CAPITOLUL 6

TRANZACŢII ŞI ACCES CONCURENT

Asa cum s-a exemplificat in primul capitol, atunci cand mai multe programe opereaza simultan pe aceleasi date pot sa apara situatii in care continutul bazei de date devine inconsistent: daca pasii aceluiasi program de rezervare de locuri rulat de la doua agentii de voiaj diferite sunt ca in tabelul de mai jos, desi se rezerva doua locuri numarul de locuri disponibile scade cu doar o unitate

Moment de timp	Agentia 1	Agentia 2	A în BD
t1	READ A		10
t2		READ A	10
t3	A = A – 1		10
t4		A = A - 1	10
t5	WRITE A		9
t6		WRITE A	9

In cazul accesului la aceleasi date, ca mai sus, se spune ca executiile celor doua programe sunt ***concurente*** sau ca exista un ***acces concurent la date***. Scopul acestui capitol este de a studia modalitatile de evitare a inconsistentelor precum si a problemelor ridicate de mecanismele folosite pentru aceasta.

6.1. Terminologie folosita

In acest prim paragraf sunt definiti principalii termeni folositi ca punct de plecare in studierea tranzactiilor si a accesului concurent la date. Pe parcurs vor fi introduse si alte notiuni care deriva din acestea:

a. Tranzactie

Notiunea de tranzactie va fi rafinata in paragrafele urmatoare. Definitia urmatoare este doar una de lucru pentru intelegerea celorlalti termeni din acest paragraf.

Definitie: O tranzactie este o singura executie a unui program.

In exemplul anterior exista doua tranzactii, T1 si T2 care erau doua executii ale aceluiasi program: T1 executia programului de rezervare lansata de Agentia 1 iar T2 este cea de la Agentia 2. Nu inseamna insa ca un set de tranzactii care opereaza simultan pe o baza de

date trebuie sa contina doar executii ale aceluiasi program. Putem avea de exemplu o tranzactie care rezerva un loc si o alta care anuleaza o rezervare de loc, ca in exemplul urmator:

Moment de timp	Tranzactia 1 lansata de Agentia 1: rezervare loc	Tranzactia 2 lansata de Agentia 2: anulare rezervare	A în BD
t1	READ A		10
t2		READ A	10
t3	A = A – 1		10
t4		A = A +1	10
t5	WRITE A		9
t6		WRITE A	11

Si in acest caz rezultatul este o inconsistenta a bazei de date deoarece numarul de locuri disponibile trebuia sa ramana constant.
Pe o aceeasi baza de date pot rula simultan mai multe tranzactii care lucreaza fiecare nu cu un singur element din baza de date (A din exemplul anterior) ci cu mai multe: fiecare tranzactie poate citi mai multe date si poate sa si scrie mai multe date in baza de date (nu neaparat cele citite ci si altele).
Operatiile efectuate de tranzactii care nu sunt de scriere sau de citire de date din baza de date nu duc la inconsistente: de exemplu nu incrementarea sau decrementarea lui A din exemplul anterior sunt cauza inconsistentelor ci ordinea in care s-au facut scrierile rezultatelor in baza de date. De aceea in unele dintre exemplele din acest capitol nu vom mai figura acest tip de operatii. Iata o executie concurenta pentru patru tranzactii:

Timp	T1	T2	T3	T4
t1	READ A			
t2	READ B			
t3		READ A		
t4		READ B		
t5			WRITE B	
t6		WRITE B		
t7				READ B
t8				READ C
T9	WRITE A			
t10		WRITE C		

Se observa ca putem avea:

- tranzactii scriu date care anterior au fost citite (ca T1 il scrie pe A, citit anterior),
- tranzactii care scriu date care nu au fost anterior citite, calculate eventual pe baza altora citite din baza de date (T2 scrie pe C pe care nu l-a citit, dar a citit A si B)
- tranzactii care doar citesc date fara sa scrie (T4)
- tranzactii care doar scriu date fara sa citeasca anterior ceva din baza de date (T3)

b. Articol (al unei baze de date)

Definitie: Un articol este o portiune a bazei de date care se poate citi sau scrie sau bloca/debloca printr-o singura operatie de READ, WRITE, LOCK respectiv UNLOCK.

In exemplul anterior am folosit articolele simbolice A, B si C. In cazurile reale un articol poate fi:

- O intreaga tabela
- O linie (sau o multime de linii) dintr-o tabela
- O celula dintr-o tabela (valoarea unei coloane de pe o linie a tabelei)
- Orice alta portiune a bazei de date care indeplineste conditia din definitie in concordanta cu facilitatile puse la dispozitie de SGBD-ul respectiv.

Exemplu: sistemul Oracle blocheaza automat orice linie modificata de o comanda de tip UPDATE pana cand tranzactia care a efectuat operatie fie ***comite*** modificarile (le face permanente in baza de date) fie le ***revoca***. In acest caz articolele sunt deci linii ale tabelei actualizate.

c. Planificare

Definitie: O planificare reprezinta ordinea in care sunt executati de SGBD pasii elementari ai unui set de tranzactii.

Planificarea este deci o lista de pasi pentru un set de tranzactii care se executa concurent si arata ca SGBD-ul executa acesti pasi in exact acea ordine.
In acest capitol vom reprezenta doar pasii care semnifica o interactiune a tranzactiei cu datele din baza de date:

- READ – citirea unui articol
- WRITE – scrierea unui articol
- LOCK (in diversele sale forme) – blocarea unui articol
- UNLOCK – deblocarea unui articol

Pentru cazurile in care operatiile de scriere nu devin permanente in baza de date decat dupa comitere putem avea si pasi de tipurile urmatoare:

- COMMIT – comiterea modificarilor efectuate de o tranzactie
- ROLLBACK – revocarea modificarilor efectuate de o tranzactie

Asa cum s-a mentionat, celelalte operatii nu ridica probleme de acces concurent. Exista mai multe moduri de a reprezenta o planificare:

- Sub forma tabelara, ca in exemplele anterioare de executie concurenta. Coloana "Timp" poate lipsi, ordinea executiei este de sus in jos.
- Sub forma unei liste.

Pentru a doua forma sa consideram urmatoarele notatii:

- $R_i(A)$ - semnifica: Tranzactia T_i citeste articolul A
- $W_i(A)$ - semnifica: Tranzactia T_i scrie articolul A

In acest caz ultima planificare (pentru T1, T2 si T3) se mai poate scrie si astfel:

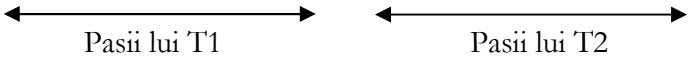
$R_1(A); R_1(B); R_2(A); R_2(B); W_3(B); W_2(B); R_4(B); R_4(C); W_1(A); W_1(C)$

d. Planificare seriala

Definitie: O planificare in care pasii fiecarei tranzactii sunt succesivi, fara sa fie intercalati pasi ai altor tranzactii se numeste planificare seriala.

Exemplu de planificare seriala pentru doua tranzactii T1 si T2:

$R_1(A); R_1(B); R_1(C); W_1(B); R_2(B); R_2(C); W_2(A); W_2(C)$



Planificarile seriale nu ridica probleme de consistenta (sunt planificari "bune" din punct de vedere al executiei concurente). De aceea unul din obiectivele acestui capitol este acela de a gasi planificari care sa se comporte la fel cu o planificare seriala.

e. Blocarea articolelor

Cum s-a mentionat si in primul capitol, una dintre metodele de a obtine planificari care sa nu ridice probleme privind consistenta datelor dupa executia tranzactiilor este aceea a blocarii articolelor.

Definitie: Blocarea unui articol de catre o tranzactie semnifica faptul ca acea tranzactie obtine din partea sistemului (SGBD) anumite drepturi speciale de acces care impiedica alte tranzactii sa efectueze anumite operatii asupra acelui articol.

Exista doua categorii de blocari:

- Blocari exclusive: celelalte tranzactii nu pot sa execute operatii asupra articolului blocat. Aceste blocari sunt denumite in literatura de specialitate si Exclusive Locks sau Write Locks.
- Blocari partajate: celelalte tranzactii pot sa execute doar anumite tipuri de operatii asupra articolului blocat. Aceste blocari sunt denumite in literatura de specialitate si Shared Locks sau Read Locks

Exemplu:

- Pentru a scrie un articol, o tranzactie trebuie sa obtina anterior un Write Lock asupra acestuia: nici o alta tranzactie nu poate citi sau scrie acel articol pana cand el nu este deblocat
- Pentru a citi un articol o tranzactie trebuie sa obtina anterior un Read Lock asupra lui. Mai multe tranzactii pot sa blocheze acelasi articol pentru citire insa nici o tranzactie nu il poate scrie. O tranzactie poate sa obtina un Write Lock pe articolul respectiv abia dupa deblocarea acestuia de catre ***toate*** tranzactiile care l-au blocat pentru citire.

Articolele pot fi deblocate unul cate unul de catre tranzactia care le-a blocat sau pot fi toate deblocate in cazul unui COMMIT sau unui ROLLBACK, in functie de modelul de blocare folosit.

6.2. Gestiunea tranzactiilor

In paragraful anterior tranzactia era definita ca o executie a unui program. In fapt un program care interactioneaza cu o baza de date contine de obicei nu o singura tranzactie ci o succesiune de tranzactii care nu se intersecteaza, fiecare dintre ele fiind finalizata fie prin comiterea modificarilor efectuate (ele devin definitive in baza de date) fie prin revocarea lor (modificarile sunt anulate). Dupa terminarea unei tranzactii celelalte operatii asupra bazei de date apartin tranzactiilor urmatoare.

Cum singurele operatii care sunt importante din punct de vedere al interactiunii dintre program si sistemul de gestiune sunt cele de citire/scriere si cele conexe (blocare, deblocare, comitere si revocare) putem defini o tranzactie si ca o succesiune de operatii de acest tip.

ACID

Pentru ca o tranzactie sa fie bine definita ea trebuie sa indeplineasca niste criterii de corectitudine care au fost sintetizate prin abrevierea ACID. Aceasta semnifica

- A – Atomicitate
- C – Consistenta
- I – Izolare
- D – Durabilitate.

Atomicitate

O tranzactie trebuie sa fie atomica in sensul ca fie toate modificarile efectuate de ea in baza de date sunt comise fie sunt toate revocate.

Exemplu: Sa luam o succesiune de actualizari in SQL care insereaza, actualizeaza si sterg linii din doua tabele STUD si SPEC:

```
INSERT INTO STUD …
UPDATE SPEC …
DELETE FROM STUD …
```

Aceste modificari trebuie ori comise ori revocate impreuna. Faptul ca de exemplu doar inserarea si stergerea sunt comise si nu si actualizarea este o incalcare a acestei reguli.
Sistemul de gestiune este cel care trebuie sa puna la dispozitie mecanismele prin care sa se asigura atomicitatea tranzactiilor inclusiv in cazul unor incidente hardware si software care pot interveni in timpul executiei unei tranzactii.

Consistenta

O tranzactie care incepe sa lucreze pe o baza de date consistenta trebuie sa o lase la final tot intr-o stare consistenta.

In acest sens o tranzactie nu poate incalca restrictiile existente la nivelul bazei de date. In cele mai multe cazuri aceste restrictii sunt modelate sub forma constrangerilor de integritate (NOT NULL, PRIMARY KEY, etc).
Daca o tranzactie contine o operatie care violeaza o constrangere de integritate atunci toate modificarile efectuate de tranzactie vor fi revocate. Mecanismele de pastrare a consistentei trebuie asigurate de SGBD.

Izolare

O tranzactie trebuie sa se comporte ca si cand operatiile efectuate de ea sunt izolate, independente de operatiile efectuate de alta tranzactie.
Nici o alta tranzactie nu trebuie sa citeasca date intermediare scrise de tranzactia respectiva. De exemplu daca o tranzactie contine succesiunea de operatii:

```
READ (A)   -- citeste valoarea veche a lui A: 5
A = A + 1
WRITE (A)  -- scrie valoarea noua a lui A:6
```

```
READ (B)   -- citeste valoarea veche a lui B:10
B = B - 1
WRITE (B)  -- scrie valoarea noua a lui B:9
```

Atunci nici o alta tranzactie nu poate citi pentru A si B doua valori dintre care una este actualizata si cealalta nu (adica 6 pentru A si 10 pentru B).
Inconsistentele prezentate in paragraful anterior erau datorate incalcarii acestui criteriu de corectitudine. Din punct de vedere al modului de asigurare a izolarii tranzactiilor tot sistemul de gestiune trebuie sa asigure mecanismele necesare.

Izolarea este obiectul controlului accesului concurent, prezentat in paragrafele urmatoare.

Durabilitate

O data comise cu succes modificarile efectuate de catre o tranzactie ele vor persista si nu mai pot fi revocate. Inclusiv in cazul unui incident hardware si software efectele tranzactiilor comise sunt regasite la recuperarea dupa incident. Din acest punct de vedere fiecare sistem de gestiune trebuie sa contina mecanisme prin care efectele tuturor traanzactiilor comise sa fie inregistrate si in jurnalele sistemului pentru a fi restaurate in caz de incident.

6.3. Serializabilitate

Asa cum s-a specificat planificarile seriale nu duc la inconsistente. In practica insa in cazul unor sisteme incarcate planificarile contin pasi intercalati ai diverselor tranzactii. Rezultatul va fi totusi corect daca efectul executiei planificarii respective este acelasi cu al uneia dintre planificarile seriale posibile ale acelorasi tranzactii. O astfel de planificare se numeste planificare serializabila.

6.3.1. Planificari serializabile

Definitie: O planificare este serializabila daca produce aceleasi efecte in baza de date cu o planificare seriala.

Exemplu: O planificare serializabila si planificarea seriala echivalenta:

Planificare serializabila:

T1	T2	T3
READ A		
READ B		
		READ C
		READ D
WRITE A		
		WRITE C
	READ A	
	WRITE A	

Planificare seriala echivalenta (se putea si T1, T2, T3):

T1	T2	T3
		READ C
		READ D
		WRITE C
READ A		
READ B		
WRITE A		
	READ A	
	WRITE A	

Exemplul 2: Planificare neserializabila si cele doua planificari seriale echivalente: Sa luam exemplul anterior cu doua tranzactii care rezerva si anuleaza o rezervare:

T1	T2	A în BD
READ A		10
	READ A	10
A = A + 1		10
	A = A +1	10
WRITE A		11
	WRITE A	11

Planificare seriala 1:

T1	T2	A în BD
READ A		10
A = A + 1		10
WRITE A		11
	READ A	11
	A = A +1	11
	WRITE A	12

Planificare seriala 2:

T1	T2	A în BD
	READ A	10
	A = A +1	10
	WRITE A	11
READ A		11
A = A – 1		11
WRITE A		12

Se observa ca exista doar doua planificari seriale diferite pentru T1 si T2 si nici una nu produce acelasi rezultat cu planificarea initiala.

6.3.2. Planificari conflict-serializabile

Exista si o alta abordare a serializabilitatii bazata pe conflictele care pot sa apara intre pasii a doua tranzactii dintr-o planificare.

Definitie: Intre doua operatii apartinand unei planificari exista un conflict daca:

- Apartin unor tranzactii diferite
- Sunt pe acelasi obiect
- Una dintre operatii este o scriere
- Cele doua operatii sunt succesive in sensul ca intre ele nu exista o operatie cu care vreuna dintre ele este in conflict.

Rezulta ca exista 3 tipuri de situatii conflictuale: Fiind date doua tranzactii T1 si T2 pot exista conflicte de tipurile R1-W2, W1-W2 si W1-R2.

Definitie: Doua planificari sunt conflict-echivalente daca:

- Contin aceleasi operatii ale acelorasi tranzactii
- Fiecare pereche de operatii conflictuale apare in aceeasi ordine in cele doua planificari.

Aceasta definitie nu spune ca nu pot sa apara anomalii in executia celor doua planificari ci ca apar aceleasi anomalii in ambele.

Definitie: O planificare este conflict-serializabila daca este conflict-echivalenta cu o planificare serial.
Observatie: Reformuland putem spune ca o planificare este conflict-serializabila daca poate fi transformata intr-o planificare seriala prin interschimbari ale operatiilor consecutive care nu sunt in conflict din doua tranzactii.

Exemplu:

1. T1	T2
Read A	
Write A	
	Read A
Read B	
	Write A
Write B	
	Read B
	Write B

3. T1	T2
Read A	
Write A	
Read B	
	Read A
Write B	
	Write A
	Read B
	Write B

2. T1	T2
Read A	
Write A	
Read B	
	Read A
	Write A
Write B	
	Read B
	Write B

4. T1	T2
Read A	
Write A	
Read B	
Write B	
	Read A
	Write A
	Read B
	Write B

Prin trei interschimbari de operatii neconflictuale s-a obtinut o planificare seriala.

Test de conflict-serializabilitate:

- Se construieste graful de dependenta astfel:
  - Nodurile sunt tranzactii
  - Pentru orice pereche de operatii aflate in conflict Oi si Oj, cu Oi in Ti si Oj in Tj, avem un arc de la nodul Ti la Tj daca Oi apare in planificare inaintea lui Oj.
- Daca acest graf nu contine cicluri planificarea este conflict-serializabila altfel nu este conflict-serializabila

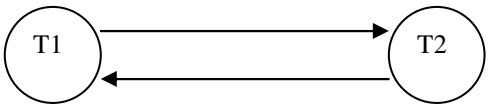
Exemplu:

T1	T2
READ A	
	WRITE A
WRITE A	

Avem 2 conflicte:

- T1-READ A cu T2-WRITE A
- T2-WRITE A cu T1-WRITE A

Rezulta ca graful are doua noduri si doua arce:



Cum exista un ciclu planificarea nu este conflict-serializabila.

Observatie: exista planificari serializabile care nu sunt conflict-serializabile. De exemplu sa presupunem ca in planificare de mai sus tranzactia T2 scrie in A exact valoarea citita de T1. Planificarea nu e conflict-serializabila dar e serializabila, avand acelasi efect cu planificarea seriala "T2 urmata de T1":

T1	T2
	WRITE A
READ A	
WRITE A	

Acest fapt se datoreaza insa doar coincidentei intre valoarea scrisa de T2 si cea citita de T1. Cum sistemul de gestiune nu face astfel de judecati pentru el potential planificarea este periculoasa putand sa duca la inconsistente. De aceea in judecarea planificarilor se considera ca la o scriere o tranzactie poate scrie orice valoare si nu doar o valoare particulara.

6.3.2. Planificari v-serializabile (view-serializability)

Exista de asemenea o a treia abordare (mai slaba) a serializabilitatii:

Definitie: Doua planificari S1 si S2 sunt v-echivalente daca pentru orice articol A:

- Daca Ti citeste valoarea initiala a lui A in S1 atunci ea face acelasi lucru si in S2
- Daca Ti citeste o valoare a lui A scrisa de Tj in S1, atunci face acelasi lucru si in S2.
- Daca Ti scrie valoarea finala a lui A in S1 atunci ea face acelasi lucru si in S2

Definitie: O Planificare este v-serializabila daca este v-echivalenta cu o planificare seriala.

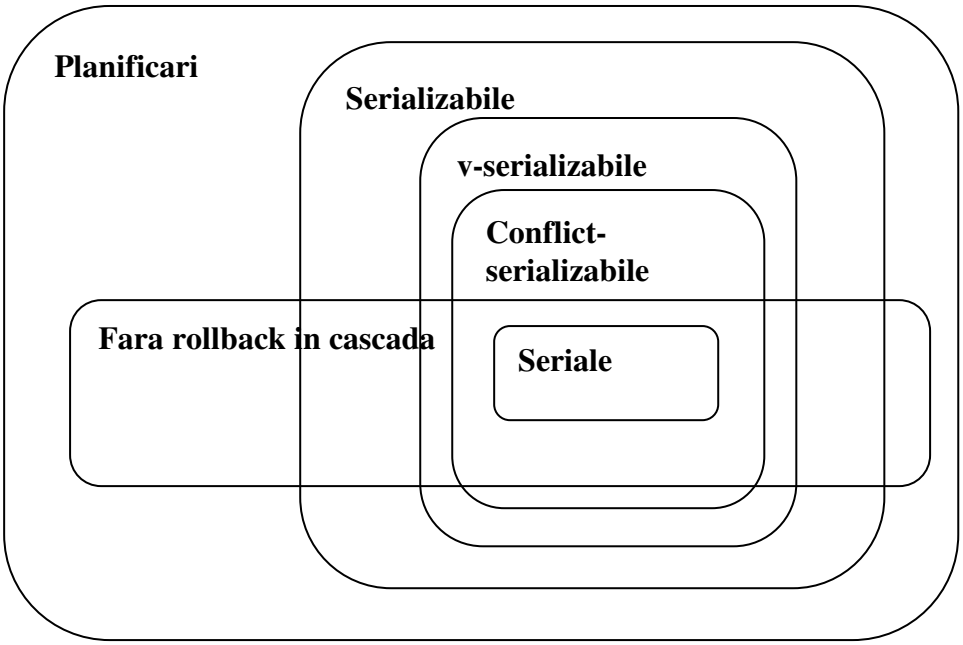
Exemplu: O planificare v-serializabila si planificarea seriala v-echivalenta:

T1	T2	T3
READ A		
	WRITE A	
WRITE A		
		WRITE A

T1	T2	T3
READ A		
WRITE A		
	WRITE A	
		WRITE A

Aceasta definitie permite planificarile de tranzactii conflict-serializabile si planificari care contin tranzactii care scriu date fara sa citeasca ceva din baza de date. Din acest punct de vedere planificarea de la punctul 6.3.2. nu e conflict-serializabila dar este v-serializabila.

Incluziunea intre diverse tipuri de planificari este urmatoarea:



6.4. Realizarea serializabilitatii prin blocari

Pentru a putea asigura serializabilitatea tranzactiilor sistemele de gestiune pun la dispozitie posibilitatea de blocare a articolelor. Daca o tranzactie blocheaza un articol, celelalte tranzactii care vor si ele sa aiba acces la acel articol pot fi puse in asteptare pana la deblocarea acestuia.
Exista mai multe modele de blocare, prezentate in continuare.

6.4.1. Modelul LOCK/UNLOCK

In cadrul acestui model exista o singura primitiva de blocare, LOCK, ea ducand la obtinerea unui acces exclusiv la articol pentru tranzactia care il blocheaza (celelalte tranzactii nu pot nici scrie nici citi articolul). Deblocarea se face cu UNLOCK.
Vom presupune in continuare ca o tranzactie

- nu blocheaza un articol deja blocat de ea
- nu deblocheaza un articol pe care nu l-a blocat.

In acest caz se poate realiza **testul de serializabilitate** astfel:

Se construieste **graful de precedenta** G astfel

- Nodurile sunt tranzactiile planificarii
- Daca pentru vreun articol (notat simbolic A) avem in S secventa
  Ti: UNLOCK A
  Tj: LOCK A
  atunci vom avea un arc in graf de la nodul Ti la nodul Tj
- Daca graful are cicluri atunci S nu e serializabila.
- Daca nu are cicluri e serializabila si planificarea seriala echivalenta se obtine prin sortarea topologica a grafului G

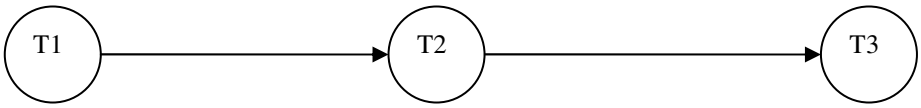
Sortarea topologica se face astfel:

- Se alege un nod care nu are arce care intra (neexistand cicluri exista cel putin un astfel de nod)
- Se listeaza tranzactia asociata nodului dupa care acesta este sters dion graf impreuna cu toate arcele care ies din el
- Procesul se reia

Exemplu:

T1	T2	T3
	LOCK A	
	UNLOCK A	
		LOCK A
		UNLOCK A
LOCK B		
UNLOCK B		
	LOCK B	
	UNLOCK B	

Graful este:



Nu are cicluri deci planificarea e serializabila. Planificarea seriala echivalenta este:

T1; T2; T3

Protocolul de blocare in doua faze

Definitie: O tranzactie respecta protocolul de blocare in doua faze daca toate blocarile preced toate deblocarile

Acest protocol ne garanteaza serializabilitatea: daca toate tranzactiile respecta cerintele protocolului se poate demonstra ca orice planificare a lor e serializabila.
De asemenea se poate demonstra ca daca o tranzactie nu respecta protocolul pot exista executii neserializabile ale acelei tranzactii in conjunctie cu alte tranzactii:

Pentru o tranzactie care contine secventa:

```
UNLOCK A
LOCK B
```

Putem avea o planificare care contine:

```
   T1                  T2
UNLOCK A
                    LOCK A
                    LOCK B
                    UNLOCK A
                    UNLOCK B
LOCK B
```

Care are un graf de precedenta care contine un ciclu.

Protocolul de blocare in 2 faze implica insa uneori operatii de roll-back in cascada:

T1	T2
LOCK A **LOCK B**	
READ A **WRITE A**	
UNLOCK A	
	LOCK A **READ A** **WRITE A** **UNLOCK A**
READ B **WRITE B**	
ROLLBACK	

In momentul Rollback pentru T1 este necesar Rollback si pentru T2 deoarece T2 a citit date scrise de T1, date care prin operatia de Rollback se pierd. O astfel de planificare se numeste planificare cu rollback in cascada (eng.: cascading aborts)

Exista pentru a evita si astfel de cazuri varianta **protocolului de blocare stricta in 2 faze** care implica eliberarea toturor articolelor blocate la sfarsitul tranzactiei. In acest caz tranzactia T2 din exemplul anterior porneste abia dupa terminarea complete a tranzactiei T1.

6.4.2. Modelul RLOCK/WLOCK/UNLOCK

In cadrul acestui model exista o doua primitive de blocare:

- RLOCK (blocare pentru citire). Oricate tranzactii pot bloca acelasi articol pentru citire dar o tranzactie nu poate bloca pentru scriere un articol blocat cu RLOCK
- WLOCK (blocare pentru citire). Duce la obtinerea unui acces exclusiv la articol pentru tranzactia care il blocheaza. Celelalte tranzactii nu mai pot sa blocheze cu RLOCK sau WLOCK acel articol.
- Deblocarea pentru ambele tipuri se face cu UNLOCK.

Vom presupune ca si anterior ca o tranzactie

- nu blocheaza un articol deja blocat de ea
- nu deblocheaza un articol pe care nu l-a blocat.

Si in acest caz se poate construi (altfel decat in paragraful anterior) un **graf de precedenta** din care se poate deduce daca planificarea e serializabila sau nu.

Observatie importanta: Si in acest caz protocolul de blocare in doua faze este valabil: Daca toate blocarile (de orice fel, la citire sau la scriere) preced toate deblocarile, planificarea este serializabila.

Test de serializabilitate pentru modelul cu blocari pentru citire si scriere

Intrare: o planificare P a mulţimii de tranzacţii T1, T2, ..., Tk.
Ieşirea: raspunsul daca planificarea P este serializabilă, si daca da planificarea serială echivalentă.
Metoda: construirea grafului de precedenta G, similar cu modelul anterior: fiecarei tranzactii ii corespunde un nod al grafului iar arcele se traseaza astfel:

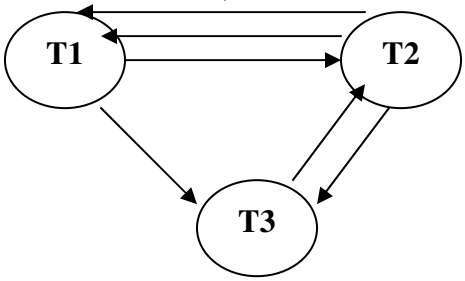
1. Fie Ti tranzactia care executa RLOCK A iar Tj urmatoarea tranzactie (diferita de Ti) care face WLOCK A. Trasam atunci un arc de la Ti la Tj.
2. Fie Ti tranzactia care face WLOCK A si Tj urmatoarea tranzactie (daca exista) care face WLOCK A. Se traseaza un arc de la Ti la Tj. De asemenea, in acest ultim caz fie Tm tranzactia care face RLOCK A dupa ce Ti elibereaza pe A dar inainte de blocarea acestuia de catre Tj. Se traseaza un arc de la Ti la Tm. Nota: Daca in cazul 2 Tj nu exista atunci Tm este urmatoarea tranzactie care face RLOCK A dupa eliberarea lui A de catre Ti.

Rezultat: Daca graful obtinut contine cicluri, P nu este serializabila.
Daca insa nu contine cicluri atunci P este serializabila iar planificarea seriala echivalenta se obtine prin sortarea topologica a grafului (ca in cazul modelului anterior, cu blocare simpla).

Exemplu:

T1	T2	T3
	WLOCK A	
		RLOCK B
1	UNLOCK A	
	5	UNLOCK B
	WLOCK B	
RLOCK A		6
	UNLOCK B	
UNLOCK A	4	
2		WLOCK A
WLOCK B		
3		UNLOCK A
UNLOCK B		
	RLOCK B	
	UNLOCK B	

- Graful este cel de mai jos. Cum are cicluri, planificarea nu este serializabila.



Arcele s-au trasat pe baza regulilor de mai sus :

- Prima regula a generat arcele de tip R-W: 4, 5.
- Prima parte a celei de-a doua reguli a generat arcele de tip W-W: 2, 6.

- A doua parte a celei de-a doua reguli a generat arcele de tip W-R: 1, 3. Arcul 3 e generat luand in considerare nota de la a doua regula.

6.4.3. Blocarea articolelor structurate ierarhic

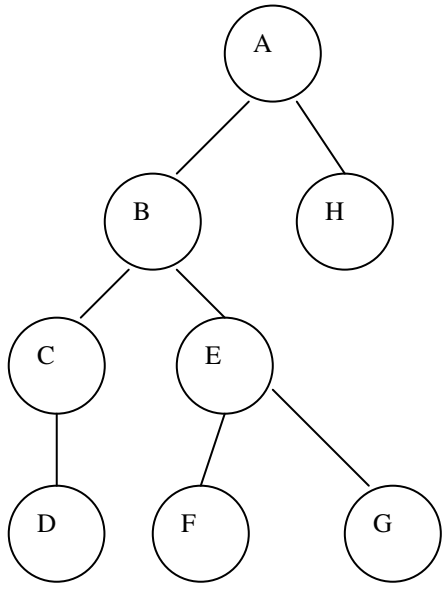
Exista cazuri in care articolele unei baze de date se pot reprezenta ca nodurile unui arbore.
In acest caz exista un protocol care garanteaza serializabilitatea numit protocol de arbore. Acesta este urmatorul:

- Cu exceptia primului articol blocat, nici un articol nu poate fi blocat decat daca tatal sau este deja blocat
- Nici un articol nu este blocat de doua ori de aceeasi tranzactie.

Exemplu:

Urmatoarea tranzactie respecta acest protocol, pentru structurarea ierarhica prezentata in continuare:

LOCK B – primul articol blocat
LOCK C – blocarea lui C este necesara pentru a realiza blocarea lui D
LOCK D
UNLOCK C – C poate fi acum deblocat
LOCK E – blocarea lui E este necesara pentru a realiza blocarea lui F
LOCK F
UNLOCK E – E poate fi acum deblocat
UNLOCK D
UNLOCK F
UNLOCK B



6.5. Controlul concurentei prin etichete timp

Se poate efectua un control al corectitudinii executiei concurente a mai multor tranzactii

- Fara mecanisme de blocare.
- Utilizand etichete-timp (timestamps)

In acest caz:

1. Fiecare articol are asociate doua etichete-timp: una de citire iar cealalta de scriere.
2. Fiecare tranzactie are asociata o eticheta timp (de exemplu: momentul lansarii tranzactiei)
3. Cand o tranzactie citeste sau scrie un articol, se actualizeaza eticheta-timp corespunzatoare a articolului respectiv la valoarea etichetei-timp a tranzactiei.

O tranzactie sesizeaza incalcarea ordinii seriale (in care caz ea trebuie abortata si relansata ulterior) in urmatoarele doua cazuri:

1. O tranzactie incearca sa citeasca un articol scris in viitor
2. O tranzactie vrea sa scrie un articol citit in viitor.

Urmatoarele operatii sunt insa valide:

1. O tranzactie incearca sa citeasca un articol citit in viitor
2. O tranzactie incearca sa scrie un articol scris in viitor (in acest caz ea nu scrie articolul dar isi continua executia).