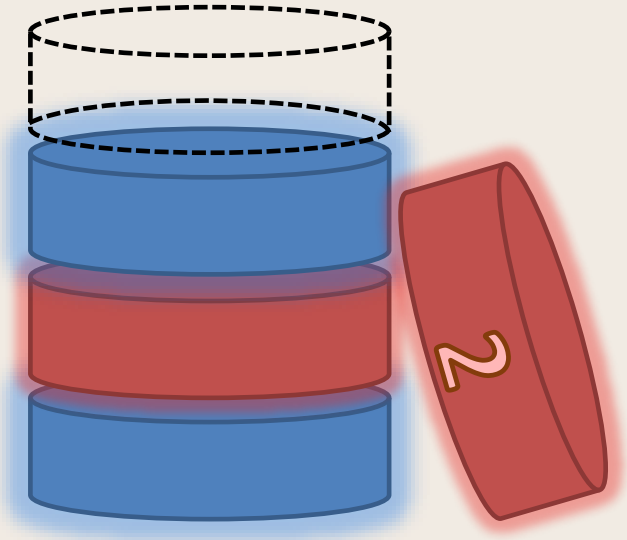
Probleme



Examinare/notare

- 8 iunie, de la ora 14:45 la 16:00.
- test grilă online (*Microsoft Forms*) :
  - 40 întrebări teoretice (cursuri 1-10)
- link chestionar trimis pe email cu o oră înainte de începere

- restanța: 14 iulie, 9:00 – 11:00, 310 FSEGA

Examinare/notare

ROUND (AVG(ROUND(AVG(P, L),2), T), 2),

- L - notă laboratoare
- P – test practic
- T – test teoretic

Problema 1

Fie următoare tranzacții:

T1: *read(A);*
read(B);
if A = 0 then B := B + 1;
write(B).

T2: *read (B);*
read (A);
if B = 0 then A := A + 1;
write(A).

Fie condiția de consistență A = 0 $\vee$ B = 0, cu valorile inițiale A = B = 0.

1. Arătați că orice execuție serială a celor două tranzacții păstrează consistența bazei de date.

2. Găsiți cel puțin o execuție concurentă a T1 şi T2 ce produce o planificare ne-serializabilă.

3. Există o execuție concurentă a T1 şi T2 ce produce o planificare serializabilă?

Problema 1 - Răspuns

1. Arătați că orice execuție serială a celor două tranzacții păstrează consistența bazei de date.

Există două execuții posibile: $\{T_1, T_2\}$ *şi* $\{T_2, T_1\}$

Caz 1:

	A	B
iniţial	0	0
după T1	0	1
după T2	0	1

date consistente

Caz 2:

	A	B
iniţial	0	0
după T2	1	0
după T1	1	0

date consistente

Problema 1 - Răspuns

2. Găsiți cel puțin o execuție concurentă a T1 şi T2 ce produce o planificare ne-serializabilă.

Orice intercalare a T1 şi T2 rezultă într-o planificare ne-serializabilă

T1	T2
read(A)	
	read(B)
	read(A)
read(B)	
if A = 0 then B = B + 1	
	if B = 0 then A = A + 1
	write(A)
write(B)	

Problema 1 - Răspuns

3 . Există o execuție concurentă a T1 şi T2 ce produce o planificare serializabilă?

De la punctul 1. ştim că o planificare serializabilă respectă condiția

$$A = 0 \ \backslash / \ B = 0.$$

Dacă începe T1 cu read(A), atunci când planificarea se termină, indiferent de momentul în care se execută comenzile din T2, B = 1. T2 va trebui să execute prima sa instrucțiune înainte de finalizarea lui T1. Atunci T2 execuția lui read(B) va da valoarea 0 pentru B. Deci la finalizarea lui T2 A = 1. Deci:

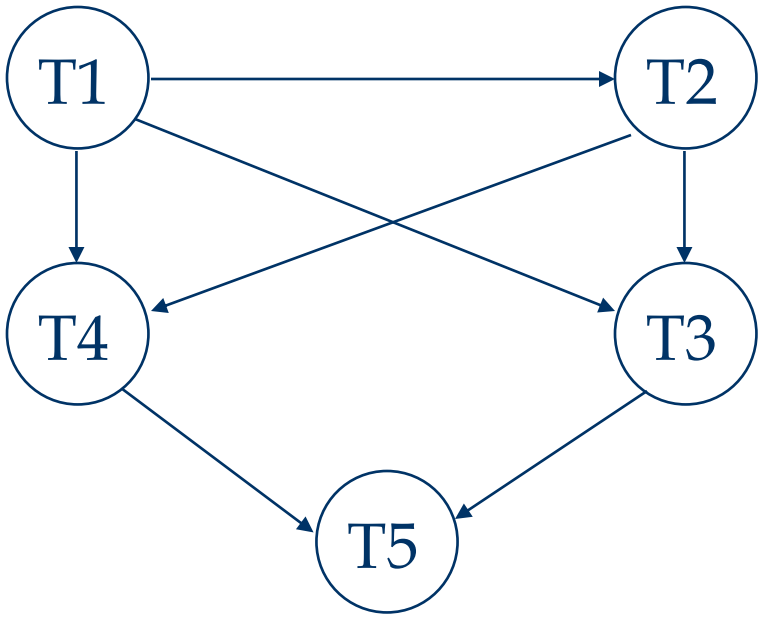
$$B = 1 \wedge A = 1 \Rightarrow \neg(A = 0 \vee B = 0).$$

Similar se demonstrează atunci când prima instrucțiune executată este a lui T2 (read(B)).

→ *Nu există nici o execuție paralelă ce sa reprezinte o planificare serializabilă.*

Problema 2

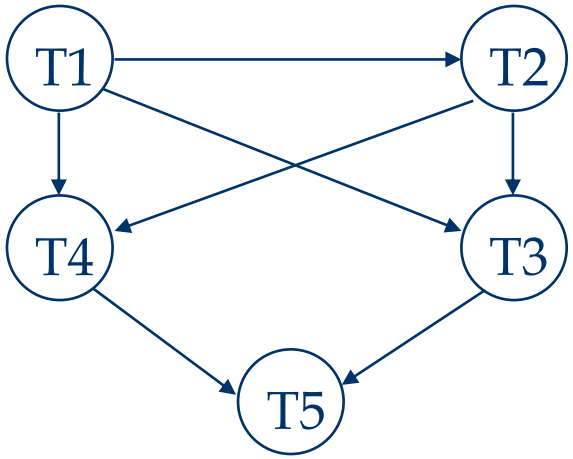
Fie următorul graf de dependenţă. Este planificarea corespunzătoare acestui graf conflict serializabilă? Justificaţi răspunsul.



Problema 2 - Răspuns

Există o planificare serializabilă corespunzătoare grafului de dependențe, deoarece graful e aciclic.

O posibilă planificare serială este obținută prin sortare topologică:
T1, T2, T3, T4, T5



Problema 3

Fie următoarele tranzacții :

T1: *read(A);*
read(B);
if A = 0 then B := B + 1;
write(B).

T2: *read (B);*
read (A);
if B = 0 then A := A + l;
write(A).

1. Adăugați intrucțiuni *lock* şi *unlock* tranzacțiilor T1 şi T2 astfel încât să fie implementat protocolul de blocare în două faze.

2. Este posibil ca execuția tranzacțiilor să intre în *deadlock*?

Problema 3 - Răspuns

1. Instrucţiuni *lock* şi *unlock* :

T1:	T2:
lock-S(A)	*lock-S(B)*
read(A)	read(B)
lock-X(B)	*lock-X(A)*
read(B)	read(A)
if A = 0 then B := B + 1	if B = 0 then A := A + 1
write(B)	write(A)
unlock(B)	*unlock(A)*
unlock(A)	*unlock(B)*

Problema 3 - Răspuns

2. Execuția tranzacțiilor poate duce la *deadlock*. De exemplu, fie următoarea planificare parțială:

T1	T2
lock-S(A)	
	lock-S(B)
	read(B)
read(A)	
lock-X(B)	
	lock-X(A)

Tranzacțiile au intrat în *deadlock*.

Problema 4

Fie următorul fişier de log:

[start_transaction, T1]
[W, T1, D, 20]
[commit, T1]
[checkpoint]
[start_transaction, T2]
[W, T2, B, 12]
[start_transaction, T4]
[W, T4, D, 15]
[start_transaction, T3]
[W, T3, C, 30]
[W, T4, A, 20]
[commit, T4]
[W, T2, D, 25]
<- system crash

Descrieți procesul de recuperare a datelor la blocarea sistemului. Specificați ce tranzacții sunt anulate, care operații sunt reexecutate şi care sunt anulate.

Problema 4 - Răspuns

- T1 comis înainte de *checkpoint*, deci toate operațiile de actualizare a datelor sunt înscrise în log şi sunt stocate pe disc. Nu este necesară reexecutarea operațiilor lui T1.
- T4 comis după *checkpoint*, deci toate operațiile sale sunt înregistrate în log dar efectul lor nu a fost neapărat stocat pe disc. Operațiile lui T4 vor trebui să fie reexecutate.
- T2 era activă la momentul întreruperii, de aceea este necesară anularea efectului operațiilor, unele dintre efectele acestor operații fiind salvate în baza de date.

Anularea operațiilor se realizează în ordine inversă

- T3 era activă de aceea este necesară anularea efectului operațiilor sale (modificarea obiectului C).

Problema 5

Fie operația join $R \otimes_{R.a = S.b} S$, executată în următorul context:
- tabela R conține 10.000 înregistrări, cu 10 înreg. pe pagină.
- tabela S conține 2.000 înregistrări, cu 10 înreg. pe pagină.
- atributul b al tabelei S este cheie primară pentru S.
- ambele tabele sunt memorate ca fişiere oarecare.
- nu există indecşi definiți pentru nici una dintre tabele.
- sunt disponibile 52 pagini în *buffer*

Care este cel mai mic cost al joinului lui R şi S folosind metodele *page-oriented nested loops*, *block nested loops* şi *sort-merge join*? Care este numărul minim de pagini disponibile în *buffer* pentru a obține acelaşi cost?

Problema 5 - Răspuns

Fie M = 1000 nr paginilor din R, N = 200 nr. paginilor din S şi B=52 nr paginilor disponibile în buffer

1. Care este costul joinului lui R şi S folosind *page-oriented nested loops join*? Care este numărul minim de pagini disponibile în *buffer* pentru a obține acelaşi cost?

Ideea de bază constă în citirea fiecărei pagini ale tabelei exterioare şi pentru fiecare pagină se scanează întreaga tabelă interioară căutâdu-se înregistrări pentru care se verifică condiția de join. Costul total ar fi:

$$\#PagExterne + (\#PagExterne * \#PagInterne)$$

Care este minimizat având cea mai mică tabelă ca tabelă exterioară.

$$CostTotal = N + (N*M) = 200.200$$

Numărul minim de pagini în buffer necesar pentru obținerea aceluiaşi cost este 3.

Problema 5 - Răspuns

2. Care este costul joinului lui R şi S folosind *block nested loops join*? Care este numărul minim de pagini disponibile în *buffer* pentru a obține acelaşi cost?

Folosind această metodă citirea tabelei exterioare se face in blocuri,, pentru fiecare bloc scanându-se tabela interioară pentru găsirea "potrivirilor". Tabela exterioară este parcursă o dată dar tabela interioară este scanată doar o dată pentru fiecare bloc, deci:

$$\lceil \#PagesInOuter \,/\, BlockSize \rceil = \lceil 200/50 \rceil = 4$$

$$Total\ Cost = N + M * \lceil N/(B-2) \rceil = 4.200$$

Dacă numărul de pagini în buffer este < 52*, numărul scanărilor tabelei exterioare este mai mare decât* $4 \rightarrow$ *numărul minim de pagini pentr a obține costul 4200 este 52*!

Sort-Merge Join ($R \otimes_{i=j} S$)

- Ordonare R şi S după câmpurile ce apar în condiția de join, apoi scanare pentru identificarea perechilor.
  - Scanarea lui R avansează până r_i curent > s_j curent, apoi se avansează cu scanarea lui S până s_j curent > r_i curent; până când r_i curent = s_j curent.
  - La acest punct toate perechile posibile între înregistrările din R cu aceeaşi valoare r_i şi toate înregistrările din S cu aceeaşi valoare s_j sunt salvate în pagina specială pentru rezultat.
  - Apoi se reia scanarea lui R şi S.
- R este scanat o dată; fiecare grup de înregistrări din S este scanat pentru fiecare înregistrare "potrivită" din R .

Rafinare algoritm Sort-Merge Join

- Se poate combina faza de interclasare din *sortarea* lui R şi S cu faza de scanare pentru join.
  - Având B > $\sqrt{L}$, unde L este dimensiunea celei mai mari tabele, şi folosind optimizarea algoritmului de sortare (ce produce monotonii de lungime 2B), numărul acestora pentru fiecare relație este < B/2.
  - Alocând o pagină pentru câte o monotonie a fiecărei relații, se va verifica expresia de join dintr-o singură trecere.
  - Cost: citire+salvare fiecare tabelă la Pas 0 + citire fiecare tabelă o dată pentru comparare (+ scriere rezultat).

  = > 3 (M+N)

Problema 5 - Răspuns

3. Care este costul joinului lui R şi S folosind *sort-merge join*? Care este numărul minim de pagini disponibile în *buffer* pentru a obține același cost?

Se poate utiliza varianta rafinată a metodei Sort-Merge Join.

$$Total\ Cost = 3 * (M+N) = 3.600$$

Numărul minim de pagini necesare este 25 → pasul inițial de sortare va împărții R în 20 şi S în 4 monotonii (de dimensiune aproximativ 50). Aceste 24 monotonii pot fi apoi interclasate într-un singur pas, cu o pagină de output.

Cu mai puțin de 25 pagini, numărul monotoniilor produse la primul pas va depăşi numărul de pagini disponibile, fiind imposibilă interclasarea acestora într-un singur pas