write (bal_z)		90	50	35	150
t_9	COMMIT	read (bal_z)	90	50	35	150
t_{10}		sum = sum + bal_z	90	50	35	185
t_{11}		COMMIT	90	50	35	185

El poblema que tenim és que sum val 185 quan hauria de valer 175 (90 + 50 + 35)

T2 no ha vist l'actualització que ha fet T1 sobre bal_x

Exercici

- Considera dos atributs (X, Y) a una base de dades i dos transaccions, T1 i T2.
  - T1 llegeix X i Y, i escriu sobre X.
  - T2 llegeix X I Y, i escriu sobre X i Y
- 1) Indica una seqüencia d'accions de T1 i T2 sobre X i Y que mostri un conflicte WR (Write-Read)
- 2) Indica una seqüencia d'accions de T1 i T2 sobre X i Y que mostri un conflicte RW (Read-Write)
- 3) Indica una seqüencia d'accions de T1 i T2 sobre X i Y que mostri un conflicte WR (Write-Write)
- Et pots basar en els tres problemes anteriors

Exercici

- Considera dos atributs (X, Y) a una base de dades i dos transaccions, T1 I T2.
  - T1 llegeix X i Y, i escriu sobre X.
  - T2 llegeix X i Y, I escriu sobre X i Y
- 1) Indica una seqüencia d'accions de T1 i T2 sobre X i Y que mostri un conflicte WR (Write-Read)

- T2: R(X), T2: R(Y), T2: W(X), **T1: R(X)**…
- T1:R(X) és una lectura corrupta perquè llegeix X abans que T2 hagi fet COMMIT (o ROLLBACK)

Exercici

- Considera dos atributs (X, Y) a una base de dades i dos transaccions, T1 I T2.
  - T1 llegeix X i Y, i escriu sobre X.
  - T2 llegeix X i Y, I escriu sobre X i Y
- 2) Indica una seqüencia d'accions de T1 i T2 sobre X i Y que mostri un conflicte RW (Read-Write)

- T2:R(X), T2: R(Y), T1: R(X), T1: R(Y), T1: **W(X),** T1: COMMIT
- Després de T1: W(X). T2 té una lectura inconsistent de X

Exercici

- Considera dos atributs (X, Y) a una base de dades i dos transaccions, T1 I T2.
  - T1 llegeix X i Y, i escriu sobre X.
  - T2 llegeix X i Y, I escriu sobre X i Y
- 3) Indica una seqüencia d'accions de T1 i T2 sobre X i Y que mostri un conflicte WR (Write-Write)

- T2: R(X), T2:R(Y), T1:R(X), T1: R(Y), T1: W(X), T1: COMMIT, T2: **W(X),** T2: COMMIT
- T2 sobreescriu X abans de que T1 hagi fet COMMIT

TEMA 6
TRANSACCIONS I CONCURRÈNCIA

Objectius

1) Conèixer concepte de transacció
2) Conèixer propietats ACID de les transaccions
3) Conèixer problemes importants relacionats amb la concurrència
4) **Conèixer concepte de serialització i planificació en sèrie i no en sèrie**
5) Conèixer tècnica de control de concurrència basada en bloqueig
6) Conèixer protocol 2PL de bloqueig / desbloqueig

Serialització i planificació

- Sembla que la solució als tres problemes anteriors passa per controlar l'accés a les dades i gestionar els *reads* i *writes*

- La solució més òbvia és que una transacció s'executi després d'una altra – però a on està la concurrència, llavors?

Serialització i planificació

- **Planificació**: seqüència d'operacions d'un conjunt de transaccions que preserva l'ordre de les operacions de cada transacció individual

- **Planificació en sèrie**: planificació en la que les operacions de cada transacció s'executen consecutivament sense operacions d'altres transaccions
  - T1 seguida de T2
  - T2 seguida de T1
  - L'ordre pot variar el resultat final. Per exemple, no és el mateix calcular el % d'interès d'un compte bancari abans o després d'ingressar o retirar diners

Serialització i planificació

- Una **planificació no en sèrie** és una planificació en la que les transaccions estan intercalades

- Una planificació (no en sèrie) és correcta si produeix el mateix resultat que alguna execució en sèrie. Llavors, la planificació és **serializable**

- Les tècniques de **control de concurrència** ens ajuden a assolir l'objectiu de la serialització

TEMA 6
TRANSACCIONS I CONCURRÈNCIA

Objectius

1) Conèixer concepte de transacció
2) Conèixer propietats ACID de les transaccions
3) Conèixer problemes importants relacionats amb la concurrència
4) Conèixer concepte de serialització i planificació en sèrie i no en sèrie
5) **Conèixer tècnica de control de concurrència basada en bloqueig**
6) Conèixer protocol 2PL de bloqueig / desbloqueig

Bloqueig

- És un procediment per controlar l'accés concurrent a les dades

- Quan una transacció accedeix a la base de dades, pot demanar un bloqueig per denegar l'accés a les dades que està manipulant – o vol manipular - a altres transaccions

- Tenim **bloqueig compartit** (llegir) i **bloqueig exclusiu** (escriure)

Bloqueig compartit i exclusiu

- Si una transacció té un bloqueig compartit, pot llegir però no escriure.
- Un bloqueig compartit no és exclusiu: N transaccions poden compatir un bloqueig compartit sobre el mateix element
  - Pensa que estem llegint – no és una operació tan crítica com escriure
- Si una transacció té un bloqueig exclusiu, pot llegir i escriure. Cap transacció pot accedir a l'element bloquejat

Procediment

- Transacció demana bloquejar un element
  - Si element no bloquejat, llavors es bloqueja
  - Si element està bloquejat, llavors el SGBD mira casos
- Si el bloqueig que es demana és compartit, i l'element té un bloqueig compartit, s'accepta
- Si l'element té un bloqueig exclusiu, la transacció ha d'esperar
- Una transacció bloqueja un element fins que el desbloqueja durant l'execució o termina

TEMA 6
TRANSACCIONS I CONCURRÈNCIA

Objectius

1) Conèixer concepte de transacció
2) Conèixer propietats ACID de les transaccions
3) Conèixer problemes importants relacionats amb la concurrència
4) Conèixer concepte de serialització i planificació en sèrie i no en sèrie
5) Conèixer tècnica de control de concurrència basada en bloqueig
6) **Conèixer protocol 2PL de bloqueig / desbloqueig**

2PL

2PL. Una transacción sigue el protocolo de bloqueo 2PL (en dos fases) si todas las operaciones de bloqueo preceden a la primera operación de desbloqueo.

Es diu que una transacció que segueix el protocol 2PL té dos fases

La fase en la que es fa gran: sol·licita i adquireix tots els bloquejos que necessita. No pot fer cap *unlock* durant aquesta fase

La fase en la que es fa petita: desbloquejos, però no pot demanar cap *lock*

Les fases habitualment s'alternen (lock- unlock – lock…)

2PL - comentari

- Si dos transaccions accedeixen a diferents elements, s'executen normalment
- Però si volen accedir al mateix element
  - En aquest protocol, si una transacció demana un bloqueig (compartit o exclusiu) sobre un element abans que una altra, la segona s'ha d'esperar fins que la primera acabi
  - S'executen en sèrie
  - Es diu que es un protocol "estricte"

2PL per prevenir el problema de l'actualització perduda

Tiempo	T_1	T_2	bal_x
t_1		BEGIN	100
t_2	BEGIN	**write_lock**(bal_x)	100
t_3	write_lock(bal_x)	read(bal_x)	100
t_4	**WAIT**	$bal_x = bal_x + 100$	100
t_5	**WAIT**	write (bal_x)	200
t_6	**WAIT**	COMMIT / UNLOCK (bal_x)	200
t_7	read(bal_x)		200
t_8	$bal_x = bal_x$-10		200
t_9	write (bal_x)		190
t_{10}	COMMIT / UNLOCK (bal_x)		190

2PL per prevenir el problema de l'actualització perduda

- T2 estableix un protocol exclusiu sobre bal_x ja que el vol modificar
- Quan T1 comença, sol·licita un bloqueig exclusiu sobre bal_x, però s'ha d'esperar
- Quan T2 finalitza, llavors T1 pot començar
- Llavors, el resultat és equivalent a fer T1 i T2 en sèrie
  - T1, T2: 100 – 10 + 100 = 190
  - T2, T1: 100 +100 – 10 = 190

2PL per prevenir el problema de la lectura corrupta

Tiempo	T_1	T_2	bal_x
t_1		BEGIN	100
t_2		**write_lock**(bal_x)	100
t_3		read(bal_x)	100
t_4	BEGIN	$bal_x = bal_x + 100$	100
t_5	write_lock(bal_x)	write (bal_x)	200
t_6	**WAIT**	**ROLLBACK / UNLOCK** (bal_x)	100
t_7	read(bal_x)		100
t_8	$bal_x = bal_x - 10$		100
t_9	write (bal_x)		90
t_{10}	COMMIT / UNLOCK (bal_x)		90

2PL per prevenir el problema de la lectura corrupta

- Com a resultat, tenim que el resultat és equivalent a fer T1, T2, i T2, T1, en sèrie
  - T1, T2 = 100 - 10 = 90
  - T2, T1 = 100 – 10 = 90

2PL per prevenir el problema de l'anàlisi inconsistent

Tiempo	T1	T2	bal_x	bal_y	bal_z	sum
t1		BEGIN	100	50	25	
t2	BEGIN	Sum = 0	100	50	25	0
t3	**write_lock**(bal_x)		100	50	25	0
t4	read(bal_x)	**read_lock**(bal_x)	100	50	25	0
t5	$bal_x = bal_x - 10$	**WAIT**	100	50	25	0
t6	write (bal_x)	**WAIT**	90	50	25	0
t7	write_lock(bal_z)	**WAIT**	90	50	25	0
t8	write(bal_z)	**WAIT**	90	50	25	0
t9	$bal_z = bal_z + 10$	**WAIT**	90	50	25	0
t10	write (bal_z)	**WAIT**	90	50	35	0
t11	**COMMIT / UNLOCK**	**WAIT**	90	50	35	0
t12		read(bal_x)	90	50	35	0
t13		sum += bal_x	90	50	35	90
t14		read_lock(bal_y)	90	50	35	90
t15		read(bal_y)	90	50	35	90
t16		sum += bal_y	90	50	35	140
t17		read_lock(bal_z)	90	50	35	140
t18		read(bal_z)	90	50	35	140
t19		sum += bal_z	90	50	35	175
t20		COMMIT / UNLOCK	90	50	35	175

Comentaris sobre el 2PL

- Adopta una aproximació pessimista, perquè retarden l'execució de les transaccions en cas de conflicte

- Té un problema: deadlock

deadlock

Tiempo	T1	T2
t_1	BEGIN	
t_2	**write_lock(bal_x)**	BEGIN
t_3	read(bal_x)	**write_lock(bal_y)**
t_4	$bal_x = bal_x - 10$	read(bal_y)
t_5	write (bal_x)	$bal_y = bal_y + 100$
t_6	**write_lock(bal_y)**	write (bal_y)
t_7	WAIT	**write_lock(bal_x)**
t_8	WAIT	WAIT
t_9	WAIT	WAIT
t_{10}	...	WAIT
T_{11}	...	...

deadlock

- Solucions?
  - Abortar alguna de les dues transaccions
  - Tenir un comptador (el temps d'espera serà X seconds; després, *abort* & *start*)

Exercici

- Considera dos atributs (X, Y) a una base de dades i dos transaccions, T1 I T2.
  - T1 llegeix X i Y, i escriu sobre X.
  - T2 llegeix X I Y, I escriu sobre X i Y
- 1) Indica una seqüencia d'accions de T1 i T2 sobre X i Y que mostri un conflicte WR (Write-Read)
- 2) Indica una seqüencia d'accions de T1 i T2 sobre X i Y que mostri un conflicte RW (Read-Write)
- 3) Indica una seqüencia d'accions de T1 i T2 sobre X i Y que mostri un conflicte WR (Write-Write)
- **Per a 1, 2 i 3, mostra com 2PL soluciona els conflictes**

Exercici

- 1) Indica una seqüencia d'accions de T1 i T2 sobre X i Y que mostri un conflicte WR (Write-Read)

- T2: R(X), T2: R(Y), T2: W(X), **T1: R(X)**.

- T1 no podria obtenir un bloqueig compartit sobre X perquè T2 tindria un bloqueig exclusiu sobre el mateix element. T1 hauria d'esperar fins que T2 alliberi X

Exercici

- 2) Indica una seqüencia d'accions de T1 i T2 sobre X i Y que mostri un conflicte RW (Read-Write)

- T2:R(X), T2: R(Y), T1: R(X), T1: R(Y), T1: **W(X)**

- T2 té un bloqueig sobre X, llavors T1 no pot demanar un nou bloqueig fins que T2 no l'alliberi

- Mateix raonament per 3).
- Recorda el comentari sobre 2PL que s'ha fet anteriorment