CONTROLUL ACCESELOR CONCURENTE LA BAZE DE DATE

Introducere.
Sistemele de programe care realizează controlul concurenței sunt acea paret a sistemului de gestiune a bazei de date (SGBD), care urmăreşte execuția „simultană" a tranzacțiilor, astfel încât să fie produse aceleaşi rezultate ca şi în cazul unei execuții secvențiale.
Formulând în mod diferit ideea de mai sus, putem defini cotrolul concurenței ca o activitate de divizare, sau dustribuire a datelor conținute în baza de date, între diferții utilizatori ai acesteia, în o astfele de maieră încât procesul acestei divizări să fie complet transparent pentru utilizatori.
O soluție primitivă în vederea atingerii obiectivului de mai sus, ar pute fi izolarea întregii baze de date în timpul execuției- sau prelucrării- fiecărei tranzacții, izolarea fiind realizată de către SGBD. Această modalitate de control este simplă, dar neeficientă. Performanțle unui sistem de baze de date ce foloseşte un astfel de mod de prelucrare se vor deteriora repede, până la un nivel inacceptabil, odată cu creşterea numărului utilizatorilor bazei de date. Drept consecință, proiectanții de siteme de gestiune au căutat să elaboreze algoritmi simpli, care să asigure un grad înalt de concurență, odată cu realizarea unor performanțe acceptabile.
În cele de mai sus, au fost folsite concepte generale care nu au fost încă definite în mod specific, în contextul tematic abordat. Acestor concepte le corespund termenii de concurență a acceselor, tranzacție, performanțe. Pe parcursul paragrafelor următoare, aceşti termeni vor fi definiți în o manieră mai specifică.

Integritatea datelor
Integritatea datelor este reprezentată de concordanța acestora cu proprietățile sistemelor din lumea reală pe care le modelează. Respectarea integrității datelor este o problemă esențială în cazul bazelor de date la care au loc accese concurente.

Schema unei baze de date conține descrierea datelor şi, implicit, „**restricțiile de integritate**" cărora trebuie să se supună aceste date.
Restricție de integritate este o aserțiune care trebuie să fie respectată, sau verificată, de către date în anumite momente determinate.
O bază de date este coerentă dacă datele pe care le conține respectă ansamblul restricțiilor de integritate implicită sau explicită ce au fost definite în contextul definirii bazei de date.
Pentru a aprofunda înțelegerea restricțiilor de integritate, să admitem exemplul unei baze de date de lucru formată din următoarele relații:

Cumpărare (CP, DC, CC),
Vânzare (CP; DV; CV),
Stoc (CP, CS),
Produs (CP, NP, NFU, NFA).

În schemele relațiilor, atributele au sensurile: CP- cod produs, DC- data cumpărării, CC- cantitatea cumpărată, DV- data vânzării, CV- cantitatea vândută, CS- cantitatea în stoc, NP- numele produsului, NFU- numele furnizorului, NFA- numele fabricantultui.
În una din clasificările adoptate, restricțiile de integritate sunt de tipurile de mai jos:

a) restricții de integritate privind un element de date. La rândul lor, acestea se pot referi la tipul da date şi la domeniul de valori (de exemplu, CC este un număr întreg, iar 1 < CV < 10 000).
b) Restricții de integritate privind mai multe elemente de date. Acestea pot fi: restricții de dependență duncțională (de exemplu, CP - > NP), restricții de dependență multi valoare (CP - >> NFU, CP - >> NFA), restricții de dependență refernțială (de exemplu, fiecare tuplu din relația Stoc trebuie să corespundă uneui tuplu din relația Produs).

c) Restricții de redundanță, car trebuie respectate de mai multe elemente de date (pentru fiecare produs, cantitatea în stoc CS trebuie să rămână egală cu diferența dintre suma cantităților cumpărate CS şi suma cantităților vândute CV).
d) Restricții ce au în vedere unii invarianți (ca exemplu, menționăm că suma generală a tuturor valorilor deținute de o bancă trebuie să rămână constantă la execuția unui transfer).
e) Restricții temporare de integritate, respectate numai în anumite momente sau intervale de timp (de pildă, în o bancă anumite valori au o valoare determinată numai la sfârşit de lună).

Pentru exprimarea şi analiza formală a restricțiilor de integritate (în afara celor temporare) se poate folosi logica predicatelor de ordinul întâi.

Tranzacții. Acțiuni.

Potrivit unei definiții larg acceptate, tranzacția este o unitate de prelucrare secvențială efectuată pentru un utilizator, care aplicată unei baze de date coerente, restitue o bază de date coerentă.
Restricțiile de integritate sunt întotdeauna invariante pentru tranzacții. În general, un sistem de baze de date execută un ansamblu de tranzacții în mod simultan, pentru diferiți utilizatori. Problemele dificile în realizarea concurenței decurg din interferențele care pot să apară între aceste tranzacții diferite, ca urmare a partajării datelor între utilizatorii bazelor de date.
Tranzacțiile sunt compuse din unități de prelucrare mai fine, ce poartă numele de acțiuni. Acțiunea este o comandă indivizibilă, executată de sistemul de gestiune în cadrul unei tranzacții. Citirea unei informații din memorie, sau înregistrarea unei informații, sunt înțelese, în general, ca acțiuni.

Probleme de concurență a acceselor.

Definirea conceptelor de tranzacție şi de acțiune permite punerea în lumină a unor probleme „elementare" ce pot să apară atunci când mai multe tranzacții încearcă să efectueze operațiuni de acces la aceleaşi elemente (sau obiecte) de date. Astfel de tentative conduc la situații de „**conflict**" între acțiuni, respectiv între tranzacții. Aşa cum se va vedea, operațiunile respective au loc la momente de timp foarte apropiate, fără însă a fi simultane în adevăratul sens al acestui termen. În cele ce urmează, ne vom referi la două tipuri de probleme: pierderile de operație şi inconsistențele.

Pierderi de operație. Un caz reprezentativ de pierdere de operație este „pierderea de actualizare", care poate fi urmărită în Fig. 1. Aşa cum se vede, o tranzacție T1 poate, în decursul efectuării unei acțiuni a_i, să citească un obiect de date într-un registru tampon de memorie (care îi aparține), urmând ca apoi să modifice valoarea acestui obiect (de exemplu, să o incrementeze cu o unitate) şi să înregistreze noua valoare în memori, în decursul unei alte acțiuni a_j (j > i). Este însă posibil ca o altă tranzacție T2

Tranzacția T1	__	**Tranzacția T2**
a_i : Read X -> x_1	---	
	---	a_k : Read X -> x_2
	---	$x_2 + 1 -> x_2$
	---	Write x_2 -> X
a_j : $x_1 + 1 -> x_1$	---	
Write x_1 -> X	---	

Fig. 1

să fi efectuat o actualizare (incrementare) a aceluiaşi element de date, printr-o acțiune a_k, amplasată între acțiunile a_i şi a_j. Actualizarea reprezentată de acțiunea a_k, în tranzacția T2, este însă pierdută.

Inconsistențe. Mai întâi, vom trata cazul în care o tranzacție afişează conținutul unei baze de date care se găseşte în stare tranzitorie. Admitem că baza de date (desigur, foarte elementară, formată

numai din elementele de date A şi B) trebuie să respecte restricția de integritate A = B. Aşa cum se vede în Fig. 2, tranzacția T2 citeşte A după ce acest element de date a fost modificat de tranzacția T1, respectiv citeşte elementul B înainte ca acesta să fi fost incrementat de T1. În acest mod, tranzacția T2 va afişa pentru A o valoare diferită de valoarea lui B.

Tranzacția T1	A=B	**Tranzacția T2**

Read A -> a_1	----	
a_1 + 1 -> a_1	----	
Write a_1 -> A	----	
	----	Read A -> a_2
	----	Print a_2
	----	Read B -> b_2
	----	Print b_2
Read B -> b_1	----	
b_1 + 1 -> b_1	----	
Write b_1 -> B	----	

Fig. 2

Să presupunem acum că tranzacția T2 încearcă să actualizeze baza de date atunci când aceasta este în regim tranzitoriu, respectiv se găseşte într-o stare de ne-coerență, în care elementele de date nu sunt egale (A este diferit de B). Evoluția tranzacțiilor T1 şi T2 este descrisă mai jos, în Fig. 3:

Tranzacția T1	A=B	**Tranzacția T2**

Read A -> a_1	----	
a_1 + 1 -> a_1	----	
Write a_1 -> A	----	
	----	Read A -> a_2
	----	a_2 x 2 -> a_2
	----	Write a_2 -> A
	----	Read B -> b_2
	----	b_2 x 2 -> b_2
	----	Write b_2 -> B
Read B -> b_1	----	
b_1 + 1 -> b_1	----	
Write b_1 -> B	----	

Fig. 3

Se poate observa imediat că prin aranjarea în timp de mai sus a acțiunilor tranzacțiilor T1 şi T2, se obține în final o bază de date necoerentă, caracterizată prin încălcarea restricției de integritate A=B. În mod obişnuit, se afirmă că printr-o asemenea dispunere în timp a acțiunilor, tranzacțiile T1 şi T2 se găsesc în stare de conflict, respective execuția (sau planificarea lor) implică un conflict. Este deci firesc să căutăm condițiile în care execuțiile de tranzacții sunt astfel planificate, încât să fie evitate conflictele.

Planificări fără conflict

Mai întâi, vor fi introduce câteva alte noțiuni care ocupă un loc însemnat în studiul acceselor concurente, respective al **planificării fără conflict** a execuției tranzacțiilor. Prin **"controlor**" (în literature străină se folosesc termenii "controller" sau "scheduler") înțelegem un modul al sistemului de gestiune a bazelor de date care urmăreşte şi comandă accesul concurrent la informațiile conținute în o baza de date. În acelaşi context, denumim **granulă** o unitate componentă a unei baze de date, care este controlată în mod individual de un controlor.

Una din problemele care se ridică în contextul prelucrării bazelor de date în regim de concurență a acceselor este alegerea unor dimensiuni adecvate ale granulelor. Aceste dimensiuni se pot înscrie în o gamă întinsă de la lungimea unei înregistrări până la lungimea unei pagimi sau a unui întreg fişier. La limită, întreaga bază de date poate reprezenta o granulă.

De dimensiunile granulelor sunt legate performanțele procesului de acces al tranzacțiilor la o bază de date. În o abordare generală, se înțelege că atunci când baza de date este divizată într-un număr mare de granule de dimensiuni mici, au acces la informație mai multe tranzacții, în acest mod realizându-se un grad mai înalt de „paralelism" în prelucrarea bazei de date. Pe de altă parte, în acest caz procesul de control al concurenței acceselor este mai complex şi va reclama un interval de timp mai lung. În practică se recomandă o dimensiune variabilă a granulelor.

Datele conținute în fiecare granulă trebuie să respecte restricții de integritate care decurg din restricțiile generale acceptate pentru ansamblul bazei de date. Vom observa aici că în procesul modificării bazei de date, granulele se modifică de asemenea, ca efect al execuției unor acțiuni. Ne reamintim că acțiunile sunt unități funcționale de prelucrare secvențială, care nu trebuie să încalce restricțiile de integritate valabile pentru datele ce aparțin granulelor şi, deci, respectă coerența internă a acestora.

Având în vedere elementele de mai sus, vom înțelege prin **operație** o succesiune de acțiuni care îndeplinesc o funcție bine definită asupra granulei, respectând coerența internă a acesteia. În funcție de tipul sistemului de gestiune şi de modul în care sunt controlate accesele concurente, natura operațiilor poate fi diferită. În unele cazuri, operațiile sunt interpretate ele însele ca procese- sau acțiuni- indivizibile, ce efectuează prelucrări elementare (înscrieri de informație, citiri, inserări etc.) asupra unor elemente de date de tipul înregistrărilor, paginilor, etc. Starea unei granule în urma efectuării unei operații, ca şi unele efecte secundare la care a condus execuția operației, poartă numele de **rezultat** al operației. De pildă, rezultatul execuției unei tranzacții care prelucrează o bază de date este starea granulelor bazei de date după încheierea tranzacției, precum şi conținutul eventualelor mesaje editate în urma acestui proces.

Planificarea (execuția) unei mulțimi de tranzacții.

O idee centrală avută în vedere pe parcursul considerațiunilor anterioare, constă în faptul că în procesul prelucrării unei baze de date, sub controlul sistemului de gestiune, este efectuată o succesiune de acțiuni care fac parte dintr-un ansamblu de tranzacții concurente. Evident că în funcție de poziția pe care ne situăm, ca observatori, în decursul prelucrării unei baze de date, putem concepe, pregăti, urmări şi controla procesul execuției tranzacțiilor concurente, sau putem analiza modul în care acest proces s-a desfăşurat, după încheierea sa. Având în vedere aceste poziții, denumirile atribuite procesului prelucrării tranzacțiilor sunt diferite. În cazul prim, se foloseşte termenul de **planificare** sau **execuție** (în literatura străină, „schedule"), iar în cel de al doile caz- termenul de **istorie** sau chiar cel de **înregistrare** sau de **jurnal** („log").

Fie mulțimea de tranzacții { T1, T2, ..., Tn }. Atunci, prin (T1, T2, ..., Tn) putem înțelege fie succesiune a tranzacțiilor efectuate, deci o „istorie" a sistemului care prelucrează accese concurente, fie o „planificare" a viitoarei execuții concurente a acestor tranzacții.

Având în vedere execuția viitoare a tranzacțiilor, putem defini **planificarea (T1, T2,...., Tn) a acestora ca o succesiune de acțiuni, obținută printr-o intercalare a acțiunilor din compunerea tranzacțiilor T1, T2, ..., Tn, respectând ordinea internă în care se găsesc aceste acțiuni în fiecare tranzacție.** Vom remarca faptul că putem înțelege în mod similar şi istoria execuției

grupului de tranzacții. Aşadar, **o planificare respectă cu exactitate ordinea acțiunilor fiecărei tranzacții constituente, fiind- prin definiție- secvențială.**
Pentru a ilustra considerațiunile de mai sus, fie exemplul traanzacțiilor T1 şi T2, ale căror acțiuni sunt însă enumerate în ordinea lor, fără a mai fi indicată amplasarea în timp (Fig. 4):

Tranzacția T1	**Tranzacția T2**
Read A -> a_1	Read A -> a_2
a_1 + 1 -> a_1	a_2 x 2 -> a_2
Write a_1 -> A	Write a_2 -> A
Read B -> b_1	Read B -> b_2
b_1 + 1 -> b_1	b_2 x 2 -> b_2
Write b_1 -> B	Write b_2 -> B

Fig. 4

În descrierea dată, tranzacțiile T1 şi T2 modifică elementele A şți B, legate prin restricția de integritate A=B. Elementele A şi B aparțin la două granule distincte, ceea ce contribute la creşterea posibilităților de concurență. În continuare (Fig. 5 şi Fig. 6), sunt prezentate două planificaări posibile pentru tranzacțiile T1 şi T2. PLanificarea din Fig. 6 este inacceptabilă, deoarece conduce la o pierdere de operație.

Tranzacția T1	**Tranzacția T2**
Read A -> a_1	
a_1 + 1 -> a_1	
Write a_1 -> A	
	Read A -> a_2
	a_2 x 2 -> a_2
	Write a_2 -> A
Read B -> b_1	
b_1 + 1 -> b_1	
Write b_1 -> B	
	Read B -> b_2
	b_2 x 2 -> b_2
	Write b_2 -> B

Fig. 5

Tranzacția T1	**Tranzacția T2**
	Read A -> a_2
	a_2 x 2 -> a_2
Read A -> a_1	
a_1 + 1 -> a_1	
	Write a_2 -> A
	Read B -> b_2
	b_2 x 2 -> b_2
Write a_1 -> A	

Read B -> b_1
b_1 + 1 -> b_1
Write b_1 -> B

Write b_2 -> B.

Fig. 6

Planificare serializabilă

Aşa cum am văzut, unele planificări de tranzacţii conduc la pierderi de operaţii sai inconsistenţe. Un obiectiv principal al controlului concurenţei acceselor este de a nu permite dacât execuţia acelor planificări, care nu sunt însoţite de pierderi de operaţie sau de inconsistenţe.

O soluţie posibilă a problemei de mai sus poate fi execuţia succesivă (serială) a tranzacţiilor. Prin adoptarea unei asemenea planificare se poate evita complet apariţia pierderilor de operaţie şi inconsistenţelor, efectuarea acţiunilor ce compun fiecare tranzacţie desfăşurându-se separat şi independent de evoluţia altor tranzacţii, fără intercalarea în timp a acestor acţiuni. Execuţia serială a unui grup de tranzacţii poartă numele de **succesiune**.

Drept consecinţă a considerentelor expuse, rezultă că în scopul excluderii posibilităţii de conflict între acţiunile tranzacţiilor, trebuie permise numai acele planificări care conduc la aceleaşi rezultate ca cele date de o succesiune, pentru fiecare dintre tranzacţii. Planificările de acest tip se numesc **serializabile. Prin definiţie, o planificare a tranzacţiilor (T1, T2, ...,Tn) este serializabilă, dacă pentru fiecare tranzacţie participantă ea conduce la aceleaşi rezultate ca o succesiune a tranzacţiilor (T1, T2, ...,Tn).**

Prin urmare, din punctul de vedere al controlului concurenţei acceselor diferitelor tranzacţii la informaţiile conţinute în o baza de date, sarcina sistemelor de gestiune (fie ele centralizate sau distribuite) este de a genera numai planificări serializabile. În acest mod, este asigurată absenţa conflictelor între acţiunile tranzacţiilor. Pentru realizarea acestui deziderat, operaţile din care este compusă execuţia tranzacţiilor trebuie să posede proprietăţi specifice.

Proprietăţi ale operaţiilor.

Vom distinge două clase de operaţii care, prin proprietăţile lor, pot asigura serializabilitatea planificării tranzacţiilor: operaţii compatibile şi operaţii permutabile.

Înainte de a defini operaţiile compatibile, vom observa că două operaţii ce fac parte din tranzacţii diferite şi care nu efectuează modificări în nicio entitate, pot fi întotdeauna executate simultan, fără a intereveni schimbări reciproce asupra rezulatelor lor. Cu alte cuvinte, de exemplu, orice fel de intercalare de operaţii care nu fac decât citiri, conduce la rezultate identice în cazul unei execuţii succesive a operaţiilor.

Prin definiţie, operaţiile O1 şi O2 sunt **compatibile** dacă şi numai dacă, oricare ar fi execuţia lor simultană , rezultatul acesteia este acelaşi cu cel al execuţiei secvenţiale (seriale) a operaţiei O1 urmată de operaţia O2, sau a operaţiei O2 urmată de O1. Remarcăm faptil că rezultatele execuţiilor O1 urmată de O2, respectiv O2 urmată de O1, pot fi diferite.

Pentru aprofundqreq considerqùiilor preyentqte pot fi folosite exemplele din Fig. 7, în care operaţiile O11 şi O21 sunt compatibile, pe când O21 şi O22, respectiv O13 şi O23 nu sunt compatibile.

O11 Read A -> a_1
a_1 x 2 -> a_1
Print a_1

O12 Read A -> a_1
a_1 x 2 -> a_1
Write a_1 -> A

O21 Read A -> a_2
a_2 + 1 -> a_2
Print a_2

O22 Read A -> a_2
a_2 + 1 -> a_2
Write a_2 -> A

O13	Read A -> a_1	O23	Read A -> a_2
	a_1 + 1 -> a_1		a_2 + 10 -> a_2
	Write a_1 -> A		Write a_2 -> A

Fig. 7

Este evident că operaţiile care prelucrează granule diferite sunt întotdeauna compatibile, întrucât pentru un asemenea caz nu pot surveni pierderi de operaţie dacă operaţiile se intercalează. Se înţelege că atunci când există posibilitatea de intercalare prin care se generează o pierdere de operaţie, operaţiile implicate nu vor fi compatibile.
Ordinea în care pot fi executate operaţiile şi rezultatele acestei execuţii determină o altă proprietate importantă a unor clase de operaţii. Să începem prin a remarca faptul că, în unele situaţii,rezultatele unei execuţii de operaţii poate depinde de ordinea în care aceasta este realizată. Pe de altă parte, în alte cazuri ordinea poate fi indiferentă. În acelaşi context, să mai observăm că proprietatea de compatibilitate a operaţiilor este independentă de ordinea axestora.
Vom defini două operaţii O1 şi O2 ca fiind **permutabile**, dacă şi numai dacă orice execuţie a acestor operaţii în ordinea O1, O2 conduce la acelaşi rezultat cu execuţia în ordinea O2, O1. Revenind la Fig. 7, operaţiile O13 şi O23 sunt permutabile, în timp ce O12 şi O22 nu sunt permutabile. Sunt întotdeauna permutabile două operaţii care prelucrează granule diferite, execuţia uneia din aceste operaţii neputând schimba rezultatul execuţiei celei de a doua operaţii şi invers.

Compatibilitate, permutabilitate, serializabilitate
În cele ce urmează, va fi pusă în lumină dependenţa proprietăţii de serializabilitate a unei execuţii de tranzacţii, de proprietăţile de compatibilitate şi permutabilitate ale operaţiilor care compun tranzacţiile.
Mai întâi, subliniem douî transformări de bază la care pot nfi supuse planificaările de tranzacţii:
-**separarea operaţiilor compatibile O1 şi O2 executate de două tranzacţii diferite**, constă în înlocuirea execuţiei simultane E(O1, O2) a acestora, printr-o secvenţă care conduce la acelaşi rezultat, respectiv (O1, O2), sau (O2, O1). Aşadar, procesul de separare permite aranjarea în succesiune a oparaţiilor compatibile executate de către tranzacţii diferite.
-**Permutarea operaţiilor permutabile O1 şi O2**, executate de două tranzacţii diferite, constă în schimbarea ordinei de execuţie a acestor operaţii (de pildă, secvenţa (O1, O2) este înlocuită cu secvenţa (O2, O1).
Luând în considerare cele două transformări definte anterior, rezultă că o **condiţie suficientă pentru ca o planificare de tranzacţii să fie serializabilă este ca prin separarea operaţiilor compatibile şi prin permutarea operaţiilor permutabile, planificarea să fie transformată în o succesiune de tranzacţii componente.**
Pentru a ilustra consideraţiunile prezentate, să examinăm exemplul planificării din Fig. 5. Dacă ne oprim numai asupra operaţiilor, planificarea poate fi reprezentată în forma:

T1 : A + 1 -> A
T2 : A x 2 -> A
T1 : B + 1 -> B
T2 : B x 2 -> B.

Operaţiile A x 2 -> A şi B + 1 -> B sunt permutabile, deoarece lucrează asupra unor granule diferite. În consecinţă, planificarea poate fi transformată în următoarea:

T1 : A + 1 -> A
T1 : B + 1 -> B

T2 : A x 2 -> A
T2 : B x 2 -> B.

Noua planificare obținută este o succesiune de tranzacții, respectiv tranzacția T1 urmată de tranzacția T2. Ca urmare, planificarea reprezentată în Fig. 5 este serializabilă.

Grafuri de precedență.
Vom defini mai întâi noțiunea de precednță, pentru orice planificare ce nu conține operații simultane (spre exemplu, planificări obținute prin separare de operații compatibile). În acest scop, să admitem că P este o planificare ce nu conține operații simultane. Pein definiție, afirmăm că în planificarea P are loc **precedența tranzacției T1 față de tranzacția T2** (adică T1 precede T2, cu notația T1 < T2), dacă şi numai dacă există două operații nepermutabile O1 şi O2, astfel încât operația O1 este executată de tranzacția T1 înainte ca operația O2 să fie executată de tranzacția T2.
Pentru a descrie interacțiunile dintre tranzacții în timpul unei execuții poate fi folosit **graful de precedență.** În graful de precedență al unei planificări se asociază un nod cu fiecare tranzacție, între nodurile T1 şi T2 fiind trasat un arc orientat de la nodul T1 la nodul T2 dacă şi numai dacă tranzacția T1 precede tranzacția T2. Potrivit unei teoreme specifice, pe care nu o demonstrăm aici, o condiție suficientă ca o planificare de tranzacții să fie serializabilă este ca graful de precedență asociat cu planificarea să nu conțină contururi închise sau bucle proprii.
Pentru exemplificare, în Fig. 8 este prezentat graful de precedență al planificării din Fig. 5. În graf nu avem conturi închise, deci planificarea este serializabilă.

T1 T2

Fig. 8

Un alt exemplu, cel următor, se referă la planificarea din Fig. 9, care nu este serializabilă.

T1 : A + 1 -> A
T2 : A x 2 -> A
T2 : B x 2 -> B
T1 : B+ 1 -> B

T1 T2

Aşa cum se vede, graful de precednță al planificării – reprezentat în aceeaşi figură- are un contur închis, fapt care potrivit teoremei enunțate confirmă ne-serializabilitatea planificării propuse.