Închideri în tranzacţii concurente

Fie două tranzacţii T1 şi T2 care trebuie să execute acelaşi program simplu P, pentru increnmentarea unui element A conţinut într-o bază de date elementară., programul este: P : Read A ; A := A + 1 ; Write A. Admitem că la început A = 10, iar tranzacţiile fac parte din planificarea dată în Fig. 1a. În Fig. 1b se arată cum se modifică valorile corespunzătoare elementului A în spaţiile de lucru asociate cu T1 şi cu T2, precum şi în baza de date.

T1	T2	Valoare A în spaţiul T1	Valoare A în spaţiul T2	Valoare A în baza de date
Read A		10		10
	Read A	10	10	10
A := A + 1		11	10	10
	A = A + 1	11	11	10
	Write A	11	11	11
Write A		11		11

a) b)

Fig. 1

În urma execuţiei programului, tranzacţiile T1 şi T2 ar urma să adauge 1 la conţinutul bazei de date, formată numai din elementul A. Datorită însă modului în care acţiunile tranzacţiilor au fost distribuite în timp, în interiorul planificării, numai tranzacţia T1 reuşeşte să incrementeze pe A cu o unitate, aşa încât rezultatul prelucrării va fi un conţinut 11 al bazei de date, în loc de 12. De fapt, a fost pierdută o operaţie.Ne putem imagina efectele de acest gen în cazul unei baze de date ample, destinate, de exemplu, rezervării de locuri.

O soluţie posibilă a problemei de mai sus stă în introducerea unui mecanism de „închidere” a accesului la elementul A în unele intervale de timp. Prin efectul acestui mecanism, o tranzacţie oarecare T trebuie să închidă obiectul de date A înainte de a efectua o operaţie de citire, urmată eventual de alte operaţii de prelucrare asupra acestuia. Ca urmare a închiderii, alte tranzacţii vor fi împiedicate de a avea acces la elementul A, atât timp cât tranzacţia T acţionează asupra sa. În prezenţa mecanismului de închidere, programul P va arăta astfel :

P : Lock A ; Read A ; A := A + 1 ; Write A ; Unlock A.

În program, comenzile Lock A şi Unlock A îndeplinesc funcţiuni evidente, de interzicere a acesului altor tranzacţii asupra obiectului A, respectiv de încheiere a intervalului de timp în care funcţionează interdicţia. O planificarea posibilă a celor două tanzacţii va fi acum:

T1	T2
Lock A	
Read A	
A := A+1	
Write A	
Unlock A	
	Lock A
	Read A
	A:= A+1
	Write A
	Unlock A

Fig. 2

În urma execuţiei tranzacţiilor, de această dată, vom avea A = 12. Dreptul de a executa comenzile Lock A şi Unlock A este acordat tranzacţiilor de către o componentă a sistemului de gestiune, numită „controlor de închideri". Vom observa că în cazul examinat, am obţinut de fapt o execuţie serială a tranzacţiilor, respectiv o succesiune. În alte cazuri, primitivele de închidere şi de deschidere pot să fie intercalate în corpul unei planificări astfel încât să nu se mai obţină succesiuni, ci execuţii seriale. De exemplu, atunci când tranzacţia T1 închide obiectul A, este posibil ca tranzacţia T2 să prelucreze un alt obiect B, acţiunile din T2 intecalându-se cu cele din T1.

În prima fază a execuţiei prezentate, tranzacţia T1 lucrează, în timp ce T2 aşteaptă. În cea de adoua fază, rolurile celor două tranazacţii sunt inversate.

Aşteptarea nedefintă

Maniera în care comenzile Lock şi Unlock sunt folosite pentru comanda execuţiei unor tranzacţii poate conduce la regimuri de lucru neodorite, în care execuţia unor tranzacţii este amânată nedefinit („livelock"). O astfel de situaţie este descrisaă în Fig. 3, în partea:

Lock A;Read A;A:= A+1;Write A;Unlock A;Lock A; ...;Unlock A;Lock A ;....;Unlock A;

(T1 în execuţie)	I (T3 în execuţie)	I (T4 în execuţie)
T2 aşteaptă	I T2 aşteaptă	I T2 aşteaptă
	aici, controlorul de închideri dă drept de „Lock" tranzacţiei T3, nu tranzacţiei T2.	aici, controlorul de închideri dă drept de „Lock" tranzacţiei T4, nu tranzacţiei T2.

Fig. 3

dreaptă a căreia sunt reprezentate numai acţiunile semnificative din punctul de vedere al prelucrării concurente, deci numai momentele Lock şi Unlock. Situaţia de mai sus poate să apară ca urmare a unei inconsistenţe în funcţionarea controlorului de închideri, care amână în mod continuu accesul tranzacţiei T" la comanda Lock. În acest mod, tranzacţia T2 poate aştepte indefinit dreptul de închidere.. Problema aşteptării nedefinite se poate soluţiona printr-o planificare adecvată, de exemplu prin folosirea strategiei „primul sosit,

primul servit". Observăm însă că această procedură poate conduce la situații de blocare a accesului.

Blocarea accesului

Să admitem că două tranzacții, T1 şi T2, prelucrează mai mult de un singur element de date (granulă). Pentru simplificarea prezentării, nu vom mai figura acțiunile de prelucrare efectivă amplasate între operatorii Lock şi Unlock. În Fig. 4 sunt date tranzacțiile simplificate (Fig.4a) şi o parte din amplasarea în timp a comenzilor de închidere (Fig.4b).

T1 : Lock A ; Lock B ; Unlock A ; Unlock B.
T2 ; Lock B ; Lock A ; Unlock B ; Unlock A. Fig. 4a

T1 nu primeşte acces aici, deoarece
accesul a fost acordat tranzacției T2, care nu l-a
restituit prin Unlock A

T1 Lock A Lock B

Fig. 4b

T2 Lock B Lock A

T2 nu va primi acces aici la Lock A,
deoarece accesul a fost anterior acordat tranzacției T1,
care încă nu l-a restituit prin Unlock A.

Mai sus, a fost admisă ipoteza că execuția celor două tranzacții începe la momente foarte apropiate. Din Fig. ¤b se observă că fiecrea tranzacție o va aştepta pe cealaltă, fără ca vreuna din ele să poată efectua prelucrările necesare (care aici nu sunt arătate). Aşteptarea poate dura nedefinit, ceea ce se poate înțelege ca o situație de blocare („deadlock").
Pentru evitarea situației de blocare, au fost imaginate rezovări diverse, din care relevăm următoarele:

1. fiecare tranzacție să solicite toate închiderile încă de la începutul execuției sale şi, în o primă variantă, să determine controlorul de închideri să satisfacă solicitarea, dacă are posibilitatea. În o a doua variantă, corespunzătoare situației în care una sau mai multe dintre închideri (firește, pe granule de date diferite) sunt în posesia altor tranzacții, controlorul de tranzacții nu acordă niciuna din închiderile solicitate de tranzacția avută în vedere, aceasta fiind lăsată în stare de aşteptare.
   În cazul ultimului exemplu, aplicarea regulii de mai sus ar fi prevenit blocarea. Dacă tranzacția T1 ar fi fost prima care a cerut dreptrile de închidere pe elementele A şi B, controlorul de închideri ar fi acordat aceste drepturi, tranzacția T2 ar fi intrat în aşteptare, între timp T1 ar fi efectuat prelucrările necesare, apoi dreptul de închidere ar fi fost atribuit tranzacției T2.
2. O altă soluție constă în ordonarea liniară a elementelor de date supuse prelucrării, urmând ca toate tranzacțiile să solicite dreptul de închidere în ordinea atribuită.

Pentru exemplul de blocare prezentat anterior, să admitem că în ordonarea aleasă elementul A precede elementul B. Atunci, ținând seama de această ordine, tranzacția T2 va solicita, în acțiunea sa, o închidere a elementului A, înainte de a cere o închidere a elementului B, dar va găsi A deja închis de tranzacția T1. Totodată, T2 nu va putea să închidă elementul B, deoarece este respectată ordonarea, aşa că închiderea elementului B va fi posibilă pentru T1, atunci când această ultimă tranzacție o va solicita (T1 respectă ordinea de asemenea- ea a închis mai întâi elementul A, aşa că poate cere dreptul de închidere pe B !). În acest mod, tranzacția T1 poate fi executată complet, eliberând până la urmă (prin Unlock) închiderile elementelor A şi B. Începând din acel moment, poate debuta acțiunea tranzacției T2.

După cum se arată în literatura de specialitate, o manieră posibilă în care poate fi tratată problema blocărilor acțiunii tranzacțiilor este de a nu întreprinde nimic în vederea prevenirii acestora. Dacă se adoptă o astfel de abordare, cererile de închidere sunt examinate periodic şi, eventual, se constată dacă apare o blocare. Procedura folosită prevede construirea unui "graf de aşteptare", ale cărui noduri corespund tranzacțiilor, iar ramurile de tipul T1 -> T2 indică faptul că tranzacția T1 aşteaptă să închidă un obiect de date asupra căruia are drept de închidere tranzacția T2. În graful obținut, fiecare eventual contur închis arată prezența unei blocări. În absența contururilor închise, nu avem blocări. Atunci când este observată o blocare, trebuie re-pornită cel puțin una dintre tranzacțiile blocate, efectele acesteia asupra bazei de date în ansamblul său urmând a fi anulate. Procesul menționat, de "întrerupere - re-pornire" putând fi complicat, dacă nu a fost manifestată prudență aspra modului în care tranzacțiile înscriu informații în baza de date, înain te ca ele să îşi încheie cu success acțiunea (deci să fie "completate")

Graful de precedență şi testarea serializabilității pentru planificări cu un model simplu de închidere

Pentru a determina corectitudinea unui planificator, trebuie să arătăm că fiecare execuție permisă de acesta este serializabilă. În acest scop, este necesar un algoritm pentru construirea grafului de precedență (numit adesea graf de serializabilitate), respectiv pentru a vedea dacă acest graf este sau nu este aciclic.
Intrarea algoritmului este o planificare oarecare P a unei mulțimi de tranzacții T1, T2,..., Tk.
Ieşirea algoritmului este reprezentată de graful de precedență G şi de decizia că P este sau nu este serializabilă.
Metoda constă în construirea grafului de precednță G, în care fiecare nod corespunde unei tranzacții din P, respectiv în trasarea ramurilor (arcelor) grafului. Pentru determinarea ramurilor, presupunem că planificarea P este dată în forma a1, a2, ...an,

unde fiecare acțiune ai este reprezentată de Tj : Lock Am, sau de Tj : Unlock Am. Notațiile Tj indică tranzacția căreia îi aparține pasul curent al algoritmului. Dacă ai este de forma Tj : Unlock Am, atunci se caută în planificare următoarea acțiune ap care urmează acțiunii ai, fiind de forma Ts : Lock Am. Dacă este găsită o asemenea acțiune, iar s este diferit de j, se trasează un arc de la nodul Tj la nodul Ts. Intuitiv, sensul existenței acestui arc constă în faptul că în orice execuție serială echivalentă cu P, tranzacția Tj trebuie să preceadă tranzacția Ts.
Prin parcurgerea procedurii descrise, se obține în final graful de serializabilitate G. Dacă G are un contur închis, planificarea P nu este serializabilă. Altfel spus, în prezența ciclurilor, ar rezulta că o tranzacție dată Ts ar preceda, în unele acțiuni, alte tranzacții, ale căror acțiuni ar conduce la tranzacția Ts, procesul fiind reluat.
În ipoteza că graful G nu are cicluri, P este serializabilă şi ar trebui găsită o execuție serială (succesiune) echivalentă cu P. Pentru aceasta, tebuie determinată o ordonare liniară a tranzacțiilor, astfel încât tranzacția Ti să preceadă tranzacția Tj oridecâte ori avem un arc orientat de la nodul Ti la nodul Tj. Definim în continuare procedura de sortare topologică, prin care se realizează ordonarea liniară urmărită. Potrivit acestei proceduri, în graf trebuie să existe un nod Ti către care nu sunt orientate ramuri, altfel s edemonstraează imediat că graful G are un contur închis. În această situație, este „listat” simbolul Ti, iar nodul Ti se elimină din graf. Procedura este repetată în continuare asupra noului graf, până când în acesta nu mai rămân noduri. Ordinea obținută prin listarea simbolurilor de tip Ti este ordinea căutată a tranzacțiilor în planificarea serială echivalentă, sau în succesiunea respectivă.
În Fig. 5 este prezentat exemplul unei planificări asupra căreia se aplică algoritmul descris mai sus, iar în Fig. 6 se găseşte graful de precedență corespunzător. Aşa cum se observă, graful de precedență are un ciclu, aşadar planificarea din Fig. 5 nu este serializabilă.

(1) T1 : Lock A
(2) T2 : Lock B
(3) T2 : Lock C
(4) T2 : Unlock B
(5) T1 : Lock B
(6) T1 : Unlock A
(7) T2 : Lock A
(8) T2 : Unlock C
(9) T2 : Unlock A
(10) T3 : Lock A
(11) T3 : Lock C
(12) T1 : Unlock B
(13) T3 : Unlock C
(14) T3 : Unlock A

Fig. 5

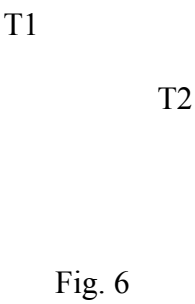

Fig. 6

Să descriem succint maniera de aplicare a algoritmului. Pentru a găsi ramurile grafului, pornim de la fiecare pas Unlock al planificării din Fig. 5. Vom începe cu pasul (4), în

care avem T2 : Unlock B. Următoarea închidere a elementului de date B apare în pasul imediat următor, (5), fiind T1 : Lock B. Ca urmare, vom trasa un arc de la nodul T2 la nodul T1. Un alt pas de tip Unlock este (6), în care se execută acțiunea T1 : Unlock A. Următorul pas în care găsim o acțiune de tip Lock asupra elementului A este (7), în care este executată acțiunea T2 : Lock A. În consecință, vom trasa un arc de la nodul T1 la nodul T2. În mod asemănător vom constata că tebuie trsat un arc de la nodul T2 la nodul T3.
Să examinăm un alt exemplu, date de planificarea din Fig. 7, pentru care graful de precednță, construit prin procedura dată, este, în mod evident, cel din Fig. 8

T1	T2	T3
	Lock A	
	Unlock A	
		Lock A
		Unlock A
Lock B		
Unlock B		
	Lock B	
	Unlock B	

Fig. 7

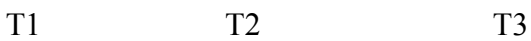

Fig. 8

Ordinea serială corespunzătoarea planificării, obținită prin sortarea topologică, este T1, T2, T3. Este interesant de remarcat că în ordinea serială tranzacția T1 precede tranzacția T3, deşi în planificarea din Fig. 7 tranzacția T1 nu începe să „funcționeze” decât după ce a fost completată tranzacția T3.

Model de planificare cu închideri separate pentru operațiunile de citire şi de înregistrare.

În ipoteza că nu vom face distincție între operațiunile de acces având obiectivul exclusiv de citire a informațiilor din baza de date („read – only”) şi operațiunile de acces care au ca scop exclusiv înregistrarea de informații („write – only”), poate fi introdus un model mai detaliat de planificare a tranzacțiilor, model în care sunt permise accese concurente

interzise în modelul mai simplu prezentat în paragraful anterior. În acest model sunt acceptate două tipuri de operații de închidere, definite în continuare.

a. Închideri de citire (sau închideri partajate). În acest regim, tranzacția T care încearcă numai să citească elementul A execută comanda Rlock A. Odată cu prelucrarea acestei comenzi, va fi împiedicată înragistrarea de către orice altă tranzacție a unei noi valori pentru A, atât timp cât tranazacția T menține închiderea pe A. Dreptul de închidere la citire pe elementul de date A poate aparține oricărui număr de tranzacții în acelaşi timp.
b. Închiderea de înregistrare (sau închiderea exclusivă). În acest caz, operația de închidere are sensul definit atunci când am prezentat algoritmul anterior, cu un model simplu de închidere. Dacă o tranzacție T urmează să modifice valoarea obiectului de date A, execută mai întâi comanda Wlock A, obținând dreptul de închidere la înregistrare. Ca urmare, nici o altă tranzacție nu va putea obține vreun drept de închidere- la înregistrare sau la citire – pe obiectul A, atât timp cât dreptul menționat aparține tranzacției T.

Atât dreptul de închidere la citire, cât şi cel de închidere la înregistrare, se cedează de către tranzacția curentă prin comanda Unlock.

Pentru a construi graful de precedență şi a verifica existența proprietății de serializabilitate a unei planificări, în cazul folosirii modelului cu închidere la înregistrare, se prezintă în continuare un algoritm specific.

Intrarea algoritmului este o planificare P a mulțimii de tranzacții T1, T2, ..., Tk.

Ieşirea constă în a stabili dacă planificarea este serializabilă, caz în care se obține şi o planificare serială echivalentă.

Metoda algoritmului este reprezentată dee construirea grafului de precedență G, într-o maniera similară cu cea folosită pentru modelul simplu de închidere. Respectiv, fiecărei tranzacții îi corespunde un nod al grafului, arcele acestuia fiind trasate potrivit procedurii care urmează.

1. Fie în planificarea P tranzacția Ti care execută o cerere de închidere la citire sau la înregistrare pe elementul de date A, iar Tj- tranzacția următoare care încearcă să închidă A la înregistrare, cu i diferit de j.
2. Fie acum în planificarea P o tanzacție Ti care închide obiectul A la înregistrare. De asemenea, admitem că Tm, cu m diferit de i, este orice tranzacție care închide la citire obiectul de date A, după ce tranzacția Ti a renunțat la dreptul său de închidere la înregistrare, dar înainte ca orice altă tranzacție să închidă A la înregistrare. În această situație, se trasează un arc de la nodul Ti la nodul Tm.

Ca şi în cazul folosirii algoritmului pentru modelul simplu de închidere, dacă graful G este aciclic, planificarea este serializabilă, iar prin sortare topologică se poate obține o planificare serială echivalentă. Atunci când în G sunt contururi închise, planificarea este ne-serializabilă, ea urmând să fie modificată.

Aplicarea algoritmului de mai sus este ilustrată în exemplul planificării din Fig. 9, graful de precedență al căreia este reprezentat în Fig. 10

	T1	T2	T3	T4
(1)			Wlock A	
(2)				Rlock B
(3)			Unlock A	
(4)	Rlock A			
(5)				Unlock B
(6)			Wlock B	
(7)		Rlock A		
(8)			Unlock B	
(9)	Wlock B			
(10)		Unlock A		
(11)	Unlock A			
(12)				Wlock A
(13)	Unlock B			
(14)		Rlock B		
(15)				Unlock A
(16)		Unlock B		

Fig. 9

Fig. 10

Aşa cum se vede din Fig. 9, planificarea conține patru tranzacții, T1, T2, T3, T4. Prima comandă Unlock se întâlneşte în pasul (3) al planificării, unde tranzacția T3 retrage închiderea la înregistrare pe A. În continuare, în paşii (4) şi (7), care succed pasului (3), urmează închiderile la citire pe obiectul A ale tranzacțiilor T1 şi T2, precum şi o închidere la înregistrare pe A în pasul (12), de către tranzacția T4. Prin urmare,

tranzacţiile T1, T2 şi T4 succed tranzacţiei T3 şi, în consecinţă, trasăm câte un arc orientat de la T3 către fiecare din nodurile T1, T2 şi T4.
Observăm, mai departe, că tranzacţiile T1 şi T2 deţin dreptul de înregistrare, respectiv de citire, asupra obiectului A, începând cu pasul (7), iar în paşii (11) şi (10) aceste tranzacţii renunţă la dreptul menţionat. În pasul ce urmează, (12), tranzacţia T4 încearcă o închidere la înregistrare pe A, încercare reuşită, deoarece potrivit punctului din descrierea algoritmului, T1 şi T2 au retras cererile lor de închidere pe A. Rezultă că în graf se pot trasa ramuri orientate de la T1 şi T2 către T4.
De asemenea, remarcăm că T4 retrage o cerere de citire pe obiectul B în pasul (5), următoarea închidere la înregistrare pe B fiind efectuată în pasul (6) de către tranzacţia T3, aşa încât vom trasa un arc de la nodul T4 la nodul T3. În acest fel, în graf va fi prezent un contur închis, deci planificarea nu este serializabilă.