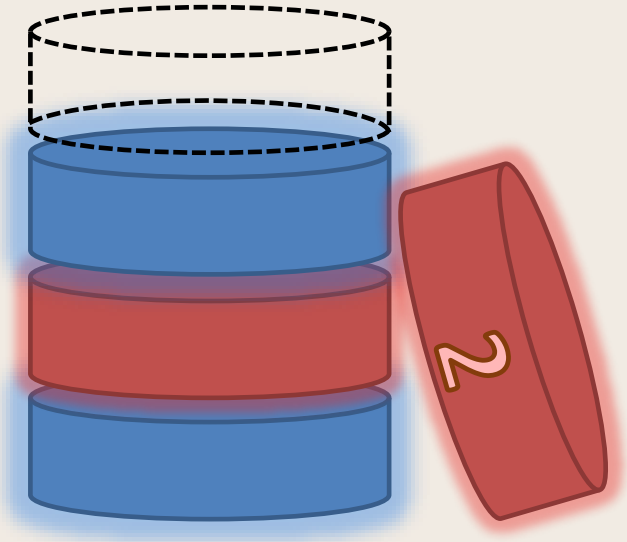
Planificarea Tranzacțiilor
Gestionarea Concurenței



Planificarea tranzacțiilor

O *planificare* reprezintă ordonarea secvențială a instrucțiunilor

(*Read* / *Write* / *Abort* / *Commit*)

a *n* tranzacții astfel încât

ordinea instrucțiunilor

fiecărei tranzacții se păstrează

Planificarea tranzacțiilor

T1:	T2:
read(A)	
read(sum)	
	read(A)
	A := A + 20
	write(A)
	commit
read(A)	
sum := sum + A	
write(sum)	
commit	

Schedule

read1(A)
read1(sum)
read2(A)

write2(A)
commit2
read1(A)

write1(sum)
commit1

Planificarea tranzacțiilor

- *Planificare serială*: este planificarea ce nu intercalează acțiuni ale mai multor tranzacții.

T1:	T2:
	read(A)
	A := A + 20
	write(A)
	commit
read(A)	
read(sum)	
read(A)	
sum := sum + A	
write(sum)	
commit	

- *Planificare non-serială*: acțiunile mai multor tranzacții concurente se interpătrund.

Planificarea tranzacțiilor

- *Planificări echivalente*: Pentru orice stare a bazei de date, efectul (asupra obiectelor bazei de date) al executării unei planificări este identic cu efectul executării celei de-a doua planificări.

- *Planificări serializabile*: este o planificare non-serială care este echivalentă cu o planificare de execuție serială a tranzacțiilor implicate. (Notă: Dacă fiecare dintre tranzacțiile implicate în planificare păstrează consistența bazei de date atunci fiecare planificare serializabilă va păstra consistența acesteia)

Serializabilitate

- Obiectivul *serializabilității* este găsirea unei planificări non-seriale care permite execuția concurentă a tranzactiilor fără ca acestea să interfereze, şi astfel să conducă la o stare a unei baze de date la care se poate ajunge şi printr-o execuție serială.

- Garantarea serializabilității tranzacțiilor concurente este importantă deoarece previne apariția inconsistențelor generate de interferența tranzițiilor.

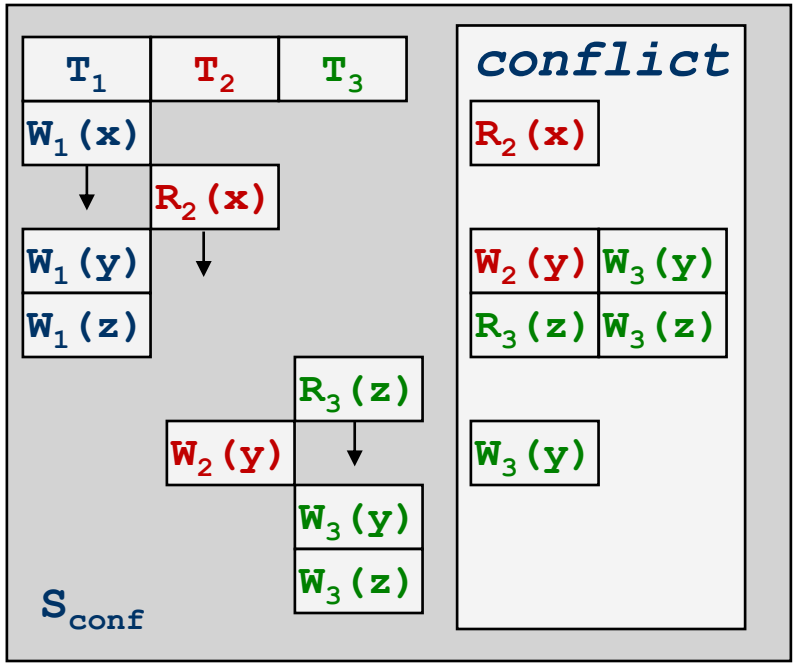
Planificarea tranzacțiilor

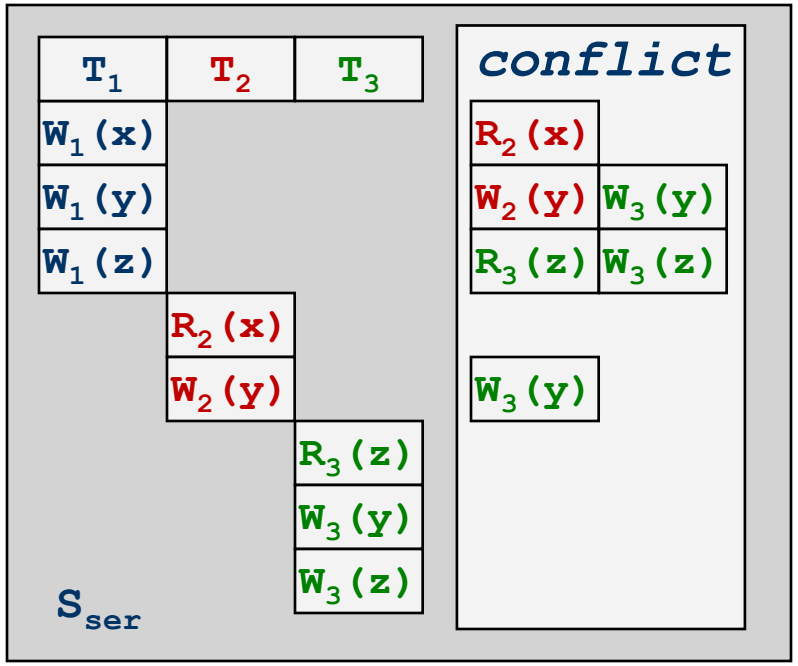
- Verificarea serializabilității: care sunt acțiunile ce nu se pot interschimba într-o tranzacție?
  - Acțiunile aparținând aceleiaşi tranzacții
  - Acțiuni aplicate de diferite tranzacții *aceluiaşi obiect,* dacă cel puțin una dintre ele este o operație de **write**. (acțiuni conflictuale!)

Planificarea tranzacțiilor

- 2 planificări sunt *conflict-echivalente* dacă:
  - Implică acțiunile aceloraşi tranzacții
  - Fiecare pereche de acțiuni conflictuale este ordonată în acelaşi mod
- Planificarea S este *conflict serializabilă* dacă S e conflict echivalentă cu o planificare serială

Conflict-serializabilitate





Planificare serială

Conflict-serializabilitate

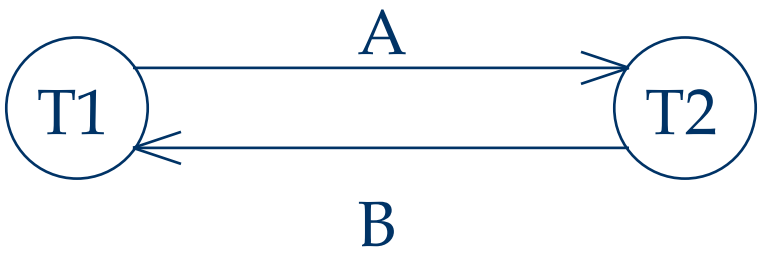
- Graf de precedență:
  - Graf orientat
  - Un nod per tranzacție
  - Arc între T_i şi T_j dacă o acțiune/operație de citire/modificare din T_j se realizează după o acțiune/operație conflictuală din T_i.

- Teoremă : O planificare este conflict-serializabilă dacă şi numai dacă graful său de precedendență nu conține circuite

Exemplu

- Planificare ce nu este conflict-serializabilă:

T1: R(A), W(A), R(B), W(B)
T2: R(A), W(A), R(B), W(B)



Graf de precedență

- Graful conține un circuit. Rezultatul lui T1 depinde de T2, şi invers.

Algoritm de Testare a Conflict-Serializabilității lui **S**

1. Pentru fiecare tranzacție T_i din **S** de crează un nod etichetat T_i în graful de precedență.
2. Pentru fiecare **S** unde T_j execută un Read(x) după un Write(x) executat de T_i crează un arc (T_i,T_j) în graful de precedență
3. Pentru fiecare caz în **S** unde T_j execută un Write(x) după un Read(x) executat de T_i crează un arc (T_i,T_j) în graful de precedență
4. Pentru fiecare caz în **S** unde T_j execută un Write(x) după un Write(x) executat de T_i crează un arc (T_i,T_j) în graful de precedență
5. **S** este conflict serializabil ddacă graful de precedență nu are circuite

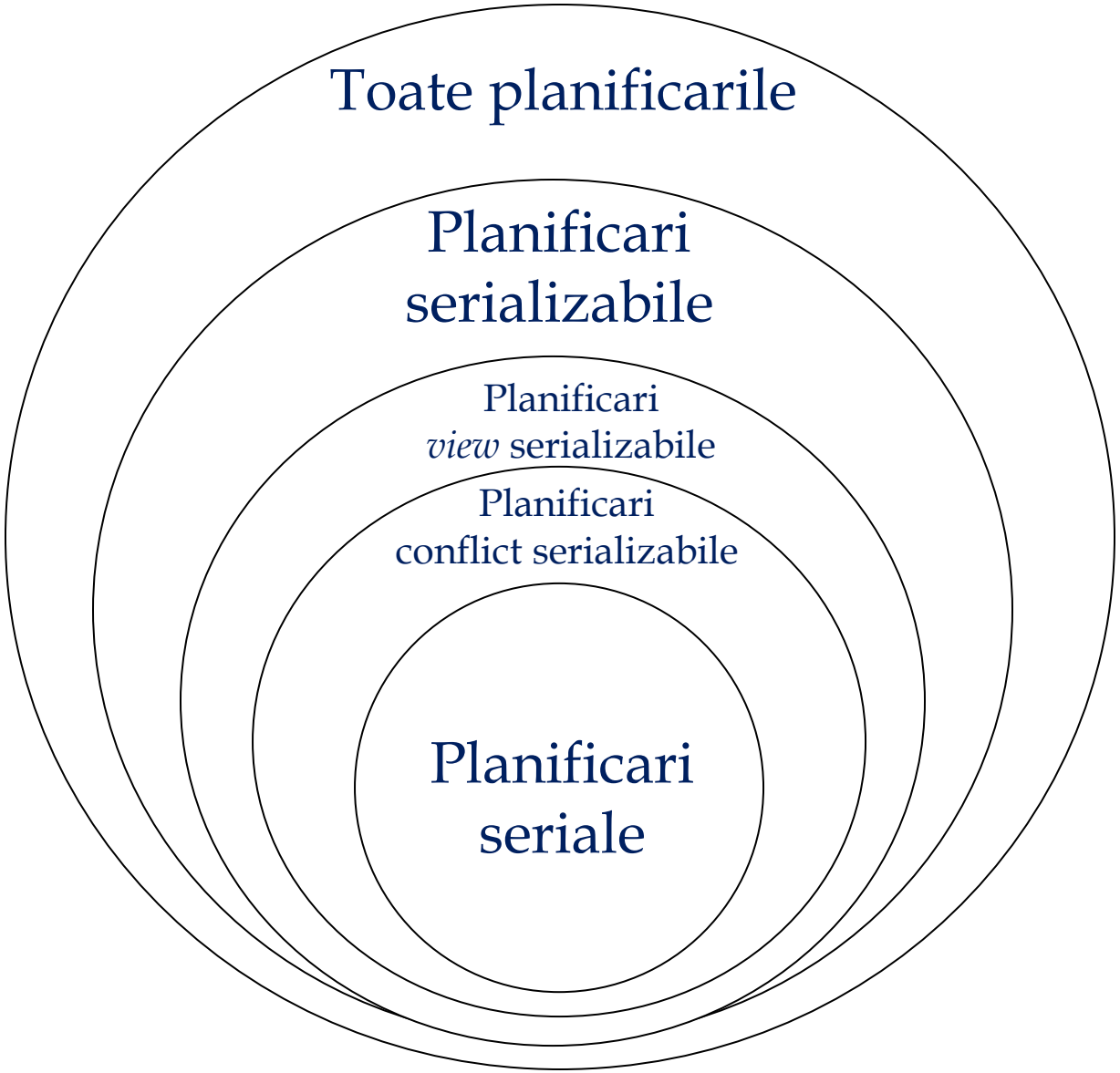
view - serializabilitate

- Planificările S_1 şi S_2 sunt view-echivalente :
  - Dacă T_i citeşte valoarea inițială a lui A in S_1, atunci T_i de asmenea citeşte valoarea inițială a lui A in S_2
  - Dacă T_i citeşte valoarea lui A modificata de T_j in S_1, atunci T_i de asmenea citeşte valoarea lui A modificata de T_j in S_2
  - Dacă T_i modifica valoarea finala a lui A in S_1, atunci T_i de asemenea modifica valoarea finala a lui A in S_2

T1:	R(A)		W(A)
T2:		W(A)	
T3:			W(A)

T1:	R(A),W(A)		
T2:		W(A)	
T3:			W(A)

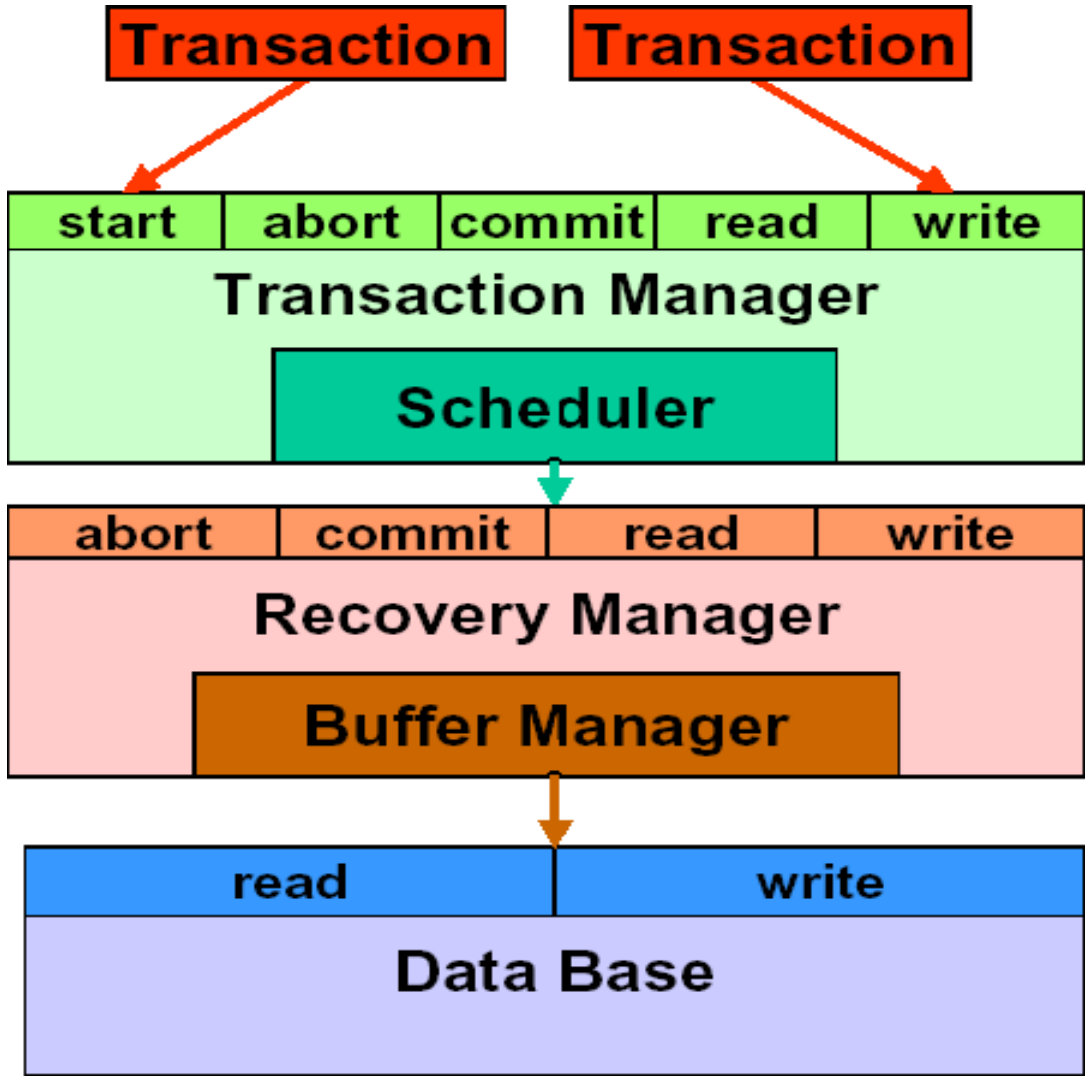
Planificarea tranzacțiilor



Serializabilitate în practică

- In practică, un SGDB nu testează serializabilitea unei planificări date. Acest lucru nu este practic deoarece intercalarea operațiilor mai multor tranzacții concurente poate fi dictată de sistemul de operare şi prin urmare este dificil de impus.
- Abordarea DBMS este să folosească protocoale specifice care sunt cunoscute că generează planificări serializabile.
- Aceste protocoale pot afecta gradul de concurență, însă elimină cazurile conflictuale.

Executarea tranzacțiilor



Phantom Reads

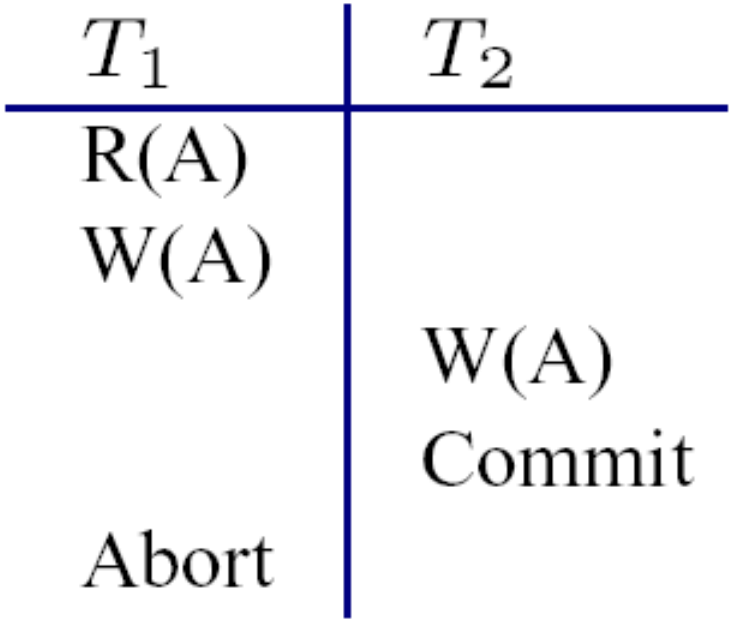
- O tranzacție re-execută o interogare şi găseşte că o altă tranzacție comisă a inserat înregistrări adiționale ce satisfac condițiile interogării
  - Dacă înregistrările au fost modificate sau şterse, este vorba de conflictul *unrepeatable read*

Exemplu:

- T_1 execută `select * from Students where age < 25`
- T_2 execută `insert into Students values(12,'Jim', 23, 7)`
- T_2 execută comit
- T_1 execută `select * from Students where age < 25`

Planificări recuperabile

- Într-o *planificare recuperabilă* tranzacțiile poat **doar citi** data care a fost **deja comisă**
- Există posibilitatea apariției **blind write**



- Care ar trebui sa fie valoarea lui A dupa **abort**??

Planificare recuperabilă

- O planificare este recuperabilă dacă pentru oricare tranzacție T comisă, comiterea lui T se efectuează după comiterea tuturor tranzacțiilor de la care T a citit un element.

Controlul concurenței bazat pe blocări

- Blocările sunt utilizate pentru a garanta planificări recuperabile/serializabile
- Un *protocol de blocare* este un set de reguli urmate de fiecare tranzacție (fiind impuse de SGBD) pentru a se asigura că, chiar şi în situațiile în care instrucțiunile tranzactiilor ar putea fi intercalate, efectul final este identic cu cel al unei executări seriale a tranzacțiilor.
- Se utilizează blocări *partajate* şi *exclusive*

Definiții

- **Blocare**: O metodă utilizată pentru controlul accesului concurent la date. Atunci când o tranzacție accesează un obiect al bazei de date, blocarea poate proteja obiectul respectiv de a fi accesat de o altă tranzacție pentru a preveni obținerea de rezultate incorecte.
- **Blocare partajată** (*shared* sau *read lock*): Dacă o tranzacție blochează un obiect in mod partajat, ea poate citi acel obiect dar nu il poate modifica.
- **Blocare exclusivă** (*exclusive* sau *write lock*): Dacă o tranzacție blochează un obiect in mod exclusiv aceasta poate citi şi modifica valoarea obiectului.

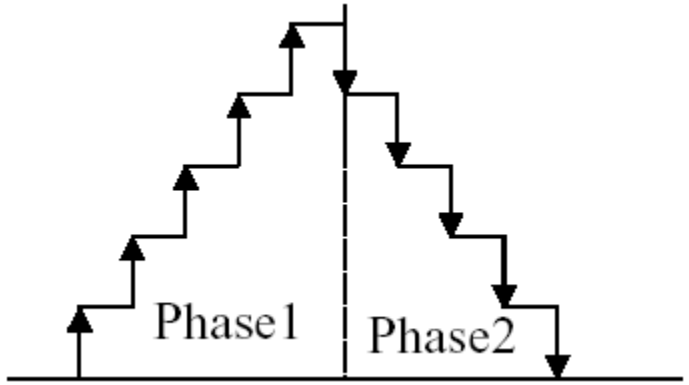
Algoritmi bazați pe blocări

- Tranzacțiile indică intenția de a bloca un obiect planificatorului (*lock manager*).
- Fiecare tranzacție care accesează un obiect pentru a-l citi sau modifica, trebuie mai întâi să blocheze obiectul respectiv.
- O tranzacție blochează un obiect până când il eliberează explicit.
- Conflicte între blocările partajate şi exclusive:

	Shared	Exclusive
Shared	Da	Nu
Exclusive	Nu	Nu

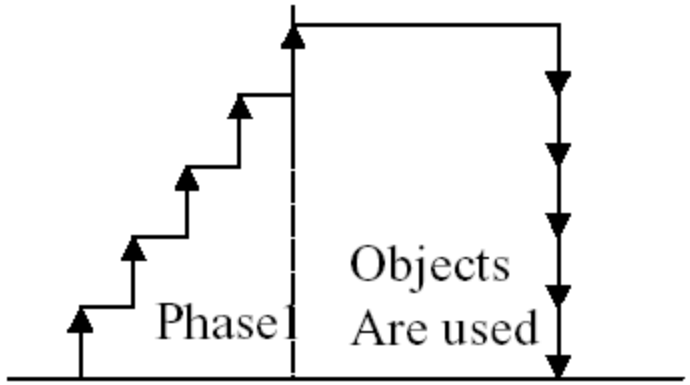
Protocol de blocare în două faze

- *2PL*:
  - O tranzacție urmează protocolul 2PL dacă toate operațiile de blocare preced prima operație de deblocare în cadrul tranzacției.
  - Faza 1 se numeşte "*faza de creştere*", aici fiind solicitate toate blocările
  - Faza 2 se numeşte "*faza de descreştere*" şi sunt eliberate toate obiectele blocate în faza anterioară



Protocol strict de blocare în două faze

- *Strict 2PL*:
  - Toate blocările sunt menținute de către tranzacție până imediat înainte de ***commit***
  - Protocolul *Strict 2PL* permite doar planificări serializabile



Gestionarea blocărilor

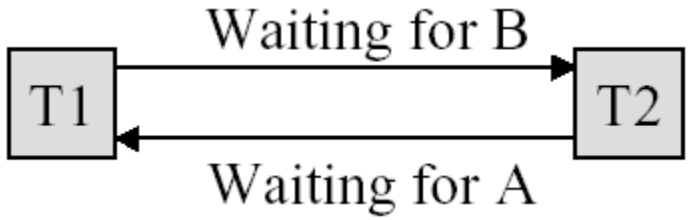
- Cererile de blocare şi deblocare de obiecte sunt gestionate de modulul de *lock management*
- Tabelă de blocări :
  - Tranzacțiile care au cel puțin o blocare
  - Tipul de blocare (*shared* sau *exclusive*)
  - Pointer către o coadă de cereri de blocare
- Operațiile de blocare şi deblocare trebuie să fie atomice

Deadlock

- *Deadlock*: Ciclu de tranzacții, fiecare aşteptând eliberarea unui obiect blocat de celelalte tranzacții.
- O tranzacție este în deadlock dacă nu mai poate continua executarea acțiunilor sale fără o intervenție externă.
- Algoritmii de control a concurenței pe bază de blocări pot cauza *deadlock*-uri.
- Metode de gestionarea *deadlock*-urilor:
  - Prevenire (garantează că nu apar *deadlock*-uri sau le anticipează)
  - Detectare (permit apariția *deadlock*-urilor şi le rezolvă atunci când apar)

Exemplu de *deadlock*

T1	T2
begin-transaction	
Write-lock(A)	begin-transaction
Read(A)	**Write-lock(B)**
A=A-100	Read(B)
Write(A)	B=B*1.06
Write-lock(B)	Write(B)
Wait	**write-lock(A)**
Wait	Wait
…	Wait
	…



Prevenire *deadlock*

- Atribuie priorități bazate pe *timestamp*. (tranzacțiile mai vechi au prioritatea cea mai mare)
- Dacă T_i doreşte acces la un obiect blocat de T_j , sunt posibile două politici:
  - *Wait-Die*: Dacă T_i are prioritate mai mare, T_i aşteaptă după T_j; altfel T_i se termină
  - *Wound-wait*: Dacă T_i are prioritate mai mare, T_j se termină; altfel T_i aşteaptă
- Dacă o tranzacție eliminată se reporneşte ulterior, va avea *timestamp*-ul original

Deadlock-urile şi expirarea timpului

- O metodă simplă de prevenire a deadlock-urilor se bazează pe expirarea timpului de aşteptare după o resursă blocată
- După cererea unei blocări, o tranzacție aşteaptă o perioadă de timp. Dacă obiectul aşteptat nu se deblochează după o anumită perioadă, tranzacția este oprită şi repornită.
- Este o soluție foarte simplă şi practică adoptată de multe SGBD-uri.

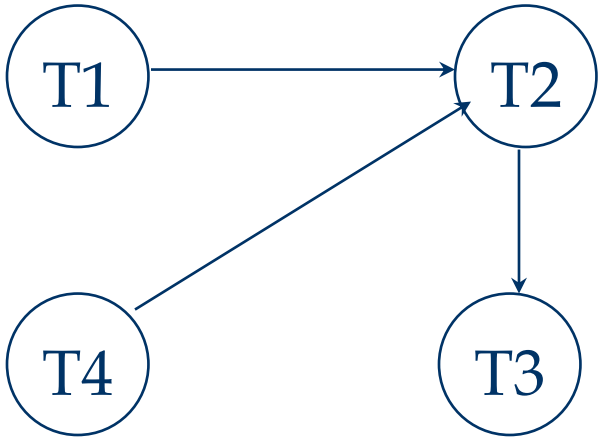
Detectarea *deadlock*-ului

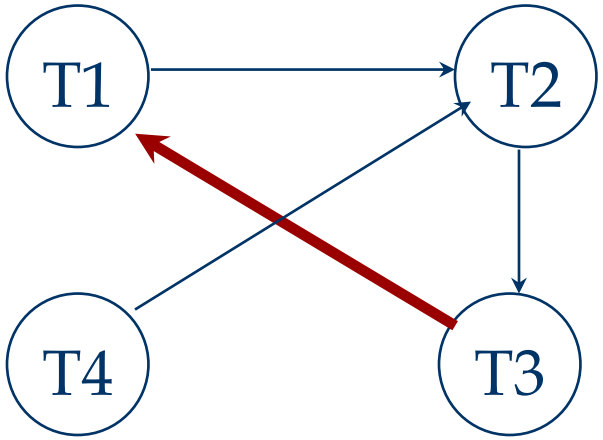
- Se crează un graf de aşteptare:
  - Nodurile sunt tranzacţii
  - Există un arc de la T_i la T_j dacă T_i aşteaptă după T_j să elibereze un obiect blocat
- Dacă este un circuit în acest graf atunci a apărut un *deadlock*.
- Periodic SGBD verifică dacă au apărut circuite în graful de aşteptare

Detectare *deadlock*

Exemplu:

T1: S(A), R(A), S(B)
T2: X(B),W(B) X(C)
T3: S(C), R(C) X(A)
T4: X(B)





Recuperarea după *deadlock*

- Cum se alege tranzacția victimă a unui *deadlock*?
  - Durata execuției unei tranzacții
  - Numărul obiectelor modificate de către tranzacție
  - Numărul obiectelor ce urmează să fie modificate de către tranzacție
- Politica de alegere a "*victimei*" trebuie să aibă în vedere echitatea: să nu fie aleasă de fiecare dată aceeaşi tranzacție ca victimă