Universitatea Politehnica Bucureşti
Masterul „Ingineria Informaţiei şi a Sistemelor de Calcul”
Sisteme de Operare Avansate

TEMA CURS

OPERAŢII CONCURENŢIALE ASUPRA BAZELOR DE DATE

ADMINISTRAREA TRANZACŢIILOR

HANU BOGDAN MARICEL

Contents

OPERAŢII CONCURENTE ASUPRA BAZELOR DE DATE. ADMINISTRAREA TRANZACŢIILOR

1. INTRODUCERE

Pentru a putea înţelege cu uşurinţă tema abordată în această lucrare, este necesar sa fie definiţi termenii principali, să se expună avantajele şi dezavantajele folosirii bazelor de date si să se indice paşii principali în realizarea unei aplicaţii de baze de date.

Baza de date:

O colecţie partajată de date, între care există relaţii logice (şi o descriere a acestor date), proiectată pentru a satisface necesităţile informaţionale ale unei organizaţii.

O bază de date este **coerentă** dacă datele pe care le conţine respectă ansamblul restricţiilor de integritate implicită sau explicită ce au fost definite în contextul definirii bazei de date.

Integritatea datelor

Este reprezentată de concordanţa acestora cu proprietăţile sistemelor din lumea reală pe care le modelează. Respectarea integrităţii datelor este o problemă esenţială în cazul bazelor de date la care au loc accese concurente.

SGBD:

Un sistem de programe care permite utilizatorului definirea, crearea şi întreţinerea bazei de date si accesul controlat la aceasta.

SQL (Structured Query Language)

Este în prezent, unul din cele mai puternice limbaje structurate pentru interogarea bazelor de date relaţionale. Este un limbaj neprocedural şi declarativ, deoarece utilizatorul descrie ce date vrea să obţină, fără a fi nevoie să stabilească modalităţile de a ajunge la datele respective. Nu poate fi considerat un limbaj de programare sau unul de sistem, ci mai degrabă face parte din categoria

limbajelor de aplicaţii, fiind orientat pe mulţimi. Foarte frecvent, este utilizat în administrarea bazelor de date client/server, aplicaţia client fiind cea care generează instrucţiunile SQL.

*Limbaj de definire a datelor (DDL)***:**

Limbajul DDL permite utilizatorilor specificarea tipurilor de date şi a structurilor, în timp ce constrângerile asupra datelor sunt stocate în baza de date.

Limbaj de manipulare a datelor (DML) **:**

Permite utilizatorilor sa insereze, sa reactualizeze, sa şteargă si sa extragă date din baza de date.

După definirea acestor limbaje le SGBD-urilor, putem redefini limbajul SQL ca fiind un limbaj de interogare a BD neprocedural, care constituie limbajul standard pentru SGBD relaţionale.

SGBD oferă **accesul controlat** la baza de date. De exemplu, poate furniza:

- un sistem de securitate, care previne accesarea bazei de date de către utilizatori neautorizaţi;
- un sistem de integritate, care menţine concordanţa datelor stocate;
- un sistem de control al concurentei, care permite accesul partajat la baza de date;
- un sistem de control al refacerii, care restaurează baza de date într-o stare precedentă concordantă, ca urmare a unei defecţiuni in hardware sau software;
- un catalog accesibil utilizatorilor, care conţine descrieri ale datelor din baza de date.

Bazele de date prezintă numeroase **avantaje** printre care:

- Controlul redundanţei datelor
- Coerenţa datelor
- Mai multe informaţii de la aceeaşi cantitate de date
- Partajarea datelor
- Integritatea crescută a datelor
- Securitatea crescută
- Aplicarea standardelor
- Economia de scala

- Echilibrul între cerinţele aflate in conflict
- Îmbunătăţirea accesibilităţii datelor si capacitaţii de răspuns
- Productivitate crescută
- Capacitatea de întreţinere îmbunătătită, prin independenţă de date
- Concurenţa îmbunătăţită
- Îmbunătăţirea serviciilor de salvare, de sigurantă si refacere

Există insă si anumite **dezavantaje** în lucrul cu bazele de date:

- Complexitatea
- Dimensiunea
- Costul sistemelor SGBD
- Costurile adiţionale pentru elemente de hardware
- Costul conversiei
- Performanţa
- Impactul crescut al unei defecţiuni

Pentru a putea realiza o aplicaţie de baze de date cu succes trebuie urmăriţi anumiţi paşi:

- Analiza problemei reale,
- Proiectarea aplicaţiei, utilizând un SGBD,
- Realizarea modulelor necesare,
- Executărea aplicaţiei,
- Aplicaţia corespunde cerintelor reale?
- Nu, se modifica anumite module

Principalele declaraţii din limbajul SQL folosite pentru o aplicaţie sunt:

SELECT	Data retrieval
INSERT UPDATE DELETE	Data Manipulation Language (DML)
CREATE ALTER DROP RENAME TRUNCATE	Data Definition Language (DDL)
COMMIT ROLLBACK SAVEPOINT	Transaction Control
GRANT REVOKE	Data Control Language (DCL)

Un obiectiv major al SGBD este de a permite mai multor utilizatori să acceseze concurent datele partajate. Dar la sistemele în care o baza de date este accesată simultan de mai mulţi utilizatori apar situaţii de conflict datorate accesului concurent la datele care constituie resursă comună. Modul de rezolvare al conflictului depinde de natura cererilor de acces la date:

- Dacă cererile de acces sunt de tip regăsire, atunci secvenţialitatea accesului la mediul de memorare este suficientă şi nu mai este nevoie de precauţii suplimentare. Atât timp cât nici unul nu face modificări ale datelor, nu are importanţă ordinea în care se asigură accesul utilizatorului la date. Deci, în acest caz, rezolvarea situaţiilor conflictuale cauzate de operaţiile de citire simultană poate fi lăsată în seama sistemului de operare, care stabileşte ordinea de satisfacere a cererilor după criteriul asigurării concurenţei maxime de operare, minimizând timpul global de satisfacere a acestor cereri.
- Dacă unele cereri sunt de tip actualizare, este necesară aplicarea unor strategii adecvate de tratare a cererilor de acces. Exemplu tipic: sistemele de rezervare a locurilor în care mai mulţi agenţi fac permanent modificări. Pericolul: două procese, care citesc şi modifică valoarea aceluiaşi obiect ar putea interacţiona atunci când sunt executate simultan. Rezultă necesitatea impunerii unor restricţii asupra execuţiei concurente a proceselor care

fac operaţii de citire şi modificare a datelor. O rezolvare imediată şi simplă ar fi blocarea bazei de date pe durata rezolvării unei cereri. Dar asta echivalează cu blocarea concurenţei şi degradarea performanţelor sistemului, făcându-l uneori neutilizabil.

În proiectarea SGBD se caută algoritmi care să permită un înalt grad de concurenţă. Algoritmii de control ai concurenţei pentru operaţiile efectuate asupra BD se împart în 2 clase:

1. algoritmi de control prin blocare
2. algoritmi de control prin marcare

2. TRANZACŢII ŞI UNITĂŢI DE ACCES

Se numeşte tranzacţie orice execuţie a unui program. Programele pot fi de la simple interogări, până la proceduri complexe, cu multe interogări ori modificări ale bazei de date. Pot exista mai multe execuţii independente ale aceluiaşi program şi fiecare dintre acestea este o tranzacţie.

D.p.d.v. al utilizatorului, o tranzacţie este o unitate singulară de execuţie care satisface două condiţii referitoare la baza de date:

• baza de date este într-o stare coerentă înaintea şi după execuţia tranzacţiei.

• baza de date poate fi într-o stare necoerentă în timpul execuţiei unei tranzacţii.

O tranzacţie poate avea două rezultate:

1. dacă este completată cu succes, se spune că tranzacţia a fost efectuată iar BD ajunge într-o nouă stare coerentă;

2. dacă nu este executată cu succes, ea este abandonată iar BD trebuie refăcută în starea coerentă dinainte. O astfel de tranzacţie este rulată înapoi sau abandonată.

O tranzacţie efectuată nu mai poate fi abandonată.

Pentru a delimita tranzacţiile, în majoritatea limbajelor de manipulare a datelor sunt disponibile instrucţiunile BEGIN TRANSACTION, COMMIT şi ROLLBACK. Dacă utilizatorul nu foloseşte aceste delimitări, de obicei întregul program este considerat ca reprezentând o

singură tranzacţie iar SGBD execută automat o instrucţiune COMMIT atunci când programul este încheiat corect, sau una ROLLBACK în caz contrar.

• BEGIN TRANSACTION – începe o nouă tranzacţie

• COMMIT – salvează orice schimbări şi încheie tranzacţia curentă

• ROLLBACK – anulează orice schimbări făcute în timpul tranzacţei curente şi încheie tranzacţia

Odată folosită instrucţiunea BEGIN TRANSACTION orice actualizare nu va fi executată instantaneu, ci numai după COMMIT sau ROLLBACK pentru a încheia tranzacţia.

Proprietăţile tranzacţiilor

Există o serie de proprietăţi pe care trebuie să le aibă toate tranzacţiile. Cele patru proprietăţi fundamentale, aşa-numitele acid, sunt:

1. caracterul atomic: reprezintă proprietatea „tot sau nimic”. O tranzacţie este o unitate indivizibilă, care ori este efectuată în întregime, ori nu este efectuată de loc;
2. coerenţa: o tranzacţie trebuie să transforme BD dintr-o stare coerentă în altă stare coerentă;
3. izolarea: tranzacţiile sunt executate independent unele de altele. Efectele parţiale ale unei tranzacţii incomplete nu trebuie să fie vizibile pentru alte tranzacţii;
4. durabilitatea: efectele unei tranzacţii încheiate cu succes sunt înregistrate definitiv în BD şi nu trebuie pierdute din cauza unei pene ulterioare.

O BD este partiţionată în mai multe unităţi de acces (items). Acestea sunt porţiuni ale BD care pot constitui obiectul unei operaţii de blocare (lock). Prin blocarea unei unităţi de acces, o tranzacţie poate împiedica accesul altor tranzacţii la unitatea blocată, până la momentul deblocării acestei unităţi de către tranzacţia care a efectuat blocarea.

Gestiunea operaţiilor de blocare precum şi arbitrarea cererilor de blocare venite din partea tranzacţiilor este realizată de o componentă specială a SGBD, numită lock manager.

Natura şi dimensiunea unităţilor de acces este stabilită de proiectantul sistemului. De ex, în cazul modelului de date relaţional, unităţile de acces pot fi de dimensiuni foarte variate, cuprinzând relaţii întregi ale BD, grupuri de tuple, tuple individuale sau chiar componente ale tuplelor.

Prin alegerea unităţilor de acces de mari dimensiuni, se reduce numărul operaţiilor de blocare, ceea ce înseamnă reducerea timpului consumat de sistem pentru gestiunea acestor operaţii, precum şi reducerea spaţiului de memorie necesar înregistrării blocajelor.

În schimb, folosind unităţi de acces de mici dimensiuni, creşte gradul de concurenţă suportat de sistem, deoarece pot fi executate în paralel un număr mai mare de tranzacţii care operează în unităţi de acces diferite.

În practică, dimensiunea potrivită a unităţilor de acces este dată de extinderea operaţiilor efectuate de tranzacţiile cu cea mai mare frecvenţă. Astfel, dacă tranzacţia tipică presupune efectuarea unor operaţii de cuplare, atunci unitatea de acces va fi relaţia. Dacă, însă, majoritatea tranzacţiilor efectuează operaţii asupra unor tuple individuale, atunci va fi convenabil să se aleagă tupla ca unitate de acces.

3. GESTIUNEA TRANZACŢIILOR

Sistemele de programe care realizează controlul concurenţei sunt acea parte a sistemului de gestiune a bazei de date (SGBD), care urmăreşte execuţa „simultană” a tranzacţiilor, astfel încât să fie produse aceleaşi rezultate ca şi în cazul unei execuţi secvenţiale.

Conceptul de gestiune a tranzacţiilor se referă la problematica menţinerii într-o stare consistentă a bazei de date în condiţiile în care accesul la date se face într-un regim concurent şi este posibilă apariţia unor defecte. În consecinţă se disting două domenii de sine stătătoare în cadrul problematicii generale a gestiunii tranzacţiilor :

Controlul concurenţei

Se ocupă cu mecanismele de sincronizare a acceselor astfel încât să fie menţinută integritatea bazei de date. Atunci când controlul concurenţei este realizat prin mecanismele de blocare (care la ora actuală sunt cele mai răspândite) mai apare o problemă, colaterală, şi anume

aceea a interblocărilor. Datorită importanţei sale problema gestiuni interblocărilor este de multe ori tratată ca o problematică de sine stătătoare a gestiunii tranzacţiilor.

Rezistenţa la defecte

Se referă la tehnicile prin care se asigură atât toleranţa sistemului faţă de apariţia unor defecte, cât şi capacitatea de recuperare a acestuia, adică posibilitatea de revenire la o stare consistentă în urma apariţiei unui defect care a cauzat intrarea lui într-o stare inconsistentă.

Prin controlul concurenţei şi rezistenţa la defecte se urmăreşte asigurărea consistenţei şi siguranţei bazei de date. O bază de date este într-o stare consistentă dacă respectă toate constrângerile de integritate a datelor definite asupra sa. În timpul operaţiilor de adăugare, ştergere şi modificare, baza de date trece dintr-o stare în alta. Evident, starea rezultată după orice prelucrare asupra bazei de date trebuie să fie o stare consistentă, chiar dacă în timpul prelucrării baza de date s-a aflat temporar într-o stare inconsistentă.

Siguranţa bazei de date se referă la toleranţa acesteia faţă de defecte şi la capacitatea de recuperare după apariţia unui defect.

O tranzacţie este o unitate logică de prelucrare care asigură consistenţa şi siguranţa bazei de date. În principiu, orice execuţe a unui program se poate considera o tranzacţie dacă baza de date este într-o stare consistentă atât înainte cât şi după execuţia sa. Consistenţa bazei de date este garantată independent de faptul că :

- tranzacţia a fost executată în mod concurent cu alte tranzacţii ;
- au apărut defecte în timpul execuţei tranzacţiei.

În general, o tranzacţie constă dintr-o secvenţă de operaţii de citire şi scriere a bazei de date, la care se adaugă o serie de operaţii de calcul. Baza de date poate fi într-o stare temporar inconsistentă în timpul executării tranzacţiei dar trebuie să fie în stări consistente atât înainte cât şi după execuţia acesteia.

O tranzacţie nu se termină întotdeauna cu succes totuşi orice tranzacţie trebuie să se termine, indiferent de situaţia existentă (chiar şi în cazul unor defecte). Dacă tranzacţia reuşeşte să

execute cu succes toate operaţiile prevăzute, atunci aceasta se va termina printr-o operaţie de validare (**commit**). În schimb, dacă dintr-un motiv sau altul tranzacţia nu reuşeşte să-şi execute complet operaţiile prevăzute, atunci se va termina printr-o operaţie de abortare (**abort** sau **rollback**). Motivele pentru care o tranzacţie se abortează sunt numeroase, ele pot fi interne tranzacţiei sau externe acesteia (ex. : detectarea de către SGBD a unei situaţii de interblocare). În cazul abortării, execuţa tranzacţiei este oprită iar efectele tuturor operaţiilor pe care le-a executăt până în acel moment sunt anulate astfel încât baza de date revine la starea de dinaintea lansării tranzacţiei.

Comanda de validare a unei tranzacţii are dublu rol :

- indică SGBD-ului momentul de la care efectele tranzacţiei pot fi reflectate în baza de date şi devin vizibile altor tranzacţii ;
- marchează momentul începând de la care efectele tranzacţiei nu mai pot fi anulate (tranzacţia nu se mai poate aborta) şi modificările efectuate în baza de de devin permanente.

Operaţia de validare este vitală în cazul sistemelor concurente, deci acolo unde este posibilă executărea în acelaşi timp a mai multor tranzacţii care accesează aceeaşi bază de date. Prin validare se pot preveni o serie de fenomene nedorite cum este abortarea în cascadă a tranzacţiilor.

Să presupunem că o tranzacţie T este abortată după ce a efectuat una sau mai multe operaţii de actualizare a bazei de date. În acest caz datele alterate de către tranzacţia T vor fi readuse la valorile pe care le-au avut înainte de a fi modificate de aceasta. Este însă posibil ca unele dîntre tranzacţiile executăte în mod concurent cu tranzacţia T să fi accesat aceste date înainte de abortarea lui T. Aceste tranzacţii vor trebui să fie, la rândul lor, abortate deoarece au avut acces la date inconsistente din baza de date iar rezultatele produse de ele pot fi compromise. Acest efect se poate propaga în conţinuare şi asupra altor tranzacţii, pe un număr nedefinit de nivele, conducând la abortarea în cascadă a tranzacţiilor. Fenomenul este cunoscut în literatura de specialitate sub numele de efect domino.

Dacă se foloseşte un mecanism de validare care respectă cele două reguli de mai sus, atunci apariţia fenomenului de abortare în cascadă devine imposibilă. Într-adevăr, conform primei reguli, nici o tranzacţie nu va putea accesa datele modificate de tranzacţia T decât după validarea acesteia. Pe de altă parte, conform regulii a doua, după validarea sa, tranzacţia T nu mai poate fi abortată, deci nu poate declanşa un lanţ de abortări în cascadă.

Validarea unei tranzacţii marchează, din punct de vedere logic, terminarea acesteia. Validarea nu se poate face înainte ca operaţiile specificate prin codul tranzacţiei să fie executăte integral şi înainte ca tranzacţia să ajungă într-o stare începând de la care există certitudinea că nu mai poate fi abortată.

Până în momentul validării, actualizările efectuate de tranzacţie sunt invizibile altor tranzacţii, au caracter tentativ şi pot fi oricând revocate (odată cu abortarea tranzacţiei). După validare actualizările se înscriu cu caracter permanent în baza de date şi devin irevocabile. După validare nu mai este posibilă abortatrea tranzacţiilor.

Tranzacţiile ar trebui să conţină doar acele comenzi DML care realizează o singură modificare asupra datelor. De exemplu un transfer de fonduri (să spunem 1000$) între două conturi ar trebui să implice un debit al unui cont de 1000$ şi un credit al altui cont de 1000$. Ambele acţiuni ar trebui să se încheie cu succes sau să dea eroare împreună. Creditul nu ar trebui executăt fără debit.

4. *ANOMALII LA INTERFERENŢĂ*

Interacţiunea necontrolată a două sau mai multe tranzacţii poate duce la apariţia unor stări inconsistente ale BD şi la producerea unor rezultate eronate. Două tranzacţii Ti şi Tj sunt susceptibile de interferenţă dacă rezultatul execuţiei lor concurente poate fi diferit de rezultatul execuţiei seriale.

Între două tranzacţii poate să apară o interferenţă dacă acestea efectuează operaţii asupra unor date comune. Dacă aceste două tranzacţii sunt executate în mod concurent, atunci spunem că sunt conflictuale.

Deci două tranzacţii Ti şi Tj sunt conflictuale dacă sunt concurente şi susceptibile de interferenţă.

În funcţie de natura operaţiilor pe care le efectuează asupra datelor comune, între două tranzacţii pot să apară mai multe tipuri de interferenţe care provoacă anomalii de interferenţă:

- anomalia de actualizare pierdută
- anomalia de citire improprie
- anomalia de citire irepetabilă (citire murdară)

Anomalia de actualizare pierdută

Corespunde unui conflict de tip scriere-scriere şi constă în faptul că rezultatul actualizării efectuate de o tranzacţie se pierde ca urmare a reactualizării aceleiaşi date de către o altă tranzacţie, fără ca reactualizarea să fie influienţată de rezultatul primei actualizări.

Exemplu: fie următoarea execuţie concurentă a două tranzacţii T1 şi T2.

Observaţie: în acest capitol vom nota simbolic operaţiile de citire şi scriere a unui atribut X din BD prin Read X, respectiv Write X.

T1	T2
	Read A
Read A	
A=A+5	
Write A	
	A=A+10
	Write A

În urma acestor tranzacţii valoarea lui A apare mărită cu 10, nu cu 15 aşa cum ar fi de aşteptat. Este ca şi cum tranzacţia T1 nici nu s-ar fi executat.

Anomalia de citire improprie

Corespunde unui conflict de tip scriere-citire şi apare atunci când o tranzacţie surprinde o stare temporar inconsistentă a BD.

Exemplu: fie următoarea execuţie concurentă a două tranzacţii T1 şi T2.

T1	T2
Read A	
A=A-10	
Write A	
	Read A
	Read B
	C=A+B
	Write C
Read B	
B=B+10	
Write B	

În urma acestor tranzacţii valoarea sumei A + B este neschimbată. Intenţia era de a reţine în C valoarea acestei sume, dar datorită interferenţei, valoarea din C este cu 10 mai mică decât cea reală.

Anomalia de citire irepetabilă

Corespunde unui conflict de tip citire-scriere şi apare atunci când aceeaşi tranzacţie găseşte valori diferite la citiri repetate ale aceleiaşi date.

Exemplu: fie următoarea execuţie concurentă a două tranzacţii T1 şi T2.

T1	T2
Read A	
B=A	
Write B	

	Read A
	A=A+10
	Write A
Read A	
C=A	
Write C	

Se observă că, deşi valorile rezultate pentru B şi C ar trebui să fie egale în urma execuţiei tranzacţiei T1, ele sunt diferite din cauza interferenţei cu tranzacţia T2.

5. PRIMITIVELE LOCK ŞI UNLOCK

Fie tranzacţiile T1 şi T2 execuţii diferite ale următoarei secvenţe de operaţii:

Read A

A = A + 1

Write A

unde A este o valoare existentă în bază. Fiecare din cele două tranzacţii citeşte valoarea lui A într-o zonă de lucru proprie, adună 1 la această valoare şi apoi scrie rezultatul în baza de date. După execuţia tranzacţiilor, este de aşteptat ca valoarea lui A să fie mărită cu 2. Totuşi, dacă tranzacţiile sunt executate concurent, este posibil ca rezultatul final să fie altul, funcţie de modul de interferenţă a tranzacţiilor.

Cea mai simplă metodă de evitare a situaţiilor de genul celei prezentate, este de a permite accesul la valoarea lui A numai pentru o singură tranzacţie, pe toată durata executării tranzacţiei. Accesul celorlalte tranzacţii va fi temporar blocat. Acest lucru se poate realiza folosind două funcţii primitive LOCK (A) şi UNLOCK (A). Aceste funcţii se numesc primitive, deoarece

secvenţa de operaţii corespunzătoare execuţiei lor nu poate fi întreruptă de alte operaţii. LOCK (A) şi UNLOCK (A) sunt operaţii indivizibile.

Dacă o tranzacţie Tk execută cu succes o primitivă LOCK (A), atunci componenta lock manager a SGBD asigură accesul exclusiv al tranzacţiei Tk la valoarea A, interzicând accesul la această valoare a oricărei alte tranzacţii atâta timp cât tranzacţia Tk nu eliberează valoarea A prin execuţia primitivei UNLOCK (A). Se spune că valoarea A este blocată în acest interval de timp. O tranzacţie poate executa cu succes o primitivă LOCK() doar asupra unei valori care nu este blocată. În acest caz valoarea returnată de funcţia LOCK() este TRUE. Orice tentativă de a executa primitiva LOCK() asupra unei valori blocate va eşua, valoarea returnată fiind FALSE. Acesta este cel mai simplu mecanism de a asigura excluderea mutuală.

Primitivele LOCK() şi UNLOCK() pot fi folosite pentru realizarea mecanismelor de sincronizare a tranzacţilor. Dacă o tranzacţie doreşte să acceseze o anumită valoare, ea va trebui să obţină accesul exclusiv la aceasta, prin executarea unei primitive LOCK(). Dacă valoarea este blocată, atunci accesul exclusiv va fi refuzat, iar tranzacţia va trebui să aştepte până la deblocarea aceste valori. Orice tranzacţie care a executat cu succes o primitivă LOCK() asupra unei valori, va trebui să execute primitiva UNLOCK() asupra aceleiaşi valori înainte de a-şi încheia execuţia. Ordonarea secvenţială a paşilor a două sau mai multe tranzacţii care respectă regulile de mai sus, se spune că este legată.

Pentru tranzacţiile T1 şi T2 considerate anterior, secvenţa de paşi folosind primitivele LOCK() şi UNLOCK() este următoarea:

```
While NOT(LOCK(A))

Read A

A = A + 1

Write A

UNLOCK(A)
```

Dacă una dintre tranzacţii, să zicem T1, obţine accesul exclusiv la valoarea A, atunci tranzacţia T2 va trebui să aştepte terminarea completă a lui T1 pentru a obţine accesul la valoarea A. La terminarea tranzacţiei T1 valoarea lui A este mărită cu 1, iar la terminarea tranzacţiei T2, valoarea lui A va fi mărită cu 2.

Rezultatul este corect, dar execuţia este pur secvenţială, nu este posibil nici un fel de paralelism între cele două tranzacţii. D.p.d.v. al timpului de execuţie, această situaţie este inacceptabilă şi de aceea se folosesc algoritmi care să relizeze ordonări secvenţiale legate, cu un grad de concurenţă a execuţiei cât mai ridicat, dar şi cu garanţia obţinerii de rezultate corecte.

6. INTERBLOCAREA

Unul dintre principalele obiective ale oricărui sistem concurent este folosirea în comun a resurselor, adică partajarea acestora. În cazul bazelor de date concurente, cea ai importantă resursă partajabilă o constituie datele. Atunci când datele sunt partajate de către un grup de tranzacţii concurente şi fiecare tranzacţie deţine controlul exclusiv al unor date particulare, este posibil să se ajungă la situaţia în care unele tranzacţii nu-şi vor putea termina niciodată execuţia.

Exemplu. Fie două tranzacţii T1 şi T2 definite prin două secvenţe de forma:

T1	T2
While NOT(LOCK(A))	While NOT (LOCK(B))
While NOT(LOCK(B))	While NOT(LOCK(A))
....	.....
Prelucrare 1	Prelucrare 2
....	....
UNLOCK(A)	UNLOCK(B)
UNLOCK(B)	UNLOCK(A)

Presupunem că tranzacţiile T1 şi T2 îşi încep execuţia aproximativ în acelaşi moment. T1 cere şi obţine blocarea lui A iar T2 cere şi obţine blocarea lui B. Apoi T1 cere bocarea lui B, dar este pusă în aşteptare deoarece unitatea de acces B este bocată de tranzacţia T2. Concomitent, T2

cere blocarea lui A şi este pusă în aşteptare până când T1 deblochează pe A, dar T1 aşteaptă deblocarea lui B de către T2. În consecinţă, nici una dintre tranzacţii nu-şi poate continua execuţia, ambele fiind puse în aşteptare la nesfârşit. O astfel de situaţie se numeşte interblocare.

Într-o situaţie de interblocare pot fi implicate un număr mai mare de tranzacţii care se aşteaptă reciproc. Interblocarea este o problemă comună tuturor sistemelor concurente. Pentru rezolvarea ei există două categorii de metode:

1. metode de prevenire şi evitare a interblocării

2. metode de detecţie şi ieşire din interblocare

Condiţii pentru interblocare

Într-un sistem concurent, poate apare situaţia de interblocare numai dacă sunt satisfăcute simultan următoarele 4 condiţii:

1. condiţia de excludere mutuală – tranzacţiile solicită controlul exclusiv al unităţilor de acces asupra cărora operează

2. condiţia de aşteptare pentru – o tranzacţie care deţine controlul exclusiv asupra unor unităţi de acces este în aşteptare pentru altele

3. condiţia de completare – nici o unitate de acces nu poate fi deblocată de către tranzacţia care o controlează înainte ca aceasta să termine toate operaţiile pe care le are de executat asupra unităţii respective

4. condiţia de aşteptare circulară – există un lanţ circular de tranzacţii cu proprietatea că fiecare tranzacţie deţine controlul asupra unei unităţi de acces solicitată de următoarea tranzacţie din lanţ.

Nesatisfacerea oricăreia din cele 4 condiţii de mai sus face imposibilă interblocarea, deci, negarea fiecărei condiţii ar putea sta la baza unei metode de prevenire a interblocării.

Negarea condiţiei 1 conduce la observaţia că nu poate apare interblocare în cazul tranzacţiilor care nu solicită accesul exclusiv la unităţile de acces. În cazul execuţiei concurente a unui set de tranzacţii care efectuează numai operaţii de citire nu poate apare interblocare.

Negarea condiţiei 2 conduce la o metodă de prevenire a interblocării care are la bază alocarea unităţilor de acces după criteriul „tot sau nimic", numită metoda cererilor anticipate.

Negarea condiţiei 3 conduce la o metodă de detecţie şi ieşire din interblocare care presupune abandonarea (renunţarea) unora dintre tranzacţiile aflate în interblocare la un moment dat.

Negarea condiţiei 4 conduce la o metodă de prevenire a interblocării bazată pe ordonarea unităţilor de acces, numită metoda ordonării.

Metoda cererilor anticipate

Blocarea unităţilor de acces de către tranzacţii se face după regula „tot sau nimic". Fiecare tranzacţie emite deodată, toate cererile de blocare necesare execuţiei sale complete, în mod anticipat, înainte de a executa orice operaţie de actualizare. Sistemul acceptă sau respinge aceste cereri în bloc. Nu este posibil ca o tranzacţie să obţină blocarea unei părţi a unităţilor pe care doreşte să le acceseze, pentru ca pe parcurs să emită alte cereri d blocare. Astfel, nici o tranzacţie care a obţinut blocarea unor unităţi de acces nu va putea fi pusă în aşteptare.

Dezavantaje:

• tranzacţiile blochează unele dintre unităţile de acces pe o durată mai mare decât este necesar, ceea ce reduce nivelul de concurenţă al sistemului.

• favorizează apariţia fenomenului de amânare nedefinită sau „infometare" a tranzacţiilor. Tranzacţiile care solicită accesul la un număr mai mare de unităţi de acces ar putea fi menţinute în aşteptare un timp nedefinit, deoarece este puţin probabil ca resursele solicitate să se disponibilizeze toate în acelaşi moment. Aceste tranzacţii au şanse mult mai mici de a fi lansate în execuţie, faţă de tranzacţiile care solicită mai puţine resurse.

• există situaţii când această tehnică nu este aplicabilă. Este posibil ca pentru o tranzacţie care blochează două unităţi de acces să nu se poată preciza de la început care sunt acestea.

Identificara celei de-a doua unităţi de acces poate să depindă de anumite valori din prima unitate de acces şi deci blocarea ei nu se poate face decât după ce s-a accesat prima unitate.

Metoda ordonării

Metoda ordonării constă în stabilirea unei relaţii de ordine peste mulţimea unităţilor de acces. Tranzacţiile pot bloca unităţile de acces numai în această ordine prestabilită.

Fie U1, U2, ..., Un unităţile de acces a căror ordonare este dată prin valoarea indicilor asociaţi. Presupunem că fiecare tranzacţie blochează unităţile de acces în ordinea crescătoare a indicilor. Dacă o tranzacţie Tx a blocat o unitate Ui, atunci T nu poate fi pusă în aşteptare decât pentru o unitate Uj, cu j>i. Dar o altă tranzacţie Ty care a blocat Uj nu poate fi în aşteptare pentru Ui, deoarece i<j. Deci interblocarea nu este posibilă.

Dezavantaje:

• Afectează deasemenea nivelul de concurenţă al sistemului prin blocarea mai mult decât este necesar al unor unităţi de acces. Fie, de ex, o tranzacţie care doreşte accesul pentru o durată scurtă de timp la unitatea Ui şi un timp mai lung la unitatea Uj. Dacă i<j, atunci unitatea Ui va trbui să fie blocată pe toată durata blocării lui Uj.

• impune restricţii programatorilor în elaborarea tranzacţiilor. Cererile de acces la date trebuie să respecte ordinea impusă de sistem.

• În cazul BD complexe, realizarea unei ordonări a unităţilor de acces poate fi foarte dificilă (nu imposibilă), din cauza posibilităţilor foarte variate de divizare în unităţi de acces.

• Acestă metodă presupune existenţa apriorică a unei asemenea divizări, ceea ce exclue posibilitatea blocărilor

7. MARCAREA TIMPULUI

Metodele de marcare a timpului pentru controlul concurenţei sunt destul de diferite de metodele de blocare. Nu este implicată nici o blocare, deci nu pot apare situaţii de impas (interblocare).

Marca de timp este un identificator unic creat de SGBD, care indică timpul relativ de începere a unei noi tranzacţii. Mărcile de timp pot fi generate folosind ceasul sistemului sau prin declanşarea unui contor logic.

Marcarea timpului este un protocol de control al concurenţei, în scopul ordonării globale a tranzacţiilor astfel încât, tranzacţiile mai vechi (cu mărci de timp mai mici) să aibă prioritate, în eventualitatea unui conflict.

Dacă o tranzacţie încearcă să citească sau să scrie o dată, aceste operaţii sunt permise numai dacă ultima reactualizare a respectivei date a fost efectuată de o tranzacţie mai veche. Altfel, tranzacţia care necisită operaţia de citire/scriere este reîncepută şi i se atribuie o nouă marcă de timp. Este necesar ca tranzacţiilor reîncepute să li se atribuie noi mărci de timp, pentru a evita să fie încontinuu abandonate şi reîncepute.

Regula de scriere a lui Thomas

Pentru a obţine o concurenţă cât mai mare prin respingerea operaţiilor scoase din uz, se poate utiliza o modificare a protocolului de ordonare a mărcilor de timp:

• Dacă tranzacţia T încearcă să scrie un articol x a cărui valoare a fost deja citită de către o tranzacţie mai nouă, tranzacţia T trebuie rulată înapoi şi reîncepută cu o nouă marcă de timp;

• Dacă tranzacţia T încearcă să scrie un articol x a cărui valoare a fost deja scrisă de către o tranzacţie mai nouă, operaţia de scriere poate fi ignorată, pentru că valoarea pe care tranzacţia T vrea să o scrie se bazează pe o valoare veche a articolului x, valoare scoasă deja din uz.

Să presupune că sunt trei tranzacţii concurente (vezi tabelul următor) şi că mărcile lor de timp sunt la un anumit moment, în ordinea

MT1 < MT2 < MT3

La momentul t8, operaţia de scriere a tranzacţiei T2 violează regula de scriere a mărcilor de timp şi drept urmare este abandonată şi reîncepută la momentul t14.

La momentul t14, operaţia de scriere a tranzacţiei T1 poate fi ignorată utilizând regula de ignorare a scrierilor scoase din uz, deoarece ar fi fost suprascrisă prin operaţia de scriere a tranzacţiei T3 de la momentul t12.

Timpul	Operaţia	T1	T2	T3
t1		Begin trans		
t2	Read x	Read x		
t3	x=x+10	x=x+10		
t4	Write x	Write x	Begin trans	
t5	Read x		Ready y	
t6	y=y+20		y=y+20	Begin trans
t7	Ready y			Ready y
t8	Write y		Write Y	
t9	y=y+30			y=y+30
t10	Write y			Write y
t11	z=100			z=100
t12	Write z			Write z
t13	z=50	z = 50	Begin trans	Commit
t14	Write z	Write z	Ready y	
t15	Ready y	Commit	y=y+20	
t16	y=y+20			
t17	Write y		Write y	
t18			Commit	

8. *REFACEREA BAZEI DE DATE*

Refacerea bazei de date este procesul de restaurare a bazei de date într-o stare corectă după apariţia unei pene.

Cauzele penelor:

• căderile sistemului (hard sau soft)

• pene de mediu (distrugerea mediului de depozitare a datelor)

• erorile soft de aplicaţie

• dezastre naturale

• neglijenţa

• sabotajul

Indiferent de cauză, există două efecte principale:

1. pierderea memoriei principale, inclusiv a bufferelor bazei de date

2. pierderea copiei de pe disc a bazei de date

Vom prezenta câteva concepte şi tehnici prin care se pot minimiza aceste efecte şi se poate permite refacerea în urma unei pene.

Tranzacţia reprezintă unitatea de reface de bază dintr-un sistem de baze de date.

Tehnici de refacere

Procedura utilizată pentru refacerea bazei de date depinde de gradul de deteriorare. Sunt posibile două situaţii:

• dacă baza de date a fost deteriorată fizic (căderea discului pe care este stocată baza de date), este necesar să se restaureze ultima copie de siguranţă şi să se aplice din nou operaţiile de reactualizare a tranzacţiilor efectuate, folosind fişierul jurnal1 Se recomandă ca fişierul jurnal să

1 *Amintim că Securitatea datelor cere ca baza de date să fie protejată împotriva unei distrugeri logice (anomalii de actualizare) sau fizice. Pentru aceasta, există instrumente care permit:*
− crearea unor puncte de repriză = salvarea din timp în timp a unor copii coerente ale bazei de date
− gestiunea unui jurnal de tranzacţii = lista operaţiilor realizate asupra bazei de date după ultimul punct de repriză

fie stocat pe un disc separat, pentru a reduce riscul deteriorării lui simultan cu deteriorarea bazei de date;

• dacă baza de date nu a fost deteriorată fizic, ci a devenit incoerentă datorită unei căderi a sistemului în timpul execuţiei unei tranzacţii, nu este necesară utilizarea copiei de siguranţă, ci poate fi restaurată utilizând imaginile anterioare şi ulterioare conţinute în fişierul jurnal.

Vom prezenta trei tehnici de refacere, cunoscute sub denumirile de reactualizare amânată, reactualizare imediată şi paginarea cu umbră.

a) Tehnica de refacere cu ajutorul reactualizării amânate

Prin utilizarea acestui protocol, reactualizările nu sunt scrise în baza de date decât după ce tranzacţia a ajuns pe punctul de a fi efectuată. Dacă tranzacţia eşuează înainte de a ajunge în acest punct, ea nu va fi modificat baza de date şi astfel nu mai sunt necesare anulări sau modificări. S-ar putea să fie nevoie doar să se reia reactualizările tranzacţiilor efectuate, deoarece este posibil ca efectul acestora să nu fi ajuns la baza de date. Pentru aceasta se foloseşte jurnalul de tranzacţii.

b) Tehnica de refacere cu ajutorul reactualizării imediate

Prin utilizarea acestui protocol, reactualizările sunt aplicate pe baza de date imediat ce au loc, fără a aştepta ca tranzacţia să ajungă pe punctul de a fi efectuată. Pe lângă reluarea reactualizărilor efectuate, acum ar putea să fie necesar să se anuleze efectele tranzacţiilor care nu au fost efectuate în momentul în care a survenit pana. Pentru aceasta se foloseşte jurnalul de tranzacţii.

Este esenţial ca înregistrările din jurnal să fie scrise înainte descrierea corespunzătoare în baza de date. Această operaţie e cunoscută sub denumirea de protocol de scriere în avans în jurnal.

c) Paginarea cu umbră

Este o alternativă a tehnicilor care folosesc jurnalul de tranzacţii. În cadrul acestei tehnici, în timpul unei tranzacţii se păstrează două tabele ale paginii: unul pentru pagina curentă, unul

pentru pagina din umbră. La începutul tranzacţiei, cele două pagini sunt identice. Pagina din umbră nu se modifică niciodată pe parcursul tranzacţiei, în timp ce pagina curentă reflectă toate reactualizările făcute pe baza de date de către tranzacţia respectivă. La încheierea tranzacţiei, tabelul paginii curente devine tabelul paginii din umbră.

Avantaje:

• eliminarea suprasarcinii datorată întreţinerii fişierului jurnal

• refacerea mai rapidă (pentru că nu sunt necesare operaţii de anulare sau reluare)

Dezavantaje:

• fragmentarea datelor

• necesitatea unei colectări periodice a „gunoaielor”.

Referinţe bibliografice:

[1]Ionescu F., Baze de date relationale si aplicatii , Editura Tehnica, Bucuresti, 2004

[2] Ionescu F., curs Baze de date pentru aplicatii stiintifice (BDAS)

[3] Ionescu F. , curs Proiectarea bazelor de date (PBD)

[4] Navroschi-Szász A., curs: Blocări, mărci de timp şi utilizarea lor in Oracle

[6] http://vega.unitbv.ro/~cataron/Courses/BD/bd.htm Curs Baze de date, Lector Angel Caţaron, Universitatea Transilvania, Brasov 2012

5] http://blogu.lu/kassak/gestiunea-tranzacţiilor/

[7] http://www.cs.cmu.edu/~mihaib/articles/tranzactii/tranzactii-html.html