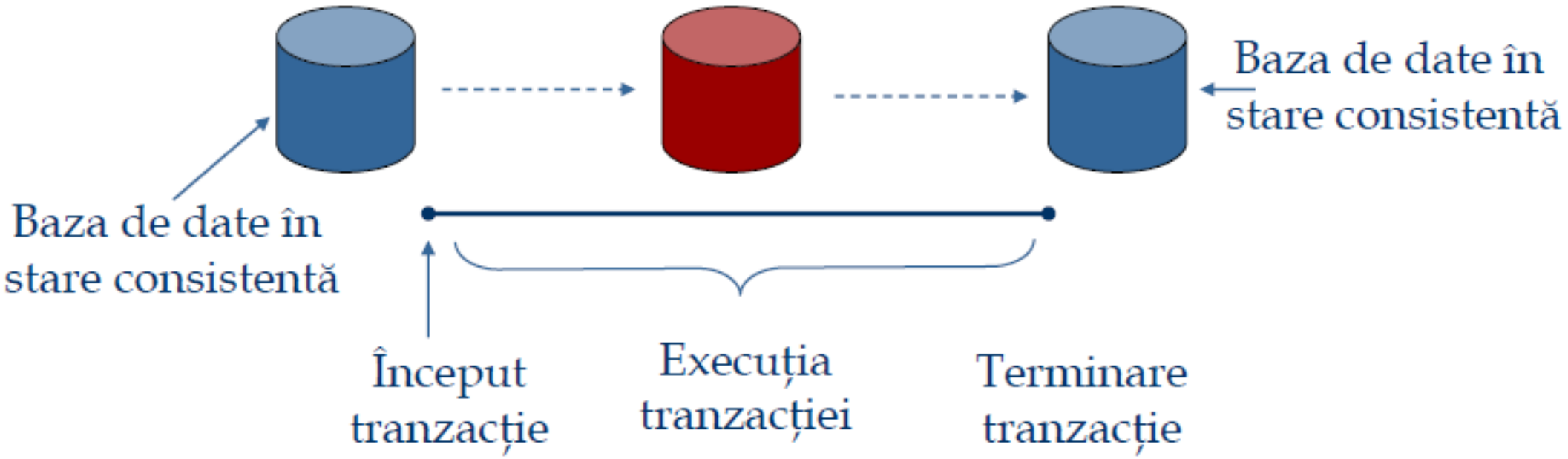
Tranzacții
Controlul concurenței

Curs 5

Tranzacții


Baza de date în stare consistentă
Baza de date în stare consistentă
Început tranzacție
Execuția tranzacției
Terminare tranzacție

Tranzacții

- Execuția concurentă este esențială pentru performanța unui SGBD

- Deoarece harddisk-ul este accesat frecvent, iar accesul este relativ lent, este preferabil ca CPU-ul să fie ocupat cu alte task-uri executate concurent

Tranzacții

- Tranzacție – o secvență de operații de citire / scriere care se execută pe o bază de date (care poate fi accesată simultan de mai mulți utilizatori); secvența de operații este tratată ca o unitate logică de lucru

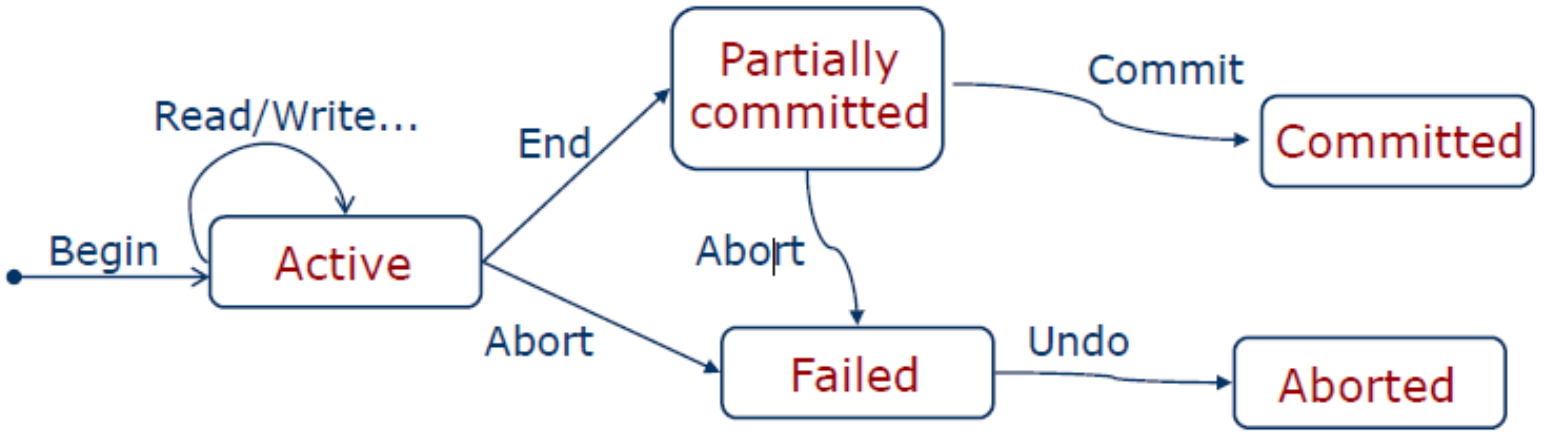
```
BEGIN TRAN
--op care preia bani din contul A
--op care depune bani în contul B
COMMIT / ROLLBACK TRAN
```

Tranzacții

- Begin –transaction
- Read
- Write
- End –transaction
- Commit –transaction
- Abort –transaction
- Undo
- Redo

Stările tranzacţiilor

- Active: tranzacţia este în execuţie
- Partially Committed: tranzacţia urmează să se finalizeze
- Committed: terminare cu succes
- Failed: execuţia normală a tranzacţiei nu mai poate continua
- Aborted: terminare cu *roll back*



Concurenţa într-un SGBD

- Un utilizator transmite unui SGBD mai multe tranzacţii spre execuţie
- Concurenţa este implimentată de SGBD prin intercalarea operaţiilor mai multor tranzacţii (citiri/modificări ale obiectelor bazei de date)
- Fiecare tranzacţie trebuie să lase baza de date într-o stare consistentă
- Constrângeri de integritate (intră în responsabilitatea SGBD)
- SGBD nu “înţelege” semantica datelor (intră în responsabilitatea programatorului).

- ***Probleme****:* Efectul *intercalării* tranzacţiilor şi *blocări*

Proprietățile unei tranzacții: ACID

- **A**tomicitate (totul sau nimic)
- **C**onsistență (garantare constrângeri de integritate)
- **I**zolare (concurența este invizibilă, serializabilitate)
- **D**urabilitate (acțiunile tranzacțiilor executate persistă)

Atomicitate

- Se execută toate operațiile din cadrul tranzacției sau nu se execută niciuna
- O tranzacţie se poate termina cu succes, după execuţia tuturor acţiunilor sale, sau poate eşua (uneori forţat de SGBD) după execuţia anumitor acţiuni.

- Utilizatorii (programatorii) pot privi o tranzacţie ca o operaţie indivizibilă.
  - SGBD salvează in ***loguri*** toate acţiunile unei tranzacţii pentru a le putea anula la nevoie.

- Acţiunea prin care se asigură atomicitatea tranzacţiilor la apariţia unor erori poartă numele de recuperarea datelor (***crash recovery***)

Consistență

- O tranzacție preia datele într-o stare consistentă și le lasă într-o stare consistentă, indiferent dacă se termină cu succes (comisă) sau se anulează

- Tranzacţiile păstrează constrângerile de integritate ale bazelor de date.

- Tranzacţiile sunt programe *corecte*

Izolare

- Tranzacțiile sunt protejate de efectele programării concurente a altor tranzacții (vezi anomalii de interferență)

- O tranzacţie nu poate partaja modificările operate până nu este finalizată
  - Condiţie necesară pentru evitarea eşecurilor in cascadă.

- Dacă mai multe tranzacţii sunt executate concurent, rezultatul trebuie sa fie identic cu una dintre execuţiile seriale a acestora (indiferent de ordine) – *serializabilitate*.

Durabilitate

- Dacă o tranzacție este comisă, se garantează că modificările operate vor fi stocate permanent în baza de date (chiar și în cazul unei pene de curent de exemplu)

- Recuperarea datelor

Tranzacții – Exemplu

T1: BEGIN A=A+100, B=B-100 END

T2: BEGIN A=1.06*A, B=1.06*B END

- Prima tranzacţie transferă 100€ din contul B în contul A.
- Cea de-a doua tranzacţie adaugă o dobandă de 6% sumelor din ambele conturi.

Tranzacții – Exemplu

- O posibilă intercalarea operaţiilor (*plan*):

  T1: A=A+100, B=B-100

  T2: A=1.06*A, B=1.06*B

  Este OK (echivalent cu planul serial T1, T2).

- O a doua variantă:

  T1: A=A+100, B=B-100

  T2: A=1.06*A, B=1.06*B

- Cum “vede” SGBD al doileaplan:

  T1: R(A), W(A), R(B), W(B)

  T2: R(A), W(A), R(B), W(B)

Operaţii conflictuale

- Daca două tranzacţii citesc date, acestea nu intră în conflict şi ordinea de execuţie nu este importantă
- Dacă două tranzacţii citesc şi modifică date complet separate, acestea nu intră în conflict şi ordinea de execuţie nu contează.

- Dacă o tranzacţie modifică o dată iar o altă tranzacţie citeşte sau modifică aceeaşi dată, ordinea de execuţie este importantă.
- Conflict WR : T2 citeşte date modificate anterior de T1
- Conflict RW: T2 modifică date citite anterior de T1
- Conflict WW: T2 modifică date modificate anterior de T1

Anomalii ale execuţiei concurente

- *Reading Uncommitted Data* (conflict WR, "*dirty reads*"):
  - T1: R(A), W(A), R(B), W(B), **A**
  - T2: R(A), W(A), **C**
- *Unrepeatable Reads*(conflict RW):
  - T1: R(A), R(A), W(A), **C**
  - T2: R(A), W(A), **C**
- *Overwriting Uncommitted Data*(Conflict WW, "*blind writes*"):
  - T1:W(A), W(B), **C**
  - T2: W(A), W(B), **C**

Anomalii

- dirty reads
- non-repeatable reads
- lost updates
- phantoms
- deadlocks

Exemplu

- Filme **(CodFilm, Titlu, Regizor, An, Nominalizari)**

Dirty reads

- o tranzacție citește date care au fost modificate de altă tranzacție, dar nu au fost comise

T1	T2
UPDATE Filme SET Regizor = 'Spielberg' WHERE CodFilm = 3	
	SELECT Regizor FROM Filme WHERE CodFilm = 3
Abort	
	Commit

Non-repeatable reads

- o tranzacție citește de două ori aceeași dată, însă între aceste două operații o altă tranzacție operează modificări pe data respectivă

T1

UPDATE Filme

SET Regizor = 'Spielberg'

WHERE CodFilm = 3

Commit

T2

SELECT Regizor

FROM Filme

WHERE CodFilm = 3

SELECT Regizor

FROM Filme

WHERE CodFilm = 3

Commit

Lost updates

- două tranzacții modifică aceeași dată; posibile probleme suprascrieri

T1

UPDATE Filme
SET Nominalizari = Nominalizari + 5
WHERE CodFilm = 3
Commit

T2

UPDATE Filme
SET Nominalizari = Nominalizari + 2
WHERE CodFilm = 3

Commit

Phantoms

- o tranzacție citește de două ori un interval de rânduri, însă între aceste două operații o altă tranzacție introduce rânduri în intervalul respectiv

T1	T2
	SELECT *
	FROM Filme
	WHERE CodFilm BETWEEN 1 AND 5
INSERT Filme (CodFilm, ...)	
VALUES (2, ...)	
Commit	
	SELECT *
	FROM Filme
	WHERE CodFilm BETWEEN 1 AND 5
	Commit

Deadlock

- cererea unei tranzacții pentru blocarea unei resurse e blocată de cealaltă tranzacție și reciproc

T1

```
UPDATE Filme
SET Nominalizari = Nominalizari + 5
WHERE CodFilm = 3
```

```
UPDATE Filme
SET Nominalizari = Nominalizari + 5
WHERE CodFilm = 5
```

T2

```
UPDATE Filme
SET Nominalizari = Nominalizari + 5
WHERE CodFilm = 5
```

```
UPDATE Filme
SET Nominalizari = Nominalizari + 5
WHERE CodFilm = 3
```

Controlul concurenţei bazat pe blocări (lock)

- Blocările sunt utilizate pentru a garanta planificări serializabile
- Un *protocol de blocare* este un set de reguli urmate de fiecare tranzacţie (fiind impuse de SGBD) pentru a se asigura că, chiar şi în situaţiile în care instrucţiunile tranzactiilor ar putea fi intercalate, efectul final este identic cu cel al unei executări seriale a tranzacţiilor.
- **Blocare**: O metodă utilizată pentru controlul accesului concurent la date. Atunci când o tranzacţie accesează un obiect al bazei de date, blocarea poate proteja obiectul respectiv de a fi accesat de o altă tranzacţie pentru a preveni obţinerea de rezultate incorecte.

Controlul concurenţei bazat pe blocări (lock)

- **Blocare partajată** (*shared* sau *read lock, S* - SQL Server): Dacă o tranzacţie blochează un obiect în mod partajat, ea poate citi acel obiect dar nu îl poate modifica.
- **Blocare exclusivă** (*exclusive* sau *write lock,* X - SQL Server): Dacă o tranzacţie blochează un obiect în mod exclusiv aceasta poate citi şi modifica valoarea obiectului.
- Alte tipuri de blocări:
  - U (update) – blocare shared care urmează sa se transforme în exclusive
  - key range etc

Blocări

- Compatibilitatea blocărilor:

	S	**U**	**X**
S	Yes	Yes	No
U	Yes	No	No
X	No	No	No

- Nivel de granularitate: cheie index, rând, tabel etc
- Dynamic Management Views
  - sys.dm_tran_locks – cereri către Lock Manager pentru blocări care au fost acordate sau așteaptă să fie acordate
  - sys.dm_tran_active_transactions

Niveluri de izolare

`SET TRANSACTION ISOLATION LEVEL` **`READ UNCOMMITTED`**

- pentru scriere sunt necesare blocări X (la toate nivelurile de izolare)
- nu se solicită blocări S la citire

⇒ o tranzacție poate vedea modificări necomise efectuate de altă tranzacție

Niveluri de izolare

`SET TRANSACTION ISOLATION LEVEL` **`READ COMMITTED`**

- nivel de izolare implicit în SQL SERVER
- la citirea unei resurse, tranzacția solicită o blocare S pe resursa respectivă, blocare care este eliberată la finalul operației

⇒ tranzacția nu mai poate vedea modificări necomise efectuate de altă tranzacție

- o altă tranzacție poate să modifice datele citite de tranzacția curentă, înainte de finalizarea acesteia

Niveluri de izolare

`SET TRANSACTION ISOLATION LEVEL` **`REPEATABLE READ`**

- la citirea unei resurse, tranzacția solicită o blocare S pe resursa respectivă, dar blocarea S se eliberează doar la finalul tranzacției

⇒ o altă tranzacție nu mai poate modifica datele citite de tranzacția curentă, înainte de finalizarea acesteia

- o altă tranzacție ar putea insera un rând în intervalul de tupluri citite de tranzacția curentă, înainte de finalizarea acesteia

Niveluri de izolare

`SET TRANSACTION ISOLATION LEVEL` **`SERIALIZABLE`**

- cel mai puternic nivel de izolare
- Păstrează blocările până la finalul tranzacției
- Repeatable read + key-range locks

⇒ o altă tranzacție nu mai poate insera un rând în intervalul de rânduri citite de tranzacția curentă, înainte de finalizarea acesteia

Niveluri de izolare

	Read Uncommitted	**Read Committed**	**Repeatable Read**	**Serializable**
Lost Updates	Nu	Nu	Nu	Nu
Dirty Reads	Da	Nu	Nu	Nu
Non-repeatable Reads	Da	Da	Nu	Nu
Phantoms	Da	Da	Da	Nu

Deadlock

- *Deadlock*: Ciclu de tranzacţii, fiecare aşteptând eliberarea unui obiect blocat de celelalte tranzacţii.
- O tranzacţie este în deadlock dacă nu mai poate continua executarea acţiunilor sale fără o intervenţie externă.
- Algoritmii de control a concurenţei pe bază de blocări pot cauza *deadlock*-uri.

- Metode de gestionarea *deadlock*-urilor:
  - **Prevenire** (garantează că nu apar *deadlock*-uri sau le anticipează)
  - **Detectare** (permit apariţia *deadlock*-urilor şi le rezolvă atunci când apar)

Prevenire *deadlock*

- Atribuie priorităţi bazate pe *timestamp*. (tranzacţiile mai vechi au prioritatea cea mai mare)
- Dacă T_i doreşte acces la un obiect blocat de T_j, sunt posibile două politici:
  - ***Wait-Die***: Dacă T_i are prioritate mai mare,T_i aşteaptă după T_j; altfel T_i se termină
  - ***Wound-wait***: Dacă T_i are prioritate mai mare, T_j se termină; altfel T_i aşteaptă
- Dacă o tranzacţie eliminată se reporneşte ulterior, va avea *timestamp*-ul original

Deadlock-urile şi expirarea timpului (timeout)

- O metodă simplă de prevenirea deadlock-urilor se bazează pe expirarea timpului de aşteptare după o resursă blocată
- După cererea unei blocări, o tranzacţie aşteaptă o perioadă de timp. Dacă obiectul aşteptat nu se deblochează după o anumită perioadă, tranzacţia este oprită şi repornită.

- Este o soluţie foarte simplă şi practică adoptată de multe SGBD-uri.

Detectarea *deadlock*-ului

- Se creează un graf de aşteptare:
  - Nodurile sunt tranzacţii
  - Există un arc de la T_i la T_j dacă T_i aşteaptă după T_j să elibereze un obiect blocat

- Dacă este un circuit/ciclu în acest graf atunci a apărut un *deadlock*.
- Periodic SGBD verifică dacă au apărut circuite în graful de aşteptare

Detectarea *deadlock*-ului

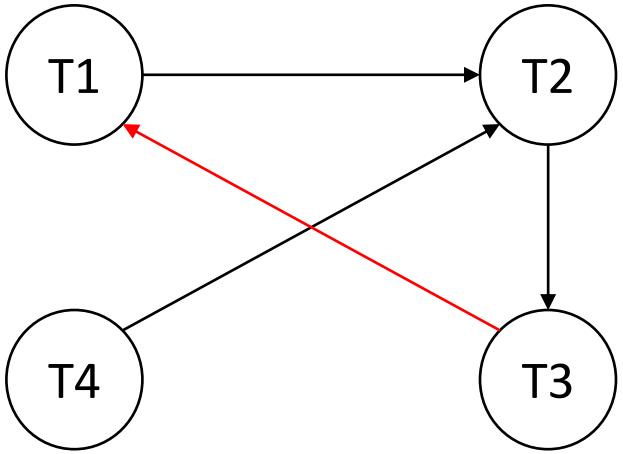
Exemplu:

T1: S(A), R(A), S(B)

T2: X(B), W(B), X(C)

T3: S(C),R(C) X(A)

T4: X(B)



Recuperarea după *deadlock*

- Cum se alege tranzacţia victimă a unui *deadlock*?
  - Durata execuţiei unei tranzacţii
  - Numărul obiectelor modificate de către tranzacţie
  - Numărul obiectelor ce urmează să fie modificate de către tranzacţie

- Politica de alegere a "*victimei*" trebuie să aibă în vedere echitatea: să nu fie aleasă de fiecare dată aceeaşi tranzacţie ca victimă

Planificarea tranzacțiilor

- O *planificare* reprezintă ordonarea secvenţială a instrucţiunilor (*Read*/ *Write*/ *Abort*/ *Commit*) a *n* tranzacţii astfel încât ordinea instrucţiunilor fiecărei tranzacţii se păstrează

Planificarea tranzacţiilor

T1:	**T2:**
read(A)	
read(sum)	
	read(A)
	A = A+20
	write(A)
	commit
read(A)	
sum = sum + A	
write(sum)	
commit	

Schedule

read1(A)
read1(sum)
read2(A)

write2(A)
commit2(A)
read1(A)

write1(sum)
commit1

Planificarea tranzacţiilor

- *Planificare serială:* este planificarea ce nu intercalează acţiuni ale mai multor tranzacţii.

- *Planificare non-serială*: acţiunile mai multor tranzacţii concurente se întrepătrund

Planificare serială vs. non-serială

Planificare non-serială

read1(A)
read1(A)
read2(A)

write2(A)
commit2(A)
read1(A)

write1(sum)
commit1

Planificare serială

read2(A)

write2(A)
commit2(A)
read1(A)
read1(A)
read1(A)

write1(sum)
commit1

Planificarea tranzacţiilor

- ***Planificări echivalente***: Pentru orice stare a bazei de date, efectul (asupra obiectelor bazei de date) al executării unei planificări este identic cu efectul executării celei de-a doua planificări.
- ***Planificări serializabile***: este o planificare non-serială care este echivalentă cu o planificare de execuţie serială a tranzacţiilor implicate.
- Notă: Dacă fiecare dintre tranzacţiile implicate în planificare păstrează consistenţa bazei de date atunci fiecare planificare serializabilă va păstra consistenţa acesteia

Serializabilitate

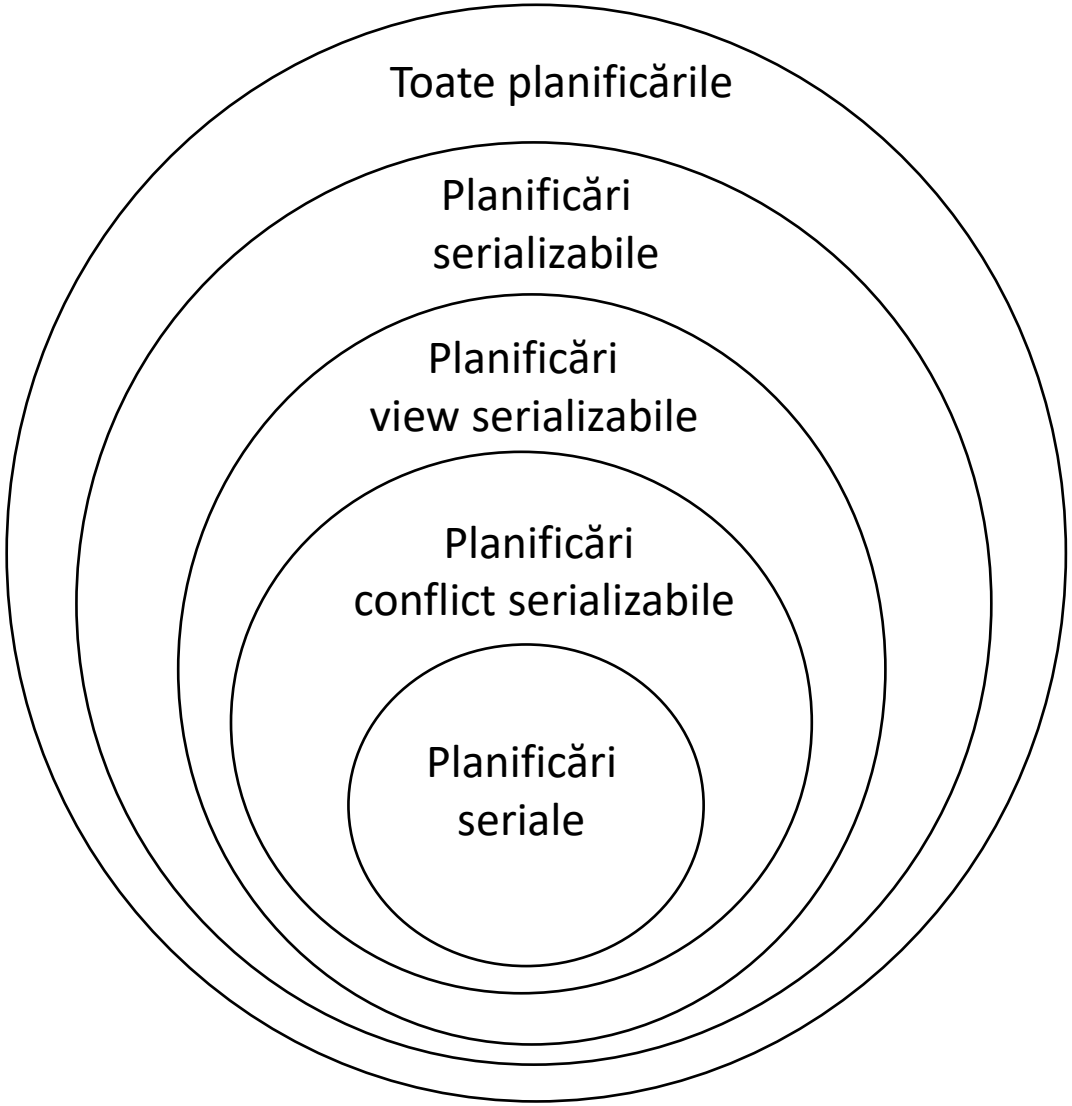
- Obiectivul ***serializabilităţii*** este găsirea unei planificări non-seriale care permite execuţia concurentă a tranzactiilor fără ca acestea să interfereze, şi astfel să conducă la o stare a unei baze de date la care se poate ajunge şi printr-o execuţie serială.

- Garantarea serializabilităţii tranzacţiilor concurente este importantă deoarece previne apariţia inconsistenţelor generate de interferenţa tranziţiilor

Planificarea tranzacţiilor

- Verificarea serializabilităţii: care sunt acţiunile ce nu se pot interschimba într-o tranzacţie?
  - Acţiunile aparţinând aceleiaşi tranzacţii
  - Acţiuni aplicate de diferite tranzacţii *aceluiaşi obiect,* dacă cel puţin una dintre ele este o operaţie de **write**. (acţiuni conflictuale!)

Transaktionen Ausführungspläne



Serializabilitate în practică

- În practică, un SGDB nu testează serializabilitea unei planificări date.
- Acest lucru nu este practic deoarece intercalarea operaţiilor mai multor tranzacţii concurente poate fi dictată de sistemul de operare şi prin urmare este dificil de impus.
- Abordarea DBMS este să folosească protocoale specifice care sunt cunoscute că generează planificări serializabile.
- Aceste protocoale pot afecta gradul de concurenţă, însă elimină cazurile conflictuale.