

6. OPERAȚII CONCURENTE ASUPRA BAZELOR DE DATE. ADMINISTRAREA TRANZACȚIILOR

Reamintim că un **obiectiv major al SGBD este de a permite mai multor utilizatori să acceseze concurent datele partajate**. Dar la sistemele în care o BD este **accesată simultan de mai mulți utilizatori** apar situații 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 secvențialitatea accesului la mediul de memorare este suficientă și nu mai este nevoie de precauții suplimentare. Atât timp cât nici unul nu face modificări ale datelor, nu are importanță ordinea în care se asigură accesul utilizatorului la date. Deci, în acest caz, rezolvarea situațiilor conflictuale cauzate de operațiile de citire simultană poate fi lăsată în seama **sistemului de operare**, care stabilește ordinea de satisfacere a cererilor după criteriul **asigurării concurenței 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 mulți agenți fac permanent modificări. Pericolul: două procese, care citesc și modifică valoarea aceluiași obiect ar putea interacționa atunci când sunt executate simultan. Rezultă necesitatea **impunerii unor restricții asupra execuției concurente** a proceselor care fac operații de citire și modificare a datelor. O rezolvare imediată și simplă ar fi blocarea BD pe durata rezolvării unei cereri. Dar asta echivalează cu blocarea concurenței și degradarea performanțelor sistemului, făcându-l uneori neutilizabil.

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

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

6.1 Tranzacții și unități de acces

Se numește **tranzacție** orice execuție 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 execuții independente ale aceluiași program și fiecare dintre acestea este o tranzacție.

D.p.d.v. al utilizatorului, **o tranzacție este o unitate singulară de execuție** care satisface două condiții referitoare la BD:

- BD este într-o stare **coerentă înaintea și după** execuția tranzacției.
- BD poate fi într-o stare **necoerentă în timpul execuției** unei tranzacții.

O tranzacție poate avea **două rezultate**:

1. dacă este completată cu succes, se spune că tranzacția 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 tranzacție este rulată înapoi sau abandonată.

O tranzacție efectuată nu mai poate fi abandonată.

Pentru a **delimita tranzacțiile**, în majoritatea limbajelor de manipulare a datelor sunt disponibile instrucțiunile **BEGIN TRANSACTION**, **COMMIT** și **ROLLBACK**. Dacă utilizatorul nu folosește aceste delimitări, de obicei **întregul program** este considerat ca reprezentând **o singură tranzacție** iar SGBD execută automat o instrucțiune COMMIT atunci când programul este încheiat corect, sau una ROLLBACK în caz contrar.

- **BEGIN TRANSACTION** – începe o nouă tranzacție
- **COMMIT** – salvează orice schimbări și încheie tranzacția curentă
- **ROLLBACK** – anulează orice schimbări făcute în timpul tranzacției curente și încheie tranzacția

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

Proprietățile tranzacțiilor

Există o serie de proprietăți pe care trebuie să le aibă toate tranzacțiile. Cele patru proprietăți fundamentale, așa-numitele **acid**, sunt:

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

O BD este partiționată în mai multe **unități de acces (items)**. Acestea sunt porțiuni ale BD care pot constitui obiectul unei **operații de blocare (lock)**. Prin blocarea unei unități de acces, **o tranzacție poate împiedica accesul altor tranzacții la unitatea blocată**, până la momentul deblocării acestei unități de către tranzacția care a efectuat blocarea.

Gestiunea operațiilor de blocare precum și **arbitrarea** cererilor de blocare venite din partea tranzacțiilor este realizată de o componentă specială a SGBD, numită **lock manager**.

Natura și dimensiunea unităților de acces este stabilită de proiectantul sistemului. De ex, în cazul modelului de date relațional, unitățile de acces pot fi de dimensiuni foarte variate, cuprinzând **relații întregi** ale BD, **grupuri de tuple**, **tuple individuale** sau chiar **componente** ale tuplelor.

Prin alegerea **unităților de acces de mari dimensiuni**, se reduce numărul operațiilor de blocare, ceea ce înseamnă reducerea timpului consumat de sistem pentru gestiunea acestor operații, precum și reducerea spațiului de memorie necesar înregistrării blocajelor.

În schimb, folosind **unități de acces de mici dimensiuni**, crește gradul de concurență suportat de sistem, deoarece pot fi executate în paralel un număr mai mare de tranzacții care operează în unități de acces diferite.

În practică, dimensiunea potrivită a unităților de acces este dată de extinderea operațiilor efectuate de **tranzacțiile cu cea mai mare frecvență**. Astfel, dacă tranzacția tipică presupune efectuarea unor operații de cuplare, atunci unitatea de acces va fi relația. Dacă, însă, majoritatea tranzacțiilor efectuează operații asupra unor tuple individuale, atunci va fi convenabil să se aleagă tupla ca unitate de acces.

6.2 Anomalii de interferență

Interacțiunea necontrolată a două sau mai multe tranzacții poate duce la apariția unor stări inconsistente ale BD și la producerea unor rezultate eronate. Două tranzacții **T_i** și **T_j** sunt **susceptibile de interferență** dacă **rezultatul execuției lor concurente poate fi diferit de rezultatul execuției seriale**.

Între două tranzacții poate să apară o interferență dacă acestea efectuează operații asupra unor date comune. Dacă aceste două tranzacții sunt executate în mod concurent, atunci spunem că sunt **conflictuale**.

Deci două tranzacții **T_i** și **T_j** sunt **conflictuale** dacă sunt **concurente** și **susceptibile de interferență**.

În funcție de natura operațiilor pe care le efectuează asupra datelor comune, între două tranzacții pot să apară mai multe tipuri de interferențe care provoacă **anomalii de interferență**:

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

6.2.1 Anomalia de actualizare pierdută

Corespunde unui conflict de tip **scriere-scriere** și constă în faptul că rezultatul actualizării efectuate de o tranzacție se pierde ca urmare a reactualizării aceleiași date de către o altă tranzacție, fără ca reactualizarea să fie influențată de rezultatul primei actualizări.

Exemplu: fie următoarea execuție concurentă a două tranzacții **T1** și **T2**.

Observație: în acest capitol vom nota simbolic operațiile de citire și 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 tranzacții valoarea lui A apare mărită cu 10, nu cu 15 așa cum ar fi de așteptat. Este ca și cum tranzacția **T1** nici nu s-ar fi executat.

6.2.2 Anomalia de citire improprie

Corespunde unui conflict de tip **scriere-citire** și apare atunci când o tranzacție surprinde o stare temporară inconsistentă a BD.

Exemplu: fie următoarea execuție concurentă a două tranzacții **T1** și **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 tranzacții valoarea sumei $A + B$ este neschimbată. Intenția era de a reține în C valoarea acestei sume, dar datorită interferenței, valoarea din C este cu 10 mai mică decât cea reală.

6.2.3 Anomalia de citire irepetabilă

Corespunde unui conflict de tip **citire-scriere** și apare atunci când aceeași tranzacție găsește valori diferite la citiri repetate ale aceleiași date.

Exemplu: fie următoarea execuție concurentă a două tranzacții **T1** și **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, deși valorile rezultate pentru B și C ar trebui să fie egale în urma execuției tranzacției **T1**, ele sunt diferite din cauza interferenței cu tranzacția **T2**.

6.3 Primitivele LOCK și UNLOCK

Fie tranzacțiile **T1** și **T2** execuții diferite ale următoarei secvențe de operații:

```

Read A
A = A + 1
Write A

```

unde **A** este o valoare existentă în bază. Fiecare din cele două tranzacții citește valoarea lui **A** într-o zonă de lucru proprie, adună 1 la această valoare și apoi scrie rezultatul în baza de date. După execuția tranzacțiilor, este de așteptat ca valoarea lui **A** să fie mărită cu 2. Totuși, dacă tranzacțiile sunt executate concurrent, este posibil ca rezultatul final să fie altul, funcție de modul de interferență a tranzacțiilor.

Cea mai simplă metodă de evitare a situațiilor de genul celei prezentate, este de a **permite accesul la valoarea lui A numai pentru o singură tranzacție**, pe toată durata executării tranzacției. Accesul celorlalte tranzacții va fi temporar blocat. Acest lucru se poate realiza folosind două funcții primitive **LOCK (A)** și **UNLOCK (A)**. Aceste funcții se numesc **primitive**, deoarece secvența de operații corespunzătoare execuției lor nu poate fi întreruptă de alte operații. **LOCK (A)** și **UNLOCK (A)** sunt operații indivizibile.

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

Primitivele **LOCK()** și **UNLOCK()** pot fi folosite pentru realizarea mecanismelor de **sincronizare** a tranzacțiilor. Dacă o tranzacție dorește să acceseze o anumită valoare, ea va trebui să obțină accesul exclusiv la aceasta, prin executarea unei primitive **LOCK()**. Dacă valoarea este blocată, atunci accesul exclusiv va fi refuzat, iar tranzacția va trebui să aștepte până la deblocarea aceste valori. Orice tranzacție care a executat cu succes o primitivă **LOCK()** asupra unei valori, va trebui să execute primitiva **UNLOCK()** asupra aceleiași valori înainte de a-și încheia execuția. Ordonarea secvențială a pașilor a două sau mai multe tranzacții care respectă regulile de mai sus, se spune că este **legată**.

Pentru tranzacțiile **T1** și **T2** considerate anterior, secvența de pași folosind primitivele **LOCK()** și **UNLOCK()** este următoarea:

```

While NOT(LOCK(A))
  Read A
  A = A + 1
  Write A
UNLOCK(A)

```

Dacă una dintre tranzacții, să zicem **T1**, obține accesul exclusiv la valoarea **A**, atunci tranzacția **T2** va trebui să aștepte terminarea completă a lui **T1** pentru a obține accesul la valoarea **A**. La

terminarea tranzacției **T1** valoarea lui **A** este mărită cu 1, iar la terminarea tranzacției **T2**, valoarea lui **A** va fi mărită cu 2.

Rezultatul este corect, dar execuția este **pur secvențială**, nu este posibil nici un fel de paralelism între cele două tranzacții. D.p.d.v. al timpului de execuție, această situație este inacceptabilă și de aceea se folosesc **algoritmi care să relizeze ordonări secvențiale legate**, cu un grad de concurență a execuției cât mai ridicat, dar și cu garanția obținerii de rezultate corecte.

6.4 Interblocarea

Unul dintre principalele obiective ale oricărui sistem concurrent 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 tranzacții concurente și fiecare tranzacție deține controlul exclusiv al unor date particulare, este posibil să se ajungă la situația în care unele tranzacții nu-și vor putea termina niciodată execuția.

Exemplu. Fie două tranzacții **T1** și **T2** definite prin două secvențe 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)

Presupunem că tranzacțiile **T1** și **T2** își încep execuția aproximativ în același moment. **T1** cere și obține blocarea lui **A** iar **T2** cere și obține blocarea lui **B**. Apoi **T1** cere blocarea lui **B**, dar este pusă în așteptare deoarece unitatea de acces **B** este blocată de tranzacția **T2**. Concomitent, **T2** cere blocarea lui **A** și este pusă în așteptare până când **T1** deblochează pe **A**, dar **T1** așteaptă deblocarea lui **B** de către **T2**. În consecință, nici una dintre tranzacții nu-și poate continua execuția, ambele fiind puse în așteptare la nesfârșit. O astfel de situație se numește **interblocare**.

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

1. metode de **prevenire și evitare** a interblocării
2. metode de **dectecție și ieșire** din interblocare

Condiții pentru interblocare

Într-un sistem concurrent, poate apare situația de interblocare numai dacă sunt satisfăcute simultan următoarele 4 condiții:

1. condiția de **excludere mutuală** – tranzacțiile solicită controlul exclusiv al unităților de acces asupra cărora operează
2. condiția de **așteptare pentru** – o tranzacție care deține controlul exclusiv asupra unor unități de acces este în așteptare pentru altele
3. condiția de **completare** – nici o unitate de acces nu poate fi deblocată de către tranzacția care o controlează înainte ca aceasta să termine toate operațiile pe care le are de executat asupra unității respective
4. condiția de **așteptare circulară** – există un lanț circular de tranzacții cu proprietatea că fiecare tranzacție deține controlul asupra unei unități de acces solicitată de următoarea tranzacție din lanț.

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

Negarea condiției 1 conduce la observația că nu poate apare interblocare în cazul tranzacțiilor care nu solicită accesul exclusiv la unitățile de acces. În cazul execuției concurente a unui set de tranzacții care efectuează numai operații de citire nu poate apare interblocare.

Negarea condiției 2 conduce la o metodă de prevenire a interblocării care are la bază alocarea unităților de acces după criteriul „tot sau nimic”, numită **metoda cererilor anticipate**.

Negarea condiției 3 conduce la o metodă de detecție și ieșire din interblocare care presupune abandonarea (renunțarea) unora dintre tranzacțiile aflate în interblocare la un moment dat.

Negarea condiției 4 conduce la o metodă de prevenire a interblocării bazată pe ordonarea unităților de acces, numită **metoda ordonării**.

6.4.1 Metoda cererilor anticipate

Blocarea unităților de acces de către tranzacții se face după regula „tot sau nimic”. Fiecare tranzacție emite deodată, toate cererile de blocare necesare execuției sale complete, în mod anticipat, înainte de a executa orice operație de actualizare. Sistemul acceptă sau respinge aceste cereri în bloc. Nu este posibil ca o tranzacție să obțină blocarea unei părți a unităților pe care dorește să le acceseze, pentru ca pe parcurs să emită alte cereri de blocare. Astfel, nici o tranzacție care a obținut blocarea unor unități de acces nu va putea fi pusă în așteptare.

Dezavantaje:

- tranzacțiile blochează unele dintre unitățile de acces pe o durată mai mare decât este necesar, ceea ce reduce nivelul de concurență al sistemului.
- favorizează apariția fenomenului de amânare nedefinită sau „infometare” a tranzacțiilor. Tranzacțiile care solicită accesul la un număr mai mare de unități de acces ar putea fi menținute în așteptare un timp nedefinit, deoarece este puțin probabil ca resursele solicitate să se disponibilizeze toate în același moment. Aceste tranzacții au șanse mult mai mici de a fi lansate în execuție, față de tranzacțiile care solicită mai puține resurse.
- există situații când această tehnică nu este aplicabilă. Este posibil ca pentru o tranzacție care blochează două unități de acces să nu se poată preciza de la început care sunt acestea. Identificarea celei de-a doua unități de acces poate să depindă de anumite valori din prima unitate de acces și deci blocarea ei nu se poate face decât după ce s-a accesat prima unitate.

6.4.2 Metoda ordonării

Metoda ordonării constă în stabilirea unei relații de ordine peste mulțimea unităților de acces. Tranzacțiile pot bloca unitățile de acces numai în această ordine prestabilită.

Fie U_1, U_2, \dots, U_n unitățile de acces a căror ordonare este dată prin valoarea indicilor asociați. Presupunem că fiecare tranzacție blochează unitățile de acces în ordinea crescătoare a indicilor. Dacă o tranzacție T_x a blocat o unitate U_i , atunci T nu poate fi pusă în așteptare decât pentru o unitate U_j , cu $j > i$. Dar o altă tranzacție T_y care a blocat U_j nu poate fi în așteptare pentru U_i , deoarece $i < j$. Deci interblocarea nu este posibilă.

Dezavantaje:

- Afectează deasemenea nivelul de concurență al sistemului prin blocarea mai mult decât este necesar al unor unități de acces. Fie, de ex, o tranzacție care dorește accesul pentru o durată scurtă de timp la unitatea U_i și un timp mai lung la unitatea U_j . Dacă $i < j$, atunci unitatea U_i va trebui să fie blocată pe toată durata blocării lui U_j .
- impune restricții programatorilor în elaborarea tranzacțiilor. Cererile de acces la date trebuie să respecte ordinea impusă de sistem.

- În cazul BD complexe, realizarea unei ordonări a unităților de acces poate fi foarte dificilă (nu imposibilă), din cauza posibilităților foarte variate de divizare în unități de acces.
- Această metodă presupune existența apriorică a unei asemenea divizări, ceea ce exclude posibilitatea blocărilor

6.5 Marcarea timpului

Metodele de marcarea a timpului pentru controlul concurenței sunt destul de diferite de metodele de blocare. Nu este implicată nici o blocare, deci nu pot apărea situații de impas (interblocare).

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

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

Dacă o tranzacție încearcă să citească sau să scrie o dată, aceste operații sunt permise numai dacă ultima reactualizare a respectivei date a fost efectuată de o tranzacție mai veche. Altfel, tranzacția care necesită operația de citire/scriere este reîncepută și i se atribuie o nouă marcă de timp. Este necesar ca tranzacțiilor reîncepute să li se atribuie noi mărci de timp, pentru a evita să fie încontinuu abandonate și reîncepute.

Regula de scriere a lui Thomas

Pentru a obține o concurență cât mai mare prin respingerea operațiilor scoase din uz, se poate utiliza o modificare a protocolului de ordonare a mărcilor de timp:

- Dacă tranzacția **T** încearcă să scrie un articol **x** a cărui valoare a fost deja **citită** de către o tranzacție mai nouă, tranzacția **T** trebuie rulată înapoi și reîncepută cu o nouă marcă de timp;
- Dacă tranzacția **T** încearcă să scrie un articol **x** a cărui valoare a fost deja **scrisă** de către o tranzacție mai nouă, operația de scriere poate fi ignorată, pentru că valoarea pe care tranzacția **T** vrea să o scrie se bazează pe o valoare veche a articolului **x**, valoare scoasă deja din uz.

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

$$M_{T1} < M_{T2} < M_{T3}$$

La momentul t_8 , operația de scriere a tranzacției T2 violează regula de scriere a mărcilor de timp și drept urmare este abandonată și reîncepută la momentul t_{14} .

La momentul t_{14} , operația de scriere a tranzacției T1 poate fi ignorată utilizând regula de ignorare a scrierilor scoase din uz, deoarece ar fi fost suprascrisă prin operația de scriere a tranzacției T3 de la momentul t_{12} .

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	

6.6 Refacerea bazei de date

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

Cauzele penelor:

- căderile sistemului (hard sau soft)
- pene de mediu (distrugerea mediului de depozitare a datelor)
- erorile soft de aplicație
- dezastre naturale
- neglijența
- 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 și tehnici prin care se pot minimiza aceste efecte și se poate permite refacerea în urma unei pene.

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

6.6.1 Tehnici de refacere

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

- 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 siguranță și să se aplice din nou operațiile de reactualizare a tranzacțiilor efectuate, folosind fișierul **jurnal**¹. Se recomandă ca fișierul jurnal să fie stocat pe un disc separat, pentru a reduce riscul deteriorării lui simultan cu deteriorarea bazei de date;

¹ Amintim că **Securitatea datelor** cere ca baza de date să fie protejată împotriva unei **distrugerii 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 tranzacții** = lista operațiilor realizate asupra bazei de date după ultimul punct de repriză

- dacă baza de date nu a fost deteriorată fizic, ci a devenit **incoerentă** datorită unei căderi a sistemului în timpul execuției unei tranzacții, nu este necesară utilizarea copieii de siguranță, ci poate fi restaurată utilizând imaginile anterioare și ulterioare conținute în fișierul jurnal.

Vom prezenta trei tehnici de refacere, cunoscute sub denumirile de *reactualizare amânată*, *reactualizare imediată* și *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 tranzacția a ajuns pe punctul de a fi efectuată. Dacă tranzacția eșuează înainte de a ajunge în acest punct, ea nu va fi modificat baza de date și astfel nu mai sunt necesare anulări sau modificări. S-ar putea să fie nevoie doar să se reia reactualizările tranzacțiilor efectuate, deoarece este posibil ca efectul acestora să nu fi ajuns la baza de date. Pentru aceasta se folosește jurnalul de tranzacții.

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 aștepta ca tranzacția să ajungă pe punctul de a fi efectuată. Pe lângă reluarea reactualizărilor efectuate, acum ar putea să fie necesar să se anuleze efectele tranzacțiilor care nu au fost efectuate în momentul în care a survenit pana. Pentru aceasta se folosește jurnalul de tranzacții.

Este esențial ca înregistrările din jurnal să fie scrise înainte descrierea corespunzătoare în baza de date. Această operație 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 tranzacții. În cadrul acestei tehnici, în timpul unei tranzacții se păstrează două tabele ale paginii: unul pentru pagina curentă, unul pentru pagina din umbră. La începutul tranzacției, cele două pagini sunt identice. Pagina din umbră nu se modifică niciodată pe parcursul tranzacției, în timp ce pagina curentă reflectă toate reactualizările făcute pe baza de date de către tranzacția respectivă. La încheierea tranzacției, tabelul paginii curente devine tabelul paginii din umbră.

Avantaje:

- eliminarea suprasarcinii datorată întreținerii fișierului jurnal
- refacerea mai rapidă (pentru că nu sunt necesare operații de anulare sau reluare)

Dezavantaje:

- fragmentarea datelor
- necesitatea unei colectări periodice a „gunoaielor”.