Baze de date

Universitatea “Transilvania” din Brasov

Lect.dr. Costel Aldea

costel.aldea@gmail.com

Acces concurent date

- Un obiectiv major al SGBD este de a permite mai multor utilizatori să acceseze concurent datele partajate
- La sistemele în care o BD este accesată simultan de mai mulţi utilizatori apar situaţii de conflict
  - Acestea se datorează accesului concurent la datele care constituie resursă comună
- Modul de rezolvare a conflictului depinde de natura cererilor de acces la date
  - Regăsire
  - Actualizare

Cereri regasire date

- Cereri de acces de tip regăsire
  - Secvenţialitatea accesului la mediul de memorare este suficientă
    - Nu mai este nevoie de precauţii suplimentare
  - Atât timp cât niciunul nu face modificări ale datelor, nu are importanţă ordinea în care se asigură accesul utilizatorului la date
  - În acest caz, rezolvarea situaţiilor conflictuale cauzate de operaţiile de citire simultană poate fi lăsată în seama sistemului de operare
    - Stabileşte ordinea de satisfacere a cererilor după criteriul asigurării concurenţei maxime de operare
    - Minimizează timpul global de satisfacere a acestor cereri

Cereri actualizare date

- Cereri de tip **actualizare**
  - Este necesară aplicarea unor **strategii** adecvate de tratare a cererilor de acces
  - Exemplu

    Sistemele de rezervare a locurilor în care mai mulţi agenţi fac permanent modificări
    - Pot exista două procese, care citesc şi modifică valoarea aceluiaşi obiect şi care 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

Algoritmi control concurenta

- Cereri de tip **actualizare** (continuare)
  - O rezolvare imediată şi simplă ar fi blocarea BD 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:
  - algoritmi de control prin **blocare**
  - algoritmi de control prin **marcare**

Tranzacţii

- 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 BD
- Pot exista mai multe execuţii independente ale aceluiaşi program şi fiecare dintre acestea este o tranzacţie
- O **tranzacţie este o unitate singulară de execuţie** care, din punctul de vedere al utilizatorului, satisface două condiţii referitoare la BD:
  - BD este într-o stare **coerentă** *înaintea* şi *după* execuţia tranzacţiei
  - BD poate fi într-o stare **incoerentă** *în timpul execuţiei* unei tranzacţii
- O tranzacţie este o unitate logică de prelucrare indivizibilă (atomică) a datelor unei baze de date prin care se asigură consistenţa acesteia

Tranzacţii

- O tranzacţie poate avea **două rezultate**:
  - dacă se încheie cu succes, se spune că tranzacţia a fost **efectuată** iar BD ajunge într-o nouă stare coerentă
  - 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

Tranzacţii

- Pentru a **delimita tranzacţiile**, în majoritatea limbajelor de manipulare a datelor sunt disponibile instrucţiunile
  - **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
- Dacă utilizatorul nu foloseşte aceste delimitări, de obicei **întregul program** este considerat ca reprezentând **o singură tranzacţie**

Tranzacţii

- SGBD execută automat
  - o instrucţiune **COMMIT** atunci când programul este încheiat corect
  - o instrucţiune **ROLLBACK** în caz contrar
- Odată folosită instrucţiunea **BEGIN TRANSACTION,** orice actualizare nu va fi executată instantaneu, ci numai după **COMMIT** sau **ROLLBACK** pentru a încheia tranzacţia

Proprietăţile tranzacţiilor - ACID

1. caracterul **atomic**: reprezintă proprietatea „**tot sau nimic**”
   - O tranzacţie este o unitate *indivizibilă*, care ori este efectuată în întregime, ori nu este efectuată deloc
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

Unităţi de acces

- 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**

Unităţi de acces

- **Natura şi dimensiunea unităţilor de acces** este stabilită de proiectantul sistemului
  - Exemplu

    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
  - Înseamnă reducerea timpului consumat de sistem pentru gestiunea acestor operaţii
  - Înseamnă reducerea spaţiului de memorie necesar înregistrării blocajelor

Unităţi de acces

- 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**
  - 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 tuplul ca unitate de acces

Anomalii de 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**

Anomalii de interferenţă

- Deci două tranzacţii **Ti** şi **Tj** sunt **conflictuale** dacă sunt
  - **concurente**
  - **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 ***citire fantomă***

Anomalia de actualizare pierdută

- Corespunde unui conflict de tip scriere-scriere
- 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 influenţată de rezultatul primei actualizări

Anomalia de actualizare pierdută

- Exemplu
  - Considerăm o execuţie concurentă a două tranzacţii T1 şi T2
  - Notăm 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

  - Valoarea lui A este 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

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

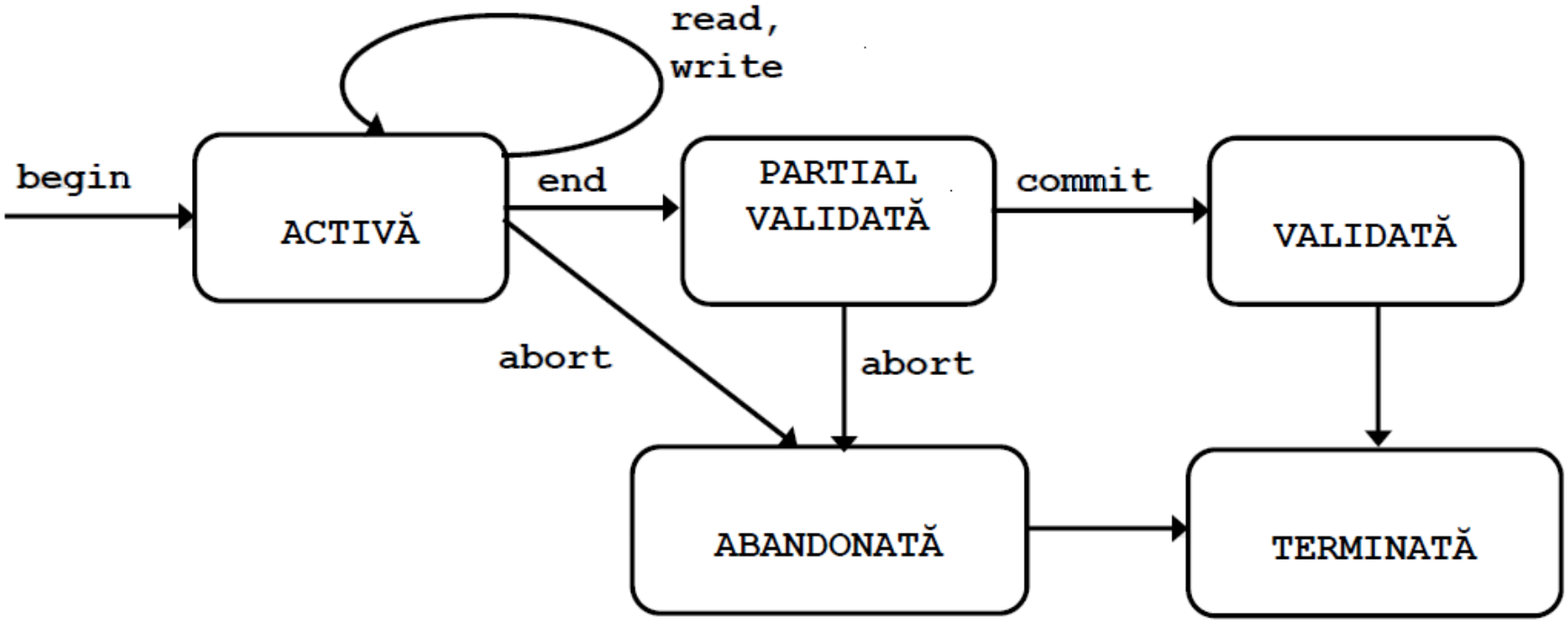
Anomalia de citire irepetabilă

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

T1	T2
Read A	
B = A	
Write A	
	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

Starile tranzactiilor



Primitivele LOCK şi UNLOCK

- Fie tranzacţiile **T1** şi **T2** execuţii diferite ale următoarei secvenţe de operaţii, unde **A** este o valoare existentă în bază:

```
Read A
A = A + 1
Write A
```

- 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

Primitivele LOCK şi UNLOCK

- **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

Primitivele LOCK şi UNLOCK

- 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**
  - Interzice 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

Primitivele LOCK şi UNLOCK

- 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

- 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

Primitivele LOCK şi UNLOCK

- 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**

Primitivele LOCK şi UNLOCK

- 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

Primitivele LOCK şi UNLOCK

- Rezultatul este corect, dar execuţia este **pur secvenţială**, nu este posibil nici un fel de paralelism între cele două tranzacţii
- Analizând timpul de execuţie, această situaţie este inacceptabilă
  - 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

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 mai 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

Interblocarea

- 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 1`
`...`	`...`
`UNLOCK(A)`	`UNLOCK(B)`
`UNLOCK(B)`	`UNLOCK(A)`

Interblocarea

- 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**

Interblocarea

- 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:
  - metode de **prevenire şi evitare** a interblocării
  - metode de **detecţie şi ieşire** din interblocare

Condiţii de aparitie a interblocarii

- Într-un sistem concurent, poate apărea situaţia de interblocare numai dacă sunt satisfăcute simultan următoarele 4 condiţii:
  - condiţia de excludere mutuală
    - tranzacţiile solicită controlul exclusiv al unităţilor de acces asupra cărora operează
  - condiţia de aşteptare
    - o tranzacţie care deţine controlul exclusiv asupra unor unităţi de acces este în aşteptare pentru altele

Condiţii de aparitie a interblocarii

- condiţia de completare
  - nicio 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
- 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

Interblocarea

- Nesatisfacerea oricăreia din cele 4 condiţii de mai sus face imposibilă interblocarea
  - 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

Interblocarea

- 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 totul 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 de blocare
- Astfel, nicio tranzacţie care a obţinut blocarea unor unităţi de acces nu va putea fi pusă în aşteptare

Metoda cererilor anticipate

- 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 - 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

Metoda cererilor anticipate

- Dezavantaje (continuare)
  - 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

Metoda ordonării

- Dezavantaje
  - Afectează deasemenea nivelul de concurenţă al sistemului prin blocarea mai mult decât este necesar al unor unităţi de acces
    - Exemplu
      - Fie 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 trebui 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

Metoda ordonării

- Dezavantaje (continuare)
  - 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

Marcarea timpului

- Metodele de marcare a timpului pentru controlul concurenţei sunt destul de diferite de metodele de blocare
  - Nu este implicată nicio blocare, deci nu pot apărea 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

- 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
  - Tranzacţia care necesită 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

Marcarea timpului - 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

Marcarea timpului - regula de scriere a lui Thomas

Timpul	Operația	T1	T2	T3
t_1		Begin_trans		
t_2	Read x	Read x		
t_3	x=x+10	x=x+10		
t_4	Write x	Write x	Begin_trans	
t_5	Read y		Read y	
t_6	y=y+20		y=y+20	Begin_trans
t_7	Read y			Read y
t_8	Write y		Write y	
T_9	y=y+30			y=y+30
t_{10}	Write y			Write y
t_{11}	z=100			z=100
t_{12}	Write z			Write z
t_{13}	z=50	z=50		Commit
t_{14}	Write z	Write z	Begin_trans	
t_{15}	Read y	Commit	Read y	
t_{16}	y=y+20		y=y+20	
t_{17}	Write y		Write y	
t_{18}			Commit	

Marcarea timpului - regula de scriere a lui Thomas

- Să presupunem că sunt trei tranzacţii concurente şi că mărcile lor de timp sunt la un anumit moment, în ordinea

$$MT1 < MT2 < MT3$$

- La momentul t_8, operaţia de scriere a tranzacţiei T2 violează regula de scriere a mărcilor de timp şi drept urmare este abandonată şi reîncepută la momentul t_{14}
- La momentul t_{14}, 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 t_{12}

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

Refacerea bazei de date

- Indiferent de cauză, există două efecte principale:
  - pierderea memoriei principale, inclusiv a bufferelor bazei de date
  - 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 refacere 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 jurnal
    - Se recomandă ca fişierul jurnal să fie stocat pe un disc separat, pentru a reduce riscul deteriorării lui simultan cu deteriorarea bazei de date

Tehnici de refacere

- 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

Tehnici de refacere

- 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

Tehnici de refacere

- 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

Tehnici de refacere

- Tehnica de refacere cu ajutorul reactualizării imediate (continuare)
  - Este esenţial ca înregistrările din jurnal să fie scrise înainte de scrierea corespunzătoare în baza de date
  - Această operaţie e cunoscută sub denumirea de protocol de scriere în avans în jurnal

Tehnici de refacere

- 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

- Paginarea cu umbră (continuare)
  - 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”

Exemplu tranzactie – bd. zboruri

- Se defineste o tranzactie pentru rezervarea unui loc la o cursa aeriana in procedura stocata sp_rezervari()
- Daca tabelul CURSE conține linia

(1, 'Bucuresti', 'Paris', 2008-12-30,1)

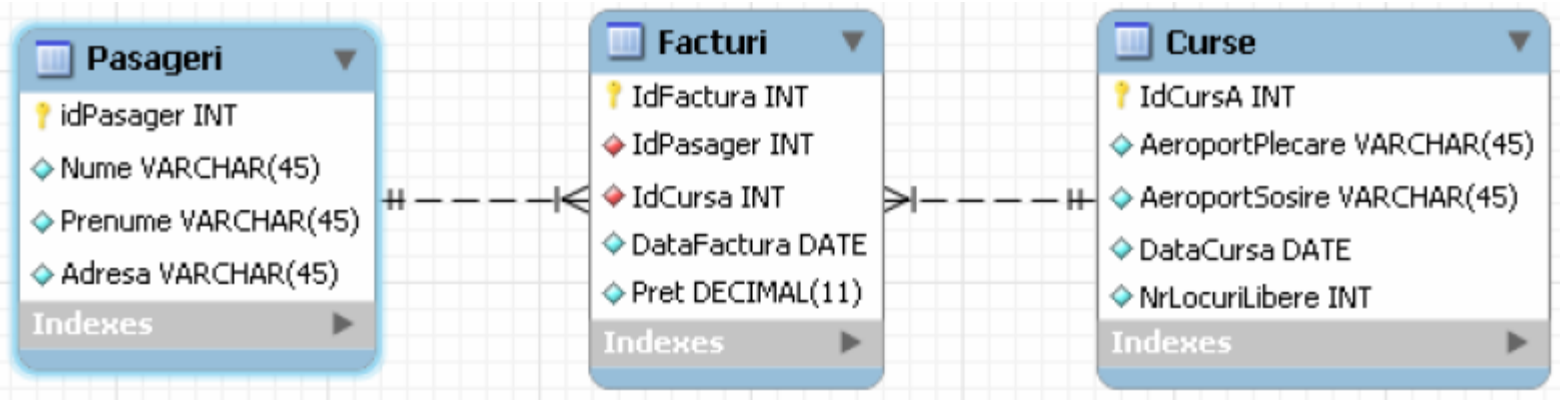
si se apeleaza:

sp_rezervari(@rez, 100, 'Ionescu', 'Ion', 'Craiova', 'Bucuresti', 'Paris', '2009-12-30', 500)

atunci:

- Se obtine @rez=1 (executie corecta)
- NrLocuriLibere in tabelul CURSE este decrementat cu 1 (mai sunt 0 locuri)
- Tabelul PASAGERI va contine si linia (100, 'Ionescu', 'Ion', 'Craiova')
- Tabelul FACTURI va contine si linia: (1, 100,1,'2008-12-28',500)

- Daca se incearca inca o rezervare, se obtine @rez=0 si nicio modificare in tabele (s-a executat rollback)



Exemplu tranzacție – bd. zboruri – procedura stocata

```
DELIMITER $$DROP PROCEDURE IF EXISTS 'zboruri'.'sp_rezervari' $$
CREATE PROCEDURE 'zboruri'.'sp_rezervari'(OUT rezultat INT, s_pasager INT, s_nume varchar(20),
    s_prenume varchar(20), s_adresa varchar(20), plecare varchar(20), sosire varchar(20), s_data date,
    s_pret decimal)
BEGIN DECLARE l_cursa, l_locuri INT;
    SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
    SET autocommit = 0;
    START TRANSACTION;
    INSERT INTO PASAGERI values(s_pasager, s_nume, s_prenume, s_adresa);
    SELECT IdCursa, NrLocuriLibere INTO l_cursa, l_locuri FROM CURSE
    WHERE AeroportPlecare=plecare AND AeroportSosire=sosire AND DataCursa = s_data;
    IF l_locuri > 0 THEN BEGIN
        UPDATE CURSE SET NrLocuriLibere = l_locuri -1;
        INSERT INTO FACTURI(IdPasager, IdCursa, DataFactura, Pret)
        values(s_pasager, l_cursa, CURDATE(), s_pret);
        COMMIT;
        SET rezultat = 1;END;
    ELSE BEGIN ROLLBACK;SET rezultat = 0;END; END IF;
END $$
DELIMITER ;
```