Capitolul 5: Gestiunea tranzactiilor

- Tranzactii
- Anomalii de acces concurent la bazele de date
  - Actualizare pierduta
  - Citire improprie
  - Citire irepetabila
  - Citire fantoma
- Proprietatile tranzactiilor
- Operatiile efectuate de tranzactii
- Starile tranzactiilor
- Planificarea tranzactiilor
- Tehnici de control al concurentei
  - Controlul concurentei prin blocare
  - Controlul concurentei prin marci de timp
- Tehnici de refacere a bazelor de date

Tranzactii

- În mod obişnuit, un sistem SGBD deserveşte mai mulți utilizatori, care accesează concurent datele din tabele
- Execuția concurentă a mai multor procese (coresp. utilizatorilor) poate avea loc:
  - într-un sistem uniprocesor, prin partajarea (împărțirea) timpului de execuție al procesorului între mai multe procese (multiprogramare)
  - într-un sistem multiprocesor, în care mai multe procese pot fi executate în mod real simultan, pe mai multe procesoare ale sistemului (multiprocesare)
- *O tranzacție* (*transaction*) *este o unitate logică de prelucrare indivizibilă* (*atomică*) *a datelor unei baze de date prin care se asigură consistența acesteia*
  - Asigurarea consistenței înseamnă modificări corecte ale datelor; de exemplu, dacă se transferă o sumă de bani: cu cât scade valoarea din contul din care s-au retras banii, cu aceeași sumă trebuie să crească valoarea în contul în care s-au transferat
- O tranzacție trebuie să asigure consistența bazei de date in orice situatie:
  - tranzactia se execută individual sau concurent cu alte tranzacții
  - apar defecte (necatastrofice) ale sistemului în cursul execuției tranzacției
- O *tranzacție* este o operație indivizibilă de acces la baza de date care:
  - fie se execută cu succes toate acțiunile şi se termină cu o validare a modificărilor efectuate asupra bazei de date (*commit*)
  - fie nu poate efectua (din diferite motive) toate acțiunile şi este abandonată şi anulată (se anulează toate modificarile efectuate pana la abandon) (*abort*, *rollback*)

Exemplu de tranzactie

- Exemplu: un sistem de rezervare a locurilor la curse aeriene
  - PASAGERI (IdPasager, Nume, Prenume, Adresa)
  - CURSE (IdCursa, AeroportPlecare, AeroportSosire, DataCursa, NrLocuriLibere)
  - FACTURI (IdFactura, *IdPasager*, *IdCursa*, DataFactura, Pret)
- Pentru rezervarea unui loc se efectuează mai multe operații:
  - 1. Se verifică dacă există locuri libere la cursa dorită; dacă nu există locuri libere, tranzacția se oprește; dacă există locuri libere, se continuă rezervarea astfel:
  - 2. Se inserează o linie nouă în tabelul PASAGERI, cu datele pasagerului
  - 3. Se scade numarul de locuri libere in tabelul CURSE
  - 4. Se insereaza o linie noua în tabelul FACTURI
  - 5. Se emite factura si se tipărește biletul
- Probleme care pot sa apară:
  - Dacă sistemul se defectează după ce s-a executat pasul 3, s-a făcut o rezervare, dar locul nu a fost atribuit niciunui pasager și costul nu a fost facturat; datele din baza de date nu mai sunt consistente: locurile atribuite nu corespund locurilor rezervate, iar sumele facturate nu corespund rezervărilor făcute
  - Dacă nu se defectează sistemul, dar doi sau mai multi agenți atribuie ultimul loc la doi sau mai multi pasageri diferiți, atunci vor fi probleme la îmbarcarea pasagerilor
- Astfel de probleme dispar dacă toate actiunile efectuate pentru o rezervare sunt grupate ca o operație indivizibilă (atomică), care ori se execută complet, ori nu se execută deloc

Anomalii de acces concurent la bazele de date

- Unitatea de transfer a datelor între discul magnetic şi memoria principală a sistemului de calcul este ***blocul***, care corespunde unui sector de pe disc, in care se memoreaza una sau mai multe inregistrari (tupluri) și care se scrie/citește într-un/dintr-un buffer din memorie
- Un articol (data item) al bazei de date reprezintă valoarea unui atribut dintr-o înregistrare (tuplu), sau o înregistrare întreagă sau chiar un grup de inregistrări
- Operațiile de acces la un articol X al bazei de date pot fi:
  - read(X): citeşte articolul X din baza de date într-o variabilă a programului; pentru simplificarea notațiilor se va considera că variabila în care se citeşte articolul X este notată, de asemenea, cu X.
  - write(X): scrie variabila de program X în articolul X al bazei de date.
- Tranzacțiile lansate de diferiți utilizatori se pot executa concurent şi este posibil să actualizeze aceleaşi articole ale bazei de date
- Dacă execuția concurentă a tranzacțiilor este necontrolată, este posibil ca baza de date să ajungă într-o stare inconsistentă (incorectă), chiar dacă:
  - fiecare tranzacție în parte a fost executată corect
  - nu au apărut defecte de funcționare ale sistemului

Anomalii de acces concurent necontrolat la bazele de date

- ***(a) Executie corectă: X = 100 + 20 + 10 = 130***
- ***(b) Actualizare pierduta*** (*lost update*): rezulta X = 100 + 20 = 120
- ***(c) Citire improprie*** (*dirty read*)(c): T2 citeste X=120, desi ac. val nu a fost validată

Timp

T1	BD X = 100	T2
read(X): X=100		
X = X + 20 = 120		
write(X)	X = 120	
		read(X): X = 120
		X = X + 10 = 130
	X = 130	write(X)

(a) Executie corecta

T1	BD X = 100	T2
read(X): X=100		
X = X + 20 = 120		
		read(X): X = 100
		X = X + 10 = 110
	X = 110	write(X)
write(X)	**X = 120**	

(b) Actualizare pierduta

T1	BD X = 100	T2
read(X): X=100		
X = X + 20 = 120		
write(X)	X = 120	
		read(X): X = 120
		X = X + 10 = 130
	X = 130	write(X)
.....		
abort		

(c) Citire improprie

- ***Citire irepetabilă*** (*nonrepetable read*): o tranzacție citeşte un articol de două ori, iar între cele două citiri, o altă tranzacție a modificat chiar acel articol
- ***Citire fantomă*** (*phantom read*): o tranzacție prelucrează un set de linii rezultat al unei interogări si în timpul acesta o altă tranzacție insereaza sau sterge o linie

Proprietățile tranzactiilor (ACID)

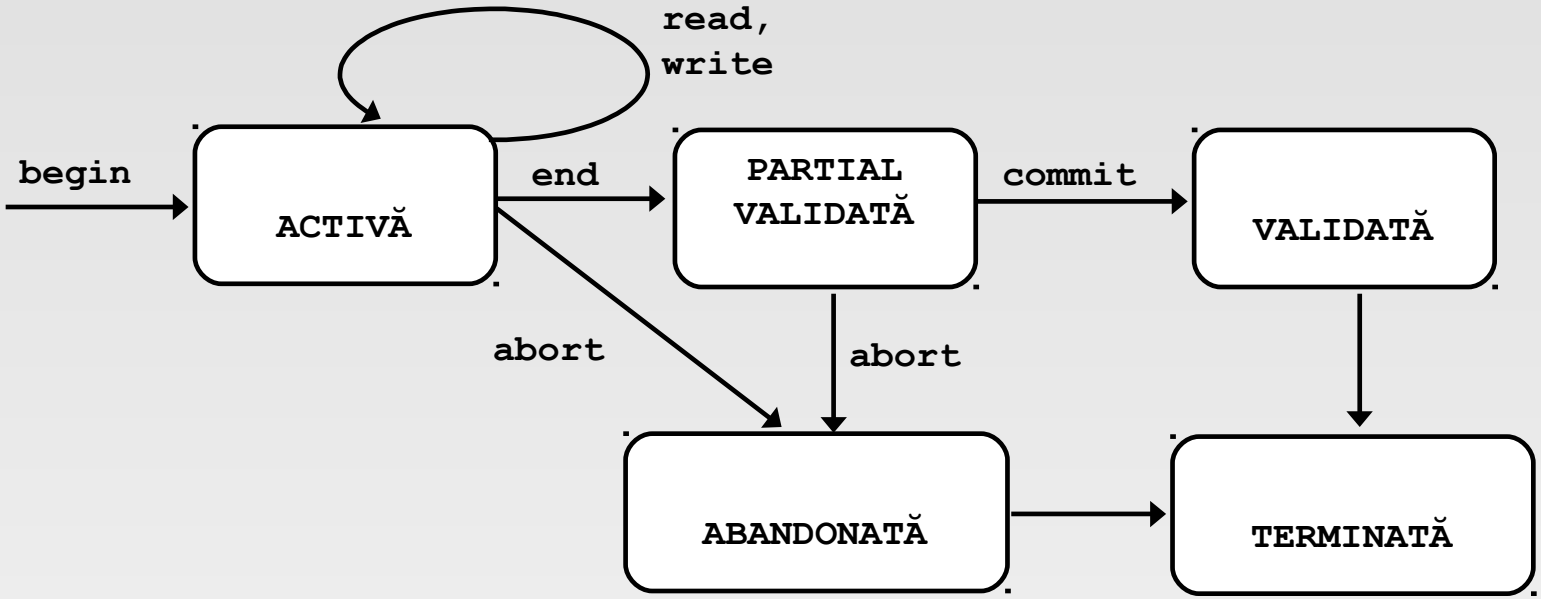
- Pentru ca tranzacțiile să se efectueze corect, și să nu apară anomalii, ele trebuie să *aibă următoarele proprietăți (ACID):*
- ***A**tomicitatea* (*atomicity*): *proprietatea unei tranzacții de a reprezenta o unitate de execuție indivizibilă (neîntreruptă)*
  - Dacă o tranzacție este întreruptă de alta, pot apare anomalii de actualizare pierdută (fig b)
- ***C**onsistența* (*consistency*): *proprietatea unei tranzacții de a se executa fie complet, fie deloc* ("totul sau nimic"), asigurând consistența bazei de date (o bază de date este consistentă dacă toate valorile memorate sunt corecte)
  - *Execuția parțială a unei tranzacții produce inconsistență (exemplele cu transferul de bani sau cu defectarea sistemului după pasul 3 din operația de rezervare a biletelor de avion)*
- ***I**zolarea* (*isolation*): *proprietatea unei tranzacții de a face vizibile modificările efectuate numai după ce a fost validată* (*committed*)
  - Dacă rezultatele parțiale ale unei tranzacții sunt vizibile altor tranzacții înainte de validarea acesteia (commit) şi dacă se întâmplă ca această tranzacție să fie abandonată şi anulată (*rollback*), atunci toate tranzacțiile care au accesat rezultatele parțiale ale acesteia vor trebui să fie anulate; aceste operații de anulare pot produce, la rândul lor alte anulări, ş.a.m.d. (anulare în cascadă)
  - Dacă tranzacțiile nu sunt izolate, pot apare anomalii de citire improprie (fig c), citire nerepetabilă sau citire fantomă
- ***D**urabilitarea* (*durability*): proprietatea prin care, după validarea unei tranzacții, modificările efectuate de aceasta în baza de date nu vor mai fi pierdute datorită unor defectări ulterioare necatastrofice a sistemului; această proprietate este asigurată de administatorul de refacere (recovery manager) (se va explica ulterior)

Operatiile efectuate de tranzactii

- Operațiile efectuate de tranzacții sunt înregistrate de administratorul de refacere (*recovery manager*) într-un fișier separat de baza de date (fișier jurnal - log file) care este folosit pentru refacerea datelor în caz de defectări ale sistemului
- Aceste operații sunt:
  - **begin**: începutul execuției unei tranzacții; trece tranzacția în stare **`ACTIVA`**
  - **read** sau **write**: operații de citire sau scriere a articolelor din baza de date
  - **end**: marchează terminarea operațiilor de scriere sau citire din baza de date, ceea ce înseamnă că tranzacția se poate termina și este trecută în starea **`PARTIAL VALIDATA`**; totuşi, este posibil să fie necesare unele operații de verificare înainte de validarea (commit) tranzacției.
  - **commit**: terminarea cu succes a tranzacției, validarea tuturor modificărilor, înscrierea lor în baza de date pe harddisk şi vizibilitatea modificărilor efectuate pentru alte tranzacții; din acest moment, modificările efectuate nu mai pot fi anulate, nici pierdute printr-o defectare ulterioară necatastrofică a sistemului (care nu distruge harddisk-ul)
  - **rollback** (sau **abort**): semnifica faptul că tranzacția a fost abandonată şi că orice efect pe care tranzacția l-a avut asupra bazei de date trebuie să fie anulat (printr-o “rulare înapoi” a operațiilor).
  - **undo**: operație similară operației rollback, dar se aplică unei singure operații, nu unei întregi tranzacții.
  - **redo**: specifică faptul că unele operații ale unei tranzacții trebuie să fie executate din nou pentru a se putea valida întreaga tranzacție.
- Ultimele două operații sunt necesare numai în anumite tehnici de refacere

Starile tranzactiilor

- Diagrama de stare a unei tranzactii:



- Pentru refacerea bazei de date, sistemul SGBD menține un *fişier jurnal (log file)*, în care memorează operațiile efectuate de fiecare tranzacție, *identificată printr-un identificator unic* (T) *generat de sistem*
  - Fişierul jurnal este memorat pe disc şi nu este afectat de erori de execuție a tranzactiilor (dar poate fi afectat de defectarea catastrofica a discului)
  - Fişierul jurnal este salvat periodic pe un suport auxiliar (bandă magnetică) (mai frecvent decât este salvată baza de date în intregime)

Planificarea tranzactiilor

- Execuția neîntreruptă (atomică) a fiecărei tranzacții asigură o funcționare corectă, dar nu permite concurența și performanțele bazei de date sunt slabe
  - Dacă o tranzacție durează foarte mult, toate celelalte tranzacții trebuie să aștepte
- Pentru creșterea perf. se folosesc *planificări ale tranzacțiilor*, care permit execuția concurentă a tranzacțiilor, fără să afecteze corectitudinea rezultatelor
- ***O planificare*** (*schedule*, sau istorie - *history*) *S* pentru *n* tranzacții T1,T2,...Tn este o ordonare și întrețesere a operațiilor tranzacțiilor astfel încât:
  - Pentru orice tranzacție Ti care participă în S, operațiile lui Ti în S respectă ordinea inițială a operațiilor din Ti
  - Alte operații (ale altor tranzacții Tj, j ≠ i) pot fi întrețesute cu operații ale tranzacției Ti
- Două operații dintr-o planificare sunt *conflictuale* (*conflicting operations*) dacă aparțin unor tranzacții diferite, accesează acelaşi articol al bazei de date şi cel puțin una dintre operații este operație de scriere
- ***Planificări seriale*** *(serial schedules)*: o planificare S se numeşte serială dacă pentru orice tranzacție T participantă în planificare, toate operațiile din T se execută consecutiv în S; altfel, planificarea se numeşte ***neserială***
- Pentru *n* tranzactii pot exista n! planificari seriale
- Orice planificare serială a unor tranzactii corecte este corectă, dar nu permite întrețeserea operațiilor şi concurența tranzacțiilor și perf obținute sunt slabe
- De aceea, în bazele de date se folosesc planificări ne-seriale, dar serializabile, care admit concurența (deci permit obținerea de performanțe ridicate), asigurând în aceelaşi timp consistența bazei de date

Planificari seriale ale tranzactiilor

- Planificarile seriale posibile ale tranzactiilor T1 si T2 sunt SA si SB:

T1	T2
read(X) – r1(X)	
X=X-N	
write(X) – w1(X)	
read(Y) – r1(Y)	
Y=Y+N	
write(Y) – w1(Y)	
	read(X) – r2(X)
	X=X+M
	write(X) – w2(X)

SA

Rez: X=X-N+M

T1	T2
	read(X) – r2(X)
	X=X+M
	write(X) – w2(X)
read(X) – r1(X)	
X=X-N	
write(X) – w1(X)	
read(Y) – r1(Y)	
Y=Y+N	
write(Y) – w1(Y)	

SB

Rez: X=X-N+M

- Notam operațiile cu baza de date: begin, read, write, commit, abort cu b, r, w, c, a, cu indice nr tranzacției şi ca param articolul citit sau scris (op. begin și end nu sunt înscrise în tabel, dar se subînțeleg la inceputul și sfârșitul fiecărei tranzacții; abort va fi înscris, dacă apare):
  SA: b1; r1(X); w1(X); r1(Y); w1(Y); c1; b2; r2(X); w2(X); c2;
  SB: b2; r2(X); w2(X); c2; b1; r1(X); w1(X); r1(Y); w1(Y); c1;
- Perechile de op. conflictuale sunt: in SA: ((r1(X), w2(X)), (w1(X), r2(X)), (w1(X), w2(X)); in SB: (r2(X), w1(X)), (w2(X), r1(X)), (w2(X), w1(X))

Planificari serializabile ale tranzactiilor

- *O planificare neserială a n tranzacții se numşte serializabilă dacă este echivalentă cu o planificare serială a celor n tranzacții*
- Două planificări sunt echiv. (d.p.v. al conflictelor) dacă operațiile din oricare pereche de operatii conflictuale se execută în aceeaşi ordine în cele două planificări
- Planificarile SC si SD sunt echivalente cu SA, deci sunt serializabile, cu rez. corect
  - Op. din per: (r1(X),w2(x)),(w1(X),r2(X)),(w1(X),w2(X)) in SC și SD sunt în ac ord ca în SA

T1	T2
read(X) – r1(X)	
X=X-N	
write(X) – w1(X)	
	read(X) – r2(X)
	X=X+M
	write(X)- w2(X)
read(Y) – r1(Y)	
Y=Y+N	
write(Y) – w1(Y)	

SC

Rez: X=X-N+M

T1	T2
read(X) – r1(X)	
X=X-N	
write(X) – w1(X)	
read(Y) – r1(Y)	
	read(X) – r2(X)
	X=X+M
Y=Y+N	
write(Y) – w1(Y)	
	write(X) –w2(X)

SD

Rez: X=X-N+M

Planificari neserializabile ale tranzactiilor

- Planific. SE si SF nu sunt sunt echiv. nici cu SA nici cu SB, deci sunt neserializ.
  - Operațiile (r1(X), w2(X)) au ord din SA, iar (r2(X), w1(X)) au ord. din SB; rez. incorect

T1	T2
read(X) – r1(X)	
X=X-N	
	read(X) – r2(X)
write(X) – w1(X)	
	X=X+M
	write(X)- w2(X)
read(Y) – r1(Y)	
Y=Y+N	
write(Y) – w1(Y)	

Rez: X=X+M

SE

T1	T2
read(X) – r1(X)	
X=X-N	
	read(X) – r2(X)
	X=X+M
write(X) – w1(X)	
read(Y) – r1(Y)	
Y=Y+N	
write(Y) – w1(Y)	
	write(X) – w2(X)

SF

Rez: X=X+M

- Testarea echivalenței unei planificări cu fiecare planificare serială posibilă prin testarea ordinii operatiilor din toate perechile de op conflict. este foarte costisitoare
- **DAR** s-a dem. ca se poate asigura echivalența prin tehnici de control al conc. tranzacțiilor, care interzic execuția în ordine incorectă a op. din perechi conflictuale
  - De exemplu: în SE sau SF se interzice execuția r2(X) înainte de w1(X)

Tehnici de control al concurentei tranzactiilor

- Pentru a elimina anomaliile de execuție concurentă a tranzacțiilor și a asigura consistența datelor, este necesar ca planificările tranzacțiilor să fie serializabile și aceasta se poate realiza prin controlul execuției concurente a tranzacțiilor
- Cele mai utilizate tehnici (protocoale) de control al execuției concurente sunt:
  - Tehnici bazate pe blocarea accesului la date prin zăvoare (*locks*)
  - Tehnici bazate pe mărci de timp (*timestamps*)
- Tehnicile de control al concurenței sunt implementate de SGBD-uri:
  - Programatorii de aplicații nu operează explicit cu zăvoare sau mărci de timp
  - Ei stabilesc opțiunile prin care sistemul SGBD execută operatiile de control
- **Un zăvor (*lock*)** asociat cu un articol X al bazei de date este o variabilă L(X) care descrie starea acelui articol în raport cu operațiile care i se pot aplica
- Tipuri de zavoare utilizate in SGBD-uri:
  - zăvoare binare, zăvoare cu stări multiple
- *Un zăvor binar* (*binary lock*) L(X) poate avea două stări:
  - L(X) = 1 - liber (sau neblocat - *free*, *unlocked*) – se poate accesa articolul X
  - L(X) = 0 - ocupat (sau blocat - *busy*, *locked*) – nu se poate accesa articolul X
- Asupra unui zăvor binar L(X) se pot executa două operații:
  - operația de blocare, lock(X) – trece zăvorul L(X) în starea blocat (ocupat)
  - operația de eliberare, unlock(X) – trece zăvorul L(X) în starea neblocat (liber)

Zăvoare binare

- Execuția operației lock(X) de către o tranzacție:
  - Dacă zăvorul articolului X este liber (L(X)=1), atunci tranzacția achiziționează zăvorul, pe care îl și blochează (îl trece în starea ocupat); apoi:
    - Execută operațiile necesare asupra articolului X
    - Execută operația unlock(X) - prin care eliberează zăvorul, trecându-l in starea liber
  - Dacă zăvorul articolului X este ocupat (L(X)=0), atunci tranzacția respectivă (procesul) este pus în așteptare până când zăvorul este eliberat (de o altă tranzacție, care şi-a terminat operațiile de acces la acel articol și a eliberat zăvorul); când zăvorul este eliberat, tranzacția îl achiziționează (îl blochează, trecându-l în starea ocupat), după care:
    - Execută operațiile necesare asupra articolului X
    - Execută operația unlock(X) - prin care eliberează zăvorul trecându-l in starea liber
- Operația de blocare se execută ca operație indivizibilă (folosind instrucțiuni speciale ale procesoarelor de tip TestAndSet)
- Regulile care trebuie să fie respectate de orice tranzacție care fol. un zăvor binar:
  1. Tranzacția trebuie să blocheze zăvorul articolului X (prin operatia lock(X)), înainte de a efectua orice operație de citire sau de scriere a articolului X
  2. Tranzacția trebuie să elibereze zăvorul unui articol X (prin operația unlock(X)) după ce a efectuat toate operațiile de citire sau de scriere a articolului X
  3. Tranzacția nu poate achizitiona un zăvor pe care îl deține deja
  4. Tranzacție nu poate elibera un zăvor pe care nu îl deține

Zăvoare cu stări multiple (1)

- Tehnica zăvoarelor binare este prea restrictivă şi uneori limitează nejustificat execuția concurentă a tranzacțiilor
  - De exemplu, mai multe tranzacții pot efectua operații de citire în mod concurent asupra aceluiași articol, fără ca acest lucru să afecteze consistența bazei de date, dar acest mod de funcționare este interzis în tehnica zăvoarelor binare
- De aceea, multe sisteme de gestiune utilizează zăvoare cu stări multiple
- ***Un zăvor cu stări multiple*** (*multiple-mode lock*) M(X) poate fi într-una din stările:
  - ***liber*** (neblocat, *unlocked*): zăvorul nu este deținut de nici o tranzacție şi prima tranzacție care lansează o operație de blocare îl poate obține
  - ***blocat pentru citire*** (sau blocat partajat,*read-locked*): oricâte tranzacții pot deține zăvorul respectiv şi pot efectua operații de citire a articolului X, dar nici o tranzacție nu poate scrie în acest articol
  - ***blocat pentru scriere*** (sau blocat exclusiv, *write-locked*): o singură tranzacție poate deține zăvorul şi poate citi sau scrie în articolul X, nici o altă tranzacție neputând accesa articolul respectiv, nici pentru scriere nici pentru citire
- Operatiile cu zăvoarele cu stări multiple:
  - read_lock(X) - blocarea pentru citire (partajată) a zăvorului M(X)
  - write_lock(X) - blocarea pentru scriere (exclusivă) a zăvorului M(X)
  - unlock(X) - deblocarea zăvorului M(X)

Zăvoare cu stări multiple (2)

- Orice tranzacție care utilizează un zăvor cu stări multiple M(X) trebuie să respecte următoarele reguli:
  1. O tranzacție trebuie să execute o operație de blocare partajată sau exclusivă a zăvorului articolului X (read_lock(X) sau write_lock(X)) înainte de a efectua orice operație de citire a articolului X
  2. O tranzacție trebuie să execute o operație de blocare exclusivă a zăvorului articolului X (write_lock(X)) înainte de a efectua orice operație de scriere a lui X
  3. O tranzacție trebuie să elibereze zăvorul unui articol X (unlock(X)) după ce a efectuat toate operațiile de citire sau de scriere a articolului X
  4. Operația de eliberare a unui zăvor poate fi executată numai de o tranzacție care deține (în mod exclusiv sau partajat) acel zavor

Blocarea folosind zăvoare binare

- Fie tranzactiile T3,T4 care operează asupra articolelor X și Y, cele două planificări seriale (a1) și (a2) si planificarile neseriale (b) si (c)
  - Daca se foloseste un singur zavor binar pentru toate articolele grupate (XY) si se respecta protocolul de utilizare a zavoarelor, se obtine planificarea serializabilă (b)
  - Daca se folosesc mai multe zăvoare (câte unul pentru fiecare articol), se pot obține planificari neserializabile (c), chiar dacă se respecta protoc. de utilizare a zavoarelor
  - Planificarea (c) este neserializabilă și rezultatul obținut este eronat deoarece:
    - operațiile din perechea ((r3(Y), w4(Y)) au ordinea din planificarea serială (a1)
    - operațiile din perechea (r4(X), w3(X)) au ordinea din planificarea serială (a2)

Blocarea folosind zăvoare binare

X=20; Y=30

T3	T4
read(Y)	
read(X)	
X=X+Y	
write(X)	
	read(X)
	read(Y)
	Y=X+Y
	write(Y)

(a1)

T3	T4
	read(X)
	read(Y)
	Y=X+Y
	write(Y)
read(Y)	
read(X)	
X=X+Y	
write(X)	

(a2)

Rez. corect:
X=50; Y=80

X=20; Y=30

T3	T4
lock(XY)	
read(Y)	
read(X)	
	lock(XY)
X= X+Y	T4 blocata
write(X)	
unlock(XY)	
	read(X)
	read(Y)
	Y=X+Y
	write(Y)
	unlock(XY)

(b)

Rez. corect:
X=50; Y=80

X=20; Y=30

T3	T4
lock(Y)	
read(Y)	
unlock(Y)	
	lock(X)
	read(x)
	unlock(X)
	lock(Y)
	read(Y)
	Y=X+Y
	write(Y)
	unlock(Y)
lock(X)	
read(X)	
X=X+Y	
write(X)	
unlock(X)	

(c)

Rez. eronat:
X=50; Y=50

Protocolul de blocare în două faze (1)

- Pentru a asigura serializabilitatea planificărilor tranzactiilor care folosesc mai multe zavoare, pe lângă regulile de utilizare a zăvoarelor, mai este necesar să se respecte un protocol privind ordinea operatiilor de blocare şi de eliberare a zăvoarelor, numit protocolul de blocare in doua faze
- ***Protocolul de blocare în două faze*** (***two-phase locking***) impune ca fiecare tranzactie să respecte protocolul de utilizare a zavoarelor și toate operațiile de blocare a zăvoarelor să preceadă prima operație de eliberare a unui zăvor
- O astfel de tranzacție poate fi divizată în două faze:
  - faza de creştere (*growing phase*), în care pot fi achiziționate noi zăvoare ale articolelor care vor fi accesate, dar nici un zăvor nu poate fi eliberat
  - faza de descreştere (*shrinking phase*), în care zăvoarele deținute pot fi eliberate, dar nici un alt zăvor nu mai poate fi achiziționat.
- S-a demonstrat că, dacă fiecare tranzacție a unei planificări respectă protocolul de blocare în două faze, atunci planificarea este serializabilă
- Planificarea tranzacțiilor din figura precedenta (c) nu respectă protocolul de blocare în două faze deoarece:
  - T3 eliberează zăvorul articolului Y (unlock(Y)) înaintea achiziționării zăvorului articolului X (lock(X))
  - T4 eliberează zăvorul articolului X (unlock(X)) înaintea achiziționării zăvorului articolului Y (lock(Y))
- Aceasta planificare este neserializabilă, cu rezultat al execuției incorect, aşa cum s-a arătat mai înainte

Protocolul de blocare în două faze (2)

- Tranzacțiile din figura alăturată (d) respectă protocolul de blocare în două faze și rezultă o planificare serializabilă, echivalentă cu planificarea serială (a) (T3, T4), cu rezultat corect X = 50, Y = 80
- Daca T3 lansează operația de blocare a zăvorului articolului X înaintea tranzacției T4, tranzacția T4 este blocată, asteptând eliberarea zăvorului articolului X, după care efectueaza restul operațiilor
- Probleme care apar la utilizarea zăvoarelor:
  - ***Impasul*** (blocaj "mort" - *deadlock*): blocarea execuției tranzacțiilor atunci când două sau mai multe tranzacții se aşteaptă una pe cealălaltă ca să elibereze un zăvor; exista tehnici de prevenire si de eliminare a impasului
  - ***Amânarea nedefinită*** (*indefinit postponement, înfometare – starvation, blocaj "viu" - livelock*): o tranzacție se află în stare de amânare nedefinită dacă ea nu poate continua execuția o perioadă lungă de timp, în timp ce toate celelalte tranzacții se execută normal; prevenirea se face prin asigurarea unei politici echilibrate de obtinere a zavoarelor

X=20, Y=30

T3	T4
lock(Y)	
read(Y)	
lock(X)	
	lock(X)
unlock(Y)	T4 blocata
read(X)	
X=X+Y	
write(X)	
unlock(X)	
	read(X)
	lock(Y)
	unlock(X)
	read(Y)
	Y=X+Y
	write(Y)
	unlock(Y)

(d)

Rez. corect X=50, Y=80

Controlul concurentei bazat pe marci de timp

- ***O marcă de timp*** *(timestamp)* este un identificator unic al unei tranzacții, creat de sistemul de gestiune a bazei de date, care se bazează pe timpul de start al tranzacției
- O marcă de timp se poate crea:
  - fie folosind valoarea curentă a ceasului sistemului de operare
  - fie folosind un numărător care este incrementat la fiecare asignare a unei noi mărci, în ordinea de lansare a tranzacțiilor
- O tranzacție T va avea o marcă de timp unică, notată TS(T)
- Pentru fiecare articol X al bazei de date se definesc două mărci de timp:
  - R_TS(X) - marca de timp de citire a articolului X; este cea mai mare marcă de timp dintre mărcile de timp ale tranzacțiilor care au citit articolul X
  - W_TS(X) - marca de timp de scriere a articolului X; este cea mai mare marcă de timp dintre mărcile de timp ale tranzacțiilor care au scris în articolul X
- Serializabilitatea planificărilor se obține dacă se impun anumite condiții ordinii de accesare a articolelor de mai multe tranzacții concurente, în funcție de mărcile de timp ale acestora

Ordonarea operatiilor după mărcile de timp

- La lansarea unei operații de citire a articolului X (read(X)):
  - **Dacă TS(T) >= W_TS(X), atunci T va executa operația de citire din articolul X şi va seta marca R_TS(X) la cea mai mare dintre valorile TS(T) şi R_TS(X)**
  - Dacă TS(T) < W_TS(X), atunci tranzacția T trebuie să fie abandonată şi rulată înapoi, deoarece o altă tranzacție cu o marcă de timp mai mare a scris deja în articolul X, înainte ca T să fi avut şansa să citească articolul X
- La lansarea unei operații de scriere a articolului X (write(X)):
  - **Dacă TS(T) >= R_TS(X) şi TS(T) >= W_TS(X), atunci T va executa operația de scriere în articolul X şi va seta W_TS(X) = TS(T)**
  - Dacă TS(T) < R_TS(X) , atunci tranzacția T trebuie să fie abandonată şi rulată înapoi, deoarece o altă tranzacție cu o marcă de timp mai mare (deci lansată după T) a citit deja valoarea lui X, înainte ca T să fi avut şansa să scrie în X
  - Dacă TS(T) < W_TS(X), atunci tranzacția T nu va executa operația de scriere, dar va putea continua cu celelalte operații. Aceasta, deoarece o altă tranzacție, cu o marcă de timp mai mare a scris deja o valoare în articolul X, care este mai recentă, iar valoarea pe care ar dori să o înscrie T este deja perimată
- Ulterior, o tranzacție T care a fost anulată şi rulată înapoi va fi relansată, dar cu o nouă marcă de timp, corespunzătoare momentului noii lansări
- Ordonarea după mărcile de timp garantează serializabilitatea planificărilor
- În acest protocol nu poate să apară impasul, dar poate apare amânarea indefinită

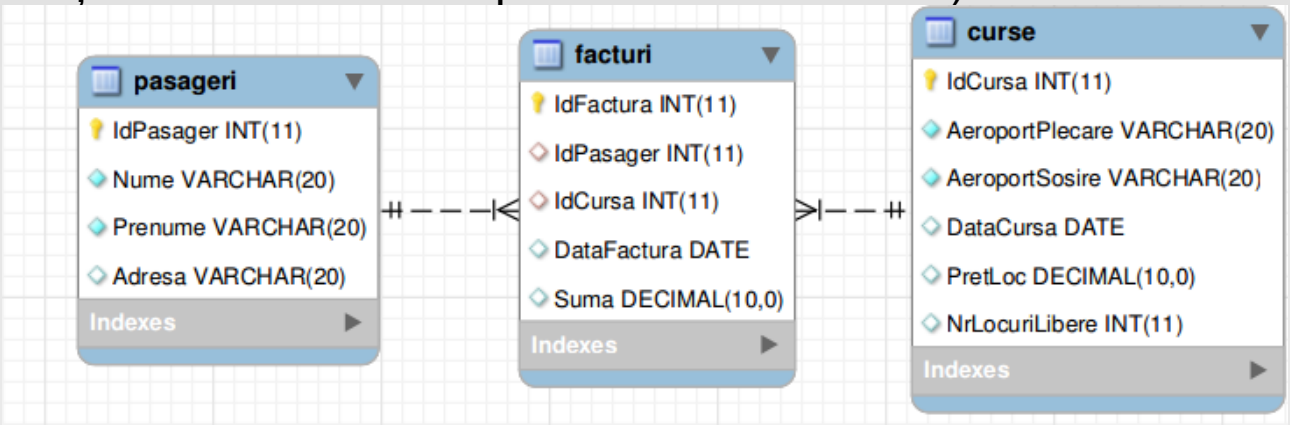
Controlul tranzacțiilor

- Tehnicile de gestiune a tranzacțiilor şi de refacere a datelor sunt incluse în SGBD-uri, iar aplicațiile de baze de date au un control limitat asupra tranzacțiilor prin intermediul unor comenzi SQL
- Comenzi SQL pentru tranzacții:
  - **`SET SESSION TRANSACTION`**: setare (definire) tranzacție cu posibile optiuni:
    Nivelul de izolare a tranzacțiilor (**`ISOLATION LEVEL`**)
    Modul de refacere a datelor (**`SET CONSTRAINTS`**)
    Modul de acces la articole - cu valori posibile **`READ ONLY, READ WRITE`** (implicit)
  - **`SET autocommit = {0 | 1}`** – modul autocommit (implicit 1): fiecare operație cu baza de date este validată imediat
  - **`START TRANSACTION`** – lansarea tranzactiei
  - **`COMMIT [WORK]`** – terminarea cu validare a tranzactiei
  - **`ROLLBACK [WORK]`** – abandonarea si rularea inapoi a tranzactiei
- În orice nivel de izolare (**`ISOLATION LEVEL`**) este interzisă pierderea actualizarilor, dar se admit unele citiri incorecte, asa cum se vede in tabel
- Nivelul de izolare se obține prin tipurile de zăvoare folosite (binare sau cu stări multiple) și prin poziționarea punctelor de salvare (checkpoints – așa cum se va vedea la refacerea datelor)

Nivel de izolare	Citire impropie	Citire nerepetabila	Citire fantoma
READ UNCOMMITTED	DA	DA	DA
READ COMMITTED	NU	DA	DA
REPEATABLE READ	NU	NU	DA
SERIALIZABLE	NU	NU	NU

Exemplu de tranzactie in MySQL (1)

- In MySQL o tranzactie se lanseaza cu comanda START TRANSACTION
- Fie baza de date ZBORURI cu tabelele descrise la inceputul capitolului (tabelele pasageri și facturi au PK de tip AUTOINCREMENT):



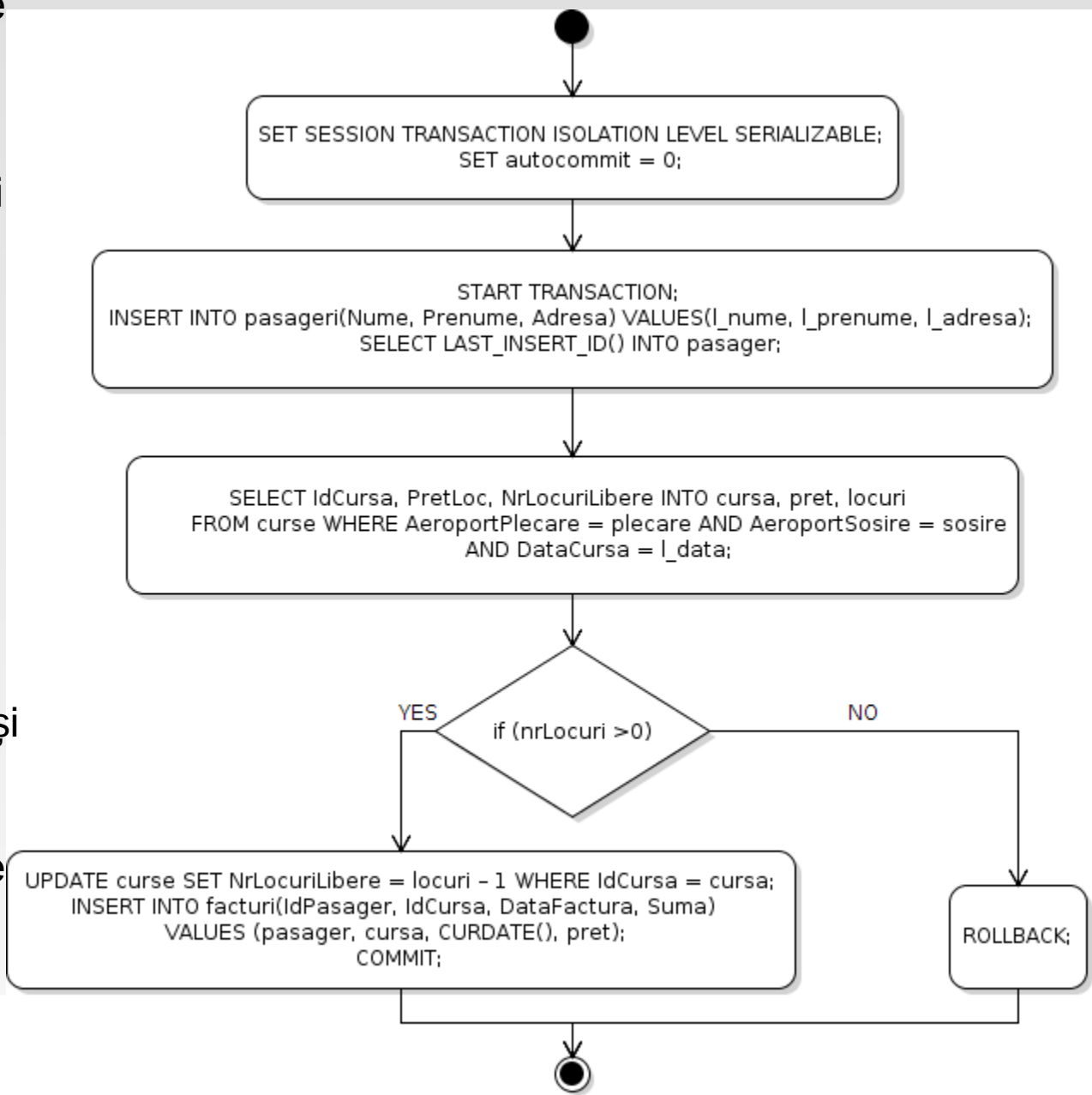
- Se defineste o tranzactie pentru rezervarea unui loc la o cursa aeriana in procedura stocată sp_rezervari()
- Apelul procedurii sp_rezervari (exemplu):

```
call sp_rezervari(@locuri, 'Ionescu', 'Ion', 'Craiova', 'Bucuresti', 'Amsterdam', '2016-01-25');
select @locuri;
```

  - Dacă locuri = NrLocuriLibere (din tabelul curse) > 0, toate tabelele sunt modificate:
    - În tabelul pasageri este inserată o linie cu datele pasagerului
    - În tabelul curse NrLocuriLibere este decrementat cu 1
    - În tabelul facturi este inserată o linie cu IdPasager, IdCursa și Suma corespunzatoare
    - Se validează tranzacția (commit)
  - Daca locuri = NrLocuriLibere (din tabelul curse) = 0, se anulează tranzacția (rollback), nici un tabel nu este modificat

Exemplu de tranzactie în MySQL (2)

- În diagrama de activitate UML sunt prezentate operațiile principale ale tranzacției: instrucțiunile de control ale tranzacției și instrucțiunile SQL transmise SGBD-ului
- Tranzacția începe cu **`START TRANSACTION`** și inserarea datelor pasagerului în tabelul pasageri
- Dacă există locuri la cursa dorită, se actualizează datele în tabelele curse și facturi și se validează tranzacția **`(COMMIT)`**
- Dacă nu există locuri, se anulează tranzacția cu **`ROLLBACK`**, ștergându-se și datele inserate în tabelul pasageri



Exemplu de tranzactie în MySQL (3)

```
DELIMITER $$  DROP PROCEDURE IF EXISTS `zboruri`.`sp_rezervari` $$
CREATE  PROCEDURE `zboruri`.`sp_rezervari`(OUT locuri INT, l_nume varchar(20),
   l_prenume varchar(20), l_adresa varchar(20), plecare varchar(20), sosire
   varchar(20), l_data date)
BEGIN DECLARE cursa, pret, pasager INT;
  SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
  SET autocommit = 0;
  START TRANSACTION;
  INSERT INTO pasageri(Nume,Prenume,Adresa) values(l_nume,l_prenume,l_adresa);
  SELECT LAST_INSERT_ID()INTO pasager;
  SELECT IdCursa, PretLoc, NrLocuriLibere INTO cursa, pret, locuri
        FROM curse WHERE AeroportPlecare = plecare AND AeroportSosire = sosire
        AND DataCursa = l_data;
  IF locuri > 0 THEN
     BEGIN
        UPDATE curse SET NrLocuriLibere = locuri - 1 WHERE IdCursa = cursa;
        INSERT INTO facturi(IdPasager, IdCursa, DataFactura, Suma)
               VALUES (pasager, cursa, CURDATE(), pret);
        COMMIT;
     END;
  ELSE
     BEGIN
        ROLLBACK;
     END;
  END IF;
END $$
DELIMITER ;
```

Tranzacţii JDBC

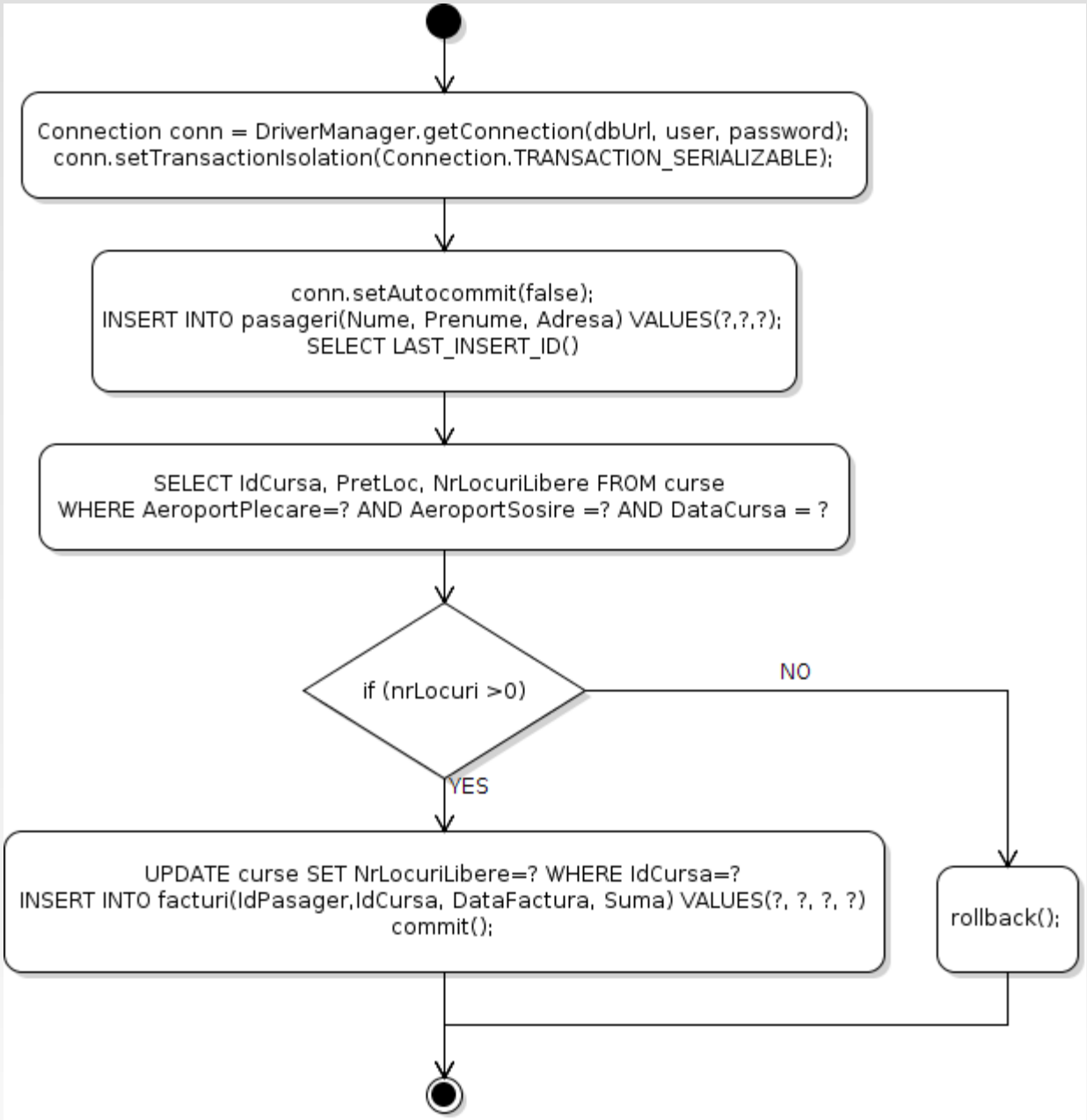
- Atunci când se creează o conexiune JDBC, aceasta este în modul auto-commit, adică fiecare operație este validată (commit) imediat după ce a fost executată
- Pentru gruparea a două sau mai multe operații cu baza de date (instrucțiuni SQL) ca tranzacții, se invalidează modul auto-commit cu **`setAutoCommit(false)`** – metodă a interfeței **`Connection`**:

```
Connection conn = DriverManager.getConnection(.......);
conn.setAutoCommit(false);
```

- Invalidarea modului auto-commit al unei conexiuni JDBC reprezintă lansarea tranzacției:
  - După invalidarea modului auto-commit, orice instrucțiune SQL care urmează este inclusă în tranzacția curentă care este validată atunci când se execută metoda **`commit()`** a interfeței **`Connection`**, sau abandonată atunci când se execută metoda **`rollback()`** a interfeței **`Connection`**
- Setarea nivelului de izolare a tranzacțiilor se realizează prin apelul metodei **`setTransactionIsolation(int level)`** a interfeței **`Connection`**, unde argum. **`level`** poate lua una din valorile constante definite în interfața **`Connection`**:
  - **`TRANSACTION_READ_UNCOMMITED`**
  - **`TRANSACTION_READ_COMMITED`**
  - **`TRANSACTION_REPETABLE_READ`**
  - **`TRANSACTION_SERIALIZABLE`**

Exemplu de tranzacție JDBC (1)

- În acest exemplu, tranzacția de rezervare a biletelor la curse aeriene pentru baza de date zboruri se definește în programul aplicație, care se conectează la baza de date prin interfață JDBC
  - Tranzacția începe cu **`setAutocommit(false)`** și inserarea datelor pasagerului în tabelul pasageri
  - Dacă există locuri la cursa dorită, se actualizează datele în tabelele curse și facturi și se validează tranzacția **`(commit())`**
  - Dacă nu există locuri, se anulează tranzacția **`(rollback())`**, ștergându-se și datele inserate în tabelul pasageri



Exemplu de tranzacție JDBC (2)

```
package zboruri;
import java.sql.*;
import java.text.SimpleDateFormat;
public class Rezervari{
  public static void main(String[] args) {
     String dbUrl = "jdbc:mysql://localhost:3306/zboruri";
     String user = "root", password = "parola";
     try {
        Class.forName("com.mysql.jdbc.Driver");
        Connection conn = DriverManager.getConnection(dbUrl, user, password);
        conn.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
        // Pregatirea argumentelor functiei rezervare()
        String nume = "Martin", prenume = "Gabriela", adresa = "Vaslui";
        String plecare = "Bucuresti", sosire = "Chicago";
        java.util.Date utilDate = new
                 SimpleDateFormat("yyyy-MM-dd").parse("2016-01-25");
        java.sql.Date dataCursa = new java.sql.Date(utilDate.getTime());

        if (rezervare(conn,nume,prenume,adresa,plecare,sosire,dataCursa) ){
             conn.commit(); System.out.println("Tranzactie validata");}
        else { conn.rollback(); System.out.println("Tranzactie anulata");}
        conn.close();
     } catch(Exception e) {e.printStackTrace(); }
  }
  // Functia de rezervare bilete
  public static boolean rezervare(Connection conn,String nume,String prenume,
    String adresa, String plecare, String sosire, java.sql.Date dataCursa)
 {.....................}
}
```

Exemplu de tranzacție JDBC (3)

```
  // Functia de rezervare bilete defineste tranzactia
public static boolean rezervare(Connection conn, String nume, String prenume,
       String adresa,  String plecare, String sosire,java.sql.Date dataCursa){
  try {
   // Setarea modului cu validare (commit) - start tranzactie
   conn.setAutoCommit(false);
   PreparedStatement pstmt;
   pstmt = conn.prepareStatement("INSERT INTO pasageri(Nume, Prenume, Adresa)
           VALUES(?,?,?)");
   pstmt.setString(1,nume);pstmt.setString(2,prenume);pstmt.setString(3,adresa);
   pstmt.executeUpdate();
   Statement stmt = conn.createStatement();
   ResultSet rs = stmt.executeQuery("SELECT LAST_INSERT_ID()");
   rs.next(); int idPasager = rs.getInt(1);
   // Preg instr de interogare pentru aflare nr locuri libere la cursa resp.
   pstmt = conn.prepareStatement("SELECT IdCursa, PretLoc, NrLocuriLibere
   FROM curse WHERE AeroportPlecare=? AND AeroportSosire =? AND DataCursa = ?");
   pstmt.setString(1,plecare); pstmt.setString(2,sosire);
   pstmt.setDate(3,dataCursa);
   // Executia interogarii
   rs = pstmt.executeQuery();
   int idCursa, nrLocuriLibere; float pretLoc;
   if (rs.next()) {
      idCursa= rs.getInt(1);pretLoc= rs.getFloat(2);nrLocuriLibere=rs.getInt(3);
   }
   else {throw new Exception("Nu exista aceasta cursa");}
   // Testarea numarului de locuri libere
   if (nrLocuriLibere > 0)  // continuare pe pagina următoare
```

Exemplu de tranzacție JDBC (4)

```
    // Functia de rezervare bilete - continuare din pagina precedenta
    if (nrLocuriLibere > 0){
      // Actualizare nr. locuri libere in tabelul CURSE
      pstmt = conn.prepareStatement("UPDATE curse SET NrLocuriLibere = ?
              WHERE IdCursa = ?");
      nrLocuriLibere = nrLocuriLibere-1;
      pstmt.setInt(1, nrLocuriLibere); pstmt.setInt(2, idCursa);
      pstmt.executeUpdate();
      // Inserare linie in tabelul FACTURI
      java.util.Date timp = new java.util.Date();
      java.sql.Date dataFactura = new java.sql.Date(timp.getTime());
      pstmt = conn.prepareStatement ("INSERT INTO facturi(IdPasager,IdCursa,
              DataFactura, Suma)VALUES(?, ?, ?, ?)");
      pstmt.setInt(1, idPasager);    pstmt.setInt(2, idCursa);
      pstmt.setDate(3, dataFactura); pstmt.setFloat(4, pretLoc);
      pstmt.executeUpdate();
      // Tranzactia va fi validata
      pstmt.close(); stmt.close();
      return true;
    }
    else { // Daca nr. de locuri libere este 0, tranzactia se va anula
      pstmt.close(); stmt.close();
      return false;
    }
  } catch (Exception e) {e.printStackTrace();
    return false;
  }
} // end metoda rezervare()
```

Proiectarea tranzactiilor

- Tranzacțiile sunt corecte dacă lasă baza de date într-o stare consistentă
- Tranzacțiile sunt cu atât mai eficiente cu cât sunt mai scurte (ca timp de execuție şi ca număr de articole ale bazei de date accesate) deoarece astfel:
  - se limiteaza frecvența de apariție a impasului (în cazul folosirii zăvoarelor)
  - scade timpul de execuție a operațiilor rollback, în cazul anulării tranzacțiilor
- Se recomandă înlocuirea, ori de câte ori este posibil a unei tranzacții complexe, cu număr mare de operații şi timp de execuție ridicat, cu mai multe tranzacții scurte
- De asemenea, pentru menținerea tranzacțiilor cât mai scurte posibil, se recomandă ca o tranzacție să nu fie pornită, decât după ce au fost pregătite toate datele (citirea datelor de intrare, parcurgerea, analiza şi prelucrarea acestora)
- Toate operațiile de gestiune a tranzacțiilor şi de refacere a datelor sunt prevăzute în diferitele componente ale sistemelor SGBD (administratorul de tranzacții, administratorul de refacere), iar aplicațiile:
  - trebuie să se prevadă tranzacții corecte (ca operații efectuate și ca program)
  - pot selecta diferite opțiuni de control al tranzacțiilor şi de refacere a datelor oferite de sistemul de gestiune respectiv

Tehnici de refacere a bazelor de date

- Refacerea unei baze de date după producerea unui defect (*database recovery*) înseamnă aducerea bazei de date într-o stare precedentă corectă, din care, eventual, se poate reconstrui o nouă stare corectă cât mai apropiată de momentul apariției defectului
  - Tehnicile de refacere a bazelor de date sunt, în general, integrate cu tehnicile de control al tranzacțiilor si depind de SGBD
  - Pentru operațiile de refacere se foloseşte fişierul jurnal (*log file*), şi (sau) o copie de salvare a bazei de date (*database backup*), stocată în general pe bandă magnetică
- ***Un punct de validare*** (*commit point*) este punctul atins de o tranzacție care a executat cu succes toate operațiile sale şi le-a înregistrat în fişierul jurnal
  - Într-un astfel de punct, o tranzacție T înscrie operația [commit] în bufferul de scriere al fișierului jurnal şi, de asemenea, scrie acest buffer în fişierul jurnal
- ***Un punct de control*** (*checkpoint*) este înscris în fişierul jurnal atunci când se scriu în fişierele bazei de date toate rezultatele operațiilor de scriere ale tranzacțiilor validate, prin scrierea bufferului de scriere în fisierele bazei de date
  - Aceasta înseamnă că toate tranzacțiile care au înregistrarea [commit] înscrisă în fişierul jurnal înaintea unui punct de control nu vor necesita reluarea operațiilor de scriere în cazul unei defectări necatastrofice a sistemului
- Administratorul de refacere al SGBD-ului (*recovery manager*) decide la ce interval de timp (sau după câte tranzacții) introduce un nou punct de control

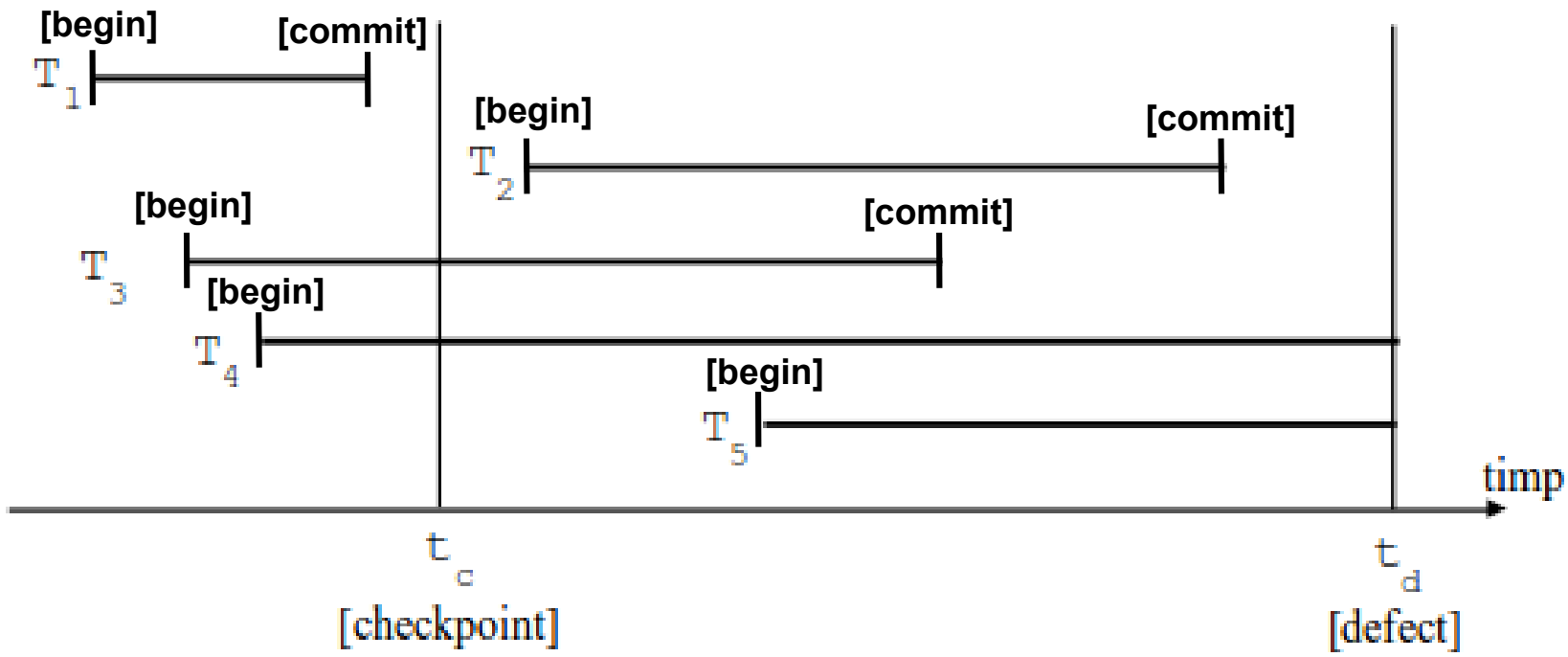
Refacerea datelor după defecte necatastrofice

- Dacă baza de date nu este distrusă fizic, dar a devenit inconsistentă datorită unui defect necatastrofic, atunci, pentru refacere, se foloseşte starea actuală a bazei de date şi fişierul jurnal
  - La apariția unui defect necatastrofic se pierd datele din memoria fizică a calculatorului (inclusiv cele înscrise în bufferele de scriere, dar ne-transferate pe hard-disk), nu și cele înregistrate pe hard-disk în fișierele bazei de date și în fișierul jurnal
- Exista două tehnici de refacere a datelor după defecte necatastrofice:
  - Refacerea cu actualizare amânată (*deferred update*)
  - Refacerea cu actualizare imediată (*immediate update*)
- ***Refacerea cu actualizare amanată*** folosește operații de reluare (REDO)
  - Pentru operații **`REDO`**, înregistrările **`write`** din fişierul jurnal conțin identificatorul tranzacției (T), articolul în care se scrie (X) și valoarea care se scrie (val) **`[write,T,X,val];`** operația **`redo`** scrie valoarea val în articolul X al bazei de date
  - În figura următoare se arată o posibilă planificare a mai multor tranzacții, un punct de control (**`checkpoint`**), înregistrat în fişierul jurnal la momentul de timp t_c şi apariția unui defect necatastrofic la momentul t_d
  - Pentru refacere, se parcurge fişierul jurnal în sens invers, începând de la ultima înregistrare, până se întâlneşte primul punct de control (**`checkpoint`**) și apoi:
  - (a) Tranzacțiile care au înregistrarea **`[commit]`** înainte de checkpoint (cum este T_1), nu sunt afectate de defectul apărut

Refacerea cu actualizare amânată

- (b) Se construieste o listă a tranzacțiilor validate LTV, în care se introduc toate tranzacțiile care au o înregistrare de validare **`[commit]`** în fişierul jurnal între ultimul punct de control şi sfârşitul fişierului jurnal; în exemplu, LTV = {T2, T3}
- (c) Se construieste o listă a tranzacțiilor nevalidate LTNV, în care se introduc toate tranzacțiile care au o înregistrare de start **`[begin]`** în fişierul jurnal, dar nu au şi înregistrarea corespunzătoare de validare **`[commit]`**; în exemplu: LTNV = {T4, T5}

- După aceasta:
  - Se execută reluarea (REDO) a tuturor operațiilor de scriere **`[write,T,X,val]`** ale tranzacțiilor validate (T_2 și T_3), în ordinea în care apar în fişierul jurnal
  - Tranzacțiile nevalidate sunt relansate: T_4, T_5



Refacerea cu actualizare imediată

- In tehnicile de refacere cu actualizare imediată, atunci când o tranzacție lansează o comandă de actualizare a bazei de date, actualizarea este efectuată imediat, fără să se mai aştepte ajungerea la un punct de validare
- In majoritatea acestor tehnici se impune ca modificarea să fie mai întâi memorată în fişierul jurnal (pe disc), înainte de a fi aplicată bazei de date; această regulă este cunoscută sub numele de “protocol de scriere în avans a fişierului jurnal” (*write-ahead log protocol*)
- În tehnica cu actualizare imediată, refacerea se execută prin anularea (folosind operații `undo`) modificărilor efectuate de o tranzacție, dacă aceasta eşuează ulterior din diferite motive; în felul acesta se efectuează rularea înapoi (rollback) a tranzacției
- Tehnica de refacere cu actualizare imediată are avantajul simplității operațiilor de scriere, care se efectuează direct în baza de date, fără să fie necesar să se aştepte atingerea unui punct de validare pentru transferarea datelor în baza de date
- Dezavantajul acestei metode este faptul că pot apare anulări în cascadă, care necesită operații complicate de refacere

Refacerea datelor după defecte catastrofice

- Dacă fișierele bazei de date au fost distruse datorită defectări discului, atunci se restaurează baza de date din copia de rezervă (salvată) a bazei de date (*database backup*)
- Ultima copie salvată se încarcă de pe bandă pe disc (după ce acesta a fost înlocuit sau reparat) şi se repornește sistemul
- Totuşi, tranzacțiile efectuate de la ultima operație de salvare a bazei de date până în momentul apariției defectului se pierd
- Deoarece fişierul jurnal este mult mai mic decât fişierele bazei de date, se obişnuieşte ca acesta să fie salvat mai des decât baza de date însăşi
- În această situație, după încărcarea ultimei copii salvate a bazei de date se poate folosi copia salvată a fişierului jurnal pentru a reface toate tranzacțiile validate existente în copia salvată a fişierului jurnal (care este mai recentă decât copia salvată a bazei de date)