Controlul concurenței tranzacțiilor

Controlul concurenței bazat pe blocări:

Blocare = o metodă utilizată pentru controlul accesului concurent la date

- odată ce o tranzacție își anunță intenția de a modifica o anumită dată, aceasta se blochează, astfel încât o altă tranzacție, când dorește să acceseze data respectivă, să fie înștiințată că aceasta este blocată, și să aștepte

Blocare partajată (*shared* sau *read lock*) = o tranzacție dorește doar să *citească* obiectul

Blocare exclusivă (*exclusive* sau *write lock*) = o tranzacție dorește să *modifice* obiectul

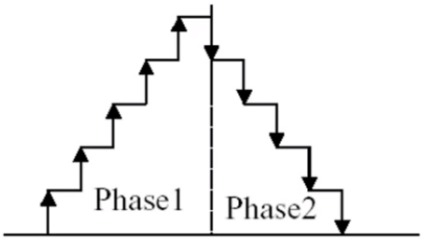
	Partajat	Exclusiv
Partajat	Da	Nu
Exclusiv	Nu	Nu

- explicație: dacă o tranzacție blochează în mod partajat un obiect, atunci poate veni o altă tranzacție să o blocheze în mod partajat, însă nu poate să o blocheze exclusiv etc.

Pentru a scăpa de anomalii complet, trebuie implementate niște politici ce trebuie urmate de toate planificările de tranzacții, astfel încât ele să nu se afecteze reciproc și să nu conducă la ceva ce nu poate fi serializabil.

Protocol de blocare în două faze (2PL – 2 Phase Locking):

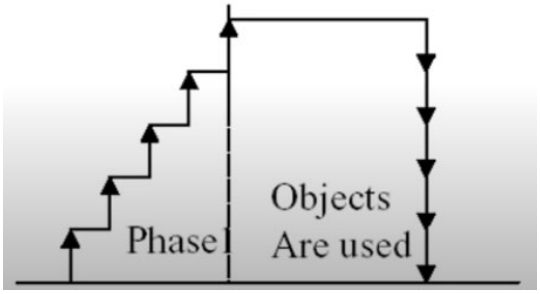
- în viața unei tranzacții, există o primă fază de blocări, și o a doua fază de deblocări
- toate operațiile de blocare preced prima operație de deblocare în cadrul tranzacției



- ordinea în care deblochez e inversă ordinii în care blochez
- pot apărea anumite probleme între două deblocări: o tranzacție a apucat să îl deblocheze pe C dar nu și pe B, iar între timp o altă tranzacție se folosește de C, îl modifică, dar după aceea, tranzacția inițială se termină fără succes, așa că se anulează orice modificare pe care ar fi putut-o face cea de-a doua tranzacție
- există un al doilea protocol pentru a elimina această problemă

Strict 2PL protocol:

- toate obiectele care s-au blocat unul după celălalt vor fi deblocate numai când se finalizează tranzacția care le-a blocat



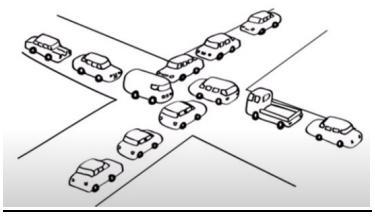
- protocolul e mult prea strict, mult prea restrictiv: micșorează capacitatea unui sistem de a executa mai multe tranzacții concurente
- marea majoritate a planificărilor ce le putem efectua sunt cele seriale

Gestionarea blocărilor:

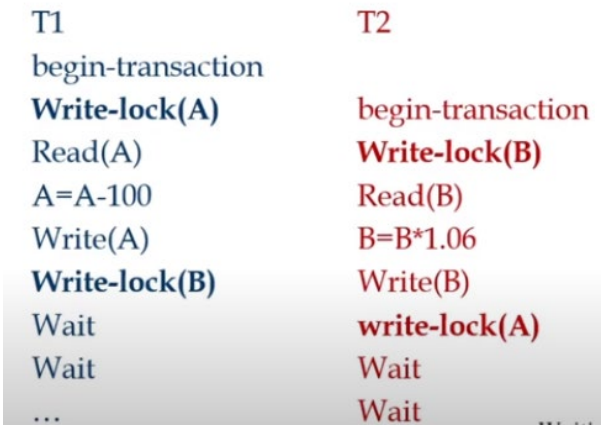
- există niște tabele sistem care memorează toate blocările – pentru fiecare înregistrare se memorează:
  - tranzacția care a inițiat blocarea
  - tipul de blocare (shared / exclusive)
  - pointer către o coadă de cereri de blocare ce așteaptă după obiectul blocat
- operațiile de blocare și deblocare trebuie să fie atomice

Deadlock:

- tranzacțiile se așteaptă reciproc



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 prin timestamp:

- cea mai veche tranzacție este cea mai puternică
- T_i dorește accesul la un obiect blocat de T_j:
  - *Wait-Die*: dacă T_i are prioritate mai mare decât T_j, T_i așteaptă după T_j; altfel, T_i se termină fără succes
  - *Wound-Wait*: dacă T_i are prioritate mai mare decât T_j, T_j se termină fără succes; altfel, T_i așteaptă
- obs: există mai multe modalități de tratare a acestei terminări: la unele sisteme de operare, în momentul în care apare un astfel de caz, tranzacția care pierde este oprită, se face roll back; alte sisteme iau tranzacția, o opresc, și o pun într-o coadă de așteptare, ca să fie executată din nou de la început, cu timestamp-ul original (!!!)

În practică, foarte multe SGBD-uri nu merg pe nicio politică. În schimb, se uită la o tranzacție și văd de cât timp așteaptă după o resursă. Dacă a trecut de un anumit număr de milisecunde / secunde, atunci tranzacția respectivă se termină.

Detectarea deadlock-ului:

- se crează un graf de așteptare:
  - nodurile sunt tranzacții
  - există un arc de la T_i la T_j dacă T_i așteaptă după T_j să elibezere un obiect blocat
- dacă există un circuit în acest graf, atunci a apărut un deadlock
- SGBD verifică periodic dacă au apărut circuite în graful de așteptare

Cum se alege tranzacția victimă a unui deadlock?

- durata execuției unei tranzacții – deoarece se elimină cea care a durat cel mai puțin
- numărul obiectelor modificate de către tranzacție
- numărul obiectelor care urmează să fie modificate – dacă o tranzacție urmează să blocheze multe obiecte, înseamnă că, dacă alegem o alta ca victimă, se poate ajunge din nou la deadlock din cauza tranzacției care blochează prea multe

!!! Nu trebuie să fie aleasă de fiecare dată aceeași tranzacție ca victimă.