Transaccions a les bases de dades

- Un dels objectius més importants dels SGBD és permetre l'accés simultani de múltiples usuaris a la mateixa BD i preservar-ne la integritat
- Objectius:
  - Comprendre el problema que intenta resoldre la gestió de transaccions, evitant interferències entre usuaris que utilitzen el sistema simultàniament
  - Saber que és una transacció, quines propietats compleix i com s'utilitza
  - Comprendre les funcions que ha d'acomplir un SGBD en la gestió de transaccions
  - Conèixer el funcionament de les reserves, la tècnica més comuna per a controlar la concurrència
  - Tenir coneixements bàsics de com un SGBD pot evitar que es perdi informació o que se'n malmeti, mitjançant còpies de seguretat i estructures amb informació de canvis

Definició i propietats de les transaccions

- Una **transacció** és un conjunt d'operacions de lectura i/o actualització de la BD que acaba confirmant o cancel·lant els canvis que s'han dut a terme
- SQL estàndard ofereix les sentències següents per indicar l'inici i l'acabament de transaccions:
  - Inici Explícit: START TRANSACTION
  - Inici Implícit: qualsevol instrucció SQL
  - Acabament: COMMIT o ROLLBACK
- Tota transacció ha de complir les **propietats ACID**:
  - **A**tomicitat
  - **C**onsistència
  - A**Ï**llament
  - **D**efinitivitat

Garantir les propietats ACID de les transaccions no és sols missió de l'SGBD, sinó també de les aplicacions que l'utilitzen i del seu desenvolupador

Introducció

- En els SGBDs el concepte de transacció representa la unitat de treball a efectes de concurrència i d'integritat
- Exemple de transacció: imaginem que una aplicació d'una entitat bancària ofereix la possibilitat de transferir una certa quantitat de diners Q d'un compte origen a un compte destí

Transferència de quantitat *Q* de *CompteOrigen* a *CompteDestí*

Número operació	Instruccions
1	SaldoOrigen := llegir_saldo(CompteOrigen) Comprovar que és més gran o igual que *Q*
2	SaldoDestí := llegir_saldo(CompteDestí)
3	escriure_saldo(CompteOrigen, SaldoOrigen − *Q*)
4	escriure_saldo(CompteDestí, SaldoDestí + *Q*)
5	registrar_moviment("Transferència", CompteOrigen, CompteDestí, *Q*) /* Crear un registre per a anotar la transferència en una taula de moviments, posant-hi també la data i l'hora, per exemple */

A) Imaginem que un usuari comença a executar una d'aquestes transferències i just després del tercer pas, una apagada fa que el procés no acabi ⇒

les operacions que s'executen en fer la transferència s'han d'efectuar completament o no s'han d'efectuar en absolut

Introducció

I) Suposem que dos usuaris diferents (A i B) intenten fer dues transferències al mateix temps i al mateix compte destí. Imaginem que passarà si els passos de les transaccions s'executen concurrentment en l'ordre següent:

Execució concurrent de dues transferències			
Número operació	**Transferència usuari A (Q = 10)**	**Número operació**	**Transferència usuari B (Q = 20)**
1	SaldoOrigen := llegir_saldo(CompteOrigen1) Comprovar que és més gran o igual que 10		
2	SaldoDestí := llegir_saldo(CompteDestí)		
		1	SaldoOrigen := llegir_saldo(CompteOrigen2) Comprovar que és més gran o igual que 20
		2	SaldoDestí := llegir_saldo(CompteDestí)
3	escriure_saldo(CompteOrigen1, SaldoOrigen − 10)		
4	escriure_saldo(CompteDestí, SaldoDestí + 10)		
5	registrar_moviment("Transferència", CompteOrigen1, CompteDestí, 10)		
		3	escriure_saldo(CompteOrigen2, SaldoOrigen − 20)
		4	escriure_saldo(CompteDestí, SaldoDestí + 20)
		5	registrar_moviment("Transferència", CompteOrigen2, CompteDestí, 20)

És necessari impedir que l'accés concurrent de diversos usuaris produeixi resultats anòmals

Introducció

C) Imaginem que un error de programació de la funció de transferència fa que el saldo del compte destí rebi com a nou valor la quantitat que s'ha transferit, en lloc de sumar-la al saldo anterior

És missió dels dissenyadors/programadors que les transaccions verifiquin els requeriments dels usuaris

D) Plantegem-nos que passaria si, després d'utilitzar l'aplicació durant uns quants dies, i en un moment de plena activitat, es produeix un error fatal del disc en què s'emmagatzema la BD

Cal que hi hagi mecanismes per a evitar la pèrdua tant de les dades més antigues com de les actualitzacions més recents

Interferències entre transaccions

- Les interferències es produeixen si les transaccions no s'aïllen adequadament entre si. Distingim quatre grans tipus d'interferències:
  - Actualització perduda
  - Lectura no confirmada
  - Lectura no repetible
  - Anàlisi inconsistent i fantasmes
- **Actualització perduda**: aquesta interferència es produeix quan es perd el canvi que ha efectuat una operació d'escriptura:

Transacció T_1 (reintegrament de 10)	Transacció T_2 (reintegrament de 25)
Saldo := llegir_saldo(Compte)	
	Saldo := llegir_saldo(Compte)
escriure_saldo(Compte, Saldo − 10)	
	escriure_saldo(Compte, Saldo − 25)
COMMIT	
	COMMIT

Interferències entre transaccions

- **Lectura no confirmada**: aquesta situació es produeix quan una transacció llegeix una dada que ha estat modificada per una altra transacció que després avorta

Transacció T_1 (reintegrament de 10)	Transacció T_2 (reintegrament de 25)
	Saldo := llegir_saldo(Compte)
	escriure_saldo(Compte, Saldo − 25)
Saldo := llegir_saldo(Compte)	
escriure_saldo(Compte, Saldo − 10)	
COMMIT	
	ABORT

Interferències entre transaccions

- **Lectura no repetible**: aquesta situació es produeix si una transacció llegeix dues vegades la mateixa dada i obté valors diferents, a causa d'una modificació efectuada per una altra transacció

Transacció T_1 (lectura de saldo)	Transacció T_2 (reintegrament de 25)
Saldo := llegir_saldo(Compte)	
	Saldo := llegir_saldo(Compte)
	escriure_saldo(Compte, Saldo − 25)
	COMMIT
Saldo := llegir_saldo(Compte)	
COMMIT	

- **Anàlisi inconsistent i fantasmes**: els tres tipus d'interferències anteriors tenen lloc respecte a una única dada de la BD, però també es produeixen interferències respecte a la visió que dues transaccions tenen d'un conjunt de dades

Interferències entre transaccions

Fantasmes

Suposem que en executar dues transaccions es produeix la seqüència d'esdeveniments següents:

- T_1 llegeix tots els registres que verifiquen una condició C
- T_2 actualitza (o insereix) un registre, que no complia la condició C, de manera que passa a complir-la
- T_1 torna a llegir els registres inicials o alguna informació que en depèn

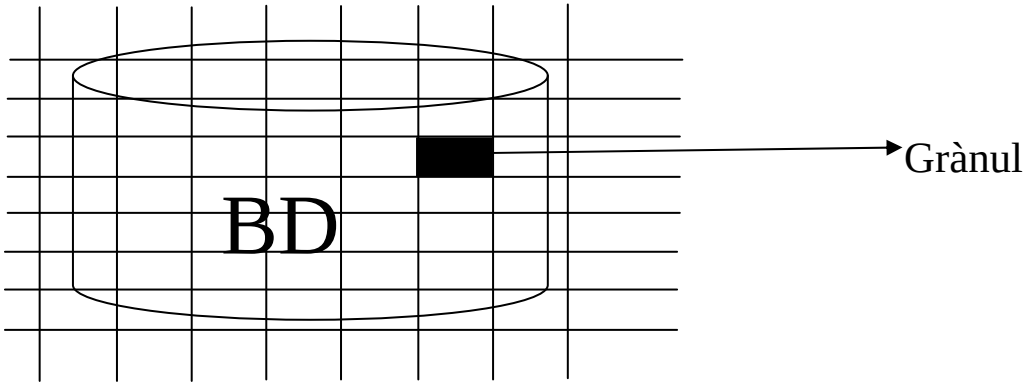
Transacció T_1 (llistat de comptes)	Transacció T_2 (creació de comptes)
Llegir tots els comptes del banc. Obtenim Compte1 i Compte2	
	crear_compte(Compte3)
	escriure_saldo(Compte3, 100)
Sumar els saldos de tots els comptes. Obtenim saldo de Compte1 + + saldo de Compte2 + 100 (El Compte3, amb saldo 100, és el fantasma)	
COMMIT	
	COMMIT

Serialitzabilitat

- La **teoria de la serialitzabilitat** defineix de manera precisa les condicions que s'han de complir per a considerar que les transaccions estan aïllades entre si correctament

  Important: la teoria de la serialitzabilitat assumeix que les transaccions sempre confirmen els seus resultats

- La serialitzabilitat considera que les transaccions estan formades per dos tipus molt senzills d'**accions** (operacions) sobre dades elementals anomenats **grànuls**:
  - Accions de lectura (R(G))i accions d'escriptura (W(G))
  - G és el gràNul, és a dir, la unitat de dades controlada individualment per l'SGBD (pot variar en funció del gestor): pàgina (bloc) de disc, un registre (una tupla) etc.



Serialitzabilitat

- L'execució concurrent d'un conjunt de transaccions (a on es preserva l'ordre d'accions dins de cada transacció) rep el nom **d'horari o història**:

# temps	T_1	T_2
1	R(A)	
2		R(A)
3	R(B)	
4		W(A)
5	R(C)	
6	commit	
7		commit

Serialitzabilitat

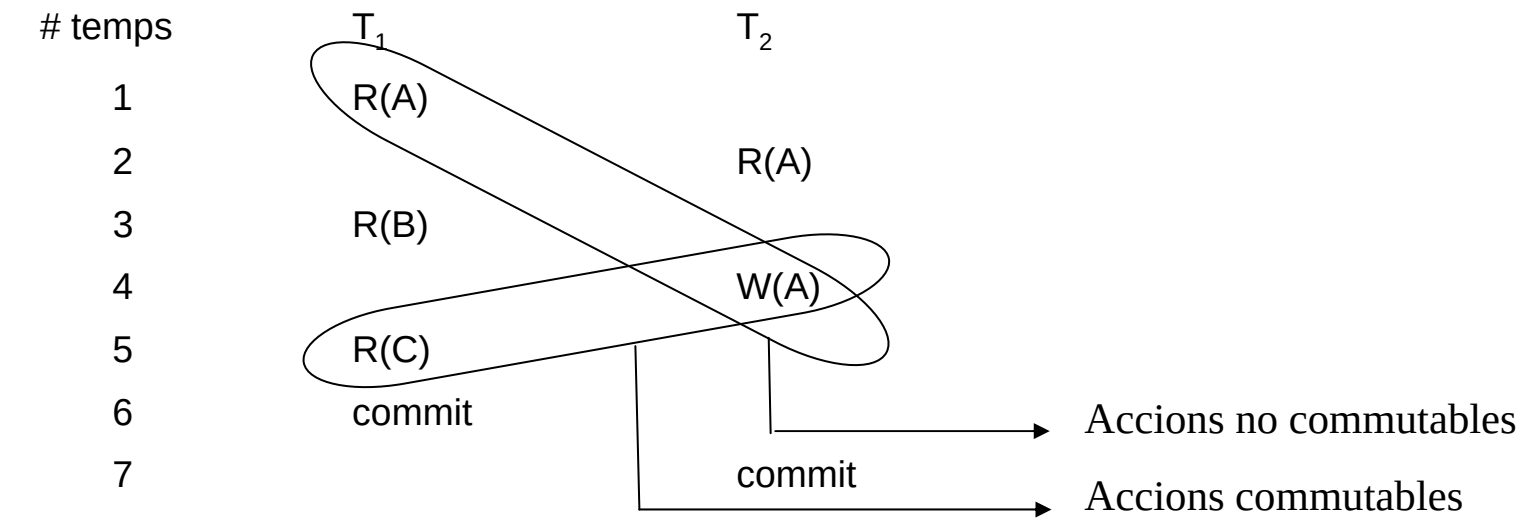
- Un **horari serial** és aquell a on no hi ha encavalcament entre les accions de les transaccions implicades:

# temps	T_1	T_2
1	R(A)	
2	R(B)	
3	R(C)	
4	commit	
5		R(A)
6		W(A)
7		commit

Donat un conjunt de ***n*** transaccions tindrem ***n!*** horaris serials possibles

Serialitzabilitat

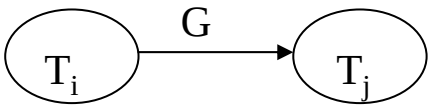
- En un horari, dues accions es consideren **accions conflictives** (**o accions no commutables**) si pertanyen a transaccions distintes i l'ordre en què s'executen pot afectar el valor del gränul que hagi llegit una de les transaccions o el valor final del gränul



Les situacions de conflicte només poden aparèixer
quan les dues accions operen sobre el **mateix gränul**
i almenys una de les dues accions és d'**escriptura**

Serialitzabilitat

- **Graf de precedències**



Si $\exists op_i(G) \in T_i \wedge \exists op_j(G) \in T_j$ t.q. $op_i \wedge op_j$ no són commutables i op_i s'ha executat abans que op_j

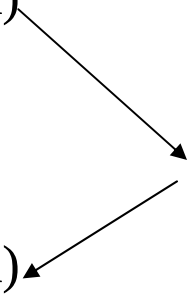
Actualització Perduda

# temps	T_1	T_2
1	R(A)	
2		R(A)
3	W(A)	
4		W(A)
5	commit	
6		commit

Serialitzabilitat

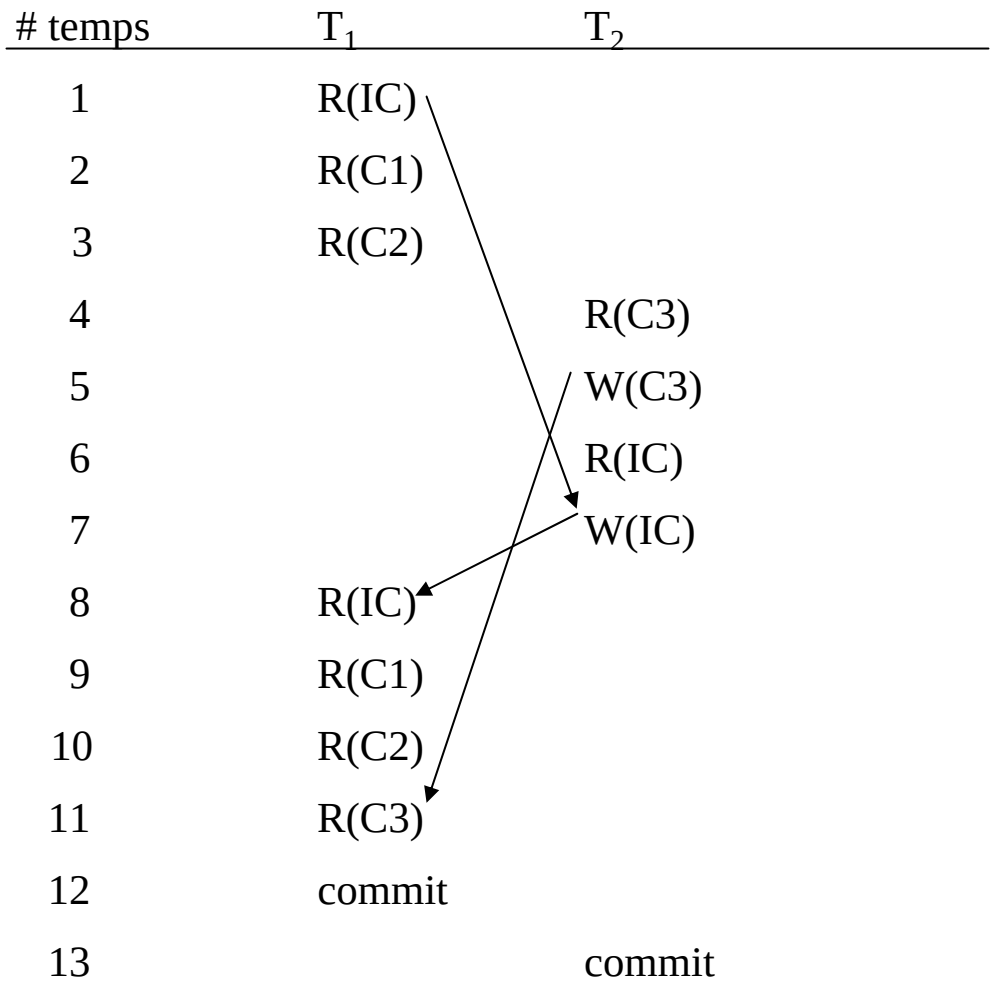
Lectura no repetible

# temps	T_1	T_2
1	R(A)	
2		R(A)
3		W(A)
4	R(A)	
5	commit	
6		commit



Serialitzabilitat

Fantasmes



Serialitzabilitat

- Un horari (història) es considera correcte si l'ordre relatiu de tots els parells d'accions no commutables és el mateix que en algun horari serial. En aquest cas, el graf de precedències no té cicles

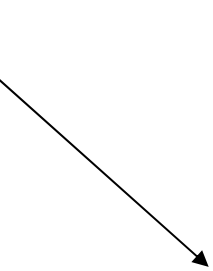
  ⇓

  Els horaris correctes s'anomenen **horaris serialitzables**

  ⇓

  Un horari serialitzable **sempre** produeix **el mateix resultat** que algun horari serial

# temps	T_1	T_2
1	R(A)	
2		R(A)
3	R(B)	
4		W(A)
5	R(C)	
6	commit	
7		commit

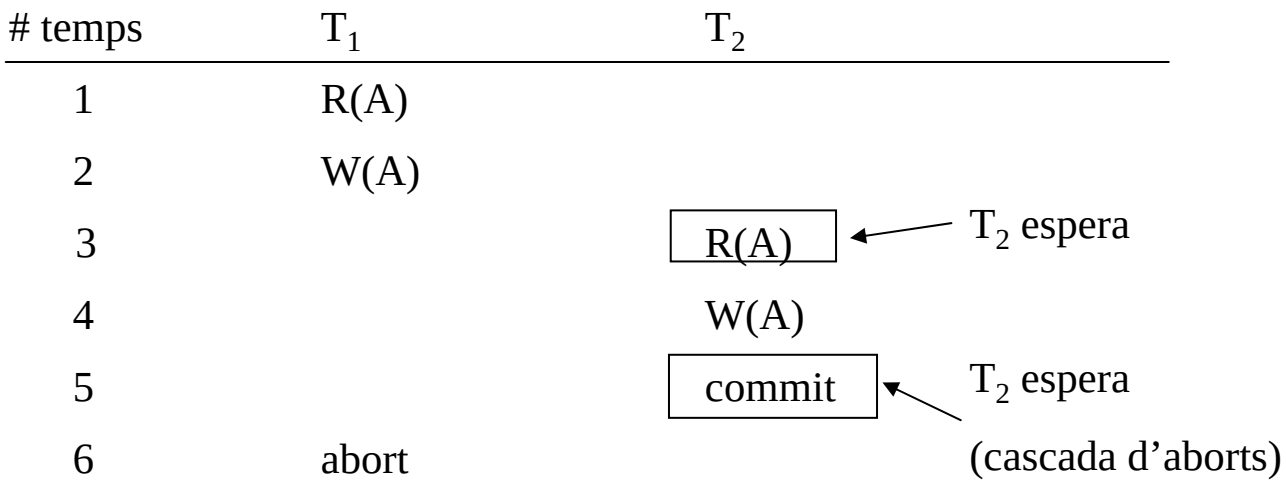


Horari serialitzable? **Sí**

Horari serial equivalent? **T_1; T_2**

Recuperabilitat

- Hem vist que algunes interferències es produeixen quan cancel·lem transaccions. Cancel·lar una transacció suposa desfer-ne tots els canvis i recuperar el valor anterior dels grànuls
- La serialitzabilitat ignora la possibilitat de cancel·lacions. Per tant, per evitar les interferències que provoquen hem d'exigir noves condicions a l'execució de les transaccions
- Un horari compleix el **criteri de recuperabilitat** si cap transacció T_2 que llegeix o escriu un grànul escrit per una altra transacció T_1 confirma sense que abans ho hagi fet T_1



Tècniques per al control de concurrència

- Un SGBD pot resoldre les interferències entre transaccions de dues maneres possibles:
  - Cancel·lar automàticament transaccions problemàtiques i desfer-ne els canvis
  - Suspendre'n l'execució temporalment i reprendre-la quan desaparegui el perill d'interferència.

$$\Downarrow$$

disminució en el nivell de paral·lelisme o nivell de concurrència del sistema

- El **nivell de paralel·lisme** és la feina efectiva realitzada, és a dir, la feina realment útil per als usuaris, efectuada per unitat de temps

Control de concurrència amb reserves

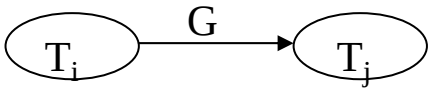
- Els SGBD garanteixen les propietats ACID de les transaccions gràcies a l'acció conjunta de dues famílies de protocols:
  - Protocols per al control de concurrència
  - Protocols de recuperació
- La tècnica més utilitzada per dur a terme el control de concurrència és la tècnica de reserves
- La idea bàsica de l'ús de reserves és que una transacció ha d'obtenir una **reserva d'un grànul amb una certa modalitat** abans de poder efectuar accions (lectures o escriptures) sobre el grànul
- Una transacció T pot demanar una reserva sobre un grànul G si executa l'acció LOCK(G,m) a on m és una modalitat que permet executar les accions desitjades sobre G
- Distingirem dues modalitats:
  - Modalitat compartida (S, ***S****hared*): permet fer lectures
  - Modalitat exclusiva (X, *e****X****clusive*): permet fer lectures i escriptures
- Per alliberar una reserva d'un grànul G, caldrà que la transacció T executi l'acció UNLOCK(G)

Control de concurrència amb reserves

- Quan una transacció demana una reserva, l'SGBD decideix si la pot concedir, cosa que farà si el tipus de reserva que se li demana no és incompatible amb cap de les reserves ja atorgades a altres transaccions sobre el mateix gránul

	Compartida (S)	Exclusiva (X)
Compartida (S)	Sí	No
Exclusiva (X)	No	No

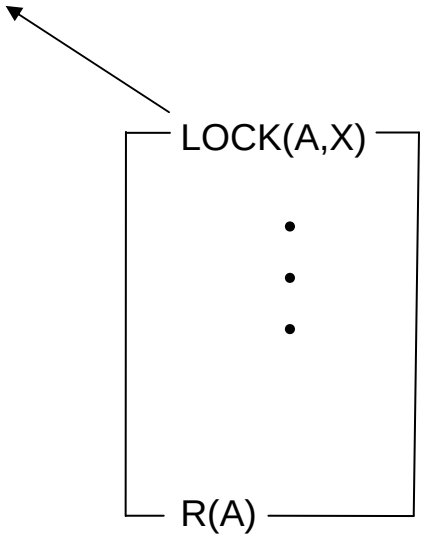
- Si una reserva no es pot concedir, se suspèn l'execució de la transacció. Sempre que s'allibera una reserva d'un gránul, l'SGBD mira si pot reprendre l'execució d'alguna transacció que s'hagi suspès a causa d'aquell gránul
- **Graf d'espera**



Si T_i està esperant a adquirir una reserva sobre un gránul G que ha estat concedida a T_j

Control de concurrència amb reserves

# temps	T_1	T_2
1	LOCK(A, S)	
2	R(A)	
3		LOCK(A,X)
4	LOCK(B,S)	
5	R(B)	
6	LOCK(C,S)	
7	R(C)	
8	UNLOCK(A)	
9		R(A)
10		W(A)
11	UNLOCK(B)	
12	UNLOCK(C)	
13		UNLOCK(A)
14	commit	
15		commit



Control de concurrència amb reserves

- Tot i que els serveis de petició i alliberament de reserves són la base sobre la qual es construeix el control de concurrència, per si mateixos no garanteixen l'aïllament de les transaccions:

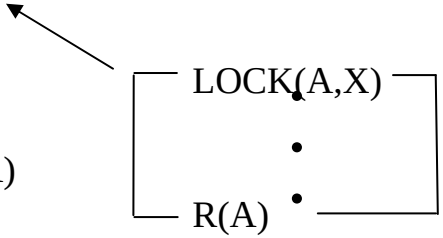
# temps	T_1	T_2
1	LOCK(A,S)	
2	R(A)	
3	UNLOCK(A)	
4		LOCK(A,X)
5		R(A)
6		W(A)
7		UNLOCK(A)
8	LOCK(A,S)	
9	R(A)	
10	UNLOCK(A)	
11	commit	
12		commit

La interferència (lectura no repetible) es produeix igualment!!!

Control de concurrència amb reserves

- Per aconseguir l'aïllament de les transaccions, les transaccions han de seguir unes certes normes a l'hora de demanar i alliberar reserves
- Una transacció segueix el **protocol de reserves en dues fases** (PR2F) si reserva qualsevol gränul en la modalitat adequada abans d'operar-hi, i mai no adquireix una reserva de qualsevol gränul després d'haver-ne alliberat qualsevol altra abans.
- **Teorema:** Si les transaccions segueixen el PR2F i fan commit ⇔ Horari serialitzable

# temps	T_1	T_2
1	LOCK(A,S)	
2	R(A)	
3		LOCK(A,X)
4	R(A)	
5	UNLOCK(A)	
6		R(A)
7	commit	
8		W(A)
9		UNLOCK(A)
10		commit



Horari sense interferències (horari serialitzable) amb horari serial equivalent T_1; T_2

Control de concurrència amb reserves

- Parlem ara de recuperabilitat

# temps	T_1	T_2
1	LOCK(A,X)	
2	R(A)	
3	W(A)	
4	UNLOCK(A)	
5		LOCK(A,S)
6		R(A)
7		UNLOCK(A)
8		commit
9	abort	

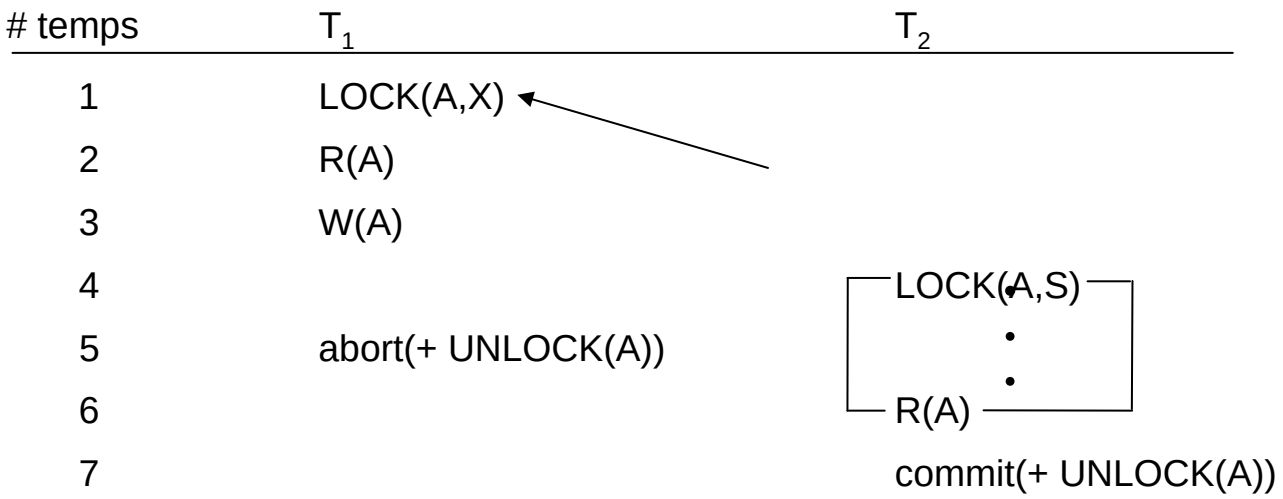
Malgrat que T_1 i T_2 segueixen el PR2F la interferència (lectura no confirmada) es produeix igualment si deixem que T_2 confirmi els seus resultats $\Rightarrow$ necessitem que les reserves es mantinguin fins l'acabament de les transaccions

$\Downarrow$

Protocol de reserves en dues fases estricte

(PR2F+ reserves fins l'acabament de les transaccions)

Control de concurrència amb reserves



Aplicació del protocol de reserves en dues fases estricte

Relaxació del nivell d'aïllament

- Hem assumit que és necessari que l'SGBD garanteixi la serialitzabilitat i la recuperabilitat de les transaccions, proporcionant una protecció total respecte davant de qualsevol tipus d'interferència
- Les principals conseqüències d'aquest fet són:
  - Δ sobrecàrrega de l'SGBD en termes de gestió d'informació de control
  - ∇ del nivell de paralel·lisme
- En determinades circumstàncies és convenient relaxar el nivell d'aïllament i possibilitar que es produeixin interferències; això és correcte si se sap que aquestes interferències no es produiran realment o si no és important que es produeixin
- L'SQL estàndard ens permet relaxar el nivell d'aïllament de la manera següent:

  SET TRANSACTION mode_accés

  ISOLATION LEVEL nivell_aïllament

  a on:
  - Mode d'accés: READ ONLY o bé READ WRITE
  - Nivell d'aïllament: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ o bé SERIALIZABLE

Relaxació del nivell d'aïllament

	Actualització perduda	Lectura no confirmada	Lectura no repetible i anàlisi inconsistent (tret de fantasmes)	Fantasmes
READ UNCOMMITTED	Sí	No	No	No
READ COMMITTED	Sí	Sí	No	No
REPEATABLE READ	Sí	Sí	Sí	No
SERIALIZABLE	Sí	Sí	Sí	Sí

- READ UNCOMMITTED protegeix les dades actualitzades, evitant que cap altra transacció les actualitzi, fins que acaba la transacció
- READ COMMITTED protegeix parcialment les lectures, impedint que una altra transacció llegeixi dades que encara no s'han confirmat
- REPEATABLE READ impedeix, fins que acaba la transacció, que una altra transacció actualitzi una dada que s'hagi llegit
- SERIALIZABLE ofereix aïllament total i evita qualsevol tipus d'interferència incloent-hi els fantasmes

Reserves i nivells d'aïllament

Actualització perduda

T_1	T_2
R(A)	
	R(A)
W(A)	
	W(A)
commit	
	commit

Lectura no repetible

T_1	T_2
R(A)	
R(A)	
	W(A)
R(A)	
commit	
	commit

Lectura no confirmada

T_1	T_2
R(A)	
W(A)	
	R(A)
	commit
abort	

Fantasmes

T_1	T_2
R(IC)	
R(C1)	
R(C2)	
	R(C3)
	W(C3)
	R(IC)
	W(IC)
R(IC)	
R(C1)	
R(C2)	
R(C3)	
commit	
	commit

READ UNCOMMITTED Reserves X fins l'acabament de la transacció. No calen reserves S per lectura

Reserves i nivells d'aïllament

Actualització perduda

T_1	T_2
R(A)	
	R(A)
W(A)	
	W(A)
commit	
	commit

Lectura no repetible

T_1	T_2
R(A)	
R(A)	
	W(A)
R(A)	
commit	
	commit

Lectura no confirmada

T_1	T_2
R(A)	
W(A)	
	R(A)
	commit
abort	

Fantasmes

T_1	T_2
R(IC)	
R(C1)	
R(C2)	
	R(C3)
	W(C3)
	R(IC)
	W(IC)
R(IC)	
R(C1)	
R(C2)	
R(C3)	
commit	
	commit

READ COMMITTED Reserves X fins l'acabament de la transacció. Reserves S fins després de la lectura

Reserves i nivells d'aïllament

Actualització perduda

T_1	T_2
R(A)	
	R(A)
W(A)	
	W(A)
commit	
	commit

Lectura no repetible

T_1	T_2
R(A)	
R(A)	
	W(A)
R(A)	
commit	
	commit

Lectura no confirmada

T_1	T_2
R(A)	
W(A)	
	R(A)
	commit
abort	

Fantasmes

T_1	T_2
R(IC)	
R(C1)	
R(C2)	
	R(C3)
	W(C3)
	R(IC)
	W(IC)
R(IC)	
R(C1)	
R(C2)	
R(C3)	
commit	
	commit

REPEATABLE READ Reserves X i S fins l'acabament de la transacció

Reserves i nivells d'aïllament

Actualització perduda

T_1	T_2
R(A)	
	R(A)
W(A)	
	W(A)
commit	
	commit

Lectura no repetible

T_1	T_2
R(A)	
R(A)	
	W(A)
R(A)	
commit	
	commit

Lectura no confirmada

T_1	T_2
R(A)	
W(A)	
	R(A)
	commit
abort	

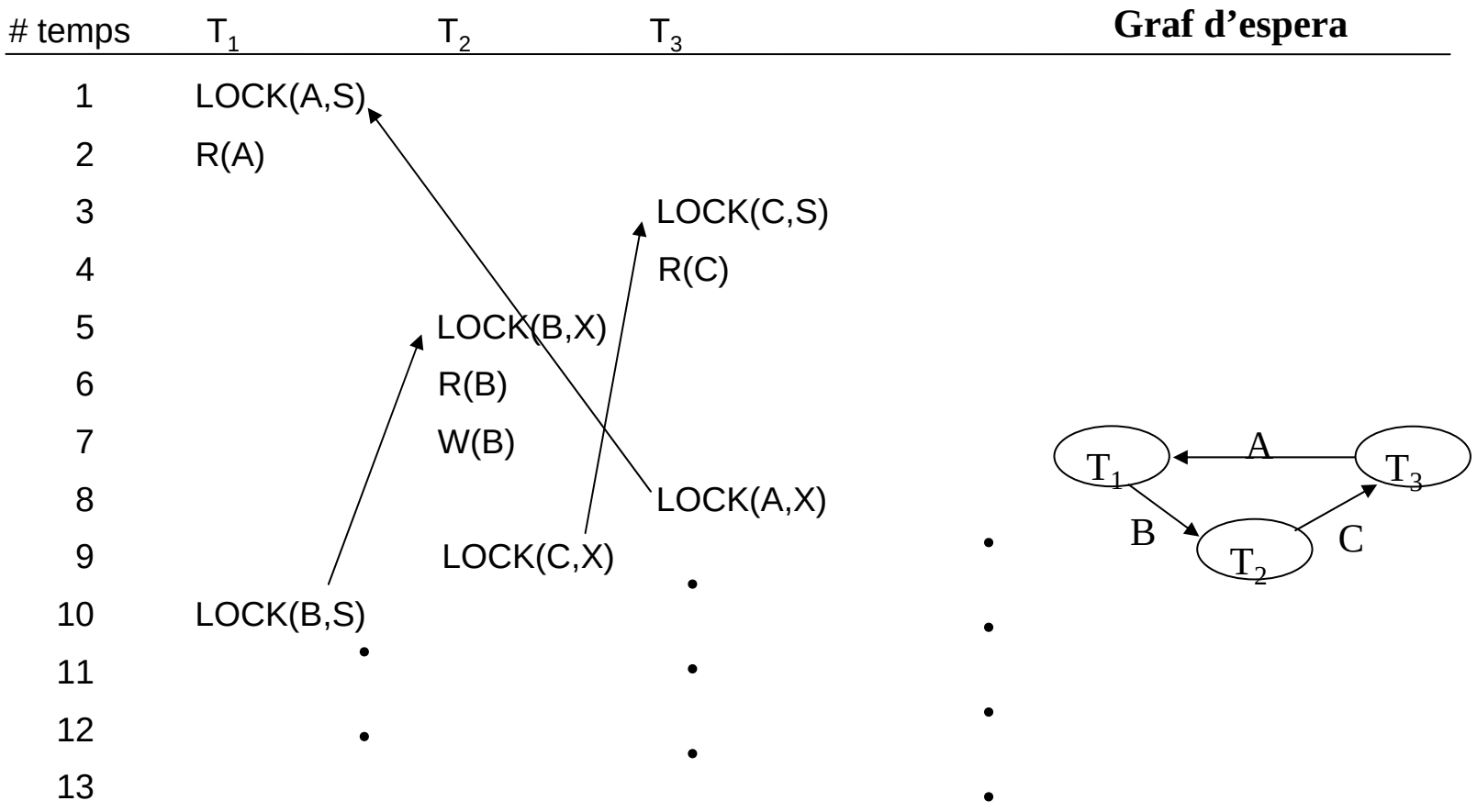
Fantasmes

T_1	T_2
R(IC)	
R(C1)	
R(C2)	
	R(C3)
	W(C3)
	R(IC)
	W(IC)
R(IC)	
R(C1)	
R(C2)	
R(C3)	
commit	
	commit

SERIALIZABLE Totes les reserves fins l'acabament de les transaccions (incloent reserves d'informació de control)

Control de concurrència amb reserves

- Problema inherent a les tècniques basades en reserves $\Rightarrow$ possibilitat de que es produeixin **esperes indefinides** (abraçades mortals)



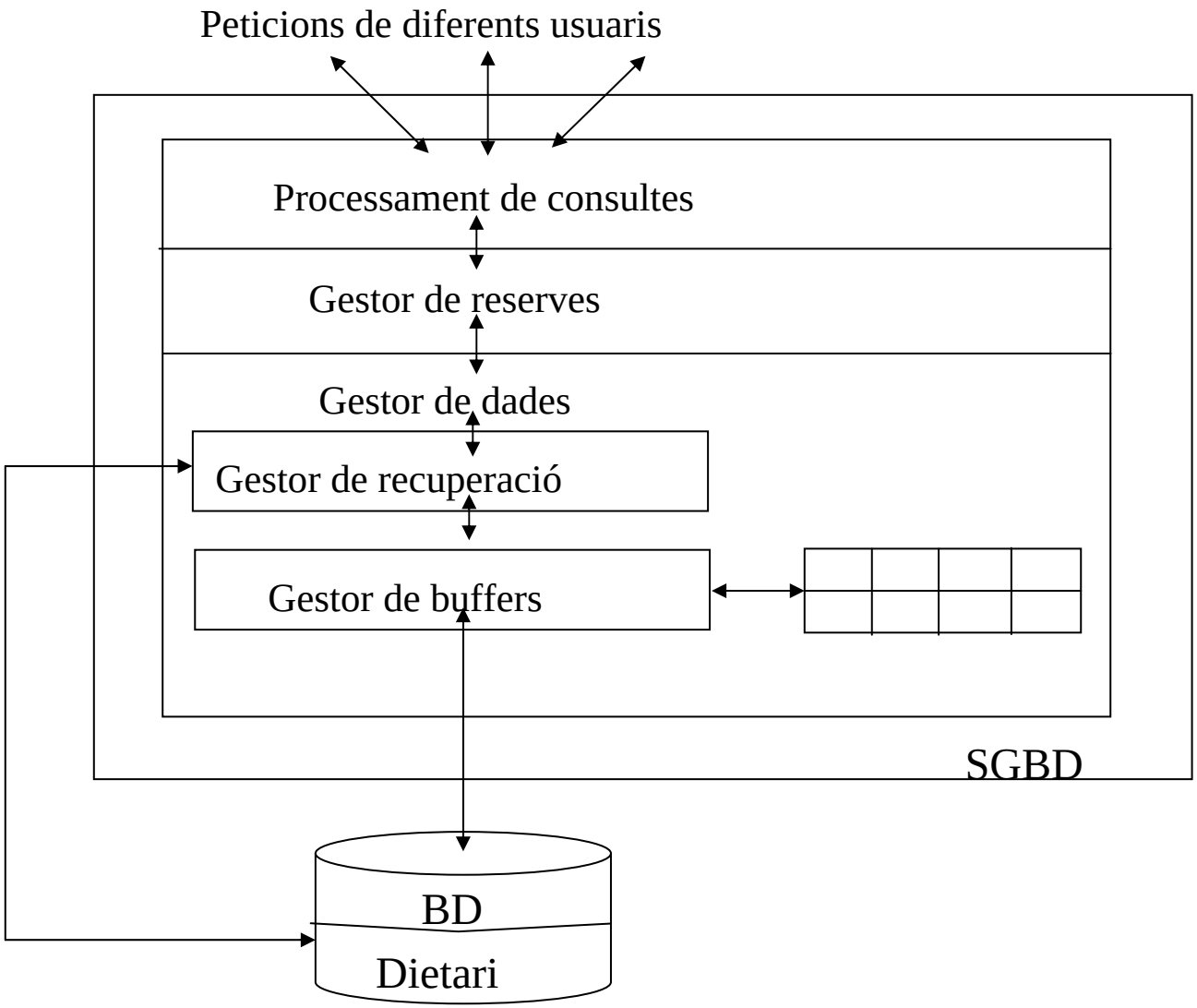
Control de concurrència amb reserves

- Davant de la possibilitat que es produeixin abraçades mortals, els SGBD basats en reserves han d'optar per una de les tres possibilitats següents:
  - Prevenir-les abans que es produeixin
  - **Detectar-les i resoldre-les una vegada s'hagin produït**
  - Definir un temps d'espera màxim que, un cop superat, faci que es cancel·li automàticament la transacció
- Un SGBD detecta les abraçades mortals buscant cicles en el graf d'espera. Aquesta cerca la pot fer en moments diferents:
  - Sempre que una transacció demana una reserva i no l'obté immediatament
  - Cada cert temps
  - Quan tenim transaccions que triguen massa temps en acabar
- Un cop que l'SGBD detecta l'abraçada mortal, la única cosa que pot fer és trencar el cicle cancel·lant una o diverses de les transaccions implicades

Recuperació

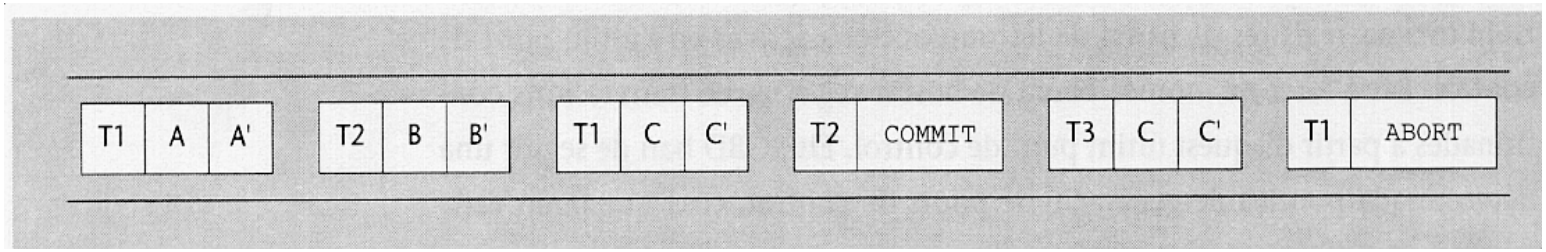
- Els protocols de recuperació de l'SGBD han de garantir l'atomicitat i la definitivitat de les transaccions.
- Objectius: mai no es poden perdre els canvis efectuats per transaccions confirmades i mai s'han de mantenir els canvis efectuats per transaccions avortades
- Situacions que posen en perill els objectius anteriors:
  - La cancel·lació voluntària d'una transacció a petició de l'aplicació
  - La cancel·lació involuntària d'una transacció
  - Una caiguda del sistema, que provocaria la cancel·lació de totes les transaccions actives
  - La destrucció total o parcial de la BD a causa de desastres o fallades de dispositius
- En funció de la situació distingim dues parts de la recuperació:
  - La **restauració**, que garanteix l'atomicitat i la definitivitat davant de cancel·lacions de les transaccions i de caigudes del sistema
  - La **reconstrucció**, que recupera l'estat de la BD davant una pèrdua total o parcial de la informació emmagatzemada a disc a causa de fallades o desastres

Recuperació



Restauració

- La restauració ha de ser capaç d'efectuar dos tipus d'operacions:
  - La **restauració cap enrere**, que implica **desfer** els canvis d'una transacció avortada
  - La **restauració cap endavant**, que implica **refer** els canvis d'una transacció confirmada
- Per a desfer i refer canvis l'SGBD utilitza una estructura de dades amb informació dels canvis, el **dietari** (en anglès **log**), que guarda informació dels canvis que han efectuat les transaccions, i de les cancel·lacions i de les confirmacions d'aquestes:



Dietari

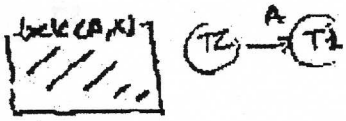
Reconstrucció

- Per a poder reconstruir l'estat d'una BD després d'una pèrdua parcial o total de les dades, cal utilitzar dues fonts d'informació:
  - Una **còpia de seguretat** (bolcat) que contingui un estat correcte de la BD
  - El contingut del dietari a partir del moment en què es va fer la còpia de seguretat

- Podem fer una classificació de les còpies de seguretat en funció de les característiques següents:
  - Les còpies de seguretat poden ser estàtiques o dinàmiques
  - Les còpies de seguretat poden ser completes o incrementals

Actualització perduda

T1	T2
lock(A,X)	
R(A)	
W(A)	lock(A,X)
Commit	
(unlock(A))	R(A)
	W(A)
	Commit
	(unlock(A))



S'EVITA!

HS = ? T1, T2

Lectura no confirmada

T1	T2
lock(A,X)	
R(A)	
W(A)	
	R(A)
	Commit
abort (unlock(A))	

NO S'EVITA!

Lectura no repetible

T1	T2
R(A)	
	lock(A,X)
	R(A)
	W(A)
R(A)	
commit	
	Commit (unlock(A))

NO S'EVITA!

Fantasmes

T1	T2
R(C1C)	
R(C1)	
R(C2)	
	lock(C3,X)
	R(C3)
	W(C3)
	R(1C)
	W(1C)
R(C1C)	
R(C1)	
R(C2)	
R(C3)	
Commit	
	Commit (unlock(C3))

NO S'EVITA!

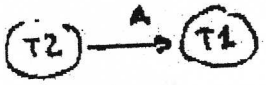
READ UNCOMMITED

ACPD COMMITED

Actualització perduda S'EVITA!

Lectura no confirmada

T1	T2
lock(A,X)	
R(A)	
W(A)	
	lock(A,S)
unlock(A) abort	
	R(A)
	commit



S'EVITA! Rise? T2

Lectura no repetible

T1	T2
lock(A,S)	
R(A)	
unlock(A)	
	lock(A,X)
	R(A)
	W(A)
lock(A,S)	
	commit (unlock(A))
R(A)	
commit	

NO S'EVITA!

Fantasmes

T1	T2
R(IC)	
lock(C1,S)	
R(C1)	
unlock(C1)	
lock(C2,S)	
R(C2)	
unlock(C2)	
	lock(C3,X)
	R(C1)
	W(C3)
	R(IC)
	W(IC)

NO S'EVITA!

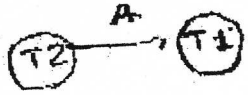
R(IC)	
lock(C1,S)	
R(C1)	
unlock(C1)	
lock(C2,S)	
R(C2)	
unlock(C2)	
lock(C3,S)	
	Commit (unlock(C3))
R(C3)	
unlock(C3)	
commit	

Actualització perduda — S'EVITA — REPETABLE READ

Lectura no confirmada — S'EVITA

Lectura no repetible

T1	T2
lock(A,S)	
R(A)	
	lock(A,X)
R(A)	
commit (unlock(A))	
	R(A)
	W(A)
	commit (unlock(A))

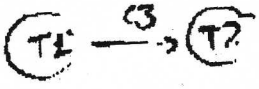


S'EVITA! HSE? T1; T2

Fantasmes

T1	T2
R(C)	
lock(C6,S)	
R(C6)	
lock(C2,S)	
R(C2)	
	lock(C3,X)
	R(C3)
	W(C3)
	R(C)
	W(C)
R(C)	
R(C6)	
R(C2)	
lock(C3)	
commit (unlock(C3))	
R(C3)	
commit (+unlock)	

NO S'EVITA!



SERIALITZABLE

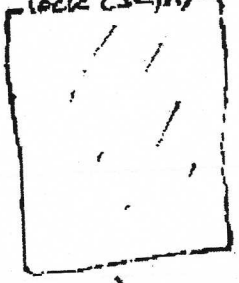
Actualització perduda S'EVITA

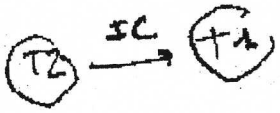
Lectura no confirmada S'EVITA

Lectura no repetible S'EVITA

Fantasmes

T1	T2
lock (IC, S)	
R(IC)	
lock (C1, S)	
R(C1)	
lock (C2, S)	
R(C2)	
	lock (C3, X)
	R(C3)
	W(C3)
	lock (IC, X)
R(IC)	
R(C1)	
R(C2)	
(unlock's) Commit	
	R(IC)
	W(IC)
	commit (unlock's)





¡SEVITA!
¿HSE? T1; T2