Adatbázisrendszerek Tranzakciók

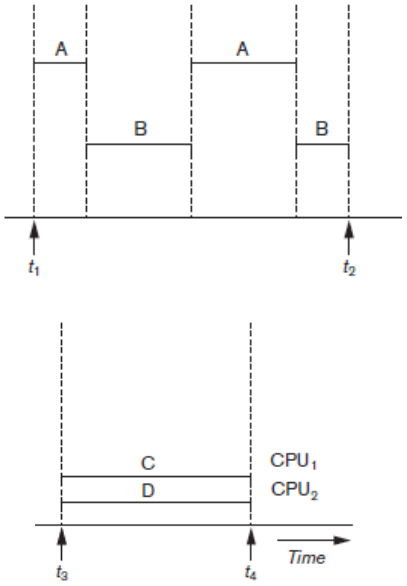
RAMEZ ELMASRI, SHAMKANT B. NAVATHE: *FUNDAMENTALS OF DATABASE SYSTEMS* KÖNYV ALAPJÁN

DBS csoportosítása konkurens felhasználók száma szerint

1. Egyfelhasználós rendszer
   Egyidejűleg legfeljebb egy felhasználó használhatja a rendszert.
2. Többfelhasználós rendszer
   Egyidejűleg (konkurens módon) több felhasználó is elérheti a rendszert.
   A legtöbb rendszer ilyen.

Konkurencia típusa

- Összefésült egyszálas feldolgozás
  A folyamatok (processzek) konkurens végrehajtása egy CPU-n fésülődik össze
- Párhuzamos feldolgozás
  A folyamatok (processzek) egyidejűleg (konkurens módon) több CPU-n hajtódnak végre.



Tranzakció

Adatbázis folyamatok egy olyan logikai egysége, amely egy vagy több adatbázis-hozzáférési műveletből (olvasás – kinyerés, írás – beszúrás, törlés, módosítás) épül fel.

A tranzakciót alkotó adatbázis-műveletek

- beágyazhatóak egy alkalmazói programba vagy
- megadhatóak explicit módon egy magas szintű lekérdező nyelv segítségével (pl.: SQL).

Tranzakció

A tranzakció határait megadhatjuk egy alkalmazói programban explicit módon **begin transaction** és **end transaction**, ahol a két határ között elhelyezkedő összes adatbázis-hozzáférési művelet alkot egy tranzakciót.

Egy alkalmazói program egynél több tranzakciót is tartalmazhat, ha több tranzakció elhatároló szerepel benne.

Read-only tranzakció
a tranzakciót alkotó adatbázis-műveletek nem módosítják az adatbázist, csak lekérdezik azt.

Read-write tranzakció
a tranzakciót alkotó adatbázis-műveletek módosítják is az adatbázist.

Adatbázis leegyszerűsített modellje

- Adatbázis: nevesített adatelemek (*named data item*) összessége
- Az adatok finomsága (*granularity*) az adatelemek (rekord, blokk (lemez), mezőérték) méretét jellemzi.
  Az ismertetett fogalmak függetlenek a konkrét finomságtól.
- Adatelem egyedi nevét a programozó tipikusan nem használja, csak egy módot biztosít az egyértelmű azonosításhoz.
- Tranzakció alapműveletei ebben a leegyszerűsített adatbázis modellben
  - **Read_item(X)**
    Beolvassa az adatbázis X elnevezésű elemét egy programváltozóba. Az egyszerűség kedvéért jelöljük a programváltozót is X-szel.
  - **Write_item(X)**
    Az X programváltozó értékét az adatbázis X elnevezésű elemébe rögzíti.

Az írás és olvasás művelete részletesen

Read_item(*X*)

- Megkeresi az *X* elemet tartalmazó *lemezblokk címét*.
- Átmásolja ezt a lemezblokkot a fő memória pufferébe (amennyiben ez a blokk nincs már benne valamelyik fő memória pufferben).
- Átmásolja az *X* elemet a pufferből az *X* nevű programváltozóba.

Write_item(*X*)

- Megkeresi az *X* elemet tartalmazó lemezblokk címét.
- Átmásolja ezt a lemezblokkot a fő memória pufferébe (amennyiben ez a blokk nincs már benne valamelyik fő memória pufferben).
- Átmásolja az *X* elemet az *X* nevű programváltozóból a puffer megfelelő területére.
- Visszamásolja a frissített blokkot a pufferből a lemezre (rögtön vagy egy későbbi időpontban).

Konkurens hozzáférés szabályozása

Tekintsünk egy egyszerű repülő helyjegyfoglaló rendszert, ahol egy rekord tárolja járatonként a foglalásokat (ellenőrzések nincsenek beépítve).

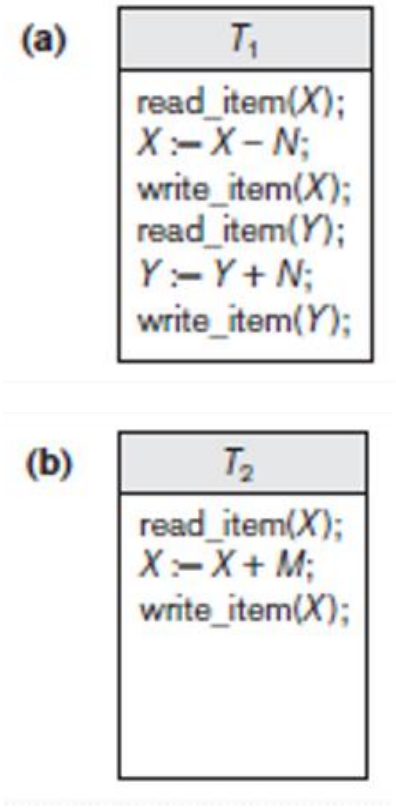
Program paraméterei
Járatszám, dátum, foglalandó helyek száma

T1: N jegy átfoglalása

T2: M jegy lefoglalása

Read-set(T1): {X,Y}, Write-set(T1:): {X,Y}

~ szempontjából a tranzakció egy meghatározott futtatása a programnak egy adott dátum, járat, helyszám esetén



Konkurens hozzáférésből adódható problémák

1. Az elvesztett frissítés problémája
2. Az ideiglenes frissítés (dirty read) problémája
3. A helytelen összegzés problémája
4. Megismételhetetlen olvasás problémája

1. Az elveszett frissítés problémája

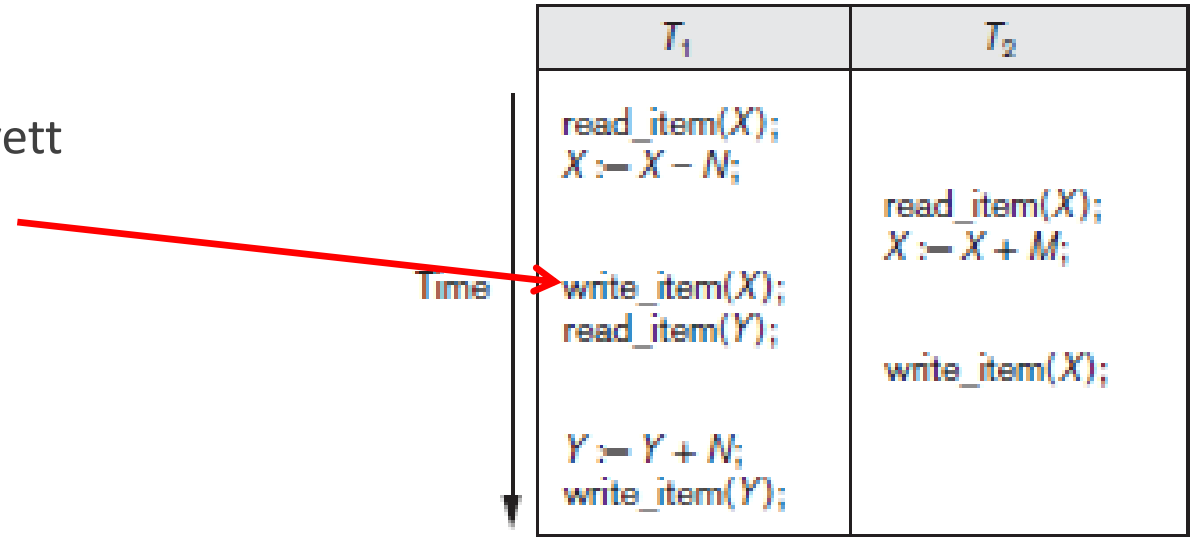
Ugyanazt az adatbázis elemet 2 tranzakció művelete is használja mely helytelen értéket eredményez.

Tegyük fel, hogy T1 és T2 megközelítőleg egyszerre kerül végrehajtásra

Pl.: X= 80, N=5, M = 4

X = 84, X = 79 helyett

Elveszett frissítés

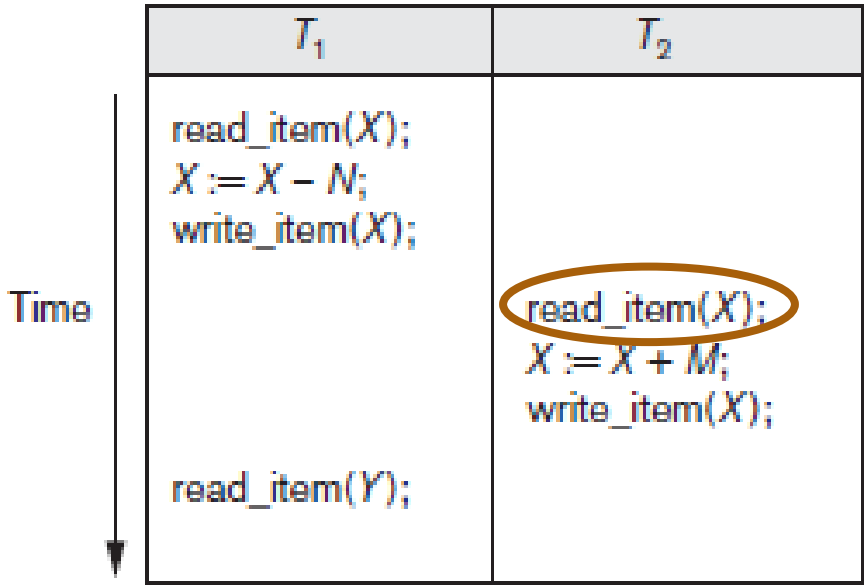


2.Ideiglenes frissítés (piszkos olvasás)

Egy adatbázis elem frissítése után a tranzakció leáll, de még a visszaállítás előtt egy másik tranzakció kiolvassa az értékét.

Piszkos olvasás
be nem fejezett tranzakcióból származó átmeneti érték olvasása.

T_1 tranzakció leáll és emiatt az X értékét vissza kell állítani az új értékre.



3. Helytelen összegzés

Aggregáló függvény használata közben másik tranzakció frissítést végez

T_1	T_3

	sum := 0;
	read_item(*A*);
	sum := *sum* + *A*;
	.

read_item(*X*);
X := *X* − *N*;
write_item(*X*);
	.

	read_item(*X*);
	sum := *sum* + *X*;
	read_item(*Y*);
	sum := *sum* + *Y*;

read_item(*Y*);
Y := *Y* + *N*;
write_item(*Y*);	

4. Megismételhetetlen olvasás problémája

Egy tranzakció ugyanannak az elemnek két különböző olvasása során különböző értékeket kap, mert egy másik tranzakció megváltoztatta az elem értékét.

Helyreállítás (Recovery)

Valahányszor egy tranzakciót elküldünk egy DBMS-hez végrehajtásra a rendszernek biztosítania kell vagy

- a tranzakció összes műveletének sikeres végrehajtását és az eredmények tartós rögzítését, vagy azt, hogy
- a tranzakciónak semmilyen hatása ne legyen.

Tranzakció

- Véglegesített (committed)
- Megszakított (aborted)

Mindent vagy semmit elv

Sikertelen tranzakció okai

1. Számítógép hiba (rendszerösszeomlás)
   A tranzakció futása alatt hardver (pl. memória), szoftver vagy hálózati hiba miatt meghiúsul a tranzakció sikeres befejezése.
2. Tranzakció vagy rendszerhiba
   Valamelyik tranzakciós művelet fut hibára (túlcsordulás, nullával osztás). Okozhatja hibás paraméterérték is illetve hibás programlogika is. A felhasználó leállíthatja a tranzakció futását.
3. A tranzakció által felismert lokális hiba vagy kivétel
   Pl.: a művelet adata nem található, fedezet hiány (olyan kivétel melyet a program képes kezelni, valójában nem is hiba)

Sikertelen tranzakció okai

4. Konkurencia szabályozás kényszerítése
   A konkurencia szabályozó metódus megszakíthatja a tranzakciót a sorbarendezhetőség elvének megsértése miatt vagy mert szükséges egy holtpont feloldásához. Általában később ezek automatikusan újra elindulnak.

5. Lemezhiba
   Hibás olvasás vagy írás (esetleg az író/olvasó fej hibája) miatt egy lemez blokk elveszítheti adatát

6. Fizikai problémák, katasztrófák
   Egyéb okokból származó hibák, mint áramkimaradás, tűz, lopás, szabotázs, tévedésből származó adatfelülírás, stb.

Az első négy gyakoribb. Ezek esetén a rendszernek rendelkeznie kel elegendő információval a gyors helyreállításhoz.

Tranzakciók állapota

A DBMS helyreállító (recovery manager) a következő műveleteket követi nyomon:

BEGIN_TRANSACTION
A tranzakció futásának indítását jelzi.

READ vagy WRITE
A tranzakció részeként elvégzett adatelem olvasását vagy írását jelzi.

END_TRANSACTION
Jelzi a READ és WRITE tranzakciós műveletek ill. a tranzakció futásának befejeződését.

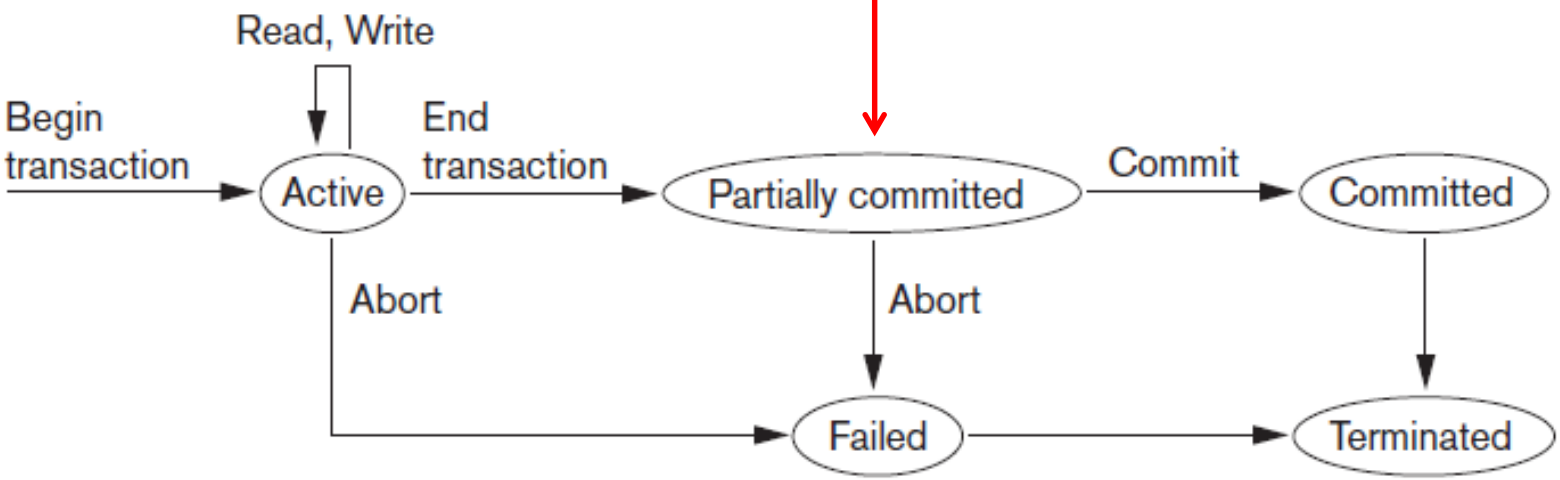
COMMIT_TRANSACTION
A tranzakció sikeres bejeződését jelzi, azaz biztonságosan rögzítésre kerültek a változások és nem lesznek visszavonva.

ROLLBACK (vagy ABORT)
A tranzakció sikertelen bejeződését jelzi, ami a már végrehajtott változtatások visszavonását eredményezi.

Tranzakció állapotdiagramja

Ellenőrzési pont (egy rendszerhiba nem fogja-e meghiúsítani a változások permanens rögzítését) – log fájl írása
változások rögzítése a log fájlban

Sikeres ellenőrzés ⇨ commit point (lsd. később) ⇨ commited state



A rendszer napló (system log)

Egy szekvenciális, csak továbbfűzhető állomány a lemezen, mely nincs kitéve csak a lemezhibának vagy katasztrófa okozta hibának.

Tipikusan egy (vagy több) memória puffer tárolja a log fájl utolsó részét, azaz a bejegyzések először mindig ide kerülnek. Amikor megtelik, vagy bizonyos körülmények fennállnak, tartalma a lemezen lévő fájlhoz hozzáfűződik.

A lemezen lévő log fájl rendszeresen archiválásra kerül a katasztrofikus hibák okozta károk kivédése érdekében.

Log bejegyzések (log records) típusai

1. **[start_transaction, *T*]**
2. **[write_item, *T*, *X*, *old_value*, *new_value*]**
   A *T* tranzakció megváltoztatta az *X* adatbázis elem értékét *old_value*-ról *new_value*-re.
3. **[read_item, *T*, *X*]**
4. **[commit, *T*]**
   A *T* tranzakció sikeresen befejeződött és a végrehajtott változtatások rögzíthetőek az adatbázisba.
5. **[abort, *T*]**

T : egyedi tranzakciós azonosító

Tranzakció commit point-ja

Egy *T* tranzakció ***commit pont***hoz ér ha minden adatbázis művelete sikeresen végrehajtásra került **és** minden műveletének az eredménye rögzítésre került a log fájlban.

A commit pont után a tranzakció végrehajtott (committed) azaz az eredményeinek véglegesen rögzítésre kellett kerülni az adatbázisban és kiíródik egy [commit, *T*] rekord a log fájlba.

Tranzakciók kívánatos tulajdonságai (ACID tulajdonságok)

1. Atomosság (atomicity)
   A tranzakció a feldolgozás atomi egysége; vagy teljes egészében végrehajtódik, vagy egyáltalán nem.

2. Konzisztencia megőrzés (consistency preservation)
   Egy tranzakció konzisztencia megőrző, ha teljes és önálló végrehajtása az adatbázist konzisztens állapotból konzisztens állapotba viszi át.

3. Elkülönítés (isolation)
   Egy tranzakciónak látszólag más tranzakcióktól elkülönítve kell végrehajtódnia. Ez azt jelenti, hogy a tranzakció végrehajtása nem állhat kölcsönhatásban semelyik másik konkurensen végrehajtott tranzakcióval sem.

4. Tartósság vagy állandóság (durability or permanency)
   Egy véglegesített tranzakció által az adatbázison véghezvitt módosításoknak megőrzésre kell kerülniük az adatbázisban. Ezeknek a módosításoknak semmilyen hiba miatt nem szabad elveszniük.

Az ACID tulajdonságok biztosításáért a DBMS konkurenciavezérlő és naplózó/helyreállító alrendszerei a felelősek.

Tranzakció ütemezés

Tranzakció ütemezés (schedule vagy history)

A T_1, T_2, …, T_n, tranzakciók egy *S* ütemezése a tranzakciókhoz tartozó műveletek összefésült, egy szálon való végrehajtásához meghatározott sorrendje.

Az ütemezés tehát a tranzakció műveletek egy olyan sorrendje, amely megfelel az egyenkénti tranzakcióbeli sorrendnek, azaz ha két, egyazon tranzakcióbeli műveletnél az egyik megelőzi a másikat, úgy ez a sorrend megmarad az összefésülés után is.

Egy ütemezés két művelete konfliktusban van, ha a következő feltételek **mindegyike** teljesül

1. Különböző tranzakciókhoz tartoznak
2. Ugyanazzal az *X* adatelemmel dolgoznak
3. Legalább az egyik művelet *write_item(X)*

Ütemezés típusok visszaállíthatóság alapján

Visszaállítható ütemezés (↔ nem visszaállítható ütemezés)
Egyetlen egy olyan **T** tranzakció sem kerül véglegesítésre míg nem véglegesítődik minden olyan **T'** tranzakció, amely olyan elemet ír ki, amelyet T beolvas. Ekkor egyetlen véglegesített tranzakciót sem szükséges visszaállítani.

Kaszkádmentes ütemezés
Minden tranzakció csak olyan adatbázis elemet olvas be, amelyet egy már elfogadott tranzakció írt ki.

Szigorú ütemezés
A tranzakciók se nem írhatnak se nem olvashatnak egy X elemet addig, míg az utolsó, X adatelemet író tranzakció véglegesítésre (vagy megszakításra) nem kerül.

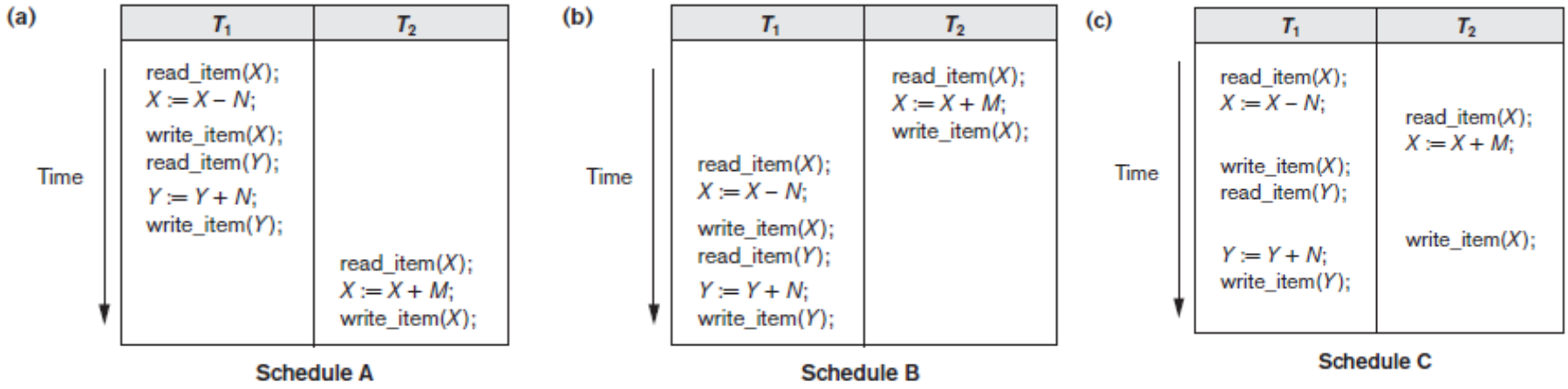
Minden szigorú ütemezés kaszkádmentes, minden kaszkádmentes visszaállítható ütemezés.

Ütemezés típusok sorbarendezhetőség alapján

Soros ütemezés (↔ nem soros)

Egy S ütemezés soros, ha az ütemezésbeli minden T tranzakció esetén az összes T-beli művelet közvetlenül egymás után kerül végrehajtásra.

Az ütemezés sorbarendezhetőségnek fogalma a helyes ütemezés azonosításához használatos.



Ütemezés típusok sorbarendezhetőség alapján

Sorba rendezhető ütemezés

Egy *n* tranzakcióból álló S ütemezés sorba rendezhető, ha ekvivalens az *n* tranzakció valamelyik soros ütemezésével.

S_1	S_2

read_item(X);
$X := X + 10$;
write_item(X);
	read_item(X);
	$X := X * 1.1$;
	write_item (X);

X=100 esetén eredmény ekvivalensek, de általában nem.

Ekvivalencia típusok

- Eredmény ekvivalencia (!)
  A két ütemezés ugyanazt a végső adatbázis állapotot eredményezi.
- Konfliktus ekvivalencia
  A két ütemezésben bármely két konfliktusos művelet (pl. ugyanazt az adatbázis elemet akarja felülírni két tranzakció) sorrendisége megegyezik.
- Nézet ekvivalencia

Konfliktus sorba rendezhető ütemezés

Az ütemezés konfliktus ekvivalens egy soros ütemezéssel.

Megjegyzések a sorbarendezhetőségről

1. A sorbarendezhetőség nem jelenti azt, hogy az ütemezés maga szeriális (soros).
2. A sorbarendezhetőségből következik, hogy az ütemezés helyes. Ez azt jelenti, hogy az adatbázis konzisztens állapotban marad a tranzakciók végrehajtása után.
3. A sorbarendezhetőséget nehéz ellenőrizni, mivel nem könnyű előre meghatározni, hogy egy ütemező hogyan fésüli össze a műveleteket.
4. A gyakorlatban protokollokat használnak a sorbarendezhetőség biztosítására.
5. Egy ütemezés kezdete és vége nem meghatározható, ezért a teljes ütemezés ellenőrzését a véglegesített tranzakciókbeli műveletek ellenőrzésére redukálják.
6. Egy gyengébb ekvivalencia ütemezés az ún. nézet (view) ekvivalencia, melyet itt nem tárgyalunk.
7. Van algoritmus a konfliktus sorbarendezhetőség ellenőrzésére, amely a precedencia gráfon alapszik.

A konkurencia vezérlés

Célja

Az elkülönítés kikényszerítése (pl. teljes kizárással) a konfliktusos tranzakciók között.

Az adatbázis konzisztenciájának megőrzése a tranzakciók konzisztencia megőrző végrehajtása révén.

Az olvasás-írás és írás-írás konfliktusok feloldása.

Pl.:

Ha egy konkurens végrehajtási környezetben a T1 konfliktusba kerül a T2-vel az A adatelem miatt, akkor a létező konkurencia kontroll dönt arról, ha T1-nek vagy T2-nek szüksége van A-ra, illetve ha más tranzakciókat vissza kell állítani vagy várakoztatni kell.

A kétfázisú zárolás módszere

Az *X* adatbázis elemen két (atomi) művelet értelmezünk

Lock(X)
A zárolás művelet biztosítja az engedélyt a tranzakció számára egy adatelem olvasására vagy írására.

Unlock(X).
A feloldás művelet törli ezeket az engedélyeket az adatelemről.

Nagyon szigorú, gyakorlatban nem alkalmazható.

Megosztott / kizárásos zárolás

Zároló operátorok: read_lock(*X*), write_lock(*X*), unlock(*X*)

megosztásos (shared lock / read_lock)
Egynél több is bejegyezhető egy adatbázis elemre olvasás céljából, azonban ekkor kizárást már nem jegyezhetünk be semmilyen tranzakció által.

kizárásos (exclusive lock / write lock)
Csak egy kizárás jegyezhető be egy adatbázis elemre egy időben és ekkor egyetlen tranzakció sem jegyezhet be megosztást erre az adatbázis elemre.

A Lock Manager kezeli a zárolásokat az adatelemeken egy zárolási tábla (lock table) segítségével.