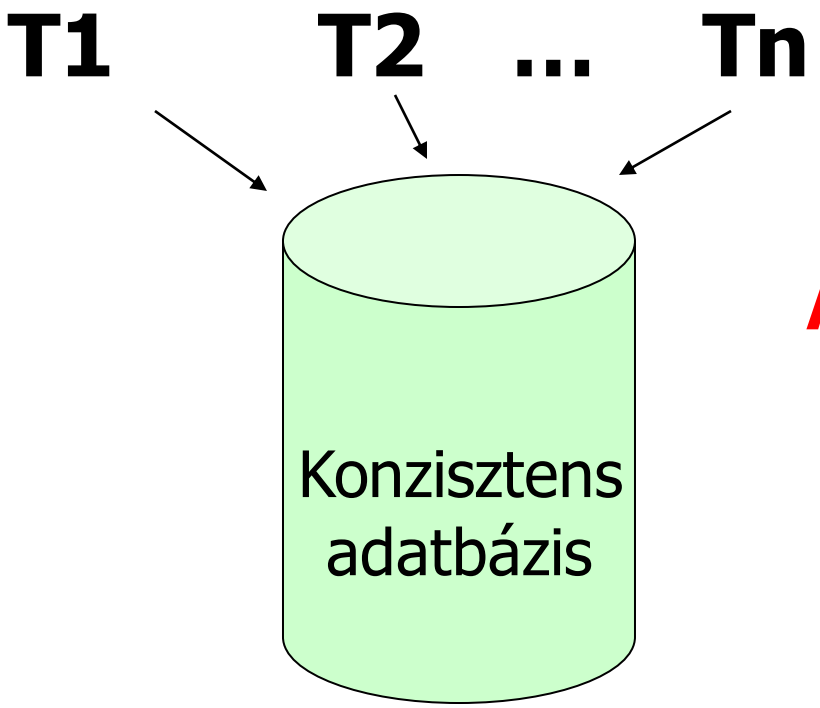
Konkurenciavezérlés

Egyszerre több tranzakció is ugyanazt az adatbázist használja.



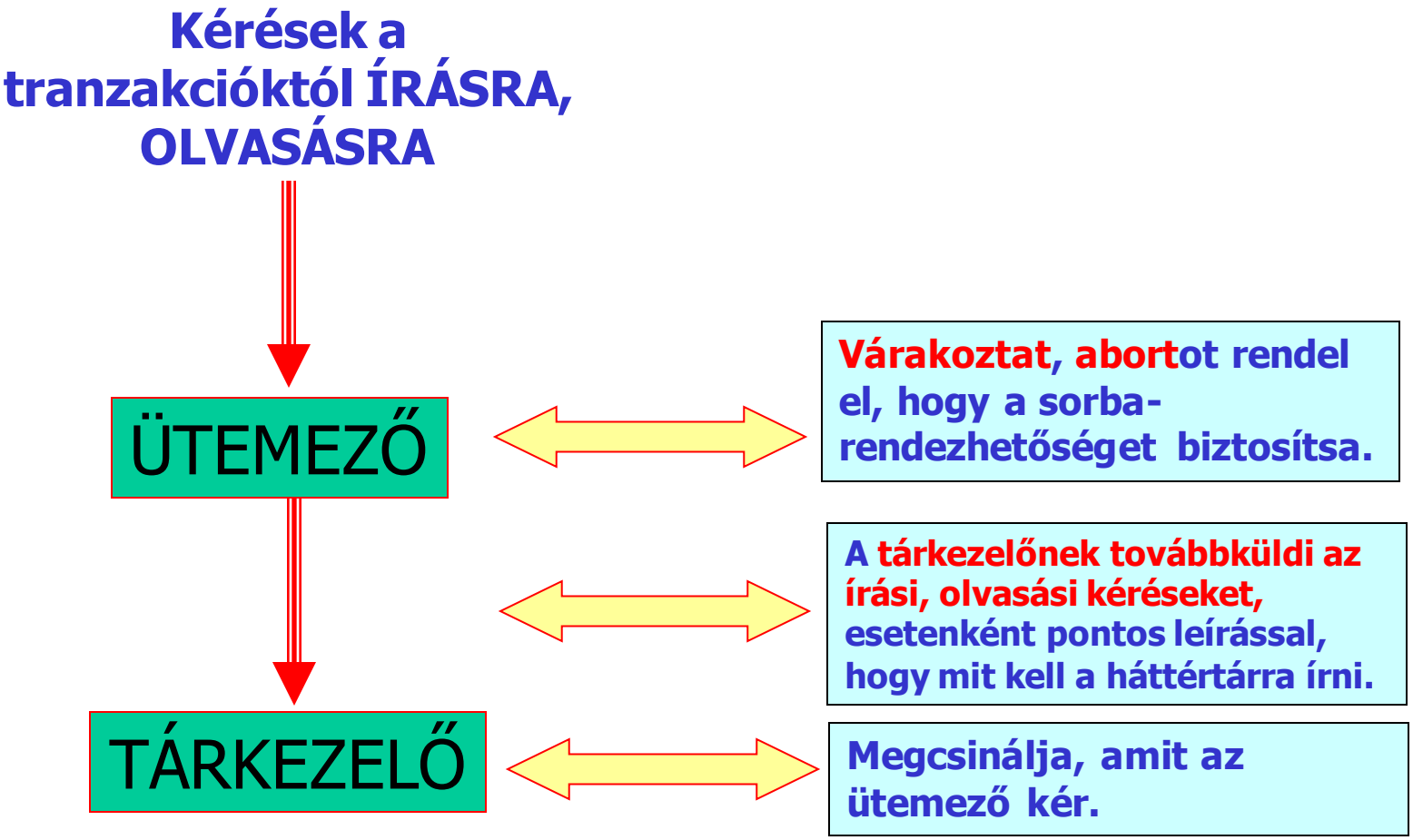
Az adatbázisnak konzisztensnek kell maradnia!

A tranzakciók közötti egymásra hatás az adatbázis-állapot inkonzisztenssé válását okozhatja, még akkor is, amikor a tranzakciók külön-külön megőrzik a konzisztenciát, és rendszerhiba sem történt.

A konkurenciavezérlés

- **A tranzakciós lépések szabályozásának feladatát az adatbázis-kezelő rendszer *ütemező* (scheduler) része végzi.**
- **Azt az általános folyamatot, amely biztosítja, hogy a tranzakciók egyidejű végrehajtása során megőrizzék a konzisztenciát, *konkurenciavezérlésnek* (concurrency control) nevezzük.**
- **Amint a tranzakciók az adatbáziselemek olvasását és írását kérik, ezek a kérések az ütemezőhöz kerülnek, amely legtöbbször közvetlenül végrehajtja azokat. Amennyiben a szükséges adatbáziselem nincs a pufferben, először a pufferkezelőt hívja meg.**
- **Bizonyos esetekben azonban nem biztonságos azonnal végrehajtani a kéréseket. Az ütemezőnek ekkor késleltetnie kell a kérést, sőt bizonyos esetben abortálnia kell a kérést kiadó tranzakciót.**
- **Az *ütemezés* (schedule) egy vagy több tranzakció által végrehajtott lényeges műveletek időrendben vett sorozata, amelyben az egy tranzakcióhoz tartozó műveletek sorrendje megegyezik a tranzakcióban megadott sorrenddel.**

Az ütemező és a tárkezelő együttműködése



• A konkurenciakezelés szempontjából a lényeges olvasási és írási műveletek a központi memória puffereiben történnek, nem pedig a lemezen. Tehát csak a READ és WRITE műveletek sorrendje számít, amikor a konkurenciával foglalkozunk, az INPUT és OUTPUT műveleteket figyelmen kívül hagyjuk.

T_1	T_2
READ(A,t)	READ(A,s)
t := t+100	s := s*2
WRITE(A,t)	WRITE(A,s)
READ(B,t)	READ(B,s)
t := t+100	s := s*2
WRITE(B,t)	WRITE(B,s)

Konzisztencia:

A=B

Egymás után futtatva, megőrzik a konzisztenciát.

Soros ütemezések

A két tranzakciónak két soros ütemezése van, az egyikben T1 megelőzi T2-t, a másikban T2 előzi meg T1-et

T_1	T_2	A	B
READ(A,t)		25	
t := t+100			
WRITE(A,t)		125	
READ(B,t)			25
t := t+100			
WRITE(B,t)			125
	READ(A,s)	125	
	s := s*2		
	WRITE(A,s)	250	
	READ(B,s)		125
	s := s*2		
	WRITE(B,s)		250

T_1	T_2	A	B
	READ(A,s)	25	
	s := s*2		
	WRITE(A,s)	50	
	READ(B,s)		25
	s := s*2		
	WRITE(B,s)		50
READ(A,t)		50	
t := t+100			
WRITE(A,t)		150	
READ(B,t)			50
t := t+100			
WRITE(B,t)			150

Mindkét soros ütemezés konzisztens (A=B) adatbázist eredményez, bár a két ütemezésben a végeredmények különböznek (250, illetve 150).

Sorbarendezhető ütemezések

Az első ütemezés egy sorbarendezhető, de nem soros. A hatása megegyezik a (T1, T2) soros ütemezés hatásával: tetszőleges konzisztens kiindulási állapotra: A = B = c-ből kiindulva A-nak is és B-nek is 2(c + 100) lesz az értéke, tehát a konzisztenciát mindig megőrizzük.

T_1	T_2	A	B
READ(A,t)		25	
t := t+100			
WRITE(A,t)		125	
	READ(A,s)	125	
	s := s*2		
	WRITE(A,s)	250	
READ(B,t)			25
t := t+100			
WRITE(B,t)			125
	READ(B,s)		125
	s := s*2		
	WRITE(B,s)		250

T_1	T_2	A	B
READ(A,t)		25	
t := t+100			
WRITE(A,t)		125	
	READ(A,s)	125	
	s := s*2		
	WRITE(A,s)	250	
	READ(B,s)		25
	s := s*2		
	WRITE(B,s)		50
READ(B,t)			50
t := t+100			
WRITE(B,t)			150

Sorbarendezhető ütemezések

A második példában szereplő ütemezés nem sorbarendezhető. A végeredmény sosem konzisztens A := 2(A + 100), B := 2B + 100, így nem lehet a hatása soros ütemezéssel megegyező. Az ilyen viselkedést a különböző konkurenciavezérlési technikákkal el kell kerülnünk.

T_1	T_2	A	B
READ(A,t)		25	
t := t+100			
WRITE(A,t)		125	
	READ(A,s)	125	
	s := s*2		
	WRITE(A,s)	250	
READ(B,t)			25
t := t+100			
WRITE(B,t)			125
	READ(B,s)		125
	s := s*2		
	WRITE(B,s)		250

T_1	T_2	A	B
READ(A,t)		25	
t := t+100			
WRITE(A,t)		125	
	READ(A,s)	125	
	s := s*2		
	WRITE(A,s)	250	
	READ(B,s)		25
	s := s*2		
	WRITE(B,s)		50
READ(B,t)			50
t := t+100			
WRITE(B,t)			150

A tranzakció szemantikájának hatása

Ez az ütemezés elvileg sorbarendezhető, de csak T2 speciális aritmetikai művelete (1-gyel szorzás) miatt. Az, hogy mivel szorzunk, lehet, hogy egy bonyolult függvény eredményeképpen dől el. Az ütemező csak az írási, olvasási műveleteket figyelve pesszimista alapon dönt a sorbarendezhetőségről:

- **Ha T tudna A-ra olyan hatással lenni, hogy az adatbázis-állapot inkonzisztenssé váljon, akkor T ezt meg is teszi.**

A feltevés miatt az ütemező szerint ez az ütemezés nem lesz sorbarendezhető.

T_1	T_2	A	B
READ(A,t)		25	
t := t+100			
WRITE(A,t)		125	
	READ(A,s)	125	
	s := s*1		
	WRITE(A,s)	125	
	READ(B,s)		25
	s := s*1		
	WRITE(B,s)		25
READ(B,t)			25
t := t+100			
WRITE(B,t)			125

A tranzakciók és az ütemezések jelölése

- **Csak a tranzakciók által végrehajtott olvasások és írások számítanak!**

$T_1: r_1(A); w_1(A); r_1(B); w_1(B);$

$T_2: r_2(A); w_2(A); r_2(B); w_2(B);$

- **Az ütemezés jelölése:**

$S: r_1(A); w_1(A); r_2(A); w_2(A); r_1(B); w_1(B); r_2(B); w_2(B);$

1. Az $r_i(X)$ vagy $w_i(X)$ azt jelenti, hogy a T_i tranzakció olvassa, illetve írja az X adatbáziselemet.
2. Egy T_i *tranzakció* az i indexű műveletekből álló sorozat.
3. Egy S *ütemezés* olyan műveletek sorozata, amelyben minden T_i tranzakcióra teljesül, hogy T_i műveletei ugyanabban a sorrendben fordulnak elő S-ben, mint a T_i -ben. S a tranzakciók műveleteinek *átlapolása* (interleaving).

Konfliktus-sorbarendezhetőség

- **Elégséges feltétel: biztosítja egy ütemezés sorbarendezhetőségét.**

- **A forgalomban lévő rendszerek ütemezői a tranzakciók sorbarendezhetőségére általában ezt az erősebb feltételt biztosítják, amelyet *konfliktus-sorbarendezhetőségnek* nevezünk.**

- **A *konfliktus* (conflict) vagy *konfliktuspár* olyan egymást követő műveletpár az ütemezésben, amelynek ha a sorrendjét felcseréljük, akkor legalább az egyik tranzakció viselkedése megváltozhat.**

Nincs konfliktus

- Legyen T_i és T_j két különböző tranzakció ($i \neq j$).

1. $r_i(X); r_j(Y)$ sohasem konfliktus, még akkor sem, ha $X = Y$,
   - mivel egyik lépés sem változtatja meg az értékeket.

2. $r_i(X); w_j(Y)$ nincs konfliktusban, ha $X \neq Y$,
   - mivel T_j írhatja Y-t, mielőtt T_i beolvasta X-et, X értéke ettől ugyanis nem változik. Annak sincs hatása T_j-re, hogy T_i olvassa X-et, ugyanis ez nincs hatással arra, hogy milyen értéket ír T_j Y-ba.

3. $w_i(X); r_j(Y)$ nincs konfliktusban, ha $X \neq Y$,
   - ugyanazért, amiért a 2. pontban.

4. $w_i(X); w_j(Y)$ sincs konfliktusban, ha $X \neq Y$.

Konfliktus

- **Három esetben nem cserélhetjük fel a műveletek sorrendjét:**

a) $r_i(X); w_i(Y)$ konfliktus,

- mivel egyetlen tranzakción belül a műveletek sorrendje rögzített, és az adatbázis-kezelő ezt a sorrendet nem rendezheti át.

a) $w_i(X); w_j(X)$ konfliktus,

- mivel X értéke az marad, amit T_j számolt ki. Ha felcseréljük a sorrendjüket, akkor pedig X-nek a T_i által kiszámolt értéke marad meg. A pesszimista feltevés miatt a T_i és a T_j által kiírt értékek lehetnek különbözőek, és ezért az adatbázis valamelyik kezdeti állapotára különbözni fognak.

b) $r_i(X); w_j(X)$ és $w_i(X); r_j(X)$ is konfliktus.

- Ha átvisszük $w_j(X)$-et $r_i(X)$ elé, akkor a T_i által olvasott X-beli érték az lesz, amit a T_j kiírt, amiről pedig feltételeztük, hogy nem szükségképpen egyezik meg X korábbi értékével. Tehát $r_i(X)$ és $w_j(X)$ sorrendjének cseréje befolyásolja, hogy T_i milyen értéket olvas X-ből, ez pedig befolyásolja T_i működését.

Konfliktus-sorbarendezhetőség

- **ELV: nem konfliktusos cserékkel az ütemezést megpróbáljuk soros ütemezéssé átalakítani. Ha ezt meg tudjuk tenni, akkor az eredeti ütemezés sorbarendezhető volt, ugyanis az adatbázis állapotára való hatása változatlan marad minden nem konfliktusos cserével.**

- **Azt mondjuk, hogy két ütemezés *konfliktusekvivalens* (conflict-equivalent), ha szomszédos műveletek nem konfliktusos cseréinek sorozatával az egyiket átalakíthatjuk a másikká.**

- **Azt mondjuk, hogy egy ütemezés *konfliktus-sorbarendezhető* (conflict-serializable schedule), ha konfliktusekvivalens valamely soros ütemezéssel.**

- **A konfliktus-sorbarendezhetőség elégséges feltétel a sorbarendezhetőségre, vagyis egy konfliktus-sorbarendezhető ütemezés sorbarendezhető ütemezés is egyben.**

- **A konfliktus-sorbarendezhetőség nem szükséges ahhoz, hogy egy ütemezés sorbarendezhető legyen, mégis általában ezt a feltételt ellenőrzik a forgalomban lévő rendszerek ütemezői, amikor a sorbarendezhetőséget kell biztosítaniuk.**

- ***Példa.*** **Legyen az ütemezés a következő:**

  $r_1(A); w_1(A); r_2(A); w_2(A); r_1(B); w_1(B); r_2(B); w_2(B);$

- **Azt állítjuk, hogy ez az ütemezés konfliktus-sorbarendezhető. A következő cserékkel ez az ütemezés átalakítható a (T_1, T_2) soros ütemezéssé, ahol az összes T_1-beli művelet megelőzi az összes T_2-beli műveletet:**

1. $r_1(A); w_1(A); r_2(A); \underline{w_2(A); r_1(B);} w_1(B); r_2(B); w_2(B);$
2. $r_1(A); w_1(A); \underline{r_2(A); r_1(B);} w_2(A); w_1(B); r_2(B); w_2(B);$
3. $r_1(A); w_1(A); r_1(B); r_2(A); \underline{w_2(A); w_1(B);} r_2(B); w_2(B);$
4. $r_1(A); w_1(A); r_1(B); \underline{r_2(A); w_1(B);} w_2(A); r_2(B); w_2(B);$
5. $r_1(A); w_1(A); r_1(B); w_1(B); r_2(A); w_2(A); r_2(B); w_2(B);$

Nem konfliktus-sorbarendezhető ütemezés

$S: r_1(A); w_1(A); r_2(A); w_2(A); r_2(B); w_2(B); r_1(B); w_1(B);$

Ez az ütemezés nem konfliktus-sorbarendezhető, ugyanis A-t T1 írja előbb, B-t pedig T2. Mivel sem A írását, sem B írását nem lehet átrendezni, semmilyen módon nem kerülhet T1 összes művelete T2 összes művelete elé, sem fordítva.

T_1	T_2	A	B
READ(A,t)		25	
t := t+100			
WRITE(A,t)		125	
	READ(A,s)	125	
	s := s*1		
	WRITE(A,s)	125	
	READ(B,s)		25
	s := s*1		
	WRITE(B,s)		25
READ(B,t)			25
t := t+100			
WRITE(B,t)			125

Sorbarendezhető, de nem konfliktus-sorbarendezhető ütemezés

- Tekintsük a T_1, T_2 és T_3 tranzakciókat és egy **soros** ütemezésüket:

  S_1: $w_1(Y)$; $w_1(X)$; $w_2(Y)$; $w_2(X)$; $w_3(X)$;

- Az S_1 ütemezés X értékének a T_3 által írt értéket, Y értékének pedig a T_2 által írt értéket adja. Ugyanezt végzi a következő ütemezés is:

  S_2: $w_1(Y)$; $w_2(Y)$; $w_2(X)$; $w_1(X)$; $w_3(X)$;

- Intuíció alapján átgondolva annak, hogy T_1 és T_2 milyen értéket ír X-be, nincs hatása, ugyanis T_3 felülírja X értékét. **Emiatt S_1 és S_2 X-nek is és Y-nak is ugyanazt az értéket adja.** Mivel S_1 soros ütemezés, és S_2-nek bármely adatbázis-állapotra ugyanaz a hatása, mint S_1-nek, ezért **S_2 sorbarendezhető**.

- Ugyanakkor mivel nem tudjuk felcserélni $w_1(X)$-et $w_2(X)$-szel, így cseréken keresztül nem lehet S_2-t valamelyik soros ütemezéssé átalakítani. Tehát S_2 sorbarendezhető, de **nem konfliktus-sorbarendezhető.**

Megelőzési gráfok és teszt a konfliktus-sorbarendezhetőségre

- **Alapötlet: ha valahol konfliktusban álló műveletek szerepelnek S-ben, akkor az ezeket a műveleteket végrehajtó tranzakcióknak ugyanabban a sorrendben kell előfordulniuk a konfliktus-ekvivalens soros ütemezésekben, mint ahogyan az S-ben voltak.**

- **Tehát a konfliktusban álló műveletpárok megszorítást adnak a feltételezett konfliktusekvivalens soros ütemezésben a tranzakciók sorrendjére.**

- **Ha ezek a megszorítások nem mondanak ellent egymásnak, akkor találhatunk konfliktusekvivalens soros ütemezést. Ha pedig ellentmondanak egymásnak, akkor tudjuk, hogy nincs ilyen soros ütemezés.**

Megelőzési gráfok és teszt a konfliktus-sorbarendezhetőségre

- **Adott a T_1 és T_2, esetleg további tranzakcióknak egy S ütemezése. Azt mondjuk, hogy T_1 *megelőzi* T_2-t, ha van a T_1-ben olyan A_1 művelet és a T_2-ben olyan A_2 művelet, hogy**

1. **A_1 megelőzi A_2-t S-ben,**
2. **A_1 és A_2 ugyanarra az adatbáziselemre vonatkoznak, és**
3. **A_1 és A_2 közül legalább az egyik írás művelet.**

- **Másképpen fogalmazva: A_1 és A_2 konfliktuspárt alkotna, ha szomszédos műveletek lennének. Jelölése: $T_1 <_S T_2$.**

- **Látható, hogy ezek pontosan azok a feltételek, amikor nem lehet felcserélni A_1 és A_2 sorrendjét. Tehát A_1 az A_2 előtt szerepel bármely S-sel konfliktusekvivalens ütemezésben. Ebből az következik, hogy ha ezek közül az ütemezések közül az egyik soros ütemezés, akkor abban T_1-nek meg kell előznie T_2-t.**

- **Ezeket a megelőzéseket a *megelőzési gráfban* (precedence graph) összegezhetjük. A megelőzési gráf csúcsai az S ütemezés tranzakciói. Ha a tranzakciókat T_i-vel jelöljük, akkor a T_i-nek megfelelő csúcsot az i egész jelöli. Az i csúcsból a j csúcsba akkor vezet irányított él, ha $T_i <_S T_j$.**

Lemma

S_1, S_2 konfliktusekvivalens $\Rightarrow$ gráf(S_1)=gráf(S_2)

Bizonyítás:

Tegyük fel, hogy gráf(S_1) $\neq$ gráf(S_2)

$\Rightarrow \exists$ olyan $T_i \rightarrow T_j$ él, amely gráf(S_1)-ben benne van, de gráf(S_2)-ben nincs benne, (vagy fordítva.)

$\Rightarrow$ S_1 = …$w_i(A)$…$r_j(A)$… p_i, q_j

S_2 = …$r_j(A)$…$w_i(A)$… konfliktusos pár

$\Rightarrow$ S_1, S_2 nem konfliktusekvivalens. Q.E.D.

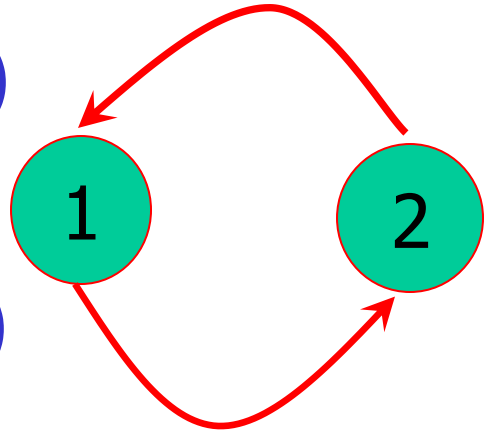
Megjegyzés:

gráf(S_1)=gráf(S_2) $\not\Rightarrow$ S_1, S_2 konfliktusekvivalens

Ellenpélda:

$S_1 = w_1(A)\ r_2(A)\ w_2(B)\ r_1(B)$

$S_2 = r_2(A)\ w_1(A)\ r_1(B)\ w_2(B)$



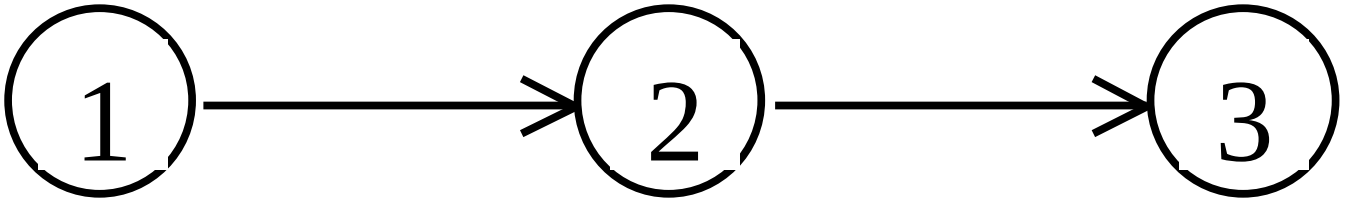
Nem lehet semmit sem cserélni!

Megelőzési gráfok és teszt a konfliktus-sorbarendezhetőségre

- ***Példa.*** **A következő S ütemezés a T_1, T_2 és T_3 tranzakciókat tartalmazza:**

S: $r_2(A); r_1(B); w_2(A); r_3(A); w_1(B); w_3(A); r_2(B); w_2(B);$

- **Az S ütemezéshez tartozó megelőzési gráf a következő:**



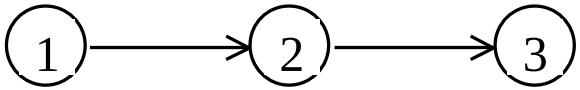
TESZT: Ha az S megelőzési gráf tartalmaz irányított kört, akkor S nem konfliktus-sorbarendezhető, ha nem tartalmaz irányított kört, akkor S konfliktus-sorbarendezhető, és a csúcsok bármelyik topologikus sorrendje megadja a konfliktusekvivalens soros sorrendet.

Megelőzési gráfok és teszt a konfliktus-sorbarendezhetőségre

- **Egy körmentes gráf csúcsainak *topologikus sorrendje* a csúcsok bármely olyan rendezése, amelyben minden a → b élre az a csúcs megelőzi a b csúcsot a topologikus rendezésben.**

S: $r_2(A); r_1(B); w_2(A); r_3(A); w_1(B); w_3(A); r_2(B); w_2(B);$

- **Az S ütemezéshez tartozó megelőzési gráf topologikus sorrendje: (T1, T2, T3).**



S′: $r_1(B); w_1(B); r_2(A); w_2(A); r_2(B); w_2(B); r_3(A); w_3(A);$

- Hogy lehet S-ből S'-t megkapni szomszédos elemek cseréjével?

1. $r_1(B); r_2(A); w_2(A); w_1(B); r_3(A); r_2(B); w_3(A); w_2(B);$
2. $r_1(B); r_2(A); w_1(B); w_2(A); r_2(B); r_3(A); w_2(B); w_3(A);$
3. $r_1(B); w_1(B); r_2(A); w_2(A); r_2(B); w_2(B); r_3(A); w_3(A);$

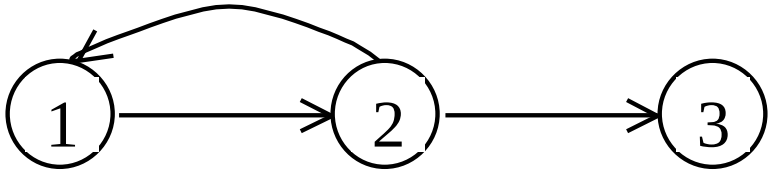
Megelőzési gráfok és teszt a konfliktus-sorbarendezhetőségre

- ***Példa.*** **Tekintsük az alábbi két ütemezést:**

S: $r_2(A); r_1(B); w_2(A); r_3(A); w_1(B); w_3(A); r_2(B); w_2(B);$

S_1: $r_2(A); r_1(B); w_2(A); r_2(B); r_3(A); w_1(B); w_3(A); w_2(B);$

- **Az** $r_2(A) \ldots w_3(A)$ **miatt** $T_2 <_{S1} T_3$.
- **Az** $r_1(B) \ldots w_2(B)$ **miatt** $T_1 <_{S1} T_2$.
- **Az** $r_2(B) \ldots w_1(B)$ **miatt** $T_2 <_{S1} T_1$.
- **Az** S_1 **ütemezéshez tartozó megelőzési gráf a következő:**



- **Ez a gráf nyilvánvalóan tartalmaz kört, ezért** S_1 **nem konfliktus-sorbarendezhető, ugyanis láthatjuk, hogy bármely konfliktusekvivalens soros ütemezésben** T_1**-nek** T_2 **előtt is és után is kellene állnia, tehát nem létezik ilyen ütemezés.**

Miért működik a megelőzési gráfon alapuló tesztelés?

Ha van kör a gráfban, akkor cserékkel nem lehet soros ütemezésig eljutni.

Ha létezik a $T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_n \rightarrow T_1$ n darab tranzakcióból álló kör, akkor a feltételezett soros sorrendben T_1 műveleteinek meg kell előzniük a T_2-ben szereplő műveleteket, amelyeknek meg kell előzniük a T_3-belieket és így tovább egészen T_n-ig. De T_n műveletei emiatt a T_1-beliek mögött vannak, ugyanakkor meg is kellene előzniük a T_1-belieket a $T_n \rightarrow T_1$ él miatt. Ebből következik, hogy ha a megelőzési gráf tartalmaz kört, akkor az ütemezés nem konfliktus-sorbarendezhető.

Miért működik a megelőzési gráfon alapuló tesztelés?

Ha nincs kör a gráfban, akkor cserékkel el lehet jutni egy soros ütemezésig.

A bizonyítás az ütemezésben részt vevő tranzakciók száma szerinti indukcióval történik:

Alapeset: Ha n = 1, vagyis csak egyetlen tranzakcióból áll az ütemezés, akkor az már önmagában soros, tehát konfliktus-sorbarendezhető.

Indukció: Legyen S a $T_1, T_2, \ldots, T_n$ n darab tranzakció műveleteiből álló ütemezés, és S-nek körmentes megelőzési gráfja van.

Ha egy véges gráf körmentes, akkor van egy olyan csúcsa, amelybe nem vezet él. Legyen a T_i egy ilyen csúcs.

Mivel az i csomópontba nem vezet él, ezért nincs S-ben olyan A művelet, amely

1. **valamelyik T_j ($i \neq j$) tranzakcióra vonatkozik,**
2. **T_i valamely műveletét megelőzi, és**
3. **ezzel a művelettel konfliktusban van.**

Így T_i minden műveletét S legelejére mozgathatjuk át, miközben megtartjuk a sorrendjüket:

S1: (T_i műveletei) (a többi $n-1$ tranzakció műveletei)

Miért működik a megelőzési gráfon alapuló tesztelés?

Ha nincs kör a gráfban, akkor cserékkel el lehet jutni egy soros ütemezésig.
(Folytatás)

S1: (T_i műveletei) (a többi $n-1$ tranzakció műveletei)

Most tekintsük S1 második részét. Mivel ezek a műveletek egymáshoz viszonyítva ugyanabban a sorrendben vannak, mint ahogyan S-ben voltak, ennek a második résznek a megelőzési gráfját megkapjuk S megelőzési gráfjából, ha elhagyjuk belőle az i csúcsot és az ebből a csúcsból kimenő éleket.

Mivel az eredeti körmentes volt, az elhagyás után is az marad, azaz a második rész megelőzési gráfja is körmentes.

Továbbá, mivel a második része n-1 tranzakciót tartalmaz, alkalmazzuk rá az indukciós feltevést.

Így tudjuk, hogy a második rész műveletei szomszédos műveletek szabályos cseréivel átrendezhetők soros ütemezéssé. Ily módon magát S-et alakítottuk át olyan soros ütemezéssé, amelyben T_i műveletei állnak legelöl, és a többi tranzakció műveletei ezután következnek valamilyen soros sorrendben. Q.E.D.

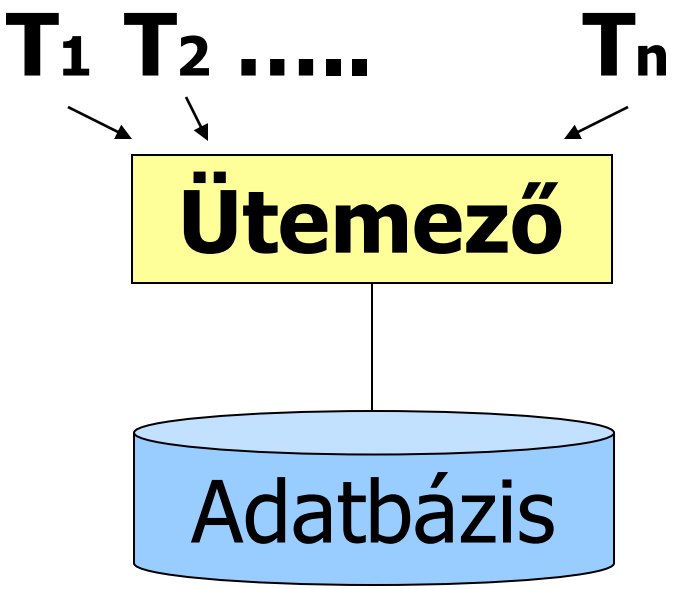
Az ütemező eszközei a sorbarendezhetőség elérésére

Passzív módszer:

- **hagyjuk a rendszert működni,**
- **az ütemezésnek megfelelő gráfot tároljuk,**
- **egy idő után megnézzük, hogy van-e benne kör,**
- **és ha nincs, akkor szerencsénk volt, jó volt az ütemezés.**

Az ütemező eszközei a sorbarendezhetőség elérésére

Aktív módszer: **az ütemező beavatkozik, és megakadályozza, hogy kör alakuljon ki.**



Az ütemező eszközei a sorbarendezhetőség elérésére

- Az ütemezőnek több lehetősége is van arra, hogy kikényszerítse a sorbarendezhető ütemezéseket:

1. zárak (ezen belül is még: protokoll elemek, pl. 2PL)
2. időbélyegek (time stamp)
3. érvényesítés

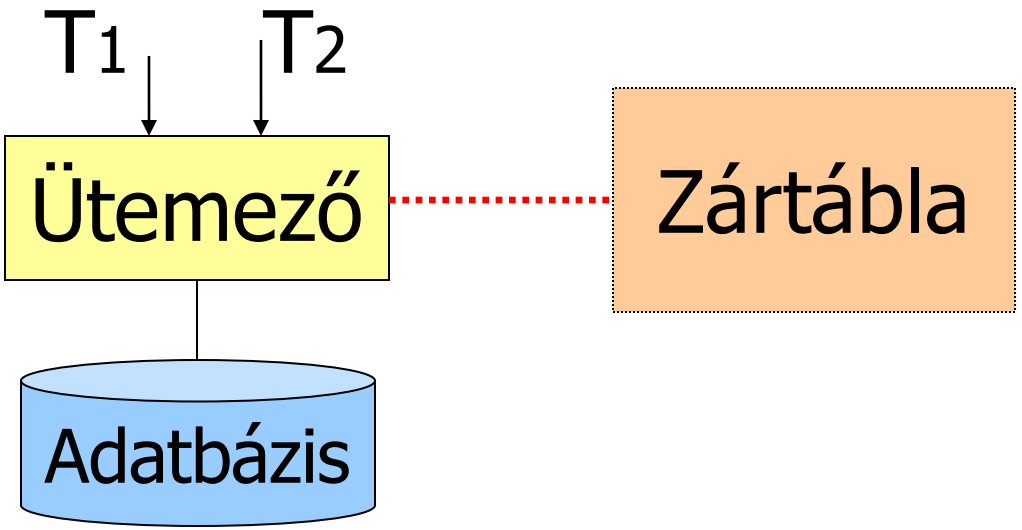
- Fő elv: inkább legyen szigorúbb és ne hagyjon lefutni egy olyan ütemezést, ami sorbarendezhető, mint hogy fusson egy olyan, ami nem az.

A sorbarendezhetőség biztosítása zárakkal

Két új műveletet vezetünk be:

- $l_i(A)$: kizárólagos zárolás (exclusive lock)
- $u_i(A)$: a zár elengedése (unlock)

A tranzakciók zárolják azokat az adatbáziselemeket, amelyekhez hozzáférnek, hogy megakadályozzák azt, hogy ugyanakkor más tranzakciók is hozzáférjenek ezekhez az elemekhez, mivel ekkor felmerülne a nem sorbarendezhetőség kockázata.



A zárak használata

- **A *zárolási ütemező* a konfliktus-sorbarendezhetőséget követeli meg, (ez erősebb követelmény, mint a sorbarendezhetőség).**

***Tranzakciók konzisztenciája* (consistency of transactions):**

1. **A tranzakció csak akkor olvashat vagy írhat egy elemet, ha már korábban zárolta azt, és még nem oldotta fel a zárat.**
2. **Ha egy tranzakció zárol egy elemet, akkor később azt fel kell szabadítania.**

***Az ütemezések jogszerűsége* (legality of schedules):**

1. **nem zárolhatja két tranzakció ugyanazt az elemet, csak úgy, ha az egyik előbb már feloldotta a zárat.**

A zárak használata

- **Kibővítjük a jelöléseinket a zárolás és a feloldás műveletekkel:**
- **$l_i(X)$: a T_i tranzakció az X adatbáziselemre *zárolást kér* (lock).**
- **$u_i(X)$: a T_i tranzakció az X adatbáziselem *zárolását feloldja* (unlock).**
- **Konzisztencia:**
- **Ha egy T_i tranzakcióban van egy $r_i(X)$ vagy egy $w_i(X)$ művelet, akkor van korábban egy $l_i(X)$ művelet, és van később egy $u_i(X)$ művelet, de a zárolás és az írás/olvasás között nincs $u_i(X)$.**

Ti: … $l_i(X)$ … $r/w_i(X)$ … $u_i(X)$ …

A zárak használata

- **Jogszerűség:**
- **Ha egy ütemezésben van olyan $l_i(X)$ művelet, amelyet $l_j(X)$ követ, akkor e két művelet között lennie kell egy $u_i(X)$ műveletnek.**

$$S = \ldots\ldots\ldots l_i(X) \ldots\ldots\ldots\ldots u_i(X) \ldots\ldots\ldots$$

nincs $l_j(X)$

A zárak használata

T_1: $l_1(A)$; $r_1(A)$; A := A+100; $w_1(A)$; $u_1(A)$;
$l_1(B)$; $r_1(B)$; B := B+100; $w_1(B)$; $u_1(B)$;

T_2: $l_2(A)$; $r_2(A)$; A := A*2; $w_2(A)$; $u_2(A)$;
$l_2(B)$; $r_2(B)$; B := B*2; $w_2(B)$; $u_2(B)$;

Mindkét tranzakció konzisztens.

A zárak használata

T_1	T_2	A	B
$l_1(A)$; $r_1(A)$;		25	
A := A+100;			
$w_1(A)$; $u_1(A)$;		125	
	$l_2(A)$; $r_2(A)$;	125	
	A := A*2;		
	$w_2(A)$; $u_2(A)$;	250	
	$l_2(B)$; $r_2(B)$;		25
	B := B*2;		
	$w_2(B)$; $u_2(B)$;		50
$l_1(B)$; $r_1(B)$;			50
B := B+100;			
$w_1(B)$; $u_1(B)$;			150

Ez az ütemezés jogszerű,

de nem sorba rendezhető.

A zárolási ütemező

- **A zároláson alapuló ütemező feladata, hogy akkor és csak akkor engedélyezze a kérések végrehajtását, ha azok jogszerű ütemezéseket eredményeznek.**

- **Ezt a döntést segíti a zártábla, amely minden adatbáziselemhez megadja azt a tranzakciót, ha van ilyen, amelyik pillanatnyilag zárolja az adott elemet.**

- **A zártábla szerkezete (egyféle zárolás esetén): Zárolások(elem, tranzakció) relációt, ahol a T tranzakció zárolja az X adatbáziselemet.**

A zárolási ütemező

T_1: $l_1(A)$; $r_1(A)$; A := A+100; $w_1(A)$; $l_1(B)$;

$u_1(A)$; $r_1(B)$; B := B+100; $w_1(B)$; $u_1(B)$;

T_2: $l_2(A)$; $r_2(A)$; A := A*2; $w_2(A)$; $l_2(B)$;

$u_2(A)$; $r_2(B)$; B := B*2; $w_2(B)$; $u_2(B)$;

T_1	T_2	A	B
$l_1(A)$; $r_1(A)$;		25	
A := A+100;			
$w_1(A)$; $l_1(B)$; $u_1(A)$;		125	
	$l_2(A)$; $r_2(A)$;	125	
	A := A*2;		
	$w_2(A)$;	250	
	$l_2(B)$; elutasítva		
$r_1(B)$; B := B+100;			25
$w_1(B)$; $u_1(B)$;			125
	$l_2(B)$; $u_2(A)$; $r_2(B)$;		125
	B := B*2;		
	$w_2(B)$; $u_2(B)$;		250

Mivel T_2-nek várakoznia kellett, ezért B-t akkor szorozza meg 2-vel, miután T_1 már hozzáadott 100-at, és ez konzisztens adatbázis-állapotot eredményez.

A kétfázisú zárolás (two-phase locking, 2PL)

T_i = ... $l_i(A)$ ..... $u_i(A)$ .....

nincs unlock

nincs lock

Minden tranzakcióban minden zárolási művelet megelőzi az összes zárfeloldási műveletet.

A kétfázisú zárolás (two-phase locking, 2PL)

T_1	T_2	A	B
$l_1(A)$; $r_1(A)$;		25	
A := A+100;			
$w_1(A)$; $u_1(A)$;		125	
	$l_2(A)$; $r_2(A)$;	125	
	A := A*2;		
	$w_2(A)$; $u_2(A)$;	250	
	$l_2(B)$; $r_2(B)$;		25
	B := B*2;		
	$w_2(B)$; $u_2(B)$;		50
$l_1(B)$; $r_1(B)$;			50
B := B+100;			
$w_1(B)$; $u_1(B)$;			150

Ez az ütemezés jogszerű,

de nem sorba rendezhető.

A tranzakciók nem kétfázisúak!

A kétfázisú zárolás (two-phase locking, 2PL)

T_1: $l_1(A)$; $r_1(A)$; A := A+100; $w_1(A)$; $l_1(B)$;

$u_1(A)$; $r_1(B)$; B := B+100; $w_1(B)$; $u_1(B)$;

T_2: $l_2(A)$; $r_2(A)$; A := A*2; $w_2(A)$; $l_2(B)$;

$u_2(A)$; $r_2(B)$; B := B*2; $w_2(B)$; $u_2(B)$;

T_1	T_2	A	B
$l_1(A)$; $r_1(A)$;		25	
A := A+100;			
$w_1(A)$; $l_1(B)$; $u_1(A)$;		125	
	$l_2(A)$; $r_2(A)$;	125	
	A := A*2;		
	$w_2(A)$;	250	
	$l_2(B)$; elutasítva		
$r_1(B)$; B := B+100;			25
$w_1(B)$; $u_1(B)$;			125
	$l_2(B)$; $u_2(A)$; $r_2(B)$;		125
	B := B*2;		
	$w_2(B)$; $u_2(B)$;		250

Elérhető egy konzisztens adatbázis-állapotot eredményező ütemezés.

A tranzakciók kétfázisúak!

A kétfázisú zárolás (two-phase locking, 2PL)

Tétel: **Konzisztens, kétfázisú zárolású tranzakciók bármely S jogszerű ütemezését át lehet alakítani konfliktusekvivalens soros ütemezéssé.**

Bizonyítás: **S-ben részt vevő tranzakciók száma (n) szerinti indukcióval.**

Megjegyzés:

A konfliktusekvivalencia csak az olvasási és írási műveletekre vonatkozik: Amikor felcseréljük az olvasások és írások sorrendjét, akkor figyelmen kívül hagyjuk a zárolási és zárfeloldási műveleteket. Amikor megkaptuk az olvasási és írási műveletek sorrendjét, akkor úgy helyezzük el köréjük a zárolási és zárfeloldási műveleteket, ahogyan azt a különböző tranzakciók megkövetelik. Mivel minden tranzakció felszabadítja az összes zárolást a tranzakció befejezése előtt, tudjuk, hogy a soros ütemezés jogszerű lesz.

A kétfázisú zárolás

Tétel: Konzisztens, kétfázisú zárolású tranzakciók bármely S jogszerű ütemezését át lehet alakítani konfliktusekvivalens soros ütemezéssé.

Bizonyítás:

Alapeset: Ha n = 1, azaz egy tranzakcióból áll az ütemezés, akkor az már önmagában soros, tehát biztosan konfliktus-sorbarendezhető.

Indukció: Legyen S a $T_1, T_2, \ldots, T_n$ n darab konzisztens, kétfázisú zárolású tranzakció műveleteiből álló ütemezés, és legyen T_i az a tranzakció, amelyik a teljes S ütemezésben a **legelső zárfeloldási műveletet** végzi, mondjuk $u_i(X)$-t.

Azt állítjuk, hogy T_i **összes olvasási és írási műveletét az ütemezés legelejére tudjuk vinni** anélkül, hogy konfliktusműveleteken kellene áthaladnunk.

A kétfázisú zárolás

Vegyünk egy konfliktusos párt, $w_i(Y)$-t és $w_j(Y)$-t.
(Hasonlóan látható be más konfliktusos párra is.)

Tekintsük T_i valamelyik műveletét, mondjuk $w_i(Y)$-t. Megelőzheti-e ezt S-ben valamely konfliktusművelet, például $w_j(Y)$? Ha így lenne, akkor az S ütemezésben az $u_j(Y)$ és az $l_i(Y)$ műveletek az alábbi módon helyezkednének el:

$$\ldots; w_j(Y); \ldots; u_j(Y); \ldots; l_i(Y); \ldots; w_i(Y); \ldots$$

Mivel T_i az első, amelyik zárat old fel, így S-ben $u_i(X)$ megelőzi $u_j(Y)$-t, vagyis S a következőképpen néz ki:

$$\ldots; w_j(Y); \ldots; u_i(X); \ldots; u_j(Y); \ldots; l_i(Y); \ldots; w_i(Y); \ldots$$

Az $u_i(X)$ művelet állhat $w_j(Y)$ előtt is. Mindkét esetben $u_i(X)$ $l_i(Y)$ előtt van, ami azt jelenti, hogy T_i nem kétfázisú zárolású, amint azt feltételeztük.

A kétfázisú zárolás

Bebizonyítottuk, hogy S legelejére lehet vinni T_i összes műveletét konfliktusmentes olvasási és írási műveletekből álló műveletpárok cseréjével.

Ezután elhelyezhetjük T_i zárolási és zárfeloldási műveleteit. Így S a következő alakba írható át:

(T_i műveletei) (a többi n-1 tranzakció műveletei)

Az n-1 tranzakcióból álló második rész szintén konzisztens 2PL tranzakciókból álló jogszerű ütemezés, így alkalmazhatjuk rá az indukciós feltevést. Átalakítjuk a második részt konfliktusekvivalens soros ütemezéssé, így a teljes S konfliktus-sorbarendezhetővé vált.

Q.E.D.

A holtpont kockázata

T_1: $l_1(A); r_1(A); A := A+100; w_1(A); l_1(B);$
$u_1(A); r_1(B); B := B+100; w_1(B); u_1(B);$

T_2: $l_2(B); r_2(B); B := B*2; w_2(B); l_2(A);$
$u_2(B); r_2(A); A := A*2; w_2(A); u_2(A);$

T_1	T_2	A	B
$l_1(A)$; $r_1(A)$;		25	
	$l_2(B)$; $r_2(B)$;		25
A := A+100;			
	B := B*2;		
$w_1(A)$;		125	
	$w_2(B)$;		50
$l_1(B)$; elutasítva	$l_2(A)$; elutasítva		

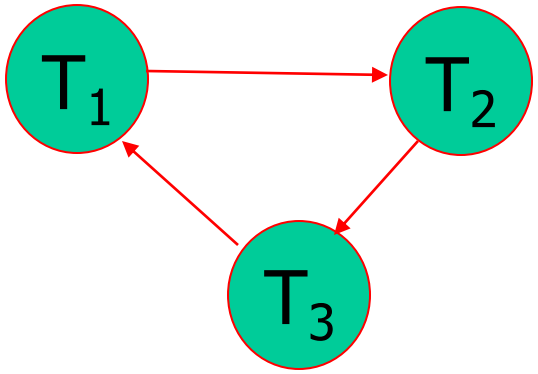
Egyik tranzakció sem folytatódhat, hanem örökké várakozniuk kell.

Nem tudjuk mind a két tranzakciót folytatni, ugyanis ha így lenne, akkor az adatbázis végső állapotában nem lehetne A=B.

Holtpont felismerése

- **A felismerésben segít a zárkérések sorozatához tartozó várakozási gráf: csúcsai a tranzakciók és akkor van él T_i-ből T_j-be, ha T_i vár egy olyan zár elengedésére, amit T_j tart éppen.**
- A várakozási gráf változik az ütemezés során, ahogy újabb zárkérések érkeznek vagy zárelengedések történnek, vagy az ütemező abortáltat egy tranzakciót.
- $l_1(A)$; $l_2(B)$; $l_3(C)$; $l_1(B)$; $l_2(C)$; $l_3(A)$

Az ütemezésnek megfelelő várakozási gráf:



Holtpont felismerése

- **Tétel**. ***Az ütemezés során egy adott pillanatban pontosan akkor nincs holtpont, ha az adott pillanathoz tartozó várakozási gráfban nincs irányított kör.***

  Bizonyítás: Ha van irányított kör a várakozási gráfban, akkor a körbeli tranzakciók egyike se tud lefutni, mert vár a mellette levőre. Ez holtpont.

  Ha a gráfban nincs irányított kör, akkor van topológikus rendezése a tranzakcióknak és ebben a sorrendben le tudnak futni a tranzakciók. (Az első nem vár senkire, mert nem megy belőle ki él, így lefuthat; ezután már a másodikba se megy él, az is lefuthat, és így tovább). Q.E.D.

Megoldások holtpont ellen

1. Rajzoljuk folyamatosan a várakozási gráfot és ha holtpont alakul ki, akkor ABORT-áljuk az egyik olyan tranzakciót, aki benne van a kialakult irányított körben.

   Ez egy megengedő megoldás (optimista), hagyja az ütemező, hogy mindenki úgy kérjen zárat, ahogy csak akar, de ha baj van, akkor erőszakosan beavatkozik. Az előző példa esetében mondjuk kilövi T_2-t, ettől lefuthat T_1, majd T_2 is.

2. Pesszimista hozzáállás: ha hagyjuk, hogy mindenki össze-vissza kérjen zárat, abból baj lehet. Előzzük inkább meg a holtpont kialakulását valahogyan. Lehetőségek:

(a) Minden egyes tranzakció előre elkéri az összes zárat, ami neki kelleni fog. Ha nem kapja meg az összeset, akkor egyet se kér el, el se indul.

   Ilyenkor biztos nem lesz holtpont, mert ha valaki megkap egy zárat, akkor le is tud futni, nem akad el. Az csak a baj ezzel, hogy előre kell mindent tudni.

(b) Feltesszük, hogy van egy sorrend az adategységeken és minden egyes tranzakció csak eszerint a sorrend szerint növekvően kérhet újabb zárakat. Itt lehet, hogy lesz várakozás, de holtpont biztos nem lesz.

Megoldások holtpont ellen

Tegyük fel, hogy T_1,..., T_n irányított kört alkot, ahol T_i vár T_{i+1}-re az A_i adatelem miatt.

Ha mindegyik tranzakció betartotta, hogy egyre nagyobb indexű adatelemre kért zárat,

akkor $A1 < A2 < A3 < An < A1$ áll fenn, ami ellentmondás.

Tehát ez a protokoll is megelőzi a holtpontot, de itt is előre kell tudni, hogy milyen zárakat fog kérni egy tranzakció.

Még egy módszer, ami szintén optimista, mint az első:

Időkorlát alkalmazása: ha egy tranzakció kezdete óta túl sok idő telt el, akkor ABORT-áljuk.

Ehhez az kell, hogy ezt az időkorlátot jól tudjuk megválasztani.

Éhezés

Másik probléma, ami zárakkal kapcsolatban előfordulhat:

éhezés: többen várnak ugyanarra a zárra, de amikor felszabadul mindig elviszi valaki a tranzakció orra elől.

Megoldás: adategységenként FIFO listában tartani a várakozókat, azaz mindig a legrégebben várakozónak adjuk oda a zárolási lehetőséget.

Különböző zármódú zárolási rendszerek

Probléma: **a T tranzakciónak akkor is zárolnia kell az X adatbáziselemet, ha csak olvasni akarja X-et, írni nem.**

- **Ha nem zárolnánk, akkor esetleg egy másik tranzakció azalatt írna X-be új értéket, mialatt T aktív, ami nem sorba rendezhető viselkedést okoz.**
- **Másrészről pedig miért is ne olvashatná több tranzakció egyidejűleg X értékét mindaddig, amíg egyiknek sincs engedélyezve, hogy írja.**

Osztott és kizárólagos zárak

Mivel ugyanannak az adatbáziselemnek két olvasási művelete nem eredményez konfliktust, így ahhoz, hogy az olvasási műveleteket egy bizonyos sorrendbe soroljuk, nincs szükség zárolásra.

Viszont szükséges azt az elemet is zárolni, amelyet olvasunk, mert ennek az elemnek az írását nem szabad közben megengednünk.

Az íráshoz szükséges zár viszont „erősebb”, mint az olvasáshoz szükséges zár, mivel ennek mind az olvasásokat, mind az írásokat meg kell akadályoznia.

Osztott és kizárólagos zárak

A legelterjedtebb zárolási séma két különböző zárat alkalmaz: az *osztott zárakat* (shared locks) vagy *olvasási zárakat*, és a *kizárólagos zárakat* (exclusive locks) vagy *írási zárakat*.

Tetszőleges X adatbáziselemet vagy egyszer lehet zárolni kizárólagosan, vagy akárhányszor lehet zárolni osztottan, ha még nincs kizárólagosan zárolva.

Amikor írni akarjuk X-et, akkor X-en kizárólagos zárral kell rendelkeznünk, de ha csak olvasni akarjuk, akkor X-en akár osztott, akár kizárólagos zár megfelel.

Feltételezzük, hogy ha olvasni akarjuk X-et, de írni nem, akkor előnyben részesítjük az osztott zárolást.

Osztott és kizárólagos zárak

Az $sl_i(X)$ jelölést használjuk arra, hogy a T_i tranzakció osztott zárat kér az X adatbáziselemre, az $xl_i(X)$ jelölést pedig arra, hogy a T_i kizárólagos zárat kér X-re.

Továbbra is $u_i(X)$-szel jelöljük, hogy T_i feloldja X zárását, vagyis felszabadítja X-et minden zár alól.

A tranzakciók konzisztenciája, a tranzakciók 2PL feltétele és az ütemezések jogszerűsége

1. *Tranzakciók konzisztenciája:* **Nem írhatunk kizárólagos zár fenntartása nélkül, és nem olvashatunk valamilyen zár fenntartása nélkül.**

Pontosabban fogalmazva: bármely T_i tranzakcióban

a) az $r_i(X)$ olvasási műveletet meg kell, hogy előzze egy $sl_i(X)$ vagy egy $xl_i(X)$ úgy, hogy közben nincs $u_i(X)$;

$sl_i(X) \ldots . r_i(X)$ vagy $xl_i(X) \ldots . r_i(X)$

b) a $w_i(X)$ írási műveletet meg kell, hogy előzze egy $xl_i(X)$ úgy, hogy közben nincs $u_i(X)$.

$xl_i(X) \ldots . w_i(X)$

Minden zárolást követnie kell egy ugyanannak az elemnek a zárolását feloldó műveletnek.

A tranzakciók konzisztenciája, a tranzakciók 2PL feltétele és az ütemezések jogszerűsége

2. *Tranzakciók kétfázisú zárolása:* **A zárolásoknak meg kell előzniük a zárak feloldását.**

Pontosabban fogalmazva: bármely T_i kétfázisú zárolású tranzakcióban egyetlen $sl_i(X)$ vagy $xl_i(X)$ műveletet sem előzhet meg egyetlen $u_i(Y)$ művelet sem semmilyen Y-ra.

$$sl_i(X) \ldots u_i(Y)$$

vagy $$xl_i(X) \ldots u_i(Y)$$

A tranzakciók konzisztenciája, a tranzakciók 2PL feltétele és az ütemezések jogszerűsége

3. ***Az ütemezések jogszerűsége:*** **Egy elemet vagy egyetlen tranzakció zárol kizárólagosan, vagy több is zárolhatja osztottan, de a kettő egyszerre nem lehet. Pontosabban fogalmazva:**

a) **Ha $xl_i(X)$ szerepel egy ütemezésben, akkor ezután nem következhet $xl_j(X)$ vagy $sl_j(X)$ valamely i-től különböző j-re anélkül, hogy közben ne szerepelne $u_i(X)$.**

$$xl_i(X) \ldots u_i(X) \ldots xl_j(X)$$

$$\text{vagy } xl_i(X) \ldots u_i(X) \ldots sl_j(X)$$

b) **Ha $sl_i(X)$ szerepel egy ütemezésben, akkor ezután nem következhet $xl_j(X)$ valamely i-től különböző j-re anélkül, hogy közben ne szerepelne $u_i(X)$.**

$$sl_i(X) \ldots u_i(X) \ldots xl_j(X)$$

A tranzakciók konzisztenciája, a tranzakciók 2PL feltétele és az ütemezések jogszerűsége

Megjegyzések:

1. **Az engedélyezett, hogy egy tranzakció ugyanazon elemre kérjen és tartson mind osztott, mind kizárólagos zárat, feltéve, hogy ezzel nem kerül konfliktusba más tranzakciók zárolásaival.** $sl_i(X) \ldots xl_i(X)$

2. **Ha a tranzakciók előre tudnák, milyen zárakra lesz szükségük, akkor biztosan csak a kizárólagos zárolást kérnének a fenti esetben, de ha nem láthatók előre a zárolási igények, lehetséges, hogy egy tranzakció osztott és kizárólagos zárakat is kér különböző időpontokban.**

A tranzakciók konzisztenciája, a tranzakciók 2PL feltétele és az ütemezések jogszerűsége

T_1: $sl_1(A); r_1(A); xl_1(B); r_1(B); w_1(B); u_1(A); u_1(B);$

T_2: $sl_2(A); r_2(A); sl_2(B); r_2(B); u_2(A); u_2(B);$

T_1	T_2
$sl_1(A);\ r_1(A);$	
	$sl_2(A);$
	$r_2(A);$
	$sl_2(B);$
	$r_2(B);$
$xl_1(B);$ elutasítva	
	$u_2(A);$
	$u_2(B);$
$xl_1(B);\ r_1(B);\ w_1(B);$	
$u_1(A);\ u_1(B);$	

Az ütemezés konfliktus-sorbarendezhető.

A konfliktusekvivalens soros sorrend a (T_2, T_1), hiába kezdődött T_1 előbb.

A tranzakciók konzisztenciája, a tranzakciók 2PL feltétele és az ütemezések jogszerűsége

Tétel: **Konzisztens 2PL tranzakciók jogszerű ütemezése konfliktus-sorbarendezhető.**

Bizonyítás: **Ugyanazok a meggondolások alkalmazhatók az osztott és kizárólagos zárakra is, mint korábban. Q.E.D.**

Megjegyzés: **Az ábrán T_2 előbb old fel zárat, mint T_1, így azt várjuk, hogy T_2 megelőzi T_1-et a soros sorrendben. Megvizsgálva az olvasási és írási műveleteket, észrevehető, hogy $r_1(A)$-t T_2 összes műveletén át ugyan hátra tudjuk cserélgetni, de $w_1(B)$-t nem tudjuk $r_2(B)$ elé vinni, ami pedig szükséges lenne ahhoz, hogy T_1 megelőzze T_2-t egy konfliktusekvivalens soros ütemezésben.**

Kompatibilitási mátrixok

- **A *kompatibilitási mátrix* minden egyes zármódhoz rendelkezik egy-egy sorral és egy-egy oszloppal.**
- **A sorok egy másik tranzakció által az X elemre elhelyezett záraknak, az oszlopok pedig az X-re kért zármódoknak felelnek meg.**
- **A kompatibilitási mátrix használatának szabálya:**

  Egy A adatbáziselemre C módú zárat akkor és csak akkor engedélyezhetünk, ha a táblázat minden olyan R sorára, amelyre más tranzakció már zárolta A-t R módban, a C oszlopban „igen" szerepel.

- **Az osztott (S) és kizárólagos (X) zárak kompatibilitási mátrixa:**

Megkaphatjuk-e ezt a típusú zárat?

Ha ilyen zár van már kiadva

	S	X
S	igen	nem
X	nem	nem

Kompatibilitási mátrixok használata

1. A mátrix alapján dönti el az ütemező, hogy egy ütemezés/zárkérés legális-e, illetve ez alapján várakoztatja a tranzakciókat. Minél több az Igen a mátrixban, annál kevesebb lesz a várakoztatás.

2. A mátrix alapján keletkező várakozásokhoz elkészített várakozási gráf segítségével az ütemező kezeli a holtpontot.

3. A mátrix alapján készíti el az ütemező a megelőzési gráfot egy zárkérés-sorozathoz:

- a megelőzési gráf csúcsai a tranzakciók és akkor van él T_i-ből T_j-be, ha van olyan A adategység, amelyre az ütemezés során Z_k zárat kért és kapott T_i , ezt elengedte, majd
- ezután A-ra legközelebb T_j kért és kapott olyan Z_l zárat, hogy a mátrixban a Z_k sor Z_l oszlopában Nem áll.

Vagyis olyankor lesz él, ha a két zár nem kompatibilis egymással, nem mindegy a két művelet sorrendje.

..... $T_i:Z_k(A)$ .... $T_i:UZ_k(A)$ .... $T_j:Z_l(A)$...

nem kompatibilis

Kompatibilitási mátrixok használata

- **A sorbarendezhetőséget a megelőzési gráf segítségével lehet eldönteni.**

- **Tétel.** ***Egy csak zárkéréseket és zárelengedéseket tartalmazó jogszerű ütemezés sorbarendezhető akkor és csak akkor, ha a kompatibilitási mátrix alapján felrajzolt megelőzési gráf nem tartalmaz irányított kört.***

Bizonyítás:

Ha van irányított kör, akkor ekvivalens soros ütemezésben is ugyanebben a sorrendben következnének a körben szereplő tranzakciók, ami ellentmondás.

A másik irány indukcióval. A topológikus sorrendben első tranzakció lépéseit előre mozgathatjuk, a hatás nem változik, és a maradék tranzakciók indukció miatt ekvivalensek egy soros ütemezéssel. Q.E.D.

A zárakra vonatkozó megelőzési gráf körmentességi feltétel erőssége

Tekintsük az

$l_1(A); r_1(A); u_1(A); l_2(A); r_2(A); u_2(A); l_1(A); w_1(A); u_1(A); l_2(B); r_2(B); u_2(B)$

ütemezést.

Ha megnézzük az írás/olvasás műveleteket ($r_1(A); r_2(A); w_1(A); r_2(B)$), akkor látszik, hogy az ütemezés hatása azonos a T_2T_1 soros ütemezés hatásával, vagyis ez egy sorbarendezhető ütemezés zárak nélkül.

De ha felrajzoljuk a zárakra vonatkozó megelőzési gráfot (és ilyenkor persze nem nézzük, hogy milyen írások/olvasások vannak, hanem a legrosszabb esetre készülünk), akkor

lesz a gráf, és mivel ez irányított kört tartalmaz, akkor ezt elvetnénk, mert nem lesz sorbarendezhető az az ütemezés, amiben már csak a zárak vannak benne.

A sorbarendezhetőség biztosítása tetszőleges zármodellben

Az ütemező egyik lehetősége a sorbarendezhetőség elérésére, hogy folyamatosan figyeli a megelőzési gráfot és ha irányított kör keletkezne, akkor ABORT-ot rendel el.

Másik lehetőség a protokollal való megelőzés. Tetszőleges zármodellben értelmes a 2PL és igaz az alábbi tétel:

Tétel. ***Ha valamilyen zármodellben egy jogszerű ütemezésben minden tranzakció követi a 2PL-t, akkor az ütemezéshez tartozó megelőzési gráf nem tartalmaz irányított kört, azaz az ütemezés sorbarendezhető.***

Bizonyítás: **Az előzőekhez hasonlóan.**

Megjegyzés: Minél gazdagabb a zármodell, minél több az IGEN a kompatibilitási mátrixban, annál valószínűbb, hogy a megelőzési gráfban nincs irányított kör, minden külön protokoll nélkül. Ez azt jelenti, ilyenkor egyre jobb lesz az ABORT-os módszer (azaz ritkán kell visszagörgetni egy tranzakciót).

Zárak felminősítése

- **L2 *erősebb* L1-nél, ha a kompatibilitási mátrixban L2 sorában /oszlopában minden olyan pozícióban „NEM” áll, amelyben L1 sorában /oszlopában „NEM” áll.**
- **Például az SX zárolási séma esetén X erősebb S-nél**

  (X minden zármódnál erősebb, hiszen X sorában és oszlopában is minden pozíción „NEM” szerepel).

	S	X
S	IGEN	NEM
X	NEM	NEM

- **Azt mondjuk, hogy a T tranzakció *felminősíti* (upgrade) az L1 zárját az L1-nél erősebb L2 zárra az A adatbáziselemen, ha L2 zárat kér (és kap) A-ra, amelyen már birtokol egy L1 zárat (azaz még nem oldotta fel L1-et). Például: $sl_i(A)$.....$Xl_i(A)$**

Zárak felminősítése

T_1: $sl_1(A); r_1(A); sl_1(B); r_1(B); xl_1(B); w_1(B); u_1(A); u_1(B);$

T_2: $sl_2(A); r_2(A); sl_2(B); r_2(B); u_2(A); u_2(B);$

T_1	T_2
$sl_1(A);\ r_1(A);$	
	$sl_2(A);\ r_2(A);$
	$sl_2(B);\ r_2(B);$
$xl_1(B);$ elutasítva	
	$u_2(A);\ u_2(B);$
$xl_1(B);\ r_1(B);$	
$w_1(B);$	
$u_1(A);\ u_1(B);$	

Felminősítés nélkül

T_1	T_2
$sl_1(A);\ r_1(A);$	
	$sl_2(A);\ r_2(A);$
	$sl_2(B);\ r_2(B);$
$sl_1(B);\ r_1(B);$	
$xl_1(B);$ elutasítva	
	$u_2(A);\ u_2(B);$
$xl_1(B);\ w_1(B);$	
$u_1(A);\ u_1(B);$	

Felminősítéssel

A T_1 tranzakció T_2-vel konkurensen tudja végrehajtani az írás előtti, esetleg hosszadalmas számításait, amely nem lenne lehetséges, ha T_1 kezdetben kizárólagosan zárolta volna B-t.

Új típusú holtpontok felminősítés esetén

T_1	T_2
$sl_1(A)$;	
	$sl_2(A)$;
$xl_1(A)$; elutasítva	
	$xl_2(A)$; elutasítva

Egyik tranzakció végrehajtása sem folytatódhat:

- **vagy mindkettőnek örökösen kell várakoznia,**
- **vagy addig kell várakozniuk, amíg a rendszer fel nem fedezi, hogy holtpont alakult ki, abortálja valamelyik tranzakciót, és a másiknak engedélyezi `A`-ra a kizáró-lagos zárat.**
- **Megoldás: módosítási zárak**

Módosítási zárak

- Az $ul_i(X)$ **módosítási zár** a T_i tranzakciónak csak X olvasására ad jogot, X írására nem. Később azonban csak a módosítási zárat lehet felminősíteni írásira, az olvasási zárat nem (azt csak módosításira).
- **A módosítási zár tehát nem csak a holtpontproblémát oldja meg, hanem a kiéheztetés problémáját is.**

	S	X	U
S	igen	nem	igen
X	nem	nem	nem
U	nem	nem	nem

NEM SZIMMETRIKUS!

Az U módosítási zár úgy néz ki, mintha osztott zár lenne, amikor kérjük, és úgy néz ki, mintha kizárólagos zár lenne, amikor már megvan.

Módosítási zárak

T_1: $ul_1(A); r_1(A); xl_1(A); w_1(A); u_1(A);$

T_2: $ul_2(A); r_2(A); xl_2(A); w_2(A); u_2(A);$

T_1	T_2
$ul_1(A);\ r_1(A);$	
	$ul_2(A);$ elutasítva
$xl_1(A);\ w_1(A);\ u_1(A);$	
	$ul_2(A);\ r_2(A);$
	$xl_2(A);\ w_2(A);\ u_2(A);$

Nincs holtpont!

Növelési zárak

- **Az `INC(A,c)` művelet:**
  `READ(A,t); t := t+c; WRITE(A,t);`
  műveletek atomi végrehajtása, ahol c egy konstans.

- **Tetszőleges sorrendben kiszámíthatók.**
- **A növelés nem cserélhető fel sem az olvasással, sem az írással.**
  - **Ha azelőtt vagy azután olvassuk be A-t, hogy valaki növelte, különböző értékeket kapunk,**
  - **és ha azelőtt vagy azután növeljük A-t, hogy más tranzakció új értéket írt be A-ba, akkor is különböző értékei lesznek A-nak az adatbázisban.**

- **Az $inc_i(X)$ művelet:**
  a T_i tranzakció megnöveli az X adatbáziselemet valamely konstanssal.
  (Annak, hogy pontosan mennyi ez a konstans, nincs jelentősége.)

- **A műveletnek megfelelő *növelési zárat* (increment lock) $il_i(X)$ -szel jelöljük.**

Növelési zárak

	S	X	I
S	igen	nem	nem
X	nem	nem	nem
I	nem	nem	igen

a) Egy konzisztens tranzakció csak akkor végezheti el x-en a növelési műveletet, ha egyidejűleg növelési zárat tart fenn rajta. A növelési zár viszont nem teszi lehetővé sem az olvasási, sem az írási műveleteket. $il_i(X) \ldots inc_i(X)$

b) Az $inc_i(X)$ művelet konfliktusban áll $r_j(X)$-szel és $w_j(X)$-szel is $j \neq i$-re, de nem áll konfliktusban $inc_j(X)$-szel.

c) Egy jogszerű ütemezésben bármennyi tranzakció bármikor fenntarthat x-en növelési zárat. Ha viszont egy tranzakció növelési zárat tart fenn x-en, akkor egyidejűleg semelyik más tranzakció sem tarthat fenn sem osztott, sem kizárólagos zárat x-en.

Növelési zárak

T_1: $sl_1(A);r_1(A);il_1(B);inc_1(B);u_1(A);u_1(B);$

T_2: $sl_2(A);r_2(A);il_2(B);inc_2(B);u_2(A);u_2(B);$

T_1	T_2
$sl_1(A);\ r_1(A);$	
	$sl_2(A);\ r_2(A);$
	$il_2(B);\ inc_2(B);$
$il_1(B);\ inc_1(B);$	
	$u_2(A);\ u_2(B);$
$u_1(A);\ u_1(B);$	

Az ütemezőnek egyik kérést sem kell késleltetnie!

Zárak nélkül: S: $r_1(A);\ r_2(A);\ inc_2(B);\ inc_1(B);$

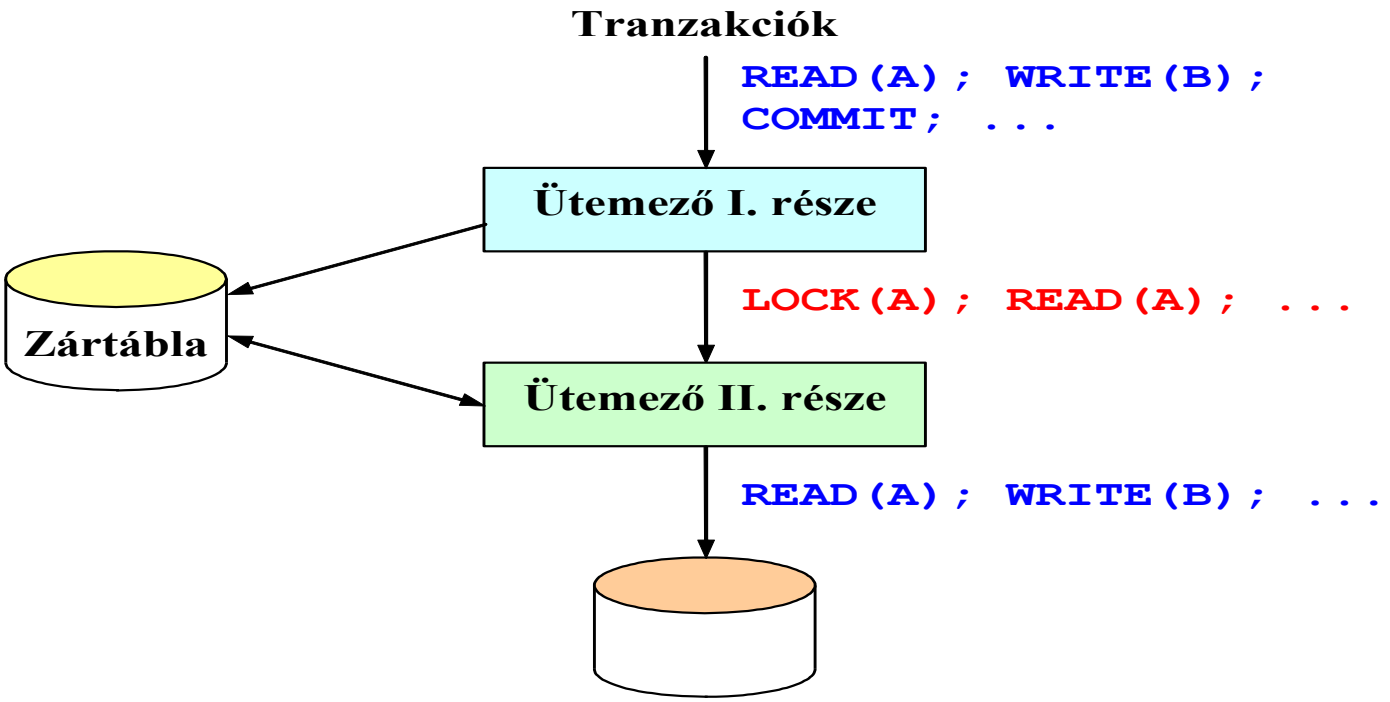
Az utolsó művelet nincs konfliktusban az előzőekkel, így előre hozható a második helyre, így a T_1,T_2 soros ütemezést kapjuk:

S1: $r_1(A);\ inc_1(B);\ r_2(A);\ inc_2(B);$

A zárolási ütemező felépítése

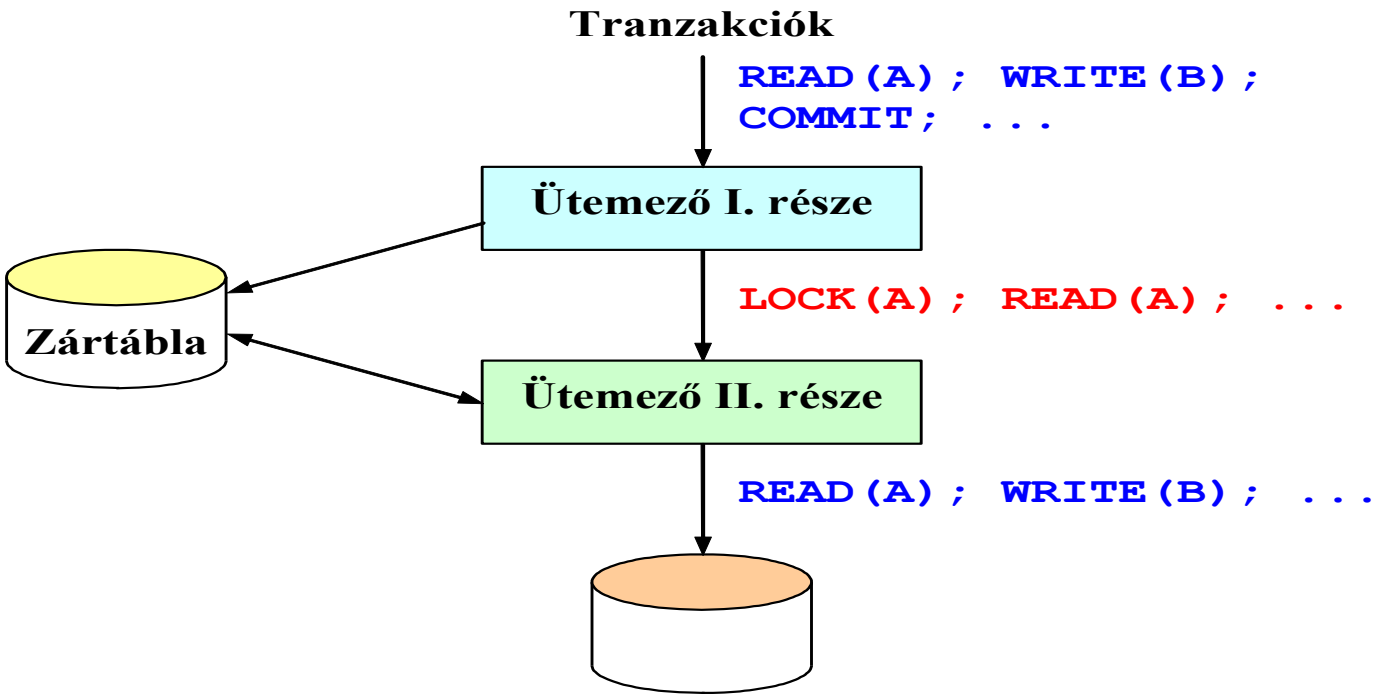
1. Maguk a tranzakciók nem kérnek zárakat, vagy figyelmen kívül hagyjuk, hogy ezt teszik. Az ütemező szúrja be a zárolási műveleteket az adatokhoz hozzáférő olvasási, írási, illetve egyéb műveletek sorába.
2. Nem a tranzakciók, hanem az ütemező oldja fel a zárakat, mégpedig akkor, amikor a tranzakciókezelő a tranzakció véglegesítésére vagy abortálására készül.

Zárolási műveleteket beszúró ütemező



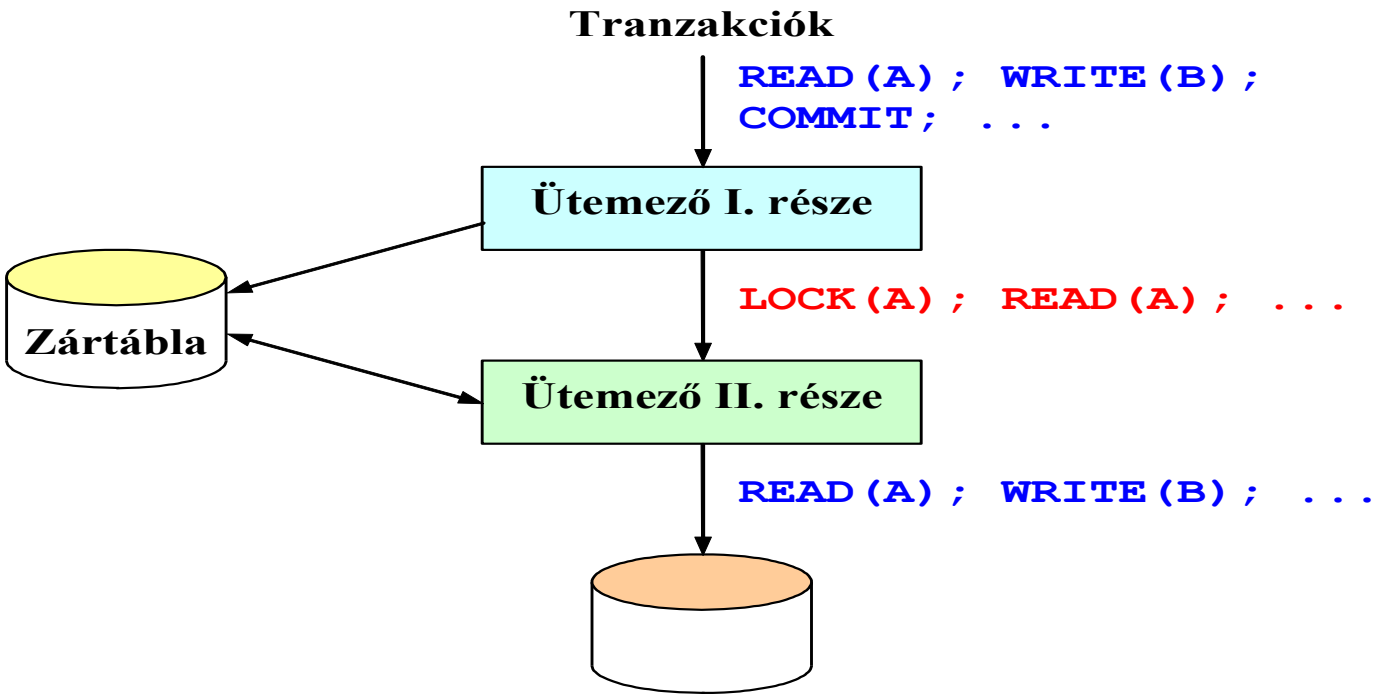
1. Az I. rész fogadja a tranzakciók által generált kérések sorát, és minden adatbázis-hozzáférési művelet elé beszúrja a megfelelő zárolási műveletet. Az adatbázis-hozzáférési és zárolási műveleteket ezután átküldi a II. részhez (a COMMIT és ABORT műveleteket nem).

Zárolási műveleteket beszúró ütemező



2. A II. rész fogadja az I. részen keresztül érkező zárolási és adatbázis-hozzáférési műveletek sorozatát. Eldönti, hogy a T tranzakció késleltetett-e (mivel zárolásra vár). Ha igen, akkor magát a műveletet késlelteti, azaz hozzáadja azoknak a műveleteknek a listájához, amelyeket a T tranzakciónak még végre kell hajtania. Ha T nem késleltetett, vagyis az összes előzőleg kért zár már engedélyezve van, akkor megnézi, hogy milyen műveletet kell végrehajtania.

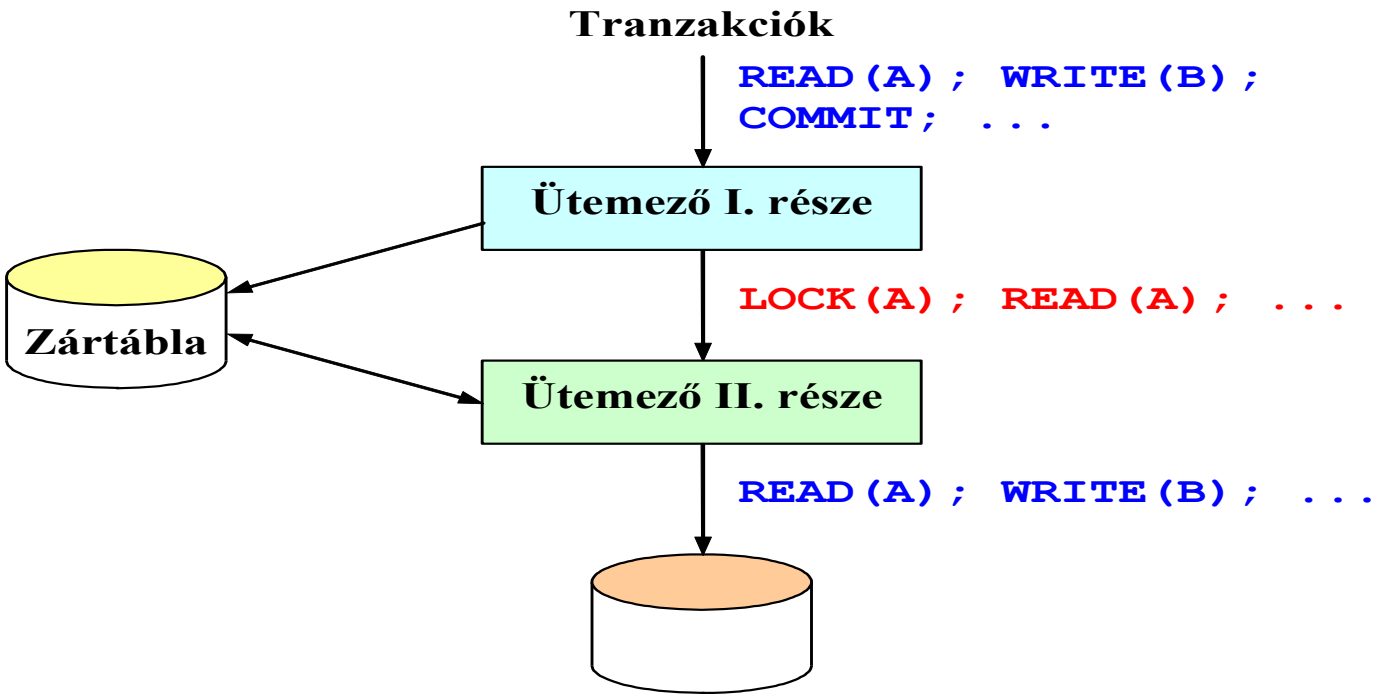
Zárolási műveleteket beszúró ütemező



a) Ha a művelet adatbázis-hozzáférés, akkor továbbítja az adatbázishoz, és végrehajtja.

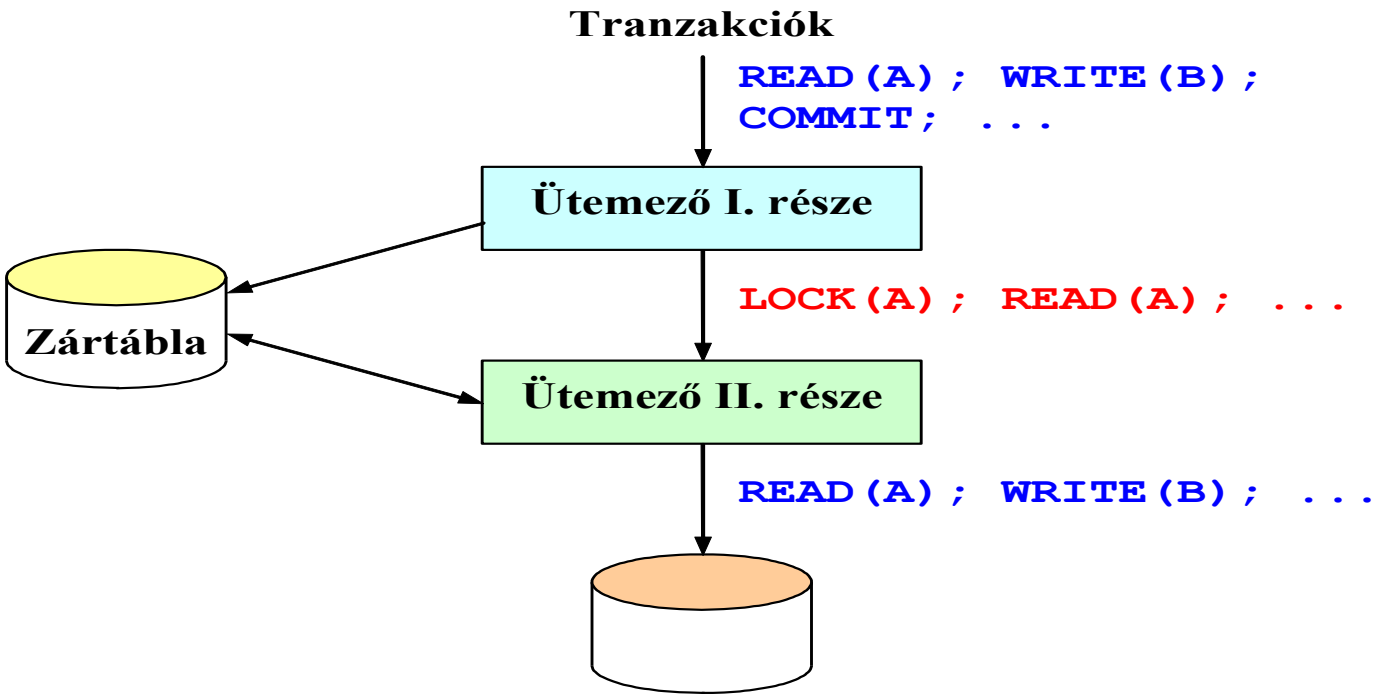
b) Ha zárolási művelet érkezik, akkor megvizsgálja a zártáblát, hogy a zár engedélyezhető-e. Ha igen, akkor úgy módosítja a zártáblát, hogy az az éppen engedélyezett zárat is tartalmazza. Ha nem, akkor egy olyan bejegyzést készít a zártáblában, amely jelzi a zárolási kérést. Az ütemező II. része ezután késlelteti a T tranzakció további műveleteit mindaddig, amíg nem tudja engedélyezni a zárat.

Zárolási műveleteket beszúró ütemező



3. Amikor a T tranzakciót véglegesítjük vagy abortáljuk, akkor a tranzakciókezelő COMMIT, illetve ABORT műveletek küldésével értesíti az I. részt, hogy oldja fel az összes T által fenntartott zárat. Ha bármelyik tranzakció várakozik ezen zárfeloldások valamelyikére, akkor az I. rész értesíti a II. részt.

Zárolási műveleteket beszúró ütemező



4. Amikor a II. rész értesül, hogy egy X adatbáziselemen felszabadult egy zár, akkor eldönti, hogy melyik az a tranzakció, vagy melyek azok a tranzakciók, amelyek megkapják a zárat X-re. A tranzakciók, amelyek megkapták a zárat, a késleltetett műveleteik közül annyit végrehajtanak, amennyit csak végre tudnak hajtani mindaddig, amíg vagy befejeződnek, vagy egy másik olyan zárolási kéréshez érkeznek el, amely nem engedélyezhető.

Zárolási műveleteket beszúró ütemező

- **Ha csak egymódú zárak vannak, akkor az ütemező I. részének a feladata egyszerű. Ha bármilyen műveletet lát az A adatbáziselemen, és még nem szúrt be zárolási kérést A-ra az adott tranzakcióhoz, akkor beszúrja a kérést. Amikor véglegesítjük vagy abortáljuk a tranzakciót, az I. rész törölheti ezt a tranzakciót, miután feloldotta a zárakat, így az I. részhez igényelt memória nem nő korlátlanul.**

Zárolási műveleteket beszúró ütemező

- **Amikor többmódú zárak vannak, az ütemezőnek szüksége lehet arra (például felminősítésnél), hogy azonnal értesüljön, milyen későbbi műveletek fognak előfordulni ugyanazon az adatbáziselemen.**

T_1: $r_1(A)$; $r_1(B)$; $w_1(B)$;

T_2: $r_2(A)$; $r_2(B)$;

Felminősítés módosítási zárral:

T_1	T_2
$sl_1(A)$; $r_1(A)$;	
	$sl_2(A)$; $r_2(A)$;
	$sl_2(B)$; $r_2(B)$;
$ul_1(B)$; $r_1(B)$;	
$xl_1(B)$; elutasítva	
	$u_2(A)$; $u_2(B)$;
$xl_1(B)$; $w_1(B)$;	
$u_1(A)$; $u_1(B)$;	

Az ütemező I. része a következő műveletsorozatot adja ki:

$sl_1(A)$; $r_1(A)$;

$sl_2(A)$; $r_2(A)$; $sl_2(B)$; $r_2(B)$;

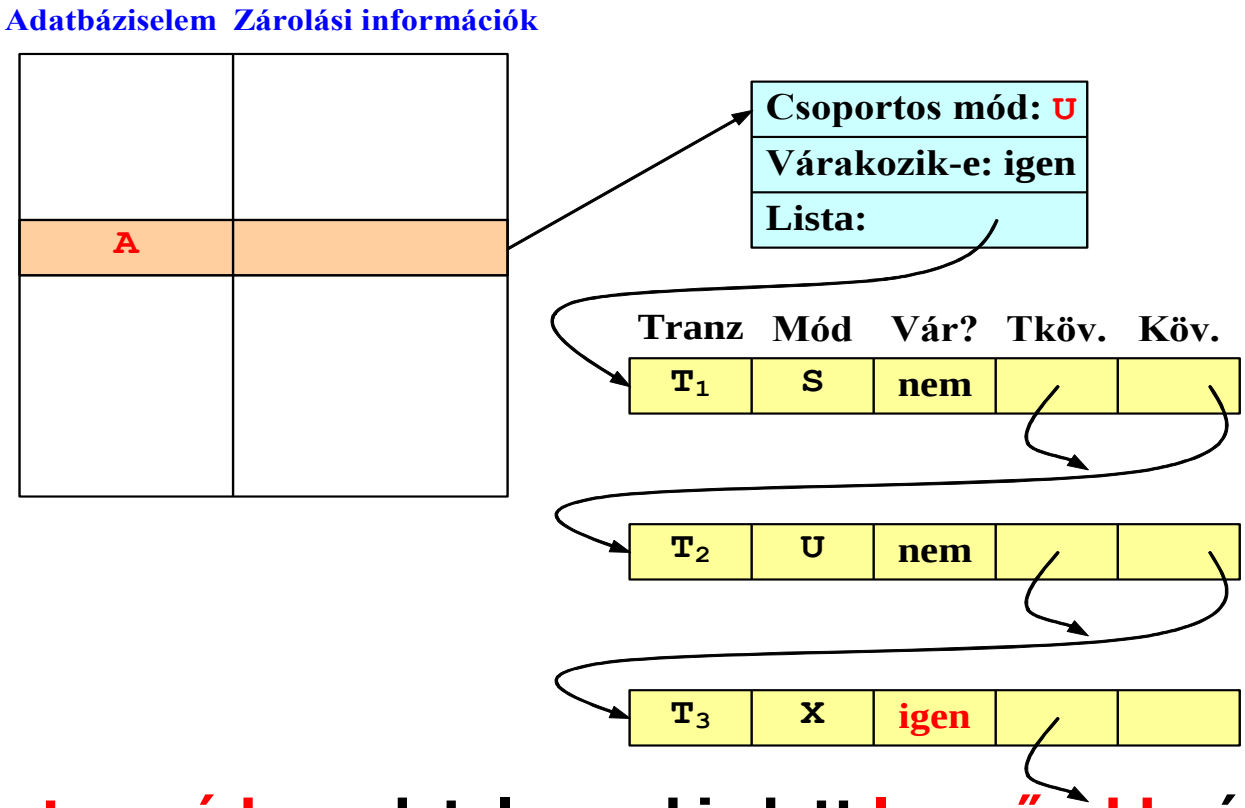
$ul_1(B)$; $r_1(B)$

$xl_1(B)$; $w_1(B)$

A II. rész viszont nem teljesítheti az $xl_1(B)$-t, várakoztatja T_1-et.

Végül T_2 végrehajtja a véglegesítést, és az I. rész feloldja a zárakat A-n és B-n. Ugyanekkor felfedezi, hogy T_1 várakozik B zárolására. Értesíti a II. részt, amely az $xl_1(B)$ zárolást most már végrehajthatónak találja. Beviszi ezt a zárat a zártáblába, és folytatja T_1 tárolt műveleteinek a végrehajtását mindaddig, ameddig tudja. Esetünkben T_1 befejeződik.

A zártábla



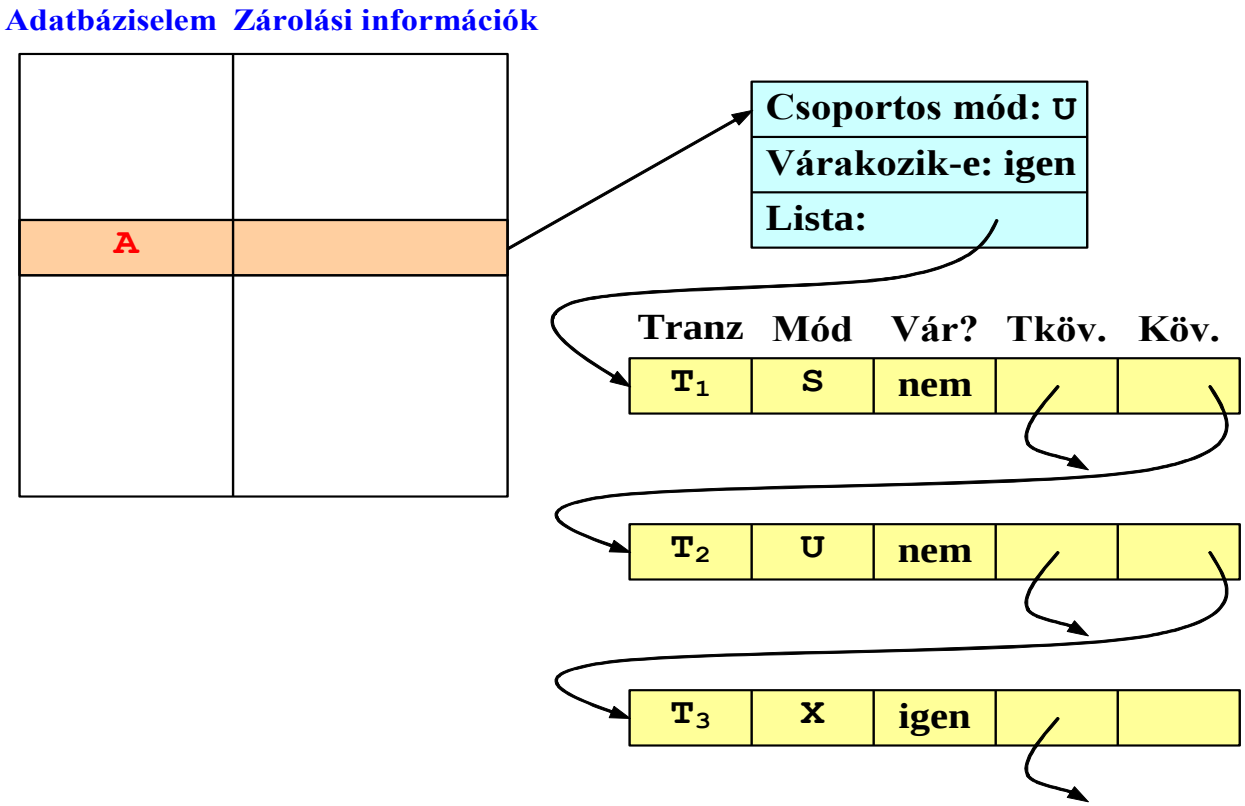
Csoportos mód az adatelemre kiadott legerősebb zár:

a) **S azt jelenti, hogy csak osztott zárak vannak;**

b) **U azt jelenti, hogy egy módosítási zár van, és lehet még egy vagy több osztott zár is;**

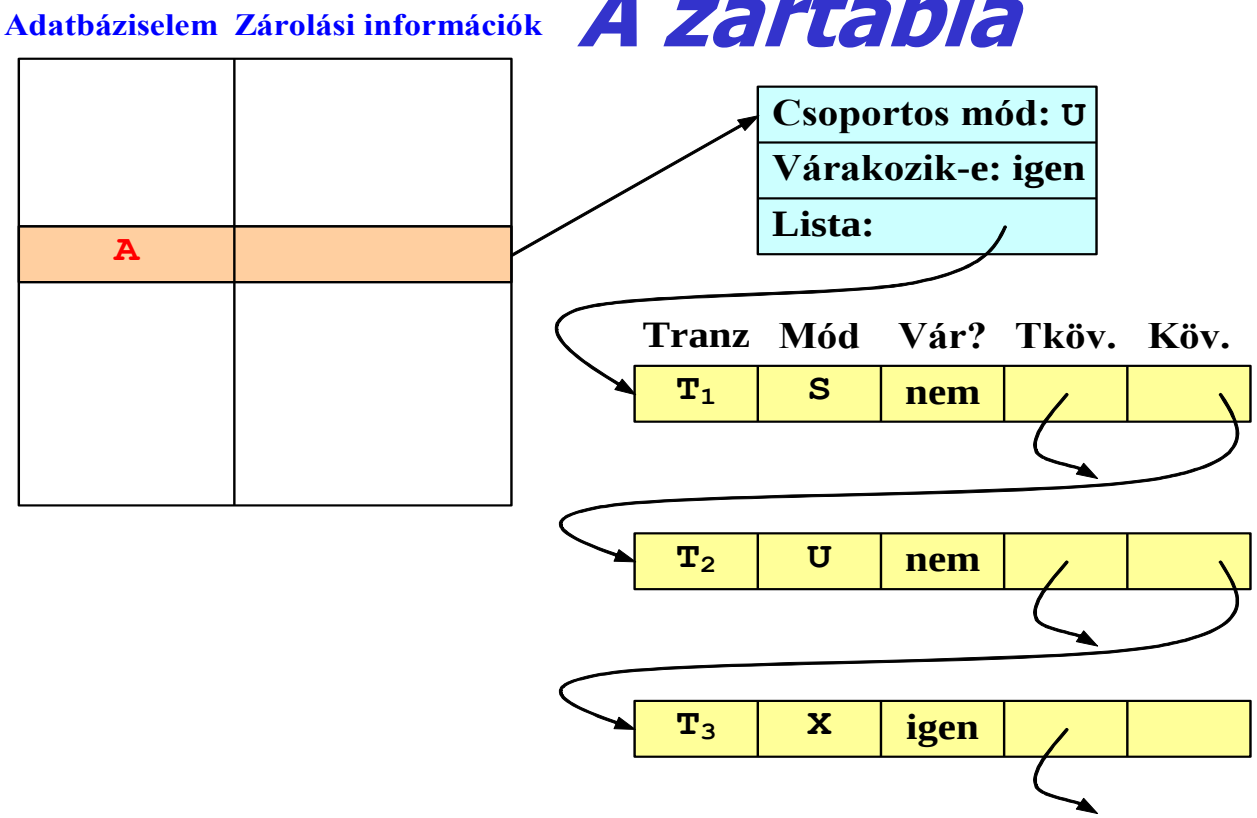
c) **X azt jelenti, hogy csak egy kizárólagos zár van, és semmilyen más zár nincs.**

A zártábla



A *várakozási bit* (waiting bit) azt adja meg, hogy van-e legalább egy tranzakció, amely az A zárolására várakozik.

A zártábla



Az összes olyan tranzakciót leíró lista, amelyek vagy jelenleg zárolják A-t, vagy A zárolására várakoznak.

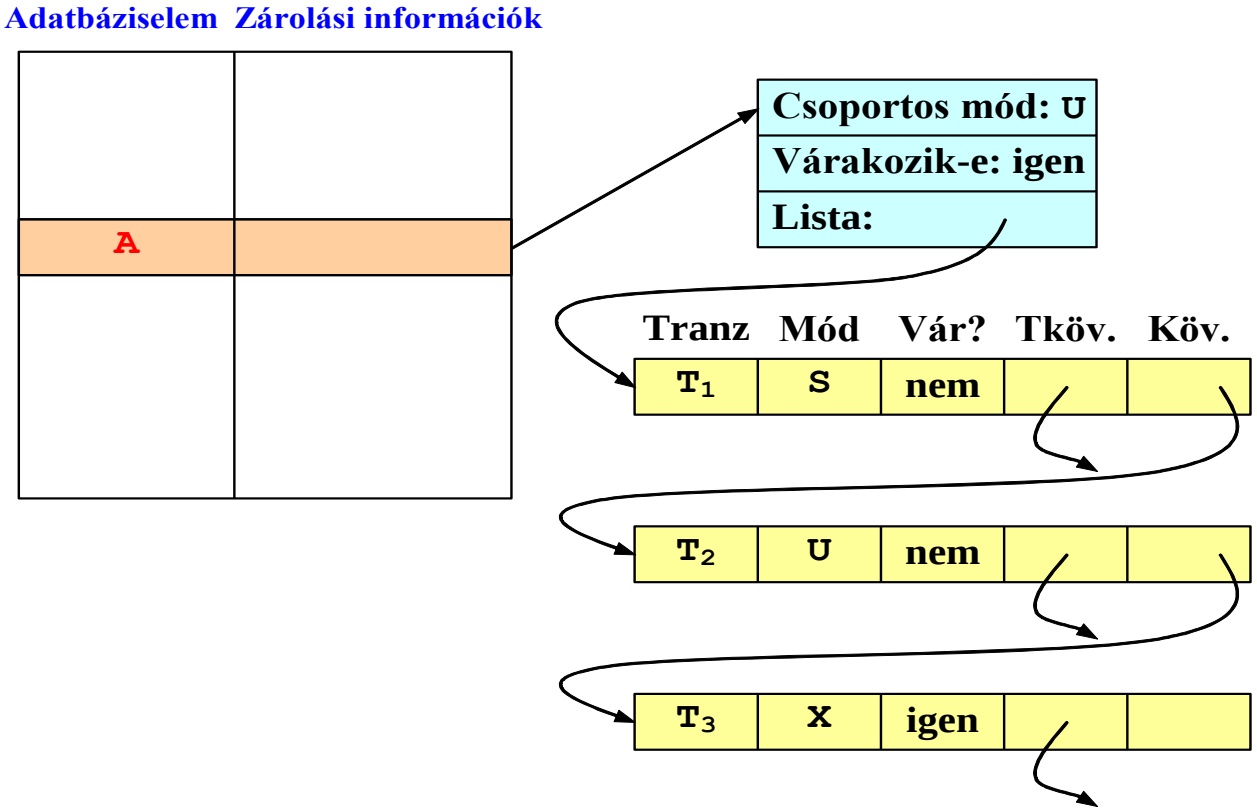
a) **a zárolást fenntartó vagy a zárolásra váró tranzakció neve;**

b) **ennek a zárnak a módja;**

c) **a tranzakció fenntartja-e a zárat, vagy várakozik-e a zárra;**

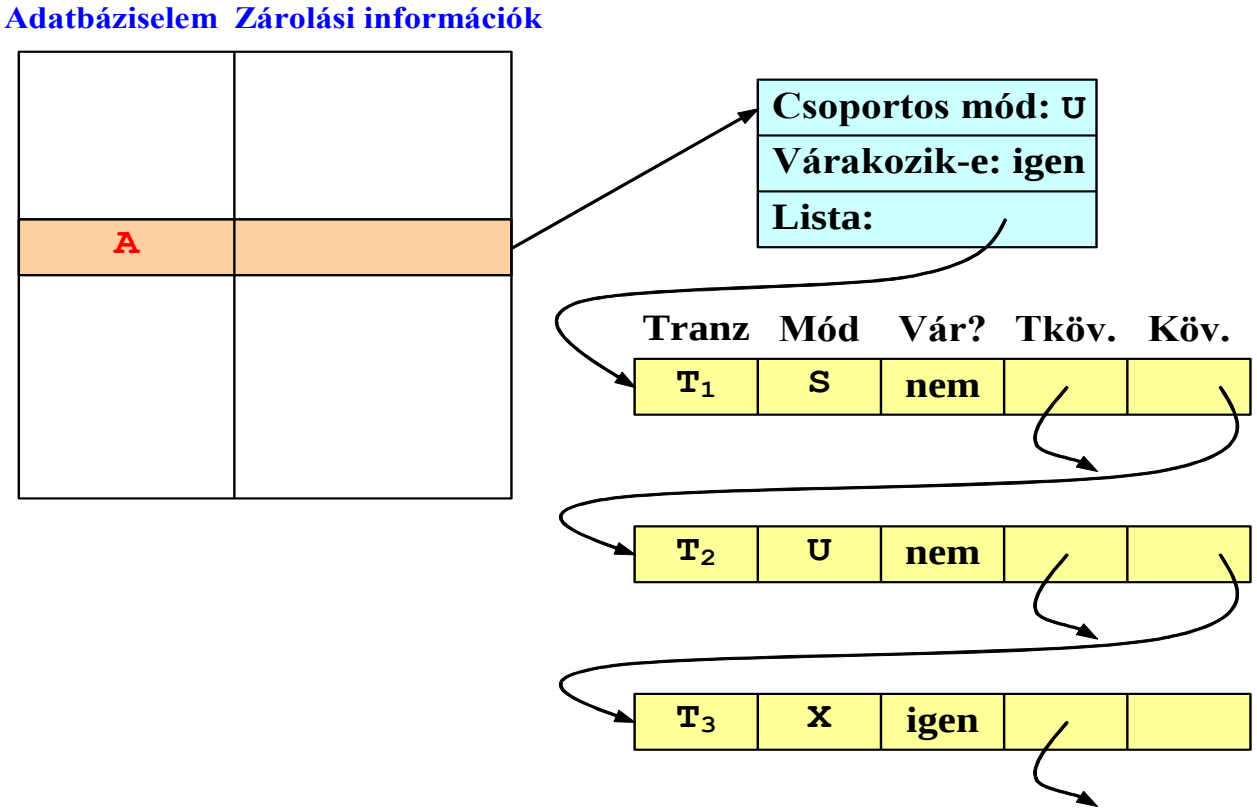
d) **az adott tranzakció következő bejegyzése Tköv.**

A zárolási kérések kezelése



Ha a T tranzakció zárat kér A-ra. Ha nincs A-ra bejegyzés a zártáblában, akkor biztos, hogy zárak sincsenek A-n, így létrehozhatjuk a bejegyzést, és engedélyezhetjük a kérést. Ha a zártáblában létezik bejegyzés A-ra, akkor megkeressük a csoportos módot, és ez alapján várakoztatunk, beírjuk a várakozási listába, vagy megengedjük a zárat (például ha T_2 X-et kér A-ra.)

A zárfeloldások kezelése



Ha T tranzakció feloldja az A-n lévő zárakat. Ekkor T bejegyzését A-ra a listában töröljük, és ha kell a csoportos módot is megváltoztatjuk. Ha maradnak várakozó tranzakciók, akkor engedélyeznünk kell egy vagy több zárat a kért zárak listájáról.

A zárfeloldások kezelése

Több különböző megközelítés lehetséges, mindegyiknek megvan a saját előnye:

1. Első beérkezett első kiszolgálása (first-come-first-served): Azt a zárolási kérést engedélyezzük, amelyik a legrégebb óta várakozik. Ez a stratégia azt biztosítja, hogy ne legyen kiéheztetés, vagyis a tranzakció ne várjon örökké egy zárra.

2. Elsőbbségadás az osztott záraknak (priority to shared locks): Először az összes várakozó osztott zárat engedélyezzük. Ezután egy módosítási zárolást engedélyezünk, ha várakozik ilyen. A kizárólagos zárolást csak akkor engedélyezzük, ha semmilyen más igény nem várakozik. Ez a stratégia csak akkor engedi a kiéheztetést, ha a tranzakció U vagy X zárolásra vár.

3. Elsőbbségadás a felminősítésnek (priority to upgrading): Ha van olyan U zárral rendelkező tranzakció, amely X zárrá való felminősítésre vár, akkor ezt engedélyezzük előbb. Máskülönben a fent említett stratégiák valamelyikét követjük.

Adatbáziselemekből álló hierarchiák kezelése

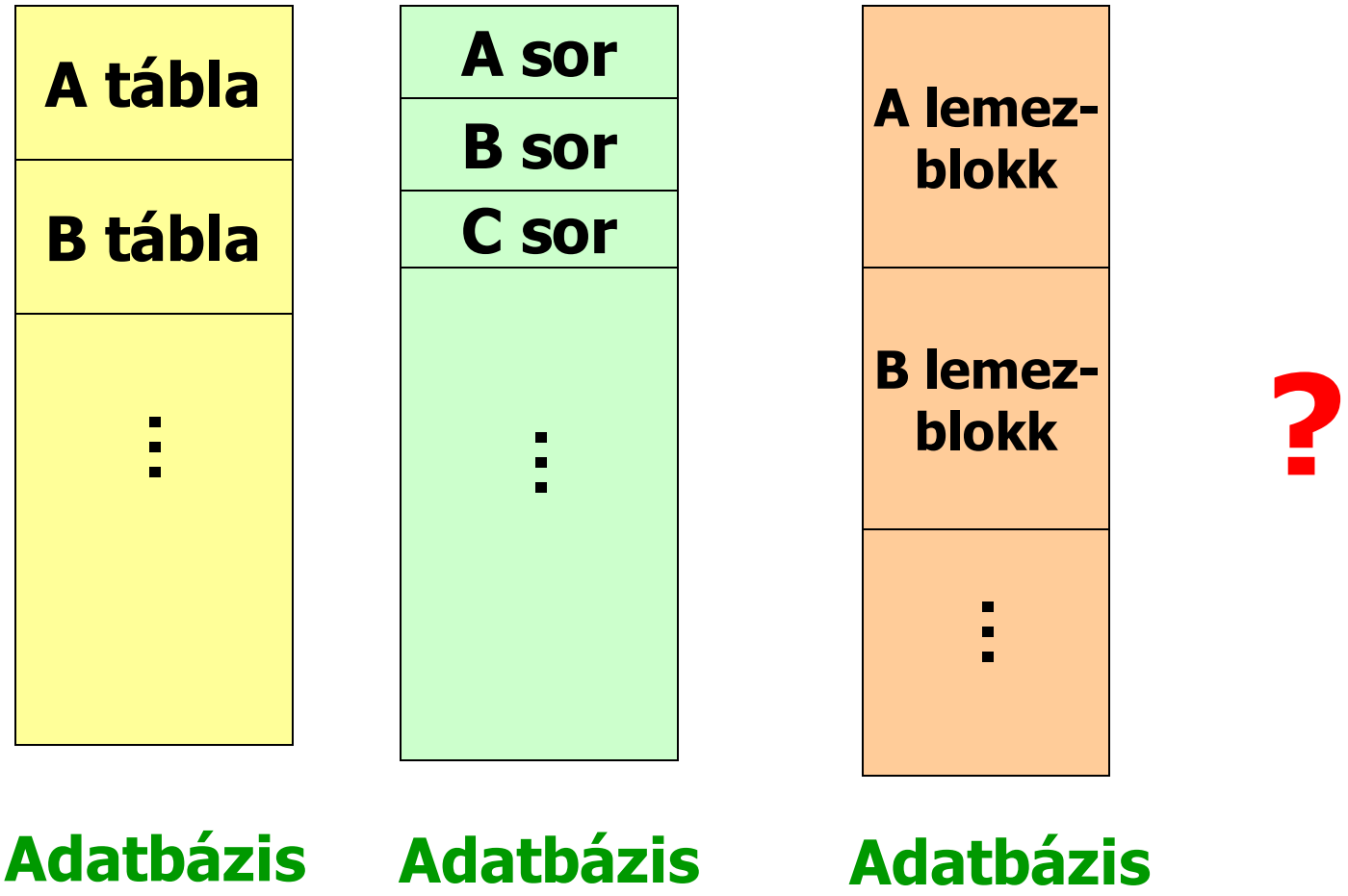
Kétféle fastruktúrával fogunk foglalkozni:

1. Az első fajta fastruktúra, amelyet figyelembe veszünk, a zárolható elemek (zárolási egységek) hierarchiája. Megvizsgáljuk, hogyan engedélyezünk zárolást mind a nagy elemekre, mint például a relációkra, mind a kisebb elemekre, mint például a reláció néhány sorát tartalmazó blokkokra vagy egyedi sorokra.

2. A másik lényeges hierarchiafajtát képezik a konkurenciavezérlési rendszerekben azok az adatok, amelyek önmagukban faszervezésűek. Ilyenek például a B-fa-indexek. A B-fák csomópontjait adatbáziselemeknek tekinthetjük, így viszont az eddig tanult zárolási sémákat szegényesen használhatjuk, emiatt egy új megközelítésre van szükségünk.

Többszörös szemcsézettségű zárak

Milyen objektumokat zároljunk?



Többszörös szemcsézettségű zárak

- **Az alapkérdés, hogy kicsi vagy nagy objektumokat zároljunk?**

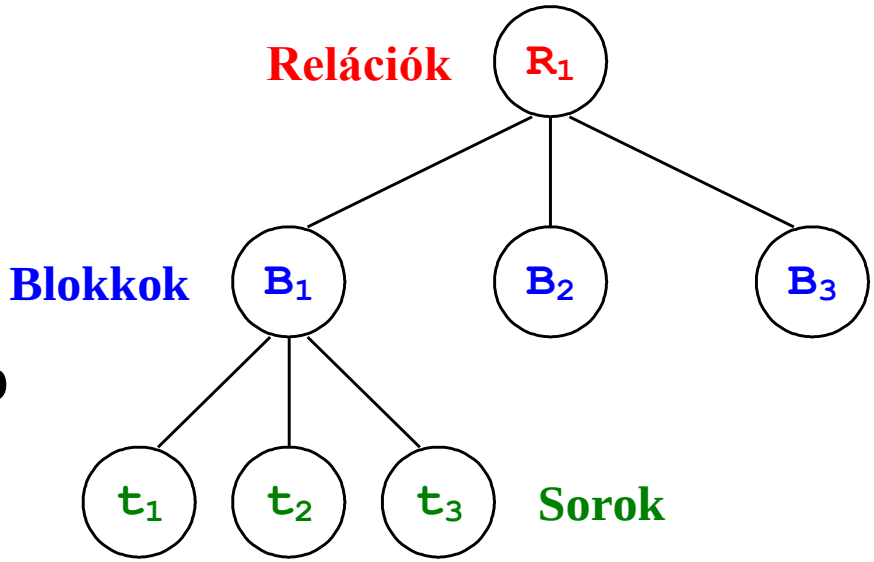
- **Ha nagy objektumokat (például dokumentumokat, táblákat) zárolunk, akkor**
  - **kevesebb zárra lesz szükség, de**
  - **csökken a konkurens működés lehetősége.**
- **Ha kicsi objektumokat, például számlákat, sorokat vagy mezőket) zárolunk, akkor**
  - **több zárra lesz szükség, de**
  - **nő a konkurens működés lehetősége.**

Megoldás: Kicsi ÉS nagy objektumokat is zárolhassunk!

Figyelmeztető zárak

Az adatbáziselemek több (például:három) szintjét különböztetjük meg:

1. a relációk a legnagyobb zárolható elemek;
2. minden reláció egy vagy több blokkból vagy lapból épül fel, amelyekben a soraik vannak;
3. minden blokk egy vagy több sort tartalmaz.



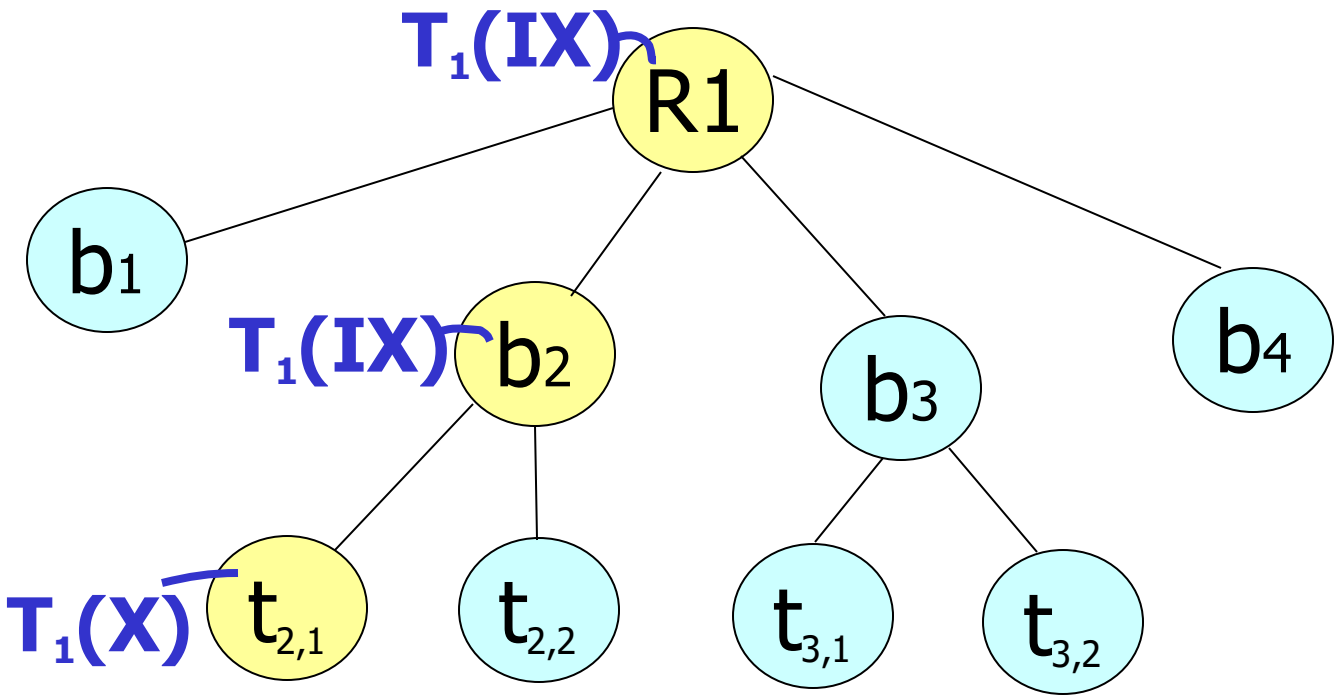
A *figyelmeztető protokoll* zárjai (warning protocol):

1. a közönséges zárak: S és X (osztott és kizárólagos),
2. figyelmeztető zárak: IS, IX (I=intention)

- Például IS azt jelenti, hogy szándékunkban áll osztott zárat kapni egy részelemen.

Figyelmeztető zárak

- **A kért zárnak megfelelő figyelmeztető zárakat kérünk az útvonal mentén a gyökérből kiindulva az adatelemig.**
- **Addig nem megyünk lejjebb, amíg a figyelmeztető zárat meg nem kapjuk.**
- **Így a konfliktusos helyzetek alsóbb szintekre kerülnek a fában.**

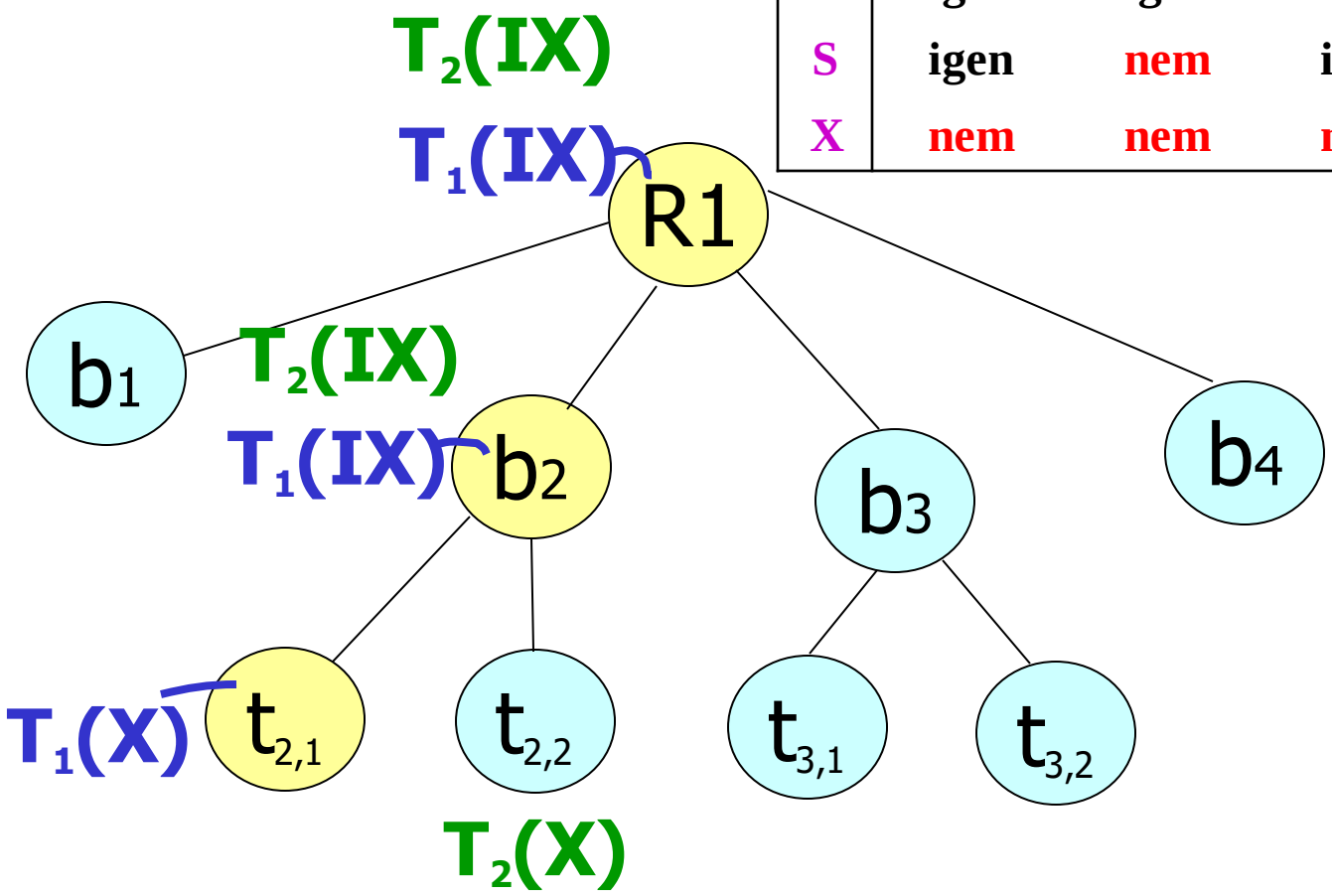


- **Megkaphatja-e T_2 az X zárat a $t_{2,2}$ sorra?**

Figyelmeztető zárak

Kompatibilitási mátrix:

	IS	IX	S	X
IS	igen	igen	igen	nem
IX	igen	igen	nem	nem
S	igen	nem	igen	nem
X	nem	nem	nem	nem

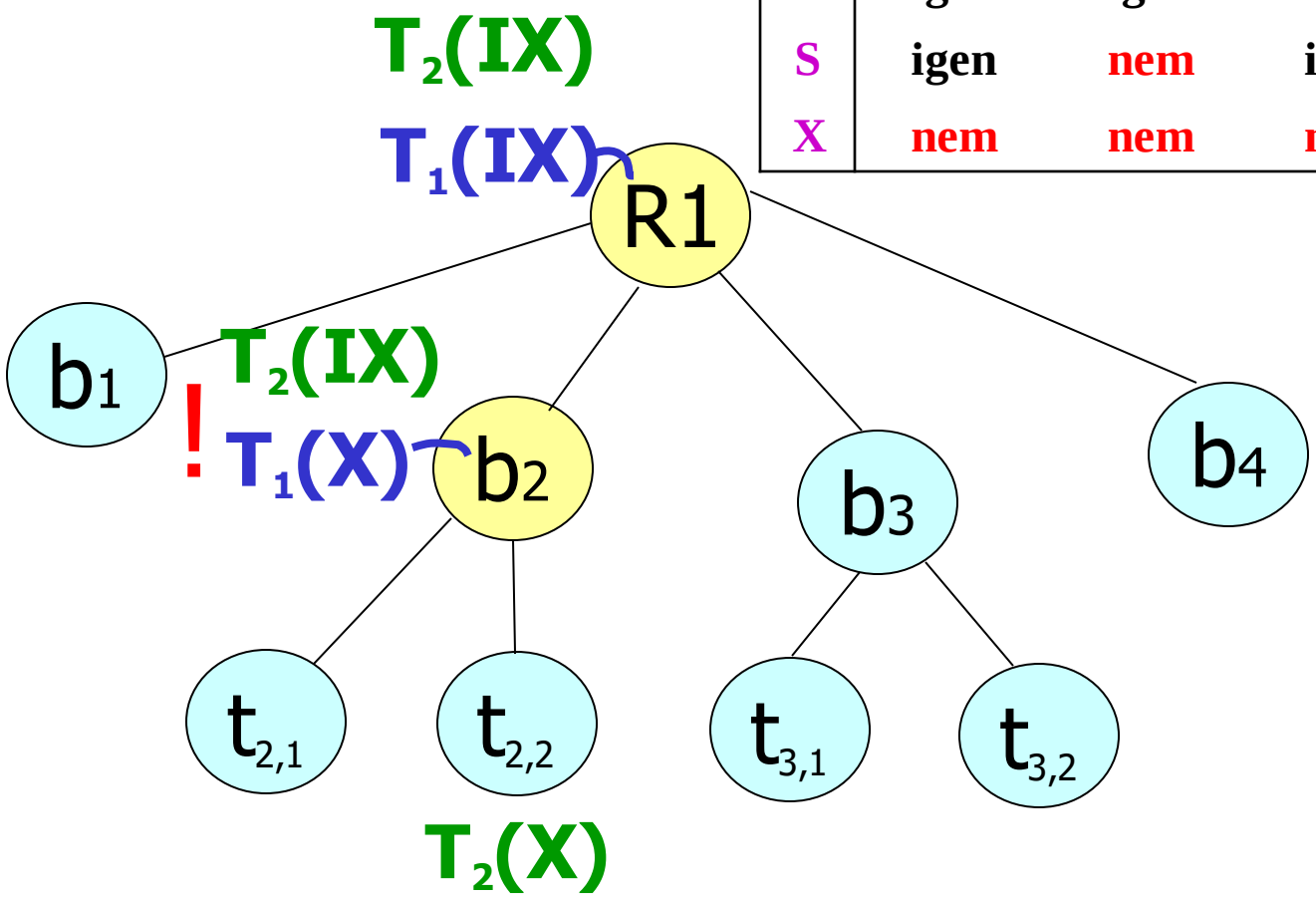


- Megkaphatja-e T_2 az X zárat a $t_{2,2}$ sorra? **IGEN**

Figyelmeztető zárak

Kompatibilitási mátrix:

	IS	IX	S	X
IS	igen	igen	igen	nem
IX	igen	igen	nem	nem
S	igen	nem	igen	nem
X	nem	nem	nem	nem

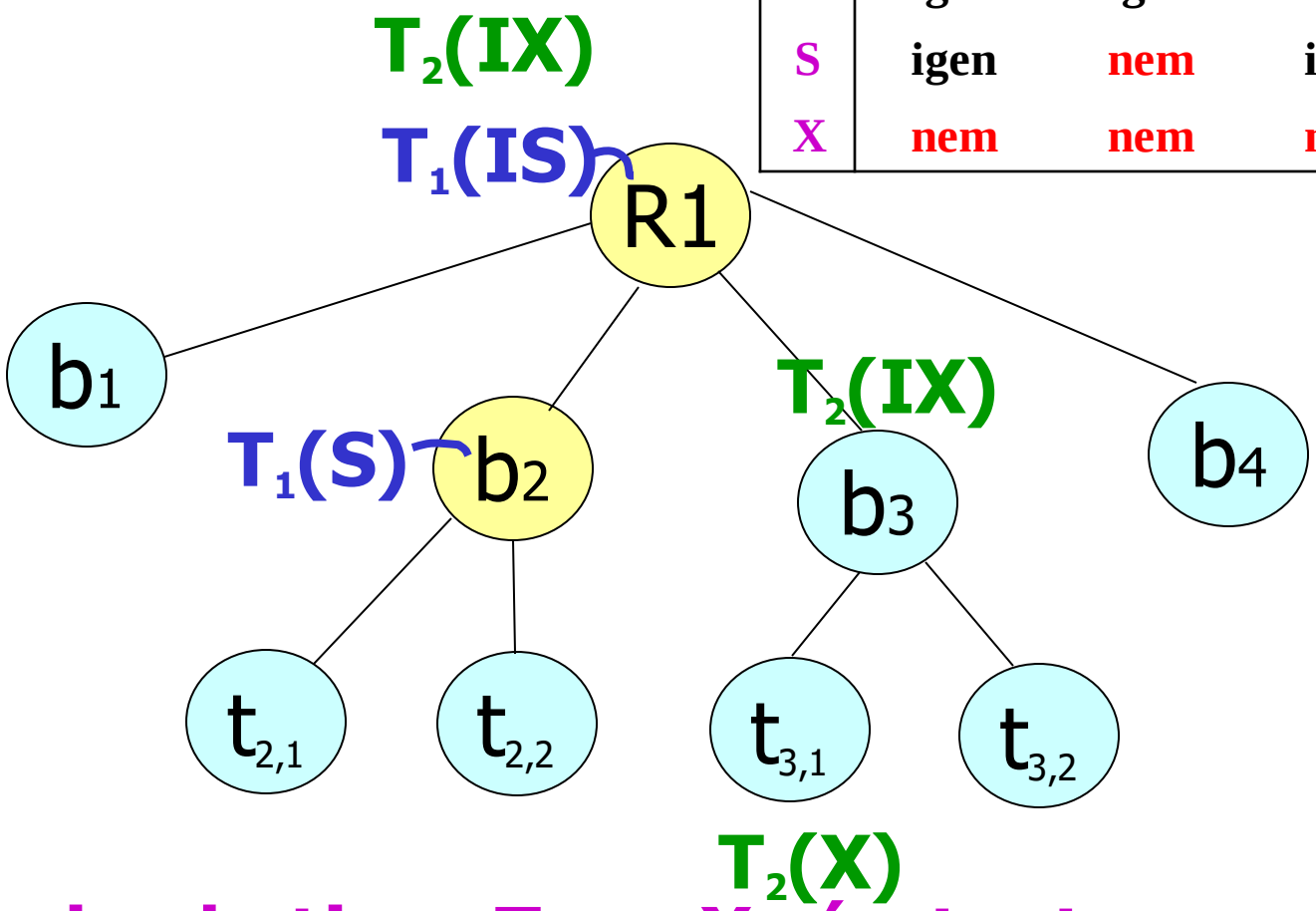


- Megkaphatja-e T_2 az X zárat a $t_{2,2}$ sorra? **NEM**

Figyelmeztető zárak

Kompatibilitási mátrix:

	IS	IX	S	X
IS	igen	igen	igen	nem
IX	igen	igen	nem	nem
S	igen	nem	igen	nem
X	nem	nem	nem	nem



- Megkaphatja-e T_2 az X zárat a $t_{3,1}$ sorra? IGEN

Figyelmeztető zárak

SOR: Ha ilyen zár van már kiadva

	IS	IX	S	X
IS	igen	igen	igen	nem
IX	igen	igen	nem	nem
S	igen	nem	igen	nem
X	nem	nem	nem	nem

Oszlop: Megkaphatjuk-e ezt a típusú zárat?

- **Ha IS zárat kérünk egy N csomópontban, az N egy leszármazottját szándékozzuk olvasni. Ez csak abban az esetben okozhat problémát, ha egy másik tranzakció már jogosulttá vált arra, hogy az N által reprezentált teljes adatbáziselemet felülírja (X). Ha más tranzakció azt tervezi, hogy N-nek csak egy részelemét írja (ezért az N csomóponton egy IX zárat helyezett el), akkor lehetőségünk van arra, hogy engedélyezzük az IS zárat N-en, és a konfliktust alsóbb szinten oldhatjuk meg, ha az írási és olvasási szándék valóban egy közös elemre vonatkozik.**

- **Ha az N csomópont egy részelemét szándékozzuk írni (IX), akkor meg kell akadályoznunk az N által képviselt teljes elem olvasását vagy írását (S vagy X). Azonban más tranzakció, amely egy részelemet olvas vagy ír, a potenciális konfliktusokat az adott szinten kezeli le, így az IX nincs konfliktusban egy másik IX-szel vagy IS-sel N-en.**

Figyelmeztető zárak

SOR: Ha ilyen zár van már kiadva

	IS	IX	S	X
IS	igen	igen	igen	nem
IX	igen	igen	nem	nem
S	igen	nem	igen	nem
X	nem	nem	nem	nem

Oszlop: Megkaphatjuk-e ezt a típusú zárat?

- Az N csomópontnak megfeleltetett elem olvasása (S) nincs konfliktusban sem egy másik olvasási zárral N-en, sem egy olvasási zárral N egy részelemén, amelyet N-en egy IS reprezentál. Azonban egy X vagy egy IX azt jelenti, hogy más tranzakció írni fogja legalább egy részét az N által reprezentált elemnek. Ezért nem tudjuk engedélyezni N teljes olvasását.

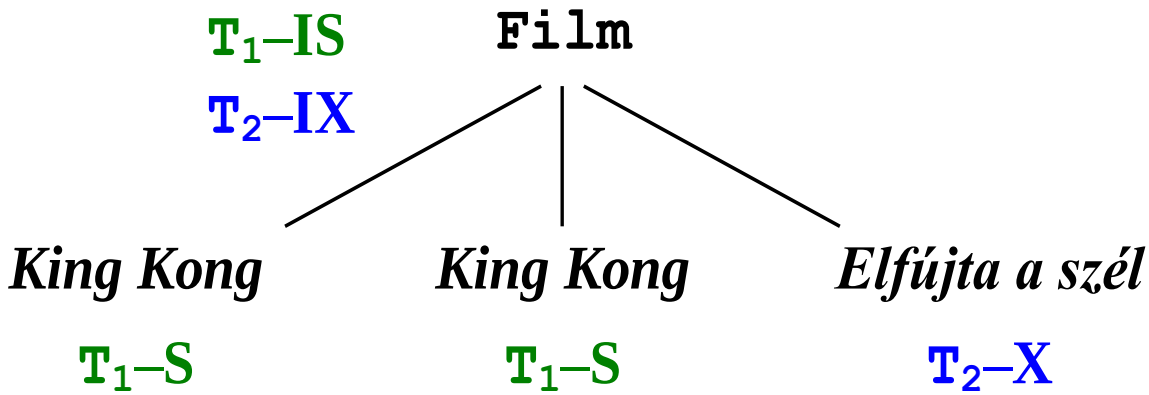
- Nem tudjuk megengedni az N csomópont írását sem (X), ha más tranzakciónak már joga van arra, hogy olvassa vagy írja N-et (S,X), vagy arra, hogy megszerezze ezt a jogot N egy részelemére (IS,IX).

Figyelmeztető zárak

T1: SELECT * FROM Film WHERE filmCím = 'King Kong';

T2: UPDATE Film SET év = 1939 WHERE filmCím = 'Elfújta a szél';

Csak tábla és sor szintű zárolást engedjünk meg.



- **Ekkor T_2-nek szüksége van a reláció IX módú zárolására, ugyanis azt tervezi, hogy új értéket ír be az egyik sorba. Ez kompatibilis T_1-nek a relációra vonatkozó IS zárolásával, így a zárat engedélyezzük.**

- **Amikor T_2 elérkezik az „Elfújta a szél" című filmhez tartozó sorhoz, ezen a soron nem talál zárat, így megkapja az X módú zárat, és módosítja a sort. Ha T_2 a „King Kong" című filmek valamelyikéhez próbált volna új értéket beírni, akkor várnia kellett volna, amíg T_1 felszabadítja az S zárakat, ugyanis az S és az X nem kompatibilisek.**

Figyelmeztető zárak

SOR: Ha ilyen zár van már kiadva

	IS	IX	S	X
IS	igen	igen	igen	nem
IX	igen	igen	nem	nem
S	igen	nem	igen	nem
X	nem	nem	nem	nem

Oszlop: Megkaphatjuk-e ezt a típusú zárat?

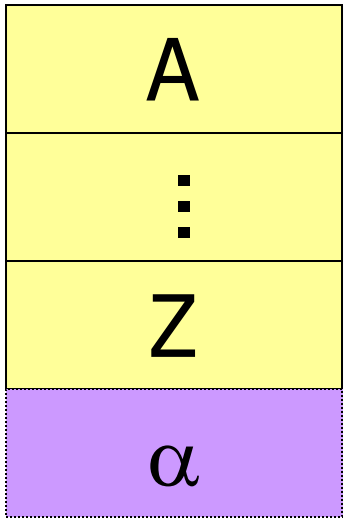
- Melyik zár erősebb a másiknál (erősebb (<): ha mindenhol "nem" szerepel, ahol a gyengébben is "nem" van, de lehet ott is "nem", ahol a gyengébben "igen" van)?
- IS<IX és S<X, de IX és S nem összehasonlítható (< csak parciális rendezés).
- A csoportos mód használatához vezessünk be egy SIX új zárat, (ami azt jelenti, hogy ugyanaz a tranzakció S és IX zárat is tett egy adatelemre). Ekkor SIX mindkettőnél erősebb, de ez a legkisebb ilyen.

Csoportos mód a szándékzárolásokhoz

- Ha mindig van egy domináns zár (vagyis minden kiadott zárnál erősebb zár) egy elemen, akkor több zárolás hatását össze tudjuk foglalni egy csoportos móddal.
- A figyelmeztető zárakat is alkalmazó zárolási séma esetén az S és az IX módok közül egyik sem dominánsabb a másiknál.
- Ugyanaz a tranzakció egy elemet az S és IX módok mindegyikében zárolhatunk egyidejűleg.
- Egy tranzakció mindkét zárolást kérheti, ha egy teljes elemet akar beolvasni, és azután a részelemeknek egy valódi részhalmazát akarja írni.
- Ha egy tranzakciónak S és IX zárolásai is vannak egy elemen, akkor ez korlátozza a többi tranzakciót olyan mértékben, ahogy bármelyik zár teszi. Vagyis elképzelhetünk egy új SIX zárolási módot, amelynek sorai és oszlopai a „nem” bejegyzést tartalmazzák az IS bejegyzés kivételével mindenhol. Az SIX zárolási mód csoportmódként szolgál, ha van olyan tranzakció, amelynek van S, illetve IX módú, de nincs X módú zárolása.

Nem ismételhető olvasás és a fantomok

Insert + delete + update műveletek



Befolyásolhatja-e egy másik tranzakció hatását?

← **Insert**

Mit zároljunk?

Egy nem létező sort?

Nem ismételhető olvasás és a fantomok

- Tegyük fel, hogy van egy T_1 tranzakció, amely egy adott feltételnek eleget tevő sorokat válogat ki egy relációból. Ezután hosszas számításba kezd, majd később újra végrehajtja a fenti lekérdezést.
- Tegyük fel továbbá, hogy a lekérdezés két végrehajtása között egy T_2 tranzakció módosít vagy töröl a táblából néhány olyan sort, amely eleget tesz a lekérdezés feltételének.
- A T_1 tranzakció lekérdezését ilyenkor *nem ismételhető (fuzzy) olvasásnak* nevezzük.
- A nem ismételhető olvasással az a probléma, hogy mást eredményez a lekérdezés másodszori végrehajtása, mint az első.
- A tranzakció viszont elvárhatja (ha akarja), hogy ha többször végrehajtja ugyanazt a lekérdezést, akkor mindig ugyanazt az eredményt kapja.

Nem ismételhető olvasás és a fantomok

Ugyanez a helyzet akkor is, ha a T_2 tranzakció beszúr olyan sorokat, amelyek eleget tesznek a lekérdezés feltételének. A lekérdezés másodszori futtatásakor most is más eredményt kapunk, mint az első alkalommal.

Ennek az az oka, hogy most olyan sorokat is figyelembe kellett venni, amelyek az első futtatáskor még nem is léteztek.

Az ilyen sorokat nevezzük *fantomoknak* (phantom).

Nem ismételhető olvasás és a fantomok

- **A fenti jelenségek általában nem okoznak problémát, ezért a legtöbb adatbázis-kezelő rendszer alapértelmezésben nem is figyel rájuk (annak ellenére, hogy mindkét jelenség nem sorbarendezhető ütemezést eredményez!).**
- **A fejlettebb rendszerekben azonban a felhasználó kérheti, hogy a nem ismételhető olvasások és a fantomolvasások ne hajtódjanak végre.**
- **Ilyen esetekben rendszerint egy hibaüzenetet kapunk, amely szerint a T_1 tranzakció nem sorbarendezhető ütemezést eredményezett, és az ütemező abortálja T_1-et.**

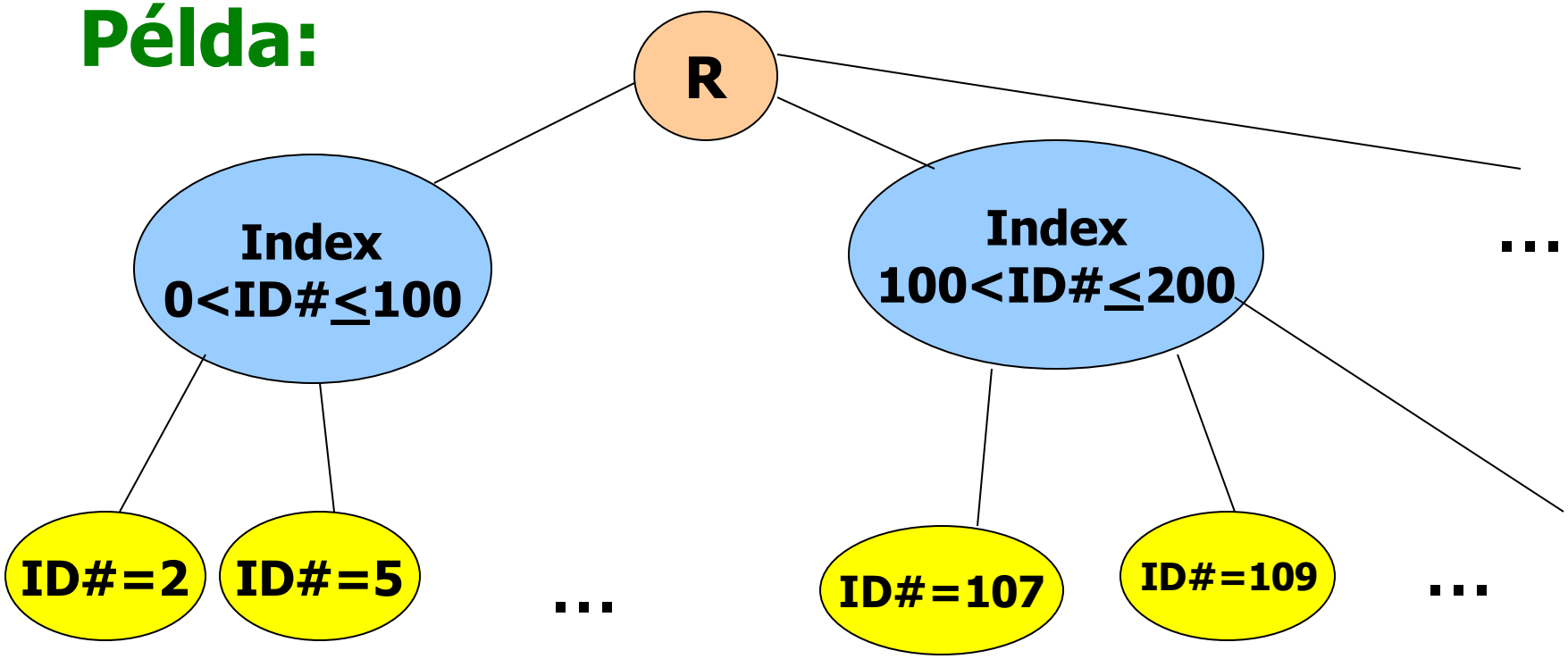
Nem ismételhető olvasás és a fantomok

- **Figyelmeztető protokoll használata esetén viszont könnyen megelőzhetjük az ilyen szituációkat, mégpedig úgy, hogy a T_1 tranzakciónak S módban kell zárolnia a teljes relációt, annak ellenére, hogy csak néhány sorát szeretné olvasni.**

- **A módosító/törlő/beszúró tranzakció ezek után IX módban szeretné zárolni a relációt. Ezt a kérést az ütemező először elutasítja és csak akkor engedélyezi, amikor a T_1 tranzakció már befejeződött, elkerülve ezáltal a nem sorbarendezhető ütemezést.**

Az R tábla sorainak elérése index alapján

Példa:



Csak egyféleképpen, a szülőkön keresztül lehet elérni egy csomópontot.

Faprotokoll

- A zárolható adategységek egy fa csúcsai.

- Például a B-fa esetén a levelekhez csak úgy juthatunk el, ha a gyökértől indulva végigjárunk egy lefele vezető utat. Ahhoz, hogy beolvashassuk azt a levelet, ami nekünk kell, előtte be kell olvasnunk az összes felmenőjét (és ha csúcsok kettévágása vagy csúcsok összevonása úgy kívánja, írnunk is kell őket).

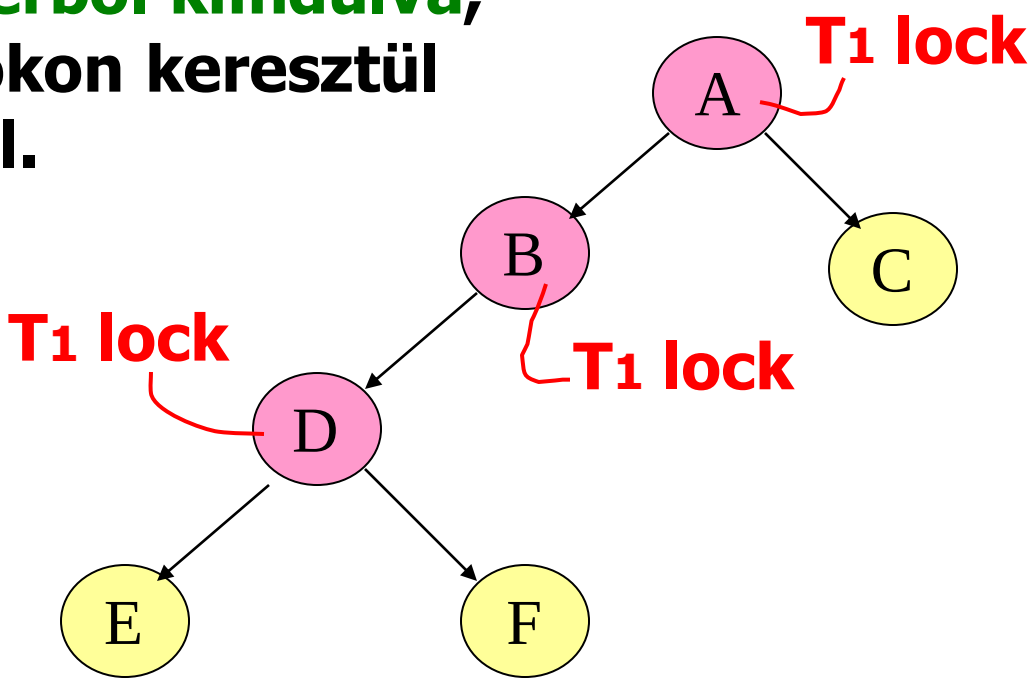
- Ilyenkor a szokásos technikák mennek ugyan, de nagyon előnytelenek lehetnek. Például a 2PL esetén egész addig kell tartani a zárat a gyökéren, amíg le nem értünk a levélhez, ami indokolatlanul sok várakozáshoz vezet.

Faprotokoll

- **Az esetek többségében egy B-fa gyökér csomópontját nem kell átírni, még akkor sem, ha a tranzakció beszúr vagy töröl egy sort.**
- **Például ha a tranzakció beszúr egy sort, de a gyökérnek az a gyereke, amelyhez hozzáférünk, nincs teljesen tele, akkor tudjuk, hogy a beszúrás nem gyűrűzik fel a gyökérig.**
- **Hasonlóan, ha a tranzakció egyetlen sort töröl, és a gyökérnek abban a gyerekében, amelyhez hozzáfértünk, a minimálisnál több kulcs és mutató van, akkor biztosak lehetünk abban, hogy a gyökér nem változik meg.**
- **Ha a tranzakció látja, hogy a gyökér biztosan nem változik meg, azonnal szeretnénk feloldani a gyökéren a zárat.**
- **Ugyanezt alkalmazhatjuk a B-fa bármely belső csomópontjának a zárolására is.**
- **A gyökéren lévő zárolás korai feloldása ellentmond a 2PL-nek, így nem lehetünk biztosak abban, hogy a B-fához hozzáférő tranzakcióknak az ütemezése sorba rendezhető lesz.**
- **A megoldás egy speciális protokoll a fa struktúrájú adatokhoz hozzáférő tranzakciók részére. A protokoll azt a tényt használja, hogy az elemekhez való hozzáférés lefelé halad a fán a sorbarendezhetőség biztosítása érdekében.**

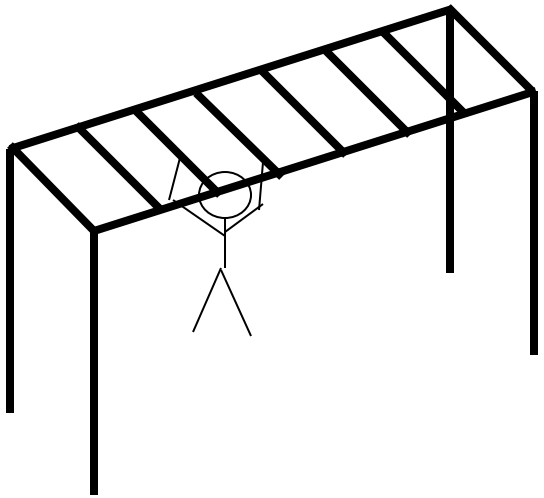
Példa

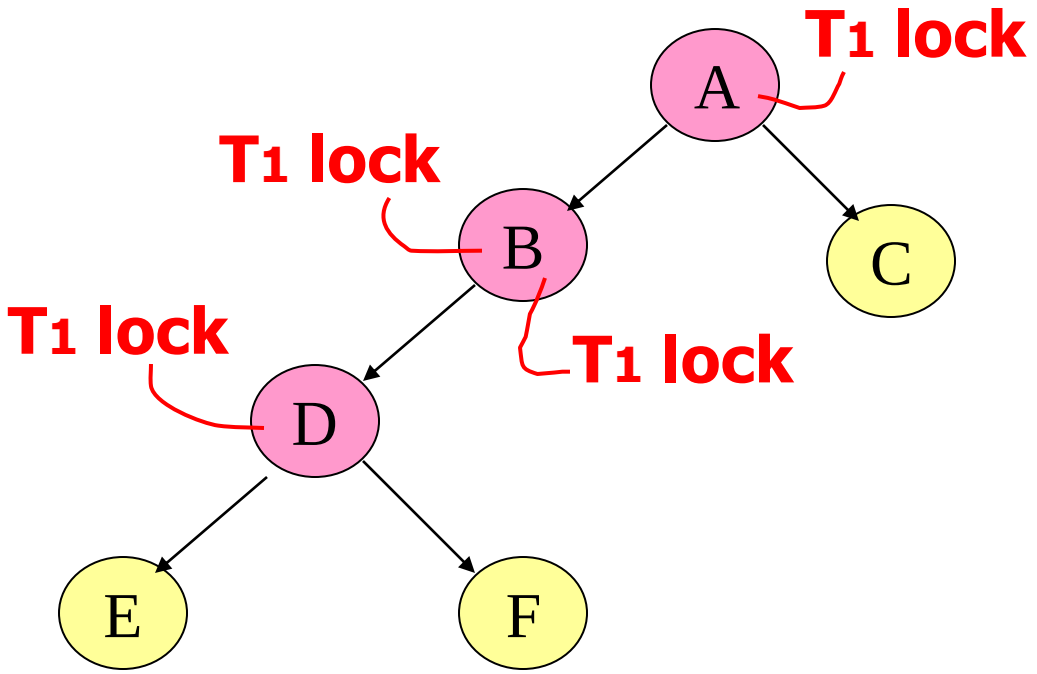
- **Az összes objektumot a gyökérből kiindulva, mutatókon keresztül érjük el.**



☛ **Elengedhetjük az A-n a zárat, ha már A-ra nincs szükségünk?**

Ötlet: Mászóka





Csak egyféle zár van, de ezt az ötletet bármely zárolási módokból álló halmazra általánosíthatjuk.

Faprotokoll szabályai

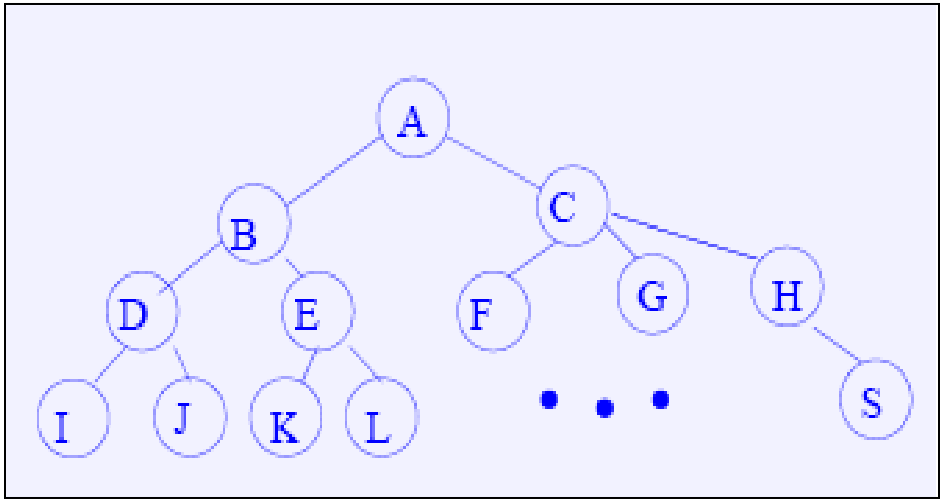
Egyszerű tranzakciómodellben vagyunk (de lehetne (S/X) modellre kibővíteni), azaz

- egy zár van csak, ezt meg kell kapni íráshoz és olvasáshoz is
- zár után mindig van UNLOCK
- nincs két különböző tranzakciónak zárja ugyanott.

A T_i tranzakció követi a faprotokollt, ha

1. Az első zárat bárhova elhelyezheti.
2. A későbbiekben azonban csak akkor kaphat zárat A-n, ha ekkor zárja van A apján.
3. Zárat bármikor fel lehet oldani (nem 2PL).
4. Nem lehet újrazárolni, azaz ha T_i elengedte egy A adategység zárját, akkor később nem kérhet rá újra (még akkor sem, ha A apján még megvan a zárja).

Tétel. *Ha minden tranzakció követi a faprotokollt egy jogszerű ütemezésben, akkor az ütemezés sorbarendezhető lesz, noha nem feltétlenül lesz 2PL.*



A B-fa paramétere legyen 3, azaz legfeljebb 3 mutatót tartalmazhat egy csúcs. A fa belső csúcsai, A-tól H-ig, mutatókat és kulcsokat tartalmaznak. a levelekben (I-től S-ig) pedig a keresési kulcs szerint rendezetten vannak a tárolt adatok. Tegyük fel, hogy egy levélben egy tárolt elem van.

Ha mondjuk az *I*-ben, *J*-ben és *K*-ban tárolt elemek keresési kulcsa 1; 3 és 10, és T_i be akar szúrni egy olyan elemet, ahol a kulcs értéke 4, akkor először olvasni kell *A*-t, *B*-t és *D*-t, majd írni is kell *D*-t.

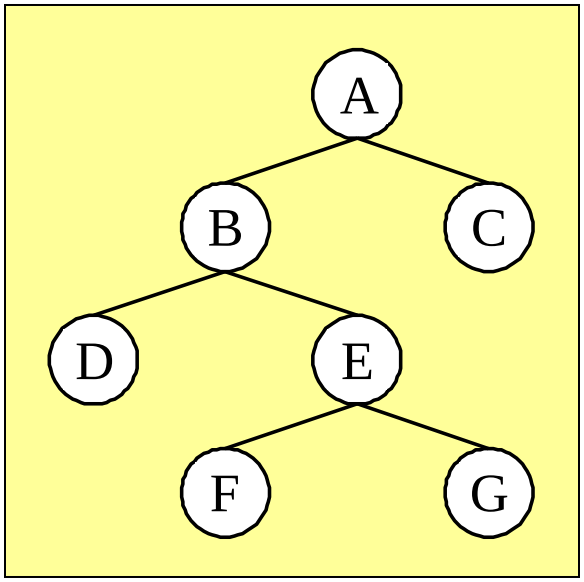
Ekkor a megfelelő (faprotokoll szerinti, legális) ütemezés eleje

$LOCK_i(A)$; $LOCK_i(B)$; $UNLOCK_i(A)$ mert *B* beolvasása után látjuk, hogy neki csak két gyereke van, ha kell is csúcskettévágás, az *A*-t biztos nem érinti, *A*-t nem kell majd írni. Csak addig kellett fogni *A*-n a zárat, amíg *B*-re is megkaptuk.

Ezután $LOCK_i(D)$; $UNLOCK_i(B)$, mert látjuk, hogy *D*-nek csak két gyereke van, ezért *B*-t biztos nem kell írni.

Innen tovább: $UNLOCK_i(D)$, amikor már megtörtént az új levél beszúrása és *D*-ben is beállítottuk a mutatókat.

Nem 2PL és ezzel nyertünk is sokat, mert amint megvolt $UNLOCK_i(A)$, akkor rögtön indulhat a következő beszúrás, ha az a fa jobb oldali ágán fut le. Ha 2PL lett volna, akkor $UNLOCK_i(D)$-ig kellene várni ezzel.



T_1	T_2	T_3
$l_1(A)$; $r_1(A)$;		
$l_1(B)$; $r_1(B)$;		
$l_1(C)$; $r_1(C)$;		
$w_1(A)$; $u_1(A)$;		
$l_1(D)$; $r_1(D)$;		
$w_1(B)$; $u_1(B)$;		
	$l_2(B)$; $r_2(B)$;	
		$l_3(E)$; $r_3(E)$;
$w_1(D)$; $u_1(D)$;		
$w_1(C)$; $u_1(C)$;		
	$l_2(E)$; elutasítva	
		$l_3(F)$; $r_3(F)$;
		$w_3(F)$; $u_3(F)$;
		$l_3(G)$; $r_3(G)$;
		$w_3(E)$; $u_3(E)$;
	$l_2(E)$; $r_2(E)$;	
		$w_3(G)$; $u_3(G)$;
	$w_2(B)$; $u_2(B)$;	
	$w_2(E)$; $u_2(E)$;	

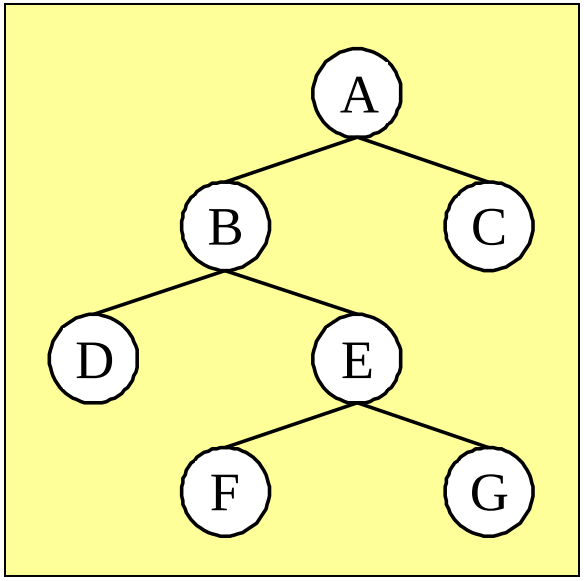
T_1 az A gyökéren kezdődik, és lefelé folytatódik B, C és D felé.

T_2 B-n kezdődik, és az E felé próbál haladni, de először elutasítjuk, ugyanis T_3-nak már van zárja E-n.

A T_3 tranzakció E-n kezdődik, és folytatja F-fel és G-vel.

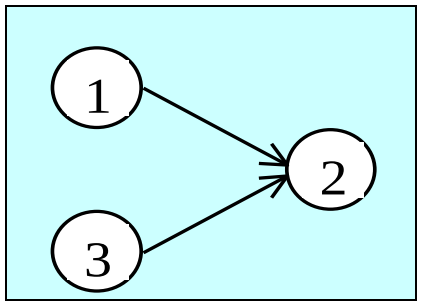
T_1 nem 2PL tranzakció, ugyanis A-n előbb töröljük a zárat, mint hogy megszerezzük a zárat D-n.

Hasonlóan T_3 sem 2PL tranzakció, de T_2 véletlenül éppen 2PL.



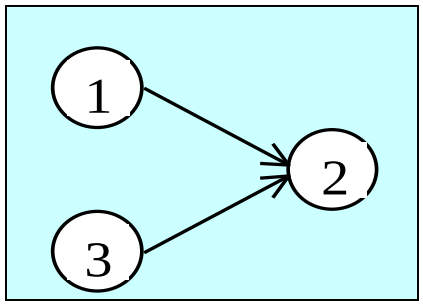
T_1	T_2	T_3
$l_1(A)$; $r_1(A)$;		
$l_1(B)$; $r_1(B)$;		
$l_1(C)$; $r_1(C)$;		
$w_1(A)$; $u_1(A)$;		
$l_1(D)$; $r_1(D)$;		
$w_1(B)$; $u_1(B)$;		
	$l_2(B)$; $r_2(B)$;	
		$l_3(E)$; $r_3(E)$;
$w_1(D)$; $u_1(D)$;		
$w_1(C)$; $u_1(C)$;		
	$l_2(E)$; elutasítva	
		$l_3(F)$; $r_3(F)$;
		$w_3(F)$; $u_3(F)$;
		$l_3(G)$; $r_3(G)$;
		$w_3(E)$; $u_3(E)$;
	$l_2(E)$; $r_2(E)$;	
		$w_3(G)$; $u_3(G)$;
	$w_2(B)$; $u_2(B)$;	
	$w_2(E)$; $u_2(E)$;	

Azt mondjuk, hogy T_i *megelőzi* T_j-t az S ütemezésben ($T_i <_S T_j$), ha a T_i és T_j tranzakciók ugyanazt a csomópontot zárolják (persze nem egy időben), hanem T_i zárolja a csomópontot először.



Ha a megelőzési gráf nem tartalmaz kört, akkor a tranzakciók bármely topologikus sorrendje egy ekvivalens soros ütemezés.

Ebben a példában vagy a (T_1, T_3, T_2) vagy a (T_3, T_1, T_2) az ekvivalens soros ütemezés.



Ha a megelőzési gráf nem tartalmaz kört, akkor a tranzakciók bármely topologikus sorrendje egy ekvivalens soros ütemezés.

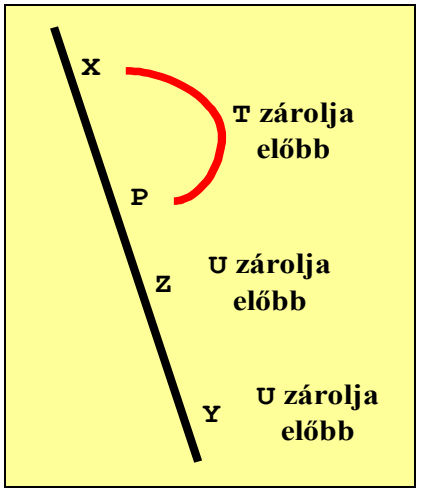
Ennek az az oka, hogy az ilyen soros ütemezésben minden egyes csomóponthoz ugyanabban a sorrendben nyúlnak a tranzakciók, mint az eredeti ütemezésben.

Ha betartjuk a faprotokollt, miért lesz a megelőzési gráf körmentes?

A bizonyítást a következő két állításra bontjuk.

Állítás: Ha két tranzakció által zárolt elemek között vannak megegyezők, akkor ezek mindegyikét ugyanabban a sorrendben zárolják.

Bizonyítás: Tekintsünk valamilyen T és U tranzakciókat, amelyek két vagy több elemet közösen zárolnak. Minden tranzakció fa formájú halmazát zárolja az elemeknek, és a két fa metszete maga is fa. Mivel most T és U közösen zárolnak elemeket, a metszet nem lehet üres fa. Emiatt van egy „legmagasabb" x elem, amelyet T és U is zárol. Indirekten tételezzük fel, hogy T zárolja X-et először, de van egy másik Y elem, amelyet U előbb zárol, mint T. Ekkor az elemekből álló fában van út X-ből Y-ba, és T-nek is és U-nak is zárolnia kell minden elemet az út mentén, ugyanis egyik sem zárolhat úgy egy csomópontot, hogy ne lenne már a szülőjén zárja.



Tekintsük az első olyan elemet az út mentén, amelyet U zárol először, legyen ez Z. Ekkor T előbb zárolja Z-nek a P szülőjét, mint U. Ekkor viszont T még mindig tartja a zárolást P-n, amikor zárolja Z-t, így U még nem zárolhatta P-t, amikor Z-t zárolja. Ellentmondás. Arra következtetünk, hogy T megelőzi U-t minden csomópontban, amelyet közösen zárolnak. Q.E.D.

$$l_T(P) .. l_U(P) .. l_U(Z) .. l_T(Z)$$

Következmény: Tekintsük a $T_1, T_2, \ldots, T_n$ tranzakciók tetszőleges halmazát, amely eleget tesz a faprotokollnak, és az S ütemezésnek megfelelően zárolja a fa valamely csomópontjait. Azok a tranzakciók, amelyek zárolják a gyökeret, ezt valamilyen sorrendben végzik:
Ha T_i előbb zárolja a gyökeret, mint T_j, akkor T_i minden T_j-vel közösen zárolt csomópontot előbb zárol, mint T_j. Vagyis $T_i <_S T_j$, de nem igaz $T_j <_S T_i$. (Igy ha minden tranzakció a gyökértől indul, akkor az a soros ütemezés, amelyben a sorrend az, ahogy a gyökeret zárolják, ekvivalens lesz az ütemezéssel.)

Mi van, ha van olyan tranzakció, amely lejjebb indul?

ÖTLET:

1. A gyökér gyerekeinek megfelelő diszjunkt részfákon indukcióval már lesz soros ütemezés.
2. A gyökeret érintő tranzakcióknak a sorrendje is megad egy soros ütemezést.
3. A részfák diszjunktak, így a különböző részfákban induló tranzakciók sorrendje tetszőleges.
4. Egyesítsük ezeket az ütemezéseket.

Állítás: A fa csomópontjainak száma szerinti teljes indukcióval megmutathatjuk, hogy a teljes tranzakcióhalmazhoz létezik az S-sel ekvivalens soros sorrend.

Bizonyítás:

Alapeset: Ha csak egyetlen csomópont van, a gyökér, akkor a megfelelő sorrend az, amelyben a tranzakciók a gyökeret zárolják.

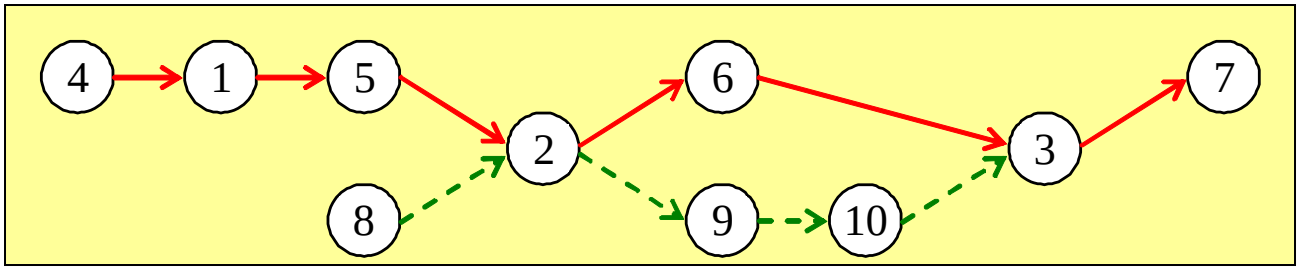
Indukció: Ha egynél több csomópont van a fában, tekintsük a gyökér mindegyik részfájához az olyan tranzakciókból álló halmazt, amelyek egy vagy több csomópontot zárolnak abban a részfában. A gyökeret záró tranzakciók több részfához is tartozhatnak, de egy olyan tranzakció, amely nem zárolja a gyökeret, csak egyetlen részfához tartozik. Például az előző táblázatban található tranzakciók közül csak T_1 zárolja a gyökeret, és az mindkét részfához tartozik: a B gyökerű és a C gyökerű fához is. T_2 és T_3 viszont csak a B gyökerű fához tartozik

Bizonyitás folytatása

- Az indukciós feltevés szerint létezik soros sorrend az összes olyan tranzakcióhoz, amelyek ugyanabban a tetszőleges részfában zárolnak csomópontokat.
- Csupán egybe kell olvasztanunk a különböző részfákhoz tartozó soros sorrendeket. Mivel a tranzakcióknak ezekben a listáiban csak azok a tranzakciók közösek, amelyek zárolják a gyökeret, és megállapítottuk, hogy ezek a tranzakciók minden közös csomópontot ugyanabban a sorrendben zárolnak, ahogy a gyökeret zárolják, nem fordulhat elő két gyökeret záró tranzakció különböző sorrendben két részlistán. Pontosabban: ha T_i és T_j előfordul a gyökér valamely C gyermekéhez tartozó listán, akkor ezek C-t ugyanabban a sorrendben zárolják, mint a gyökeret, és emiatt a listán is ebben a sorrendben fordulnak elő.
- Így felépíthetjük a soros sorrendet a teljes tranzakció-halmazhoz azokból a tranzakciókból kiindulva, amelyek a gyökeret zárolják, a megfelelő sorrendjükben, és beleolvasztjuk azokat a tranzakciókat, amelyek nem zárolják a gyökeret, a részfák soros sorrendjével konzisztens tetszőleges sorrendben. Q.E.D.

Példa a részfák sorrendjének egyesítésére

- Legyen $T_1, T_2, \ldots, T_{10}$ 10 darab tranzakció, és ezekből T_1, T_2 és T_3 ugyanebben a sorrendben zárolja a gyökeret.
- Tegyük fel, hogy a gyökérnek van két gyereke, az elsőt T_1-től T_7-ig zárolják a tranzakciók, a másikat pedig T_2, T_3, T_8, T_9 és T_{10} zárolja.
- Legyen az első részfához a soros sorrend (T_4, T_1, T_5, T_2, T_6, T_3, T_7). Ennek a sorrendnek T_1-et, T_2-t és T_3-at ebben a sorrendben kell tartalmaznia.
- A másik részfához tartozó soros sorrend legyen (T_8, T_2, T_9, T_{10}, T_3). Mint az előző esetben, a T_2 és T_3 tranzakciók, amelyek a gyökeret zárolják, abban a sorrendben fordulnak elő, ahogyan a gyökeret zárolták.



Egy lehetséges topologikus sorrend:

(T_4, T_8, T_1, T_5, T_2, T_9, T_6, T_{10}, T_3, T_7)

Konkurenciavezérlés időbélyegzőkkel

- *Eddig a zárakkal kényszerítettük ki a sorbarendezhető ütemezést.*
- *Most két másik módszert nézünk meg a tranzakciók sorbarendezhetőségének biztosítására:*

1. *Időbélyegzés* (timestamping):

- Minden tranzakcióhoz hozzárendelünk egy „időbélyegzőt”.
- Minden adatbáziselem utolsó olvasását és írását végző tranzakció időbélyegzőjét rögzítjük, és összehasonlítjuk ezeket az értékeket, hogy biztosítsuk, hogy a tranzakciók időbélyegzőinek megfelelő soros ütemezés ekvivalens legyen a tranzakciók aktuális ütemezésével.

2. *Érvényesítés* (validation):

- Megvizsgáljuk a tranzakciók időbélyegzőit és az adatbáziselemeket, amikor a tranzakció véglegesítésre kerül. Ezt az eljárást a tranzakciók *érvényesítésének* nevezzük. Az a soros ütemezés, amely az érvényesítési idejük alapján rendezi a tranzakciókat, ekvivalens kell, hogy legyen az aktuális ütemezéssel.

Konkurenciavezérlés időbélyegzőkkel

- Mindkét megközelítés optimista abban az értelemben, hogy feltételezik, nem fordul elő nem sorba rendezhető viselkedés, és csak akkor tisztázza a helyzetet, amikor ez nyilvánvalóan nem teljesül.

- Ezzel ellentétben minden zárolási módszer azt feltételezi, hogy „a dolgok rosszra fordulnak”, hacsak a tranzakciókat azonnal meg nem akadályozzák abban, hogy nem sorba rendezhető viselkedésük alakuljon ki.

- Az optimista megközelítések abban különböznek a zárolásoktól, hogy az egyetlen ellenszerük, amikor valami rosszra fordul, hogy azt a tranzakciót, amely nem sorba rendezhető viselkedést okozna, abortálják, majd újraindítják.

- A zárolási ütemezők ezzel ellentétben késleltetik a tranzakciókat, de nem abortálják őket, hacsak nem alakul ki holtpont. (Késleltetés az optimista megközelítések esetén is előfordul, annak érdekében, hogy elkerüljük a nem sorba rendezhető viselkedést.)

- Általában az optimista ütemezők akkor jobbak a zárolásinál, amikor sok tranzakció csak olvasási műveleteket hajt végre, ugyanis az ilyen tranzakciók önmagukban soha nem okozhatnak nem sorba rendezhető viselkedést.

Időbélyegzők

- **Minden egyes T tranzakcióhoz hozzá kell rendelni egy egyedi számot, a TS(T) *időbélyegzőt* (time stamp).**

- **Az időbélyegzőket növekvő sorrendben kell kiadni abban az időpontban, amikor a tranzakció az elindításáról először értesíti az ütemezőt.**

- **Két lehetséges megközelítés az időbélyegzők generálásához:**

1. **Az időbélyegzőket a rendszeróra felhasználásával hozzuk létre.**

2. **Az ütemező karbantart egy számlálót. Minden alkalommal, amikor egy tranzakció elindul, a számláló növekszik eggyel, és ez az új érték lesz a tranzakció időbélyegzője. Így egy később elindított tranzakció nagyobb időbélyegzőt kap, mint egy korábban elindított tranzakció.**

Adatelemek időbélyegzői és véglegesítési bitjei

- Minden egyes X adatbáziselemhez hozzá kell rendelnünk **két időbélyegzőt** és esetlegesen **egy további bitet**:

1. **RT(X): X olvasási ideje** (read time), amely a legmagasabb időbélyegző, ami egy olyan tranzakcióhoz tartozik, amely már olvasta X-et.

2. **WT(X): X írási ideje** (write time), amely a legmagasabb időbélyegző, ami egy olyan tranzakcióhoz tartozik, amely már írta X-et.

3. **C(X): X véglegesítési bitje** (commit bit), amely akkor és csak akkor igaz, ha a legújabb tranzakció, amely X-et írta, már véglegesítve van.

- A **C(X)** bit célja, hogy **elkerüljük azt a helyzetet**, amelyben egy T tranzakció egy másik U tranzakció által írt adatokat olvas be, és utána U-t abortáljuk. Ez a probléma, **amikor T nem véglegesített adatok „piszkos olvasását" hajtja végre**, az adatbázis-állapot inkonzisztenssé válását is okozhatja.

Fizikailag nem megvalósítható viselkedések

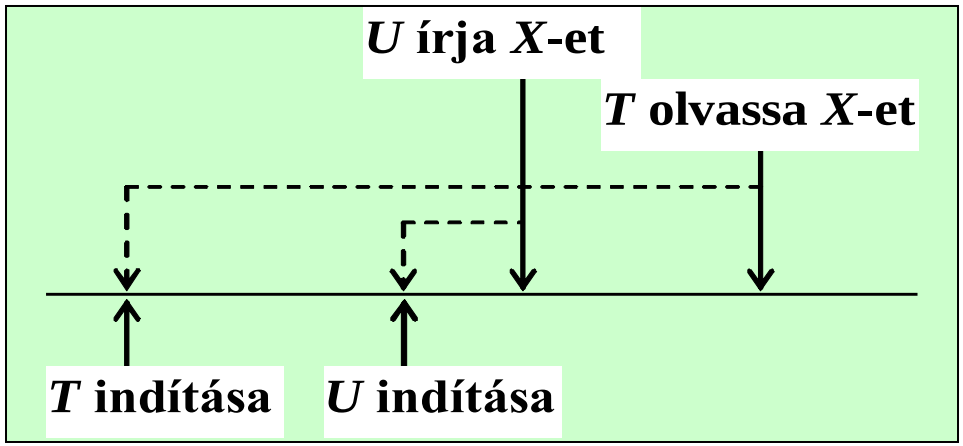
- **Az ütemező olvasáskor és íráskor ellenőrzi, hogy ez abban a sorrendben történik-e, mintha a tranzakciókat az időbélyegzőjük szerinti növekvő, soros ütemezésben hajtottunk volna végre.**

- **Ha nem, akkor azt mondjuk, hogy a viselkedés *fizikailag nem megvalósítható* és *ilyenkor beavatkozik az ütemező.***

- **Kétféle probléma merülhet fel:**
  1. **Túl késői olvasás**
  2. **Túl késői írás**

1. Túl késői olvasás

- **A T tranzakció megpróbálja olvasni az X adatbáziselemet, de X írási ideje azt jelzi, hogy X jelenlegi értékét azután írtuk, miután T-t már elméletileg végrehajtottuk, vagyis TS(T) < WT(X).**



A megoldás, hogy T-t abortáljuk, amikor ez a probléma felmerül.

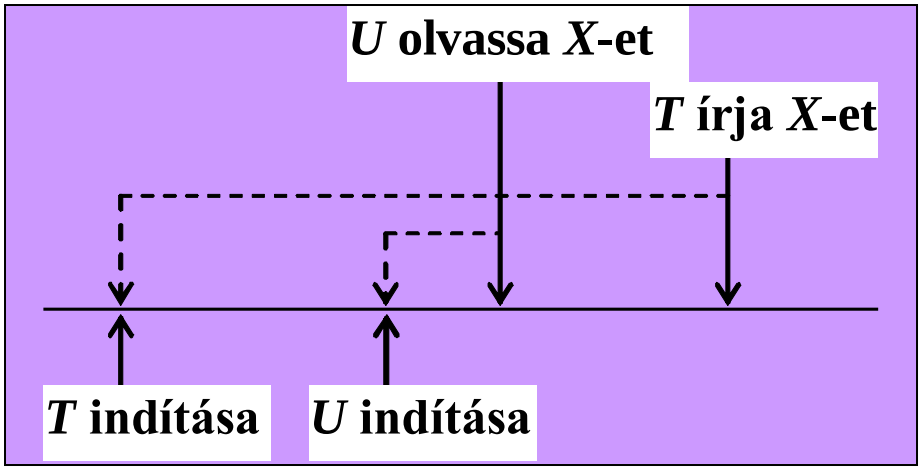
2. Túl késői írás

- **A T tranzakció megpróbálja írni az X adatbáziselemet, de X olvasási ideje azt jelzi, hogy van egy másik tranzakció is, amelynek a T által beírt értéket kellene olvasnia, ám ehelyett más értéket olvas, vagyis**

  WT(X) < TS(T) < RT(X) vagy

  TS(T) < RT(X) < WT(X).

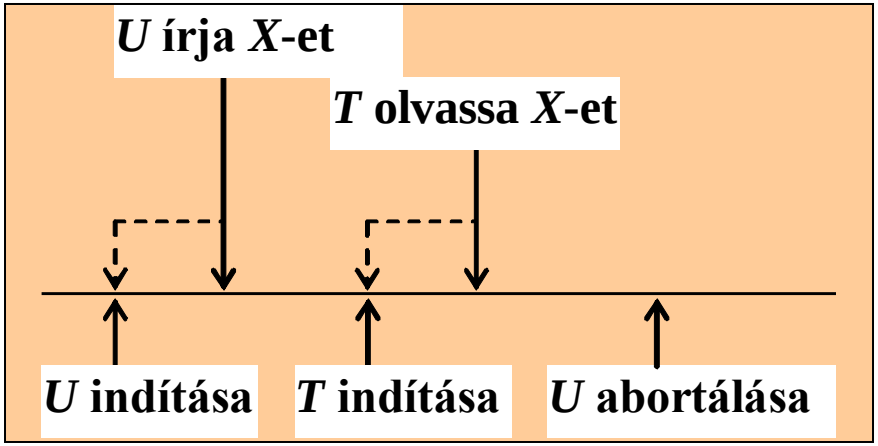
Semelyik más tranzakció sem írta X-et, amellyel felülírta volna a T által írt értéket, és ezzel érvénytelenítette volna T hatását.



A megoldás, hogy T-t abortáljuk, amikor ez a probléma felmerül.

A piszkos adatok problémái

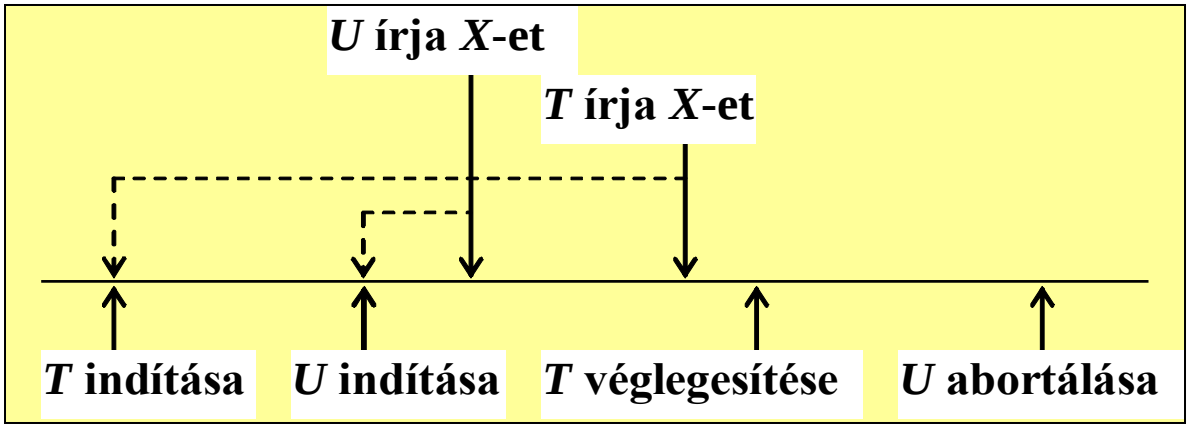
- **Bár nincs fizikailag nem megvalósítható abban, hogy T olvassa X-et, mégis jobb a T általi olvasást azutánra elhalasztani, hogy U véglegesítését vagy abortálását már elvégeztük, különben az ütemezésünk nem lesz konfliktus-sorbarendezhető. Azt, hogy U még nincs véglegesítve, onnan tudjuk, hogy a C(X) véglegesítési bit hamis.**



- **A piszkos olvasás problémája véglegesítési bit nélkül is megoldható: Amikor abortálunk egy U tranzakciót, meg kell néznünk, hogy vannak-e olyan tranzakciók, amelyek olvastak egy vagy több U által írt adatbáziselemet. Ha igen, akkor azokat is abortálnunk kell. Ebből aztán további abortálások következhetnek, és így tovább. Ezt továbbgyűrűző visszagörgetésnek nevezzük. Ez a megoldás azonban alacsonyabb fokú konkurenciát engedélyez, mint a véglegesítési bit bevezetése és a késleltetés, ráadásul előfordulhat, hogy nem helyreállítható ütemezést kapunk. Ez abban az esetben következik be, ha az egyik „abortálandó” tranzakciót már véglegesítettük.**

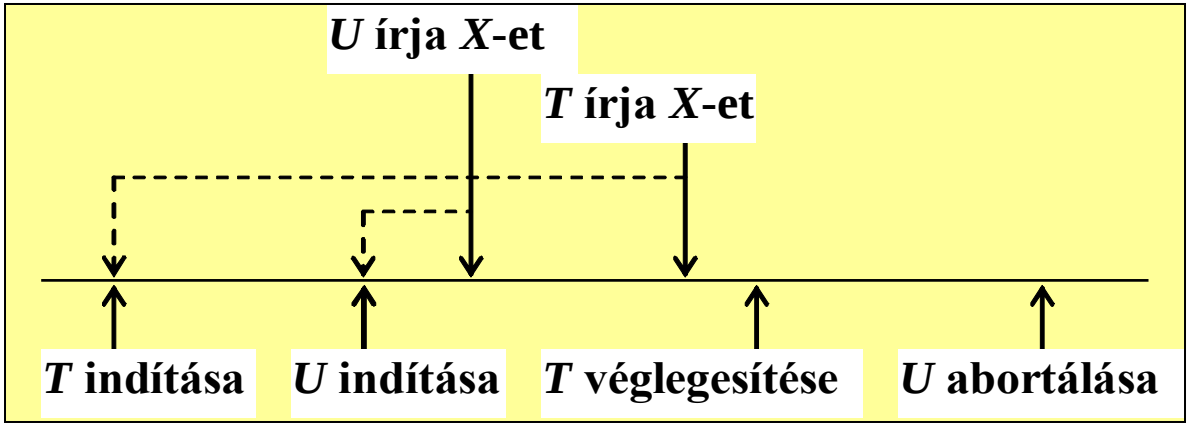
Egy másik probléma (Thomas-féle írás)

- **Itt U írja először X-et. Amikor T írni próbál, a megfelelő művelet semmit sem végez, tehát elhagyható. Nyilvánvalóan nincs más V tranzakció, amelynek X-ből a T által beírt értéket kellene beolvasnia, és ehelyett az U által írt értéket olvasná, ugyanis ha V megpróbálná olvasni X-et, abortálnia kellene a túl késői olvasás miatt. X későbbi olvasásainál az U által írt értéket kell olvasni, vagy X még későbbi, de nem T által írt értékét. Ezt az ötletet, miszerint azokat az írásokat kihagyhatjuk, amelyeknél későbbi írási idejű írást már elvégeztünk, *Thomas-féle írási szabálynak* nevezzük.**



PROBLÉMA: Ha U-t később abortáljuk, akkor X-nek az U által írt értékét ki kell törölnünk, továbbá az előző értéket és írási időt vissza kell állítanunk. Minthogy T-t véglegesítettük, úgy látszik, hogy X T által írt értékét kell a későbbi olvasásokhoz használnunk. Mi viszont kihagytuk a T általi írást, és már túl késő, hogy helyrehozhassuk ezt a hibát.

Egy másik probléma (Thomas-féle írás)

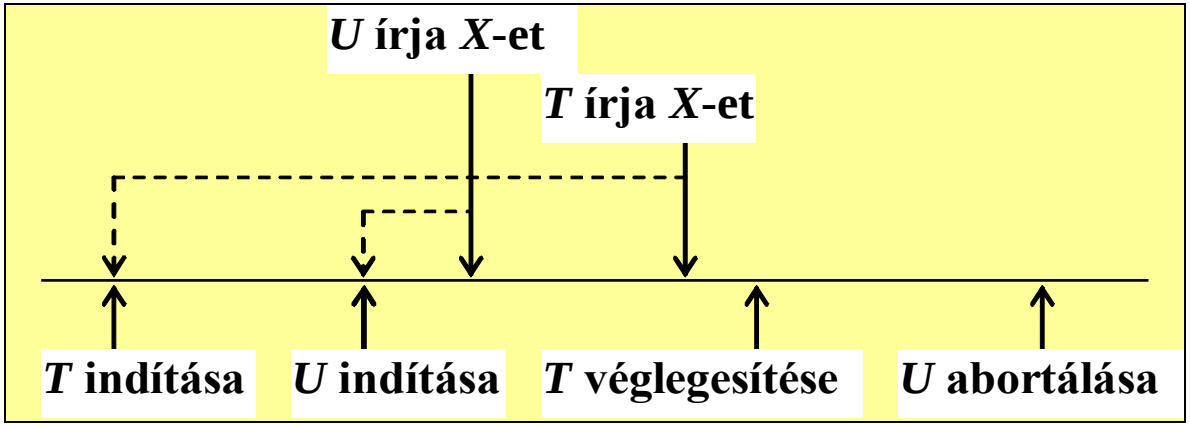


1. MEGOLDÁS: Amikor a T tranzakció írja az X adatbáziselemet, és azt látjuk, hogy X írási ideje nagyobb T időbélyegzőjénél (azaz TS(T) < WT(X)), valamint hogy az X-et író tranzakció még nincs véglegesítve (azaz C(X) hamis), akkor T-t késleltetjük mindaddig, amíg C(X) igazzá nem válik.

2. MEGOLDÁS: Amikor a T tranzakció írja az X adatbáziselemet, és azt látjuk, hogy X írási ideje nagyobb T időbélyegzőjénél (azaz TS(T) < WT(X)), T-t visszagörgetjük.

- **Nyilván ez a megoldás alacsonyabb fokú konkurenciát engedélyez, mint a véglegesítési bit bevezetése és a késleltetés, és ha el akarjuk kerülni a piszkos olvasásokat, akkor az abortálás miatt most is továbbgördülő visszagörgetéshez és nem helyreállítható ütemezéshez juthatunk.**

Egy másik probléma (Thomas-féle írás)



3. MEGOLDÁS: X minden írásánál hozzuk létre X és WT(X) egy új változatát, és csak akkor írjuk felül ezek „eredeti” változatait, ha TS(T) ≥ WT(X). Ekkor sem késleltetjük a tranzakciókat, és ha abortál az a tranzakció, melynek időbélyegzője a legnagyobb WT(X) érték, akkor megkeressük a többi letárolt WT(X) értékek közül a legnagyobbat, és ezután ezt, illetve az ehhez tartozó X értéket tekintjük „eredetinek”. Ezen az ötleten alapul a többváltozatú időbélyegzés, ami szintén megoldást nyújt a Thomas-féle írási szabály problémájára.

Látható, hogy az időbélyegzési technika alapváltozatában (amikor nem használunk véglegesítési bitet és nincs késleltetés) nem léphet fel holtponti helyzet, előfordulhat viszont továbbgyűrűző visszagörgetés és nem helyreállítható ütemezés.

Az időbélyegzőn alapuló ütemezések szabályai

- **Az ütemezőnek egy T tranzakciótól érkező olvasási vagy írási kérésre adott válaszában az alábbi választásai lehetnek:**

1. **Engedélyezi a kérést.**
2. **Abortálja T-t (ha T „megsérti a fizikai valóságot”), és egy új időbélyegzővel újraindítja. Azt az abortálást, amelyet újraindítás követ, gyakran *visszagörgetésnek* (rollback) nevezzük.**
3. **Késlelteti T-t, és később dönti el, hogy abortálja T-t, vagy engedélyezi a kérést (ha a kérés olvasás, és az olvasás piszkos is lehet, illetve ha a kérés írás, és alkalmazzuk a Thomas-féle írási szabályt).**

- **Összegezhetjük azokat a szabályokat (4 szabályt), amelyeket az időbélyegzőket használó ütemezőnek követnie kell ahhoz, hogy biztosan konfliktus-sorbarendezhető ütemezést kapjunk.**

Az időbélyegzőn alapuló ütemezés 4 szabálya

1. Tegyük fel, hogy az ütemezőhöz érkező kérés $r_T(X)$:

a) Ha $TS(T) \geq WT(X)$, az olvasás fizikailag megvalósítható:

i. Ha C(X) igaz, engedélyezzük a kérést. Ha TS(T) > RT(X), akkor RT(X) := TS(T), egyébként nem változtatjuk meg RT(X)-et.

ii. Ha C(X) hamis, késleltessük T-t addig, amíg C(X) igazzá nem válik (azaz az X-et utoljára író tranzakció nem véglegesítődik vagy abortál).

a) Ha $TS(T) < WT(X)$, az olvasás fizikailag nem megvalósítható: Visszagörgetjük T-t, vagyis abortáljuk, és újraindítjuk egy új, nagyobb időbélyegzővel.

Az időbélyegzőn alapuló ütemezés 4 szabálya

2. Tegyük fel, hogy az ütemezőhöz érkező kérés $w_T(X)$:

a) Ha $TS(T) \geq RT(X)$ és $TS(T) \geq WT(X)$, az írás fizikailag megvalósítható, és az alábbiakat kell végrehajtani:

i. X új értékének beírása;
ii. WT(X) := TS(T);
iii. C(X) := hamis.

b) Ha $TS(T) \geq RT(X)$, de $TS(T) < WT(X)$, vagy $TS(T) < WT(X) < RT(X)$, akkor az írás fizikailag megvalósítható, de X-nek már egy későbbi értéke van.

i. Ha C(X) igaz, az X előző írását végző tranzakció véglegesítve van, így egyszerűen figyelmen kívül hagyjuk X T általi írását; megengedjük, hogy T folytatódjon, és ne változtassa meg az adatbázist.
ii. Ha viszont C(X) hamis, akkor késleltetnünk kell T-t, mégpedig az 1. a) ii) pontban leírtak szerint.

c) Ha $WT(X) \leq TS(T) < RT(X)$ vagy $TS(T) < RT(X) \leq WT(X)$, az írás fizikailag nem megvalósítható, és T-t vissza kell görgetnünk.

Az időbélyegzőn alapuló ütemezés 4 szabálya

3. Tegyük fel, hogy az ütemezőhöz érkező kérés T véglegesítése (COMMIT T).

- Meg kell találnunk (az ütemező karbantartási listája alapján) az összes olyan X adatbáziselemet, amelybe T írt utoljára (WT(X) = TS(T)), és állítsuk be minden C(X)-et igazra.

- Ha vannak X „véglegesítésére” várakozó tranzakciók az 1. a) ii) és a 2. b) ii) pontoknak megfelelően (ezeket a tranzakciókat az ütemező egy másik karbantartási listáján találjuk meg), akkor meg kell ismételnünk ezen tranzakciók olvasási vagy írási kísérleteit.

Az időbélyegzőn alapuló ütemezés 4 szabálya

4. Tegyük fel, hogy az ütemezőhöz érkező kérés T abortálása (ABORT T) vagy visszagörgetése, mint az 1. b) vagy a 2. c) esetben.

- Ekkor visszavonjuk az abortált tranzakció azon írásait, amelyek olyan X adatbáziselemekre vonatkoznak, amelyekre WT(X) = TS(T). Ez azt jelenti, hogy visszaállítjuk a T által írt adatbáziselemeknek és azok írási idejének régi értékét, valamint igazra állítjuk a véglegesítési bitet, ha az írási időhöz tartozó tranzakció már véglegesítődött.

- Ezután bármely olyan tranzakcióra, amely egy X elem T általi írása miatt várakozik (1. a) ii) és 2. b) ii)), meg kell ismételnünk az olvasási vagy írási kísérletet, és meglátjuk, hogy a művelet most jogszerű-e.

Példa időbélyezésre

T_1	T_2	T_3	A	B	C
200	150	175	RT = 0	RT = 0	RT = 0
			WT = 0	WT = 0	WT = 0
			C = igaz	C = igaz	C = igaz
$r_1(B)$;				RT = 200	
	$r_2(A)$;		RT = 150		
		$r_3(C)$;			RT = 175
$w_1(B)$;				WT = 200 C = hamis	
$w_1(A)$;			WT = 200 C = hamis		
	$w_2(C)$;				
	abortál				
véglegesítődik			C = igaz	C = igaz	
		$w_3(A)$;			

Az események előfordulásának ideje szokás szerint lefelé nő. Legyen kezdetben minden adatbáziselemhez az olvasási és az írási idő is 0. A tranzakciók abban a pillanatban kapnak időbélyegzőt, amikor értesítik az ütemezőt az elindításukról. Most például bár T_1 hajtja végre az első adathozzáférést, mégsem neki van a legkisebb időbélyegzője. Tegyük fel, hogy T_2 az első, amelyik az indításáról értesíti az ütemezőt, T_3 volt a következő, és T_1-et indítottuk el utoljára.

Példa időbélyezésre

T_1	T_2	T_3	A	B	C
200	150	175	RT = 0	RT = 0	RT = 0
			WT = 0	WT = 0	WT = 0
			C = igaz	C = igaz	C = igaz
r_1(B);				RT = 200	
	r_2(A);		RT = 150		
		r_3(C);			RT = 175
w_1(B);				WT = 200 C = hamis	
w_1(A);			WT = 200 C = hamis		
	w_2(C);				
	abortál				
véglegesítődik			C = igaz	C = igaz	
		w_3(A);			

- Az első műveletben T_1 beolvassa B-t. Mivel B írási ideje kisebb, mint T_1 időbélyegzője, ez az olvasás fizikailag megvalósítható, és engedélyezzük a végrehajtást. B olvasási idejét 200-ra, T_1 időbélyegzőjére állítjuk.
- A második és a harmadik olvasási művelet hasonlóan jogszerű, és mindegyik adatbáziselem olvasási idejének értékét az őt olvasó tranzakció időbélyegzőjére állítjuk.

Példa időbélyezésre

T_1	T_2	T_3	A	B	C
200	150	175	RT = 0	RT = 0	RT = 0
			WT = 0	WT = 0	WT = 0
			C = igaz	C = igaz	C = igaz
r_1(B);				RT = 200	
	r_2(A);		RT = 150		
		r_3(C);			RT = 175
w_1(B);				WT = 200 C = hamis	
w_1(A);			WT = 200 C = hamis		
	w_2(C);				
	abortál				
véglegesítődik			C = igaz	C = igaz	
		w_3(A);			

- A negyedik lépésben T_1 írja B-t. Mivel B olvasási ideje nem nagyobb, mint T_1 időbélyegzője, az írás fizikailag megvalósítható. Mivel B írási ideje nem nagyobb, mint T_1 időbélyegzője, ténylegesen végre kell hajtanunk az írást. Amikor ezt elvégeztük, B írási idejét 200-ra növeljük, amely az őt felülíró T_1 tranzakció időbélyegzője.
- Ezután hasonlóan járunk el A-val.

Példa időbélyezésre

T_1	T_2	T_3	A	B	C
200	150	175	RT = 0	RT = 0	RT = 0
			WT = 0	WT = 0	WT = 0
			C = igaz	C = igaz	C = igaz
r_1(B);				RT = 200	
	r_2(A);		RT = 150		
		r_3(C);			RT = 175
w_1(B);				WT = 200 C = hamis	
w_1(A);			WT = 200 C = hamis		
	w_2(C);				
	abortál				
véglegesítődik			C = igaz	C = igaz	
		w_3(A);			

- Ezután T_2 megpróbálja írni C-t. C-t viszont már beolvasta a T_3 tranzakció, amelyet elméletileg a 175-ös időpontban hajtottunk végre, míg T_2-nek az értéket a 150-es időpontban kellett volna beírnia. Így T_2 olyan dologgal próbálkozik, amely fizikailag nem megvalósítható viselkedést eredményezne, tehát T_2-t vissza kell görgetnünk.

Példa időbélyezésre

T_1	T_2	T_3	A	B	C
200	150	175	RT = 0	RT = 0	RT = 0
			WT = 0	WT = 0	WT = 0
			C = igaz	C = igaz	C = igaz
$r_1(B)$;				RT = 200	
	$r_2(A)$;		RT = 150		
		$r_3(C)$;			RT = 175
$w_1(B)$;				WT = 200 C = hamis	
$w_1(A)$;			WT = 200 C = hamis		
	$w_2(C)$;				
	abortál				
véglegesítődik			C = igaz	C = igaz	
		$w_3(A)$;			

- **Az utolsó lépés, hogy T_3 írja A-t. Mivel A olvasási ideje (150) kevesebb, mint T_3 időbélyegzője (175), az írás jogszerű. Viszont A-nak már egy későbbi értéke van tárolva ebben az adatbáziselemben, mégpedig a T_1 által – elméletileg a 200-as időpontban – beírt érték. T_3-at tehát nem görgetjük vissza, de be sem írjuk az értéket. (Feltesszük, hogy T_1 időközben véglegesítődött.)**

Többváltozatú időbélyegzés

- **Az időbélyegzés egyik fontos változata, a *többváltozatú időbélyegzés* (multiversion timestamping) karbantartja az adatbáziselemek régi változatait is a magában az adatbázisban tárolt jelenlegi változaton kívül.**

- **A cél az, hogy megengedjünk olyan $r_T(X)$ olvasásokat, amelyek egyébként a T tranzakció abortálását okoznák (ugyanis X jelenlegi változatát egy T-nél későbbi tranzakció írta felül).**

- **Ilyenkor T-t X megfelelő régebbi változatának beolvasásával folytatjuk.**

- **A módszer különösen hasznos, ha az adatbáziselemek lemezblokkok vagy lapok, ugyanis ekkor csak annyit kell a pufferkezelőnek biztosítania, hogy bizonyos blokkok a memóriában legyenek, amelyek néhány jelenleg aktív tranzakció számára hasznosak lehetnek.**

Többváltozatú időbélyegzés

T_1	T_2	T_3	T_4	A
150	200	175	225	RT = 0
				WT = 0
$r_1(A)$;				RT = 150
$w_1(A)$;				WT = 150
	$r_2(A)$;			RT = 200
	$w_2(A)$;			WT = 200
		$r_3(A)$;		
		abortál		
			$r_4(A)$;	RT = 225

Ezek a tranzakciók egy közönséges, időbélyegzőn alapuló ütemező alatt működnek. Amikor T_3 megpróbálja olvasni A-t, azt találja, hogy WT(A) nagyobb, mint a saját időbélyegzője, így abortálni kell. Viszont megvan A-nak a T_1 által írt, és a T_2 által felülírt régi értéke, amely alkalmas lenne T_3-nak, hogy olvassa. Ebben a változatában A-nak 150 volt az írási ideje, ami kevesebb, mint T_3 175-ös időbélyegzője. Ha A-nak ez a régi értéke hozzáférhető lenne, T_3 engedélyt kaphatna az olvasásra, még ha ez A-nak nem is a „jelenlegi" értéke.

Többváltozatú időbélyegzés

T_1	T_2	T_3	T_4	A_0	A_{150}	A_{200}
150	200	175	225			
$r_1(A)$;				olvasás		
$w_1(A)$;					létrehozás	
	$r_2(A)$;				olvasás	
	$w_2(A)$;					létrehozás
		$r_3(A)$;			olvasás	
			$r_4(A)$;			olvasás

A-nak három változata létezik: A_0, amelyik a tranzakciók elindítása előtt létezik, A_{150}, amelyet T_1 írt, és A_{200}, amelyet T_2 írt. Az ábra mutatja azt az eseménysorozatot, amikor az egyes változatokat létrehozzuk, illetve beolvassuk. T_3-at most nem kell abortálni, ugyanis be tudja olvasni A-nak egy korábbi változatát.

- **A többváltozatú időbélyegzés tehát kiküszöböli a túl késői olvasásokat.**
- **Piszkos olvasás most is előfordulhat, de most nem csak a tranzakció késleltetésével tehetünk ellene, hanem azzal is, hogy olvasáskor megkeressük az adatbáziselem utolsó (az olvasás idejénél nem későbbi) *véglegesített* változatát. Így sosem olvasunk piszkos adatot, és nem kell késleltetnünk egy tranzakciót sem.**
- **A Thomas-féle írási szabály pedig nem alkalmazható többváltozatú időbélyegzés esetén, még akkor is létrehozzuk az adatbáziselem „új” változatát, ha az régebbi, mint a legújabb változat.**

Többváltozatú időbélyegzés

- A többváltozatú időbélyegzést használó ütemező az alábbiakban különbözik a közönséges időbélyegzéses leírt ütemezőtől:

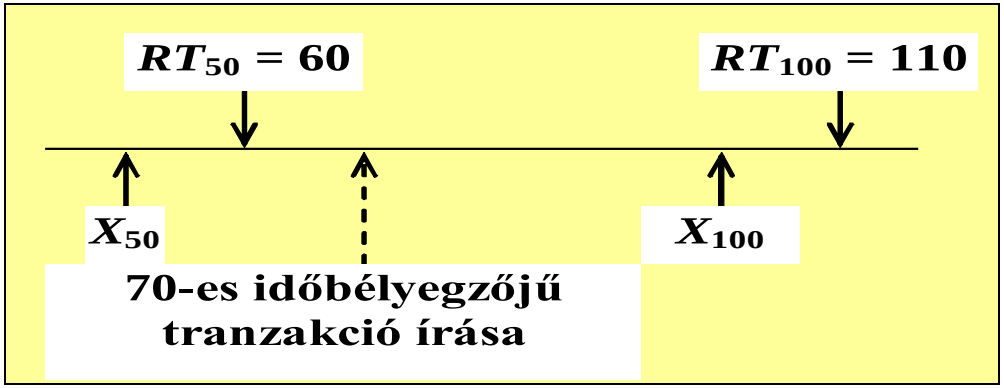
1. Amikor egy új $w_T(X)$ írás fordul elő, ha ez jogszerű, akkor az X adatbáziselemnek egy új változatát hozzuk létre, amelynek az írási ideje TS(T), és X_t-vel fogunk rá hivatkozni, ahol t = TS(T).

2. Amikor egy $r_T(X)$ olvasás fordul elő, az ütemező megkeresi X-nek azt az X_t változatát, amelyre $t \leq TS(T)$, de nincs más $X_{t'}$ változata, amelyre $t < t' \leq TS(T)$ lenne. Vagyis X-nek azt a változatát olvassa be T, amelyet T elméleti végrehajtása előtt közvetlenül írtak.

3. Az írási időket egy elem változataihoz rendeljük, és soha nem változtatjuk meg.

Többváltozatú időbélyegzés

4. Az **olvasási időket** szintén rendelhetjük a változatokhoz. Arra használjuk őket, hogy ne kelljen visszautasítanunk bizonyos írásokat, mégpedig azokat, amelyek ideje nagyobb vagy egyenlő, mint az őt időben közvetlenül megelőző változat olvasási ideje. Ha csak az utolsó változat olvasási idejét tartanánk nyilván, akkor az ilyen írásokat el kellene utasítanunk. A problémát a következő ábra szemlélteti:



X változatai X_{50} és X_{100}. X_{50} a 60-as időpontban olvasásra került, és megjelent a 70-es időbélyegzőjű T tranzakció általi új írás. Ez az írás jogszerű, mert $RT_{50} \leq TS(T)$. Ha csak az utolsó változat 110-es olvasási idejét tárolnánk, akkor erről az írásról nem tudnánk eldönteni, hogy jogszerű-e, ezért abortálnunk kellene T-t.

5. Amikor egy X_t változat t írási ideje olyan, hogy nincs t-nél kisebb időbélyegzőjű aktív tranzakció, akkor **törölhetjük X-nek az X_t-t megelőző változatait.**

Időbélyegzők és zárolások

- Bizonyos rendszerek érdekes kompromisszumot alkalmaznak:
- Az ütemező felosztja a tranzakciókat
  - csak olvasási tranzakciókra
  - és olvasási/írási tranzakciókra.

- Az olvasási/írási tranzakciókat kétfázisú zárolást használva hajtjuk végre úgy, hogy a zárolt elemek hozzáférését megakadályozzuk a többi tranzakciónak.

- A csak olvasási tranzakciókat a többváltozatú időbélyegzéssel hajtjuk végre.

- Amikor az olvasási/írási tranzakciók létrehozzák egy adatbáziselem új változatait, ezeket a változatokat úgy kezeljük, ahogyan leírtuk.

- Egy csak olvasási tranzakciónak megengedjük, hogy egy adatbáziselem bármelyik olyan változatát olvassa, amely korábban jött létre, mint a tranzakció időbélyegzője. Csak olvasási tranzakciókat emiatt soha nem kell abortálnunk, és csak nagyon ritkán kell késleltetnünk.

Konkurenciavezérlés érvényesítéssel

- Az *érvényesítés* (validation) az optimista konkurenciavezérlés másik típusa, amelyben a tranzakcióknak megengedjük, hogy zárolások nélkül hozzáférjenek az adatokhoz, és a megfelelő időben ellenőrizzük a tranzakció sorba rendezhető viselkedését.

- Az érvényesítés alapvetően abban különbözik az időbélyegzéstől, hogy itt az ütemező nyilvántartást vezet arról, mit tesznek az aktív tranzakciók, ahelyett hogy az összes adatbáziselemhez feljegyezné az olvasási és írási időt.

- Mielőtt a tranzakció írni kezdene értékeket az adatbáziselemekbe, egy „érvényesítési fázison” megy keresztül, amikor a beolvasott és kiírandó elemek halmazait összehasonlítjuk más aktív tranzakciók írásainak halmazaival. Ha fellép a fizikailag nem megvalósítható viselkedés kockázata, a tranzakciót visszagörgetjük.

Az érvényesítésen alapuló ütemező felépítése

- **Ha az érvényesítést használjuk konkurenciavezérlési módszerként, az ütemezőnek meg kell adnunk minden T tranzakcióhoz a T által olvasott és a T által írt adatbáziselemek halmazát: RS(T) az *olvasási halmaz*, WS(T) az *írási halmaz*.**

- **A tranzakciókat három fázisban hajtjuk végre:**

1. ***Olvasás.*** **Az első fázisban a tranzakció beolvassa az adatbázisból az összes elemet az olvasási halmazba, majd kiszámítja a lokális változóiban az összes eredményt, amelyet ki fog írni, és meghatározza az írási halmazt is.**
2. ***Érvényesítés.*** **A második fázisban az ütemező érvényesíti a tranzakciót oly módon, hogy összehasonlítja az olvasási és írási halmazait a többi tranzakcióéval. Ha az érvényesítés hibát jelez, akkor a tranzakciót visszagörgetjük, egyébként pedig folytatódik a harmadik fázissal.**
3. ***Írás.*** **A harmadik fázisban a tranzakció az írási halmazában lévő elemek értékeit kiírja az adatbázisba.**

Az érvényesítésen alapuló ütemező felépítése

- **Intuitív alapon minden sikeresen érvényesített tranzakcióról azt gondolhatjuk, hogy az érvényesítés pillanatában került végrehajtásra. Így az érvényesítésen alapuló ütemező a tranzakciók feltételezett soros sorrendjével dolgozik. Annak a döntésnek az alapja, hogy érvényesítsen-e egy tranzakciót vagy sem, az, hogy a tranzakciók viselkedése konzisztens legyen ezzel a soros sorrenddel.**

- **A döntés segítéséhez az ütemező fenntart három halmazt:**

- ***KEZD:*** **a már elindított, de még nem teljesen érvényesített tranzakciók halmaza. Ebben a halmazban az ütemező minden T tranzakcióhoz karbantartja KEZD(T)-t, amely T indításának időpontja.**

- ***ÉRV:*** **a már érvényesített, de a harmadik fázisban az írásokat még be nem fejezett tranzakciók halmaza. Ebben a halmazban az ütemező minden T tranzakcióhoz karbantartja KEZD(T)-t, és T érvényesítésekor ÉRV(T)-t. ÉRV(T) az az idő, amikor T végrehajtását gondoljuk a végrehajtás feltételezett soros sorrendjében.**

- ***BEF:*** **a harmadik fázist befejezett tranzakciók halmaza. Ezekhez a T tranzakciókhoz az ütemező rögzíti KEZD(T)-t, ÉRV(T)-t, és T befejezésekor BEF(T)-t. Elméletben ez a halmaz nő, de nem kell megjegyeznünk a T tranzakciót, ha BEF(T) < KEZD(U) bármely U aktív tranzakcióra (vagyis $\forall U \in KEZD \cup ÉRV$ esetén). Az ütemező így időnként tisztogathatja a *BEF* halmazt, hogy megakadályozza méretének korlátlan növekedését.**

Az érvényesítési szabályok

- **Ha az ütemező elvégzi a fenti halmazok karbantartását, akkor segítségükkel észlelheti a tranzakciók feltételezett soros sorrendjének (azaz a tranzakciók érvényesítési sorrendjének) bármely lehetséges megsértését.**

- **Mi lehet hibás, amikor a T tranzakciót megpróbáljuk érvényesíteni?**

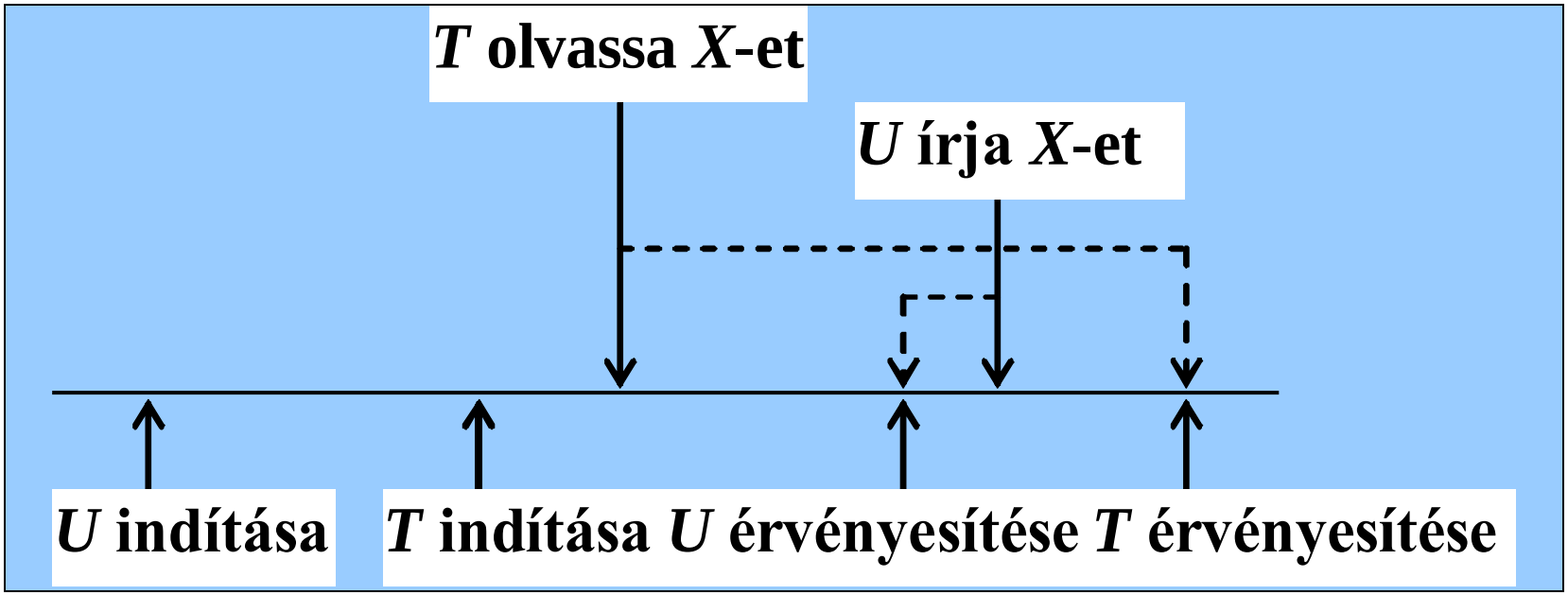
Túl korai olvasás

1. Tegyük fel, hogy van olyan U tranzakció, melyre teljesülnek a következő feltételek:

a) $U \in \textit{ÉRV} \cup \textit{BEF}$, vagyis U-t már érvényesítettük.

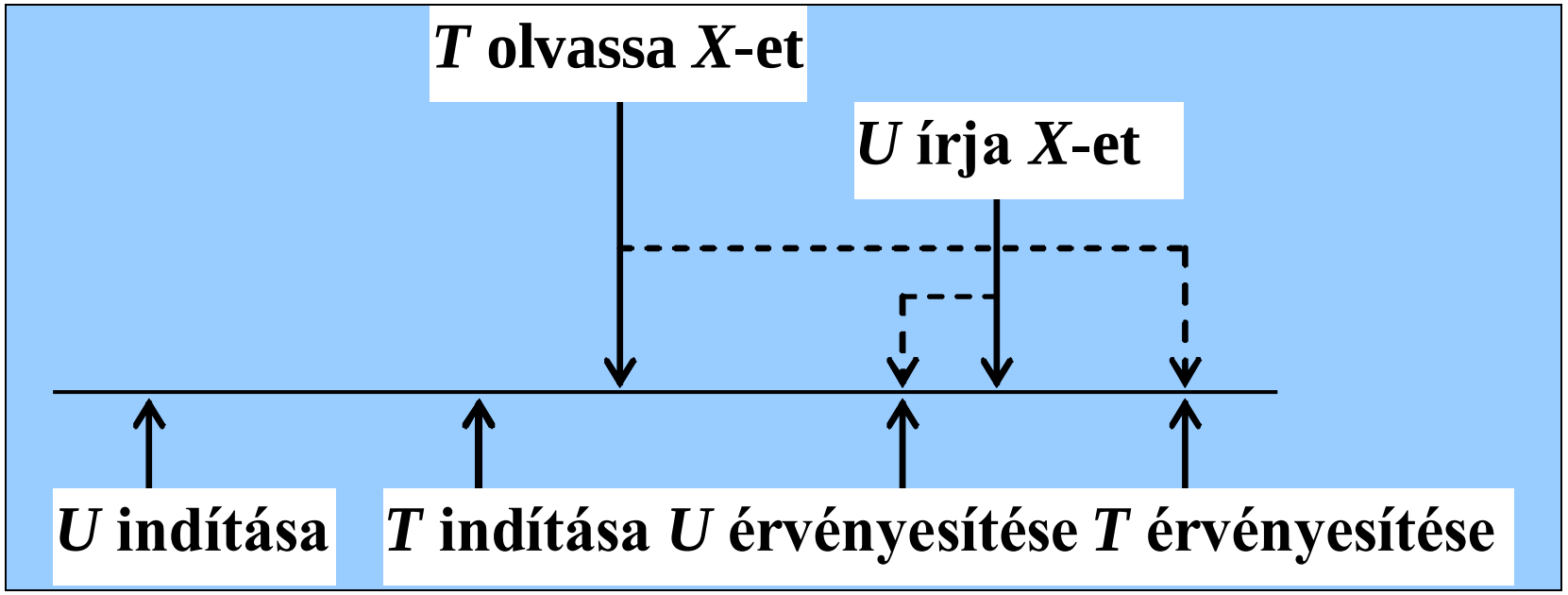
b) BEF(U) > KEZD(T), vagyis U nem fejeződött be T indítása előtt. (Ha $U \in \textit{ÉRV}$, vagyis U még nem fejeződött be T érvényesítésekor, akkor BEF(U) technikailag nem definiált, de az biztos, hogy KEZD(T)-nél nagyobbnak kell lennie.)

c) $RS(T) \cap WS(U) \neq \emptyset$, legyen X egy eleme ennek a halmaznak.



Ekkor lehetséges, hogy U azután írja X-et, miután T olvassa azt („túl korai olvasás”). Elképzelhető az is, hogy U még nem írta X-et.

Túl korai olvasás



- A pontozott vonalak kapcsolják össze a valós idejű eseményeket azzal az idővel, amikor be kellett volna következniük, ha a tranzakciókat az érvényesítés pillanatában hajtottuk volna végre.
- Mivel nem tudjuk, hogy T beolvasta-e az U-tól származó értéket, vissza kell görgetnünk T-t, hogy elkerüljük annak kockázatát, hogy T és U műveletei nem lesznek konzisztensek a feltételezett soros sorrenddel.

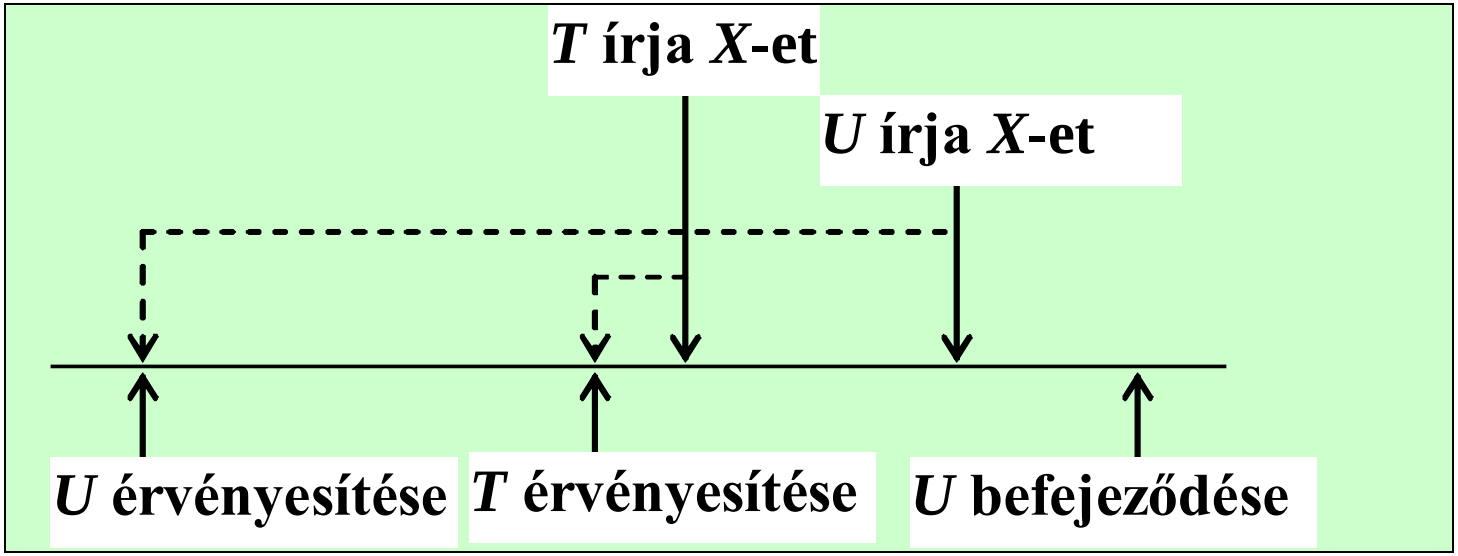
Túl korai írás

2. Tegyük fel, hogy van olyan U tranzakció, melyre teljesülnek a következő feltételek:

a) $U \in \textit{ÉRV}$, vagyis U-t már érvényesítettük.

b) BEF(U) > ÉRV(T), vagyis U-t nem fejeztük be, mielőtt T az érvényesítési fázisába lépett. (Ez a feltétel valójában mindig teljesül, mivel U még biztosan nem fejeződött be.)

c) $WS(T) \cap WS(U) \neq \emptyset$, legyen X egy eleme ennek a halmaznak.



Mind T-nek, mind U-nak írnia kell X értékét, és ha megengedjük T érvényesítését, lehetséges, hogy U előtt fogja írni X-et („túl korai írás"). Mivel nem lehetünk biztosak a dolgunkban, visszagörgetjük T-t, hogy biztosan ne szegjük meg azt a feltételezett soros sorrendet, amelyben T követi U-t.

Az érvényesítési szabályok

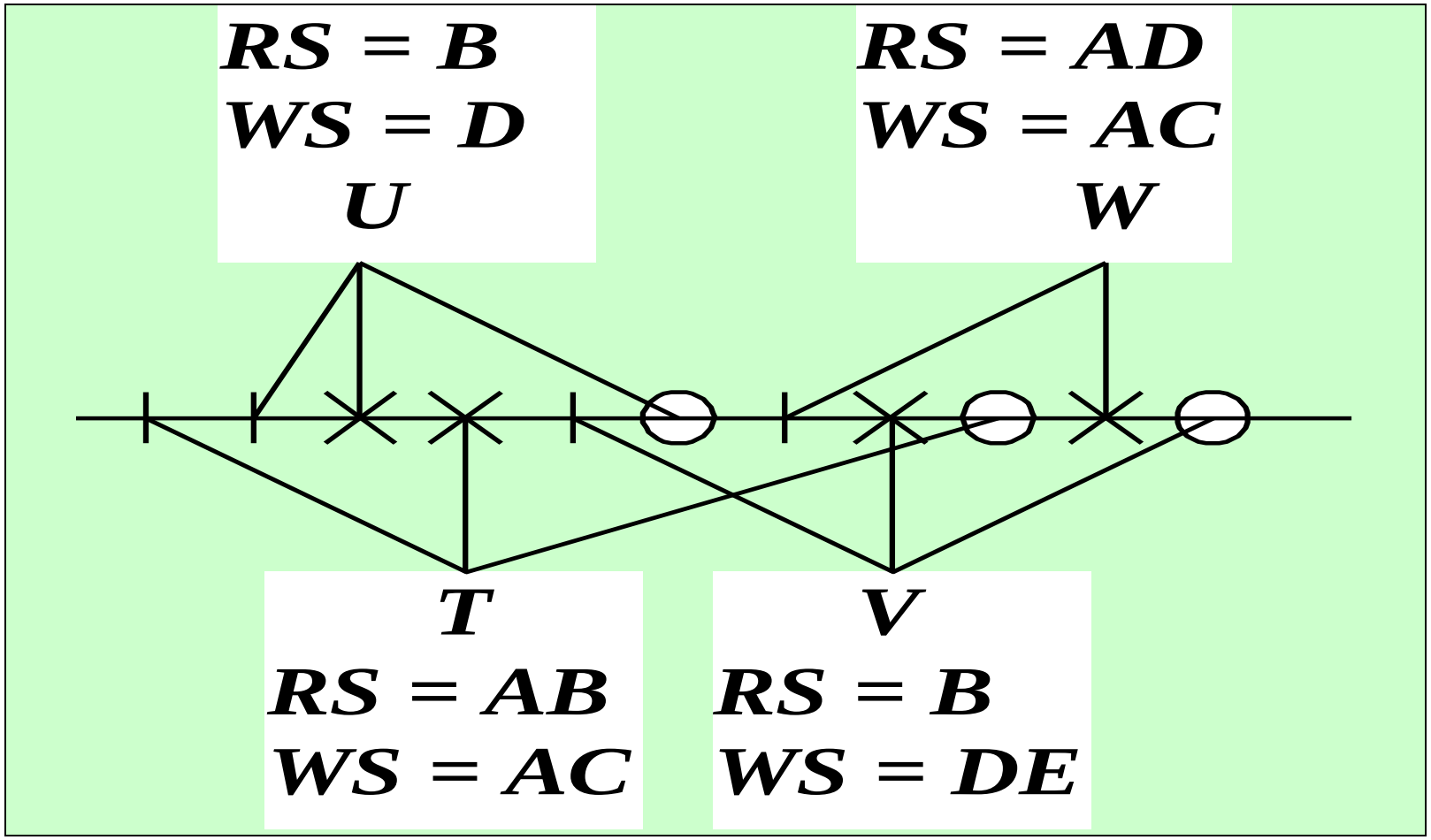
- Az előbbi két problémával kerülhetünk csak olyan helyzetbe, amikor a T által végzett írás fizikailag nem megvalósítható.
- Az 1. esetben ha U T elindítása előtt fejeződött volna be, akkor T biztosan olyan X értéket olvasna, amelyet vagy U, vagy valamely későbbi tranzakció írt.
- A 2. esetben ha U T érvényesítése előtt fejeződik be, akkor biztos, hogy U T előtt írta X-et.
- Ezek alapján a T tranzakció érvényesítésére vonatkozó észrevételeinket az alábbi szabállyal foglalhatjuk össze:

Az érvényesítési szabályok

1. Összehasonlítjuk RS(T)-t WS(U)-val, és ellenőrizzük, hogy $RS(T) \cap WS(U) = \emptyset$ minden olyan érvényesített U-ra, amely még nem fejeződött be T elindítása előtt, vagyis $U \in \mathit{ÉRV} \cup \mathit{BEF}$ és $BEF(U) > KEZD(T)$.

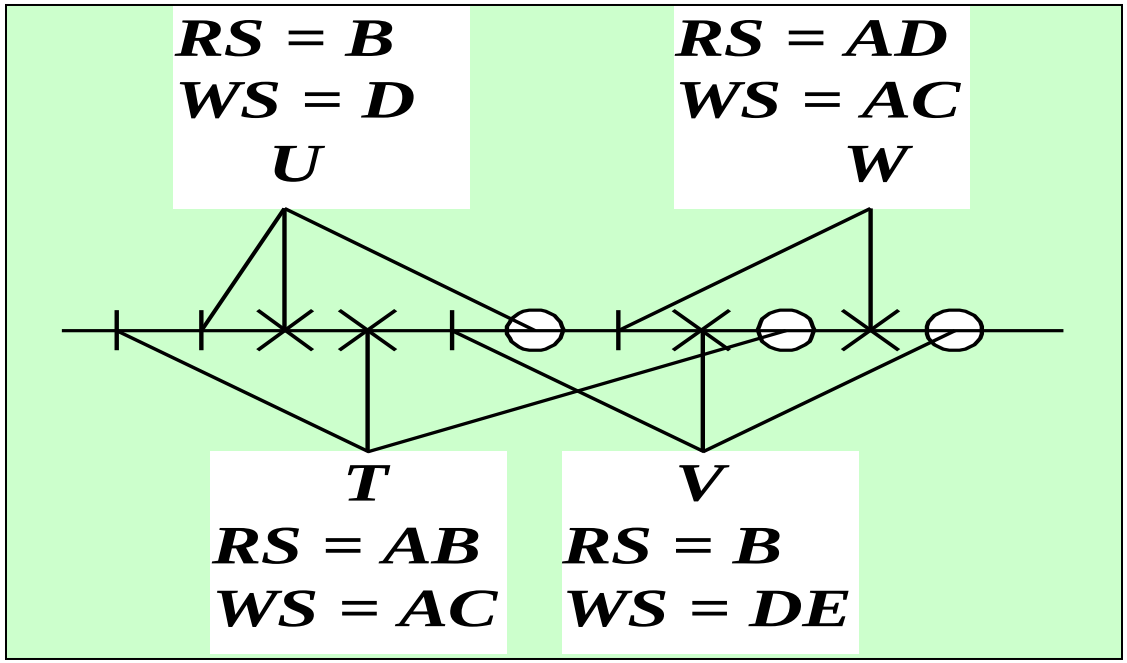
2. Összehasonlítjuk WS(T)-t WS(U)-val, és ellenőrizzük, hogy $WS(T) \cap WS(U) = \emptyset$ minden olyan érvényesített U-ra, amely még nem fejeződött be T érvényesítése előtt, vagyis $U \in \mathit{ÉRV}$ és $BEF(U) > ÉRV(T)$.

Példa: Az érvényesítési szabályok



Az ábra egy idővonalat ábrázol, amely mentén négy tranzakció (T, U, V és W) végrehajtási és érvényesítési kísérletei láthatók. I-vel jelöltük az indítást, X-szel az érvényesítést, O-val pedig a befejezést. Az ábrán láthatók az egyes tranzakciók olvasási és írási halmazai. T-t indítjuk el elsőnek, de U-t érvényesítjük elsőnek.

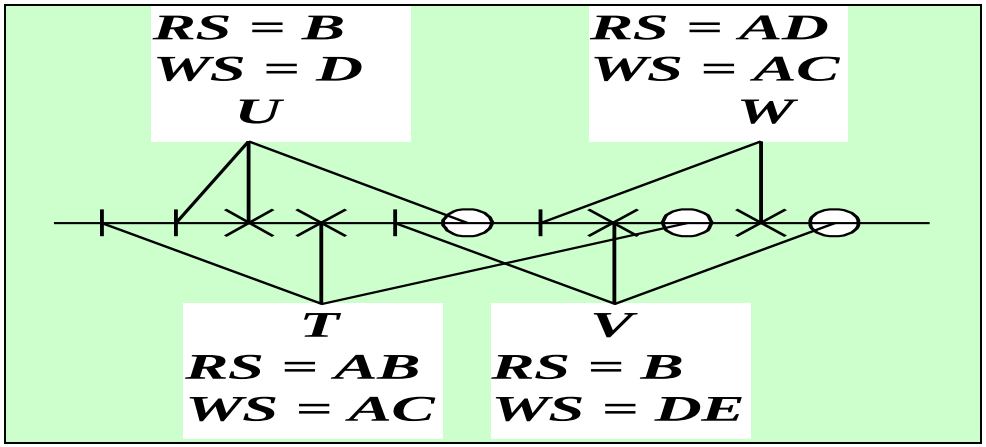
Példa: Az érvényesítési szabályok



1. Amikor U-t érvényesítjük, nincs más érvényesített tranzakció, így nem kell semmit sem ellenőriznünk. U-t érvényesítjük, és beírjuk az új értéket a D adatbáziselembe.

2. Amikor T-t érvényesítjük, U már érvényesítve van, de még nincs befejezve. Így ellenőriznünk kell, hogy T-nek sem az olvasási, sem az írási halmazában nincs semmi közös WS(U) = {D}-vel. Mivel RS(T) = {A,B} és WS(T) = {A,C}, mindkét halmazzal a metszet üres, tehát T-t érvényesítjük.

Példa: Az érvényesítési szabályok

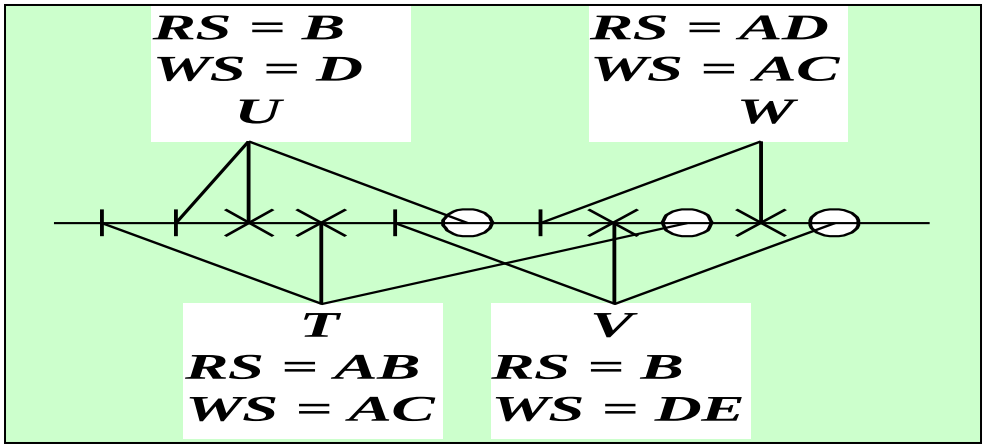


3. **Amikor V-t érvényesítjük, U már érvényesítve van és befejeződött, T pedig szintén érvényesítve van, de még nem fejeződött be. Továbbá V-t U befejeződése előtt indítottuk el. Így össze kell hasonlítanunk mind RS(V)-t, mind WS(V)-t WS(T)-vel, azonban csak RS(V)-t kell összehasonlítanunk WS(U)-val. Az eredmények:**

- **RS(V) ∩ WS(T) = {B} ∩ {A,C} = ∅;**
- **WS(V) ∩ WS(T) = {D,E} ∩ {A,C} = ∅;**
- **RS(V) ∩ WS(U) = {B} ∩ {D} = ∅.**

Ezek alapján V-t érvényesítjük.

Példa: Az érvényesítési szabályok



4. **Amikor W-t érvényesítjük, azt tapasztaljuk, hogy U már W elindítása előtt befejeződött, így nem kell elvégeznünk W és U összehasonlítását. T W érvényesítése előtt fejeződött be, de nem fejeződött be W elindítása előtt, ezért csak RS(W)-t kell összehasonlítanunk WS(T)-vel. V már érvényesítve van, de még nem fejeződött be, így össze kell hasonlítanunk mind RS(W)-t, mind WS(W)-t WS(V)-vel. Az eredmények:**

- **RS(W) ∩ WS(T) = {A,D} ∩ {A,C} = {A};**
- **RS(W) ∩ WS(V) = {A,D} ∩ {D,E} = {D};**
- **WS(W) ∩ WS(V) = {A,C} ∩ {D,E} = ∅.**

Mivel a metszetek nem mind üresek, W-t nem érvényesítjük, hanem visszagörgetjük, így nem ír értéket sem A-ba, sem C-be.

A három konkurenciavezérlési technika működésének összehasonlítása

- **Hasonlítsuk őket össze először a tárigény szempontjából:**

1. ***Zárolások:*** **A zártábla által lefoglalt tár a zárolt adatbáziselemek számával arányos.**
2. ***Időbélyegzés:*** **Egy naiv megvalósításban minden adatbáziselem olvasási és írási idejéhez szükségünk van tárra, akár hozzáférünk az adott elemhez, akár nem. Egy körültekintőbb megvalósítás azonban az összes olyan időbélyegzőt mínusz végtelen értékűnek tekinti, amely a legkorábbi aktív tranzakciónál korábbi tranzakcióhoz tartozik, és nem rögzíti ezeket. Ez esetben a zártáblával analóg méretű táblában tudjuk tárolni az olvasási és írási időket, amelyben csak a legújabban elért adatbáziselemek szerepelnek.**
3. ***Érvényesítés:*** **Tárat használunk az időbélyegzőkhöz és minden jelenleg aktív tranzakció olvasási/írási halmazaihoz, hozzávéve még egy pár olyan tranzakciót, amelyek azután fejeződnek be, miután valamelyik jelenleg aktív tranzakció elkezdődött.**

A három konkurenciavezérlési technika működésének összehasonlítása

- **Hasonlítsuk őket össze először a tárigény szempontjából:**

- **Így mindegyik megközelítésben az összes aktív tranzakcióra felhasznált tár a tranzakciók által hozzáfért adatbáziselemek számának az összegével megközelítőleg arányos.**

- **Az időbélyegzés és az érvényesítés kicsit több helyet használhat fel, ugyanis nyomon kell követnünk a korábban véglegesített tranzakciók bizonyos hozzáféréseit, amelyeket a zártábla nem rögzítene.**

- **Az érvényesítéssel kapcsolatban egy lényeges probléma, hogy a tranzakcióhoz tartozó írási halmazt az írások elvégzése előtt kell már ismernünk (de a tranzakció számításainak befejeződése után).**

A három konkurenciavezérlési technika működésének összehasonlítása

- Összehasonlíthatjuk a módszereket abból a szempontból is, hogy **késleltetés nélkül befejeződnek-e** a tranzakciók. A három módszer hatékonysága attól függ, hogy vajon a tranzakciók közötti egymásra hatás erős vagy gyenge, azaz milyen valószínűséggel akar egy tranzakció hozzáférni egy olyan elemhez, amelyhez egy konkurens tranzakció már hozzáfért:

- A **zárolás késlelteti** a tranzakciókat, azonban elkerüli a visszagörgetéseket, még ha erős is az egymásra hatás. Az **időbélyegzés** és az **érvényesítés nem késlelteti a tranzakciókat**, azonban visszagörgetést okozhatnak, amely a késleltetésnek egy problémásabb formája, azonfelül erőforrásokat is pazarol.

- Ha gyenge az egymásra hatás, akkor sem az időbélyegzés, sem az érvényesítés nem okoz sok visszagörgetést, és előnyösebbek lehetnek a zárolásnál, ugyanis ezeknek általában alacsonyabbak a költségei, mint a zárolási ütemezőnek.

- Amikor szükséges a visszagörgetés, az időbélyegzők hamarabb feltárják a problémákat, mint az érvényesítés, amely mindig hagyja, hogy a tranzakció elvégezze az összes belső munkáját, mielőtt megnézné, hogy vissza kell-e görgetni a tranzakciót.

Az Oracle konkurenciavezérlési technikája

- **Az Oracle alapvetően a zárolás módszerét használja a konkurenciavezérléshez.**
- **Felhasználói szinten a zárolási egység lehet a tábla vagy annak egy sora.**
- **A zárakat az ütemező helyezi el és oldja fel, de lehetőség van arra is, hogy a felhasználó (alkalmazás) kérjen zárat.**
- **Az Oracle alkalmazza a kétfázisú zárolást, a figyelmeztető protokollt és a többváltozatú időbélyegzőket is némi módosítással.**

Többszintű konkurenciavezérlés Oracle-ben

- **Az Oracle minden lekérdezés számára biztosítja az olvasási konzisztenciát, azaz a lekérdezés által olvasott adatok egy időpillanatból (a lekérdezés kezdetének pillanatából) származnak.**
  - **Emiatt a lekérdezés sohasem olvas piszkos adatot,**
  - **és nem látja azokat a változtatásokat sem, amelyeket a lekérdezés végrehajtása alatt véglegesített tranzakciók eszközöltek.**

  Ezt *utasítás szintű olvasási konzisztenciának* nevezzük.

- **Kérhetjük egy tranzakció összes lekérdezése számára is a konzisztencia biztosítását, ez a *tranzakció szintű olvasási konzisztencia*.**
  - **Ezt úgy érhetjük el, hogy a tranzakciót sorba rendezhető**
  - **vagy csak olvasás módban futtatjuk.**
  - **Ekkor a tranzakció által tartalmazott összes lekérdezés a tranzakció indításakor fennálló adatbázis-állapotot látja, kivéve a tranzakció által korábban végrehajtott módosításokat.**

Többszintű konkurenciavezérlés Oracle-ben

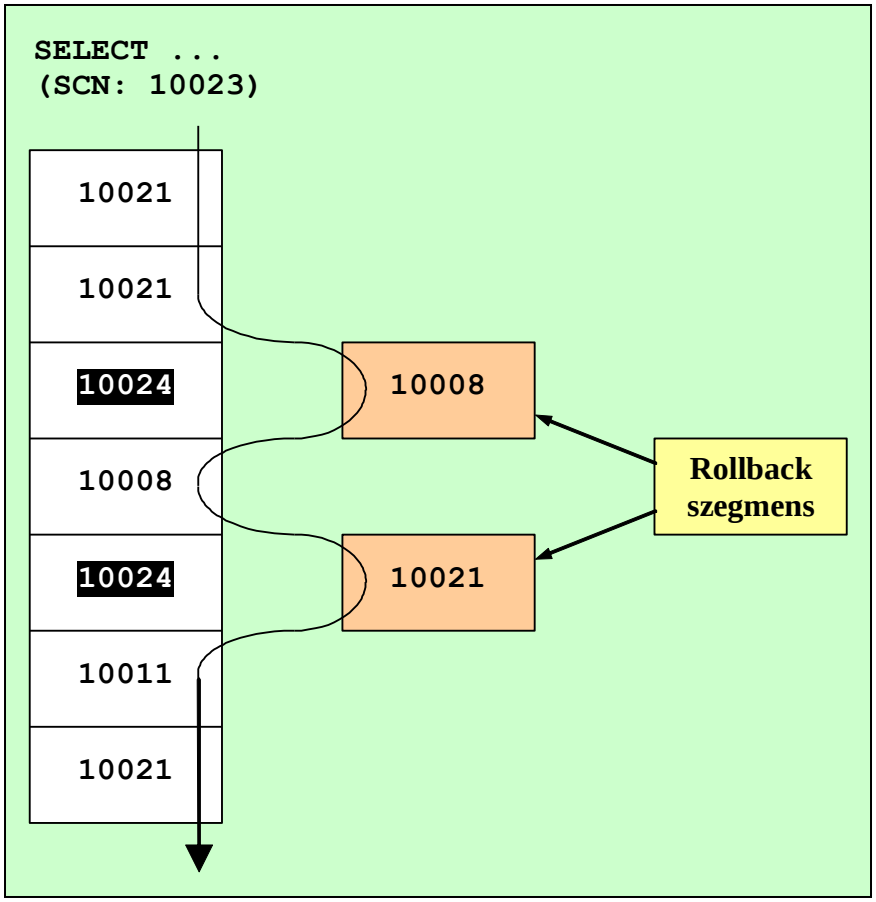
- **A kétféle olvasási konzisztencia eléréséhez az Oracle a rollback szegmensekben található információkat használja fel.**

- **A rollback szegmensek tárolják azon adatok régi értékeit, amelyeket még nem véglegesített vagy nemrég véglegesített tranzakciók változtattak meg.**

- **Amint egy lekérdezés vagy tranzakció megkezdi működését, meghatározódik a *system change number* (SCN) aktuális értéke. Az SCN a blokkokhoz mint adatbáziselemekhez tartozó időbélyegzőnek tekinthető.**

Többszintű konkurenciavezérlés Oracle-ben

- Ahogy a lekérdezés olvassa az adatblokkokat, összehasonlítja azok SCN-jét az aktuális SCN értékkel, és csak az aktuálisnál kisebb SCN-nel rendelkező blokkokat olvassa be a tábla területéről.
- A nagyobb SCN-nel rendelkező blokkok esetén a rollback szegmensből megkeresi az adott blokk azon verzióját, amelyhez a legnagyobb olyan SCN érték tartozik, amely kisebb, mint az aktuális, és már véglegesített tranzakció hozta létre.



A 10023 előtt indult tranzakciók módosításait már elvileg láthatja.

A 10024-es blokkok esetén a régi példányokat a rollback szegmensből olvassuk ki.

A tranzakcióelkülönítési szintek

- **Az SQL92 ANSI/ISO szabvány a tranzakcióelkülönítés négy szintjét definiálja, amelyek abban különböznek egymástól, hogy az alábbi három jelenség közül melyeket engedélyezik:**

- ***piszkos olvasás:*** **a tranzakció olyan adatot olvas, amelyet egy másik, még nem véglegesített tranzakció írt;**

- ***nem ismételhető (fuzzy) olvasás:*** **a tranzakció újraolvas olyan adatokat, amelyeket már korábban beolvasott, és azt találja, hogy egy másik, már véglegesített tranzakció módosította vagy törölte őket;**

- ***fantomok olvasása:*** **a tranzakció újra végrehajt egy lekérdezést, amely egy adott keresési feltételnek eleget tevő sorokkal tér vissza, és azt találja, hogy egy másik, már véglegesített tranzakció további sorokat szúrt be, amelyek szintén eleget tesznek a feltételnek.**

A négy tranzakcióelkülönítési szint a következő:

	piszkos olvasás	nem ismételhető olvasás	fantomok olvasása
nem olvasásbiztos (read uncommitted)	lehetséges	lehetséges	lehetséges
olvasásbiztos (read committed)	nem lehetséges	lehetséges	lehetséges
megismételhető olvasás (repeatable read)	nem lehetséges	nem lehetséges	lehetséges
sorbarendezhető (serializable)	nem lehetséges	nem lehetséges	nem lehetséges

Az Oracle ezek közül az

1. **olvasásbiztos és a**
2. **sorbarendezhető elkülönítési szinteket ismeri, valamint egy**
3. ***csak olvasás* (read-only) módot, amely nem része a szabványnak.**

Az Oracle tranzakcióelkülönítési szintjei

1. ***Olvasásbiztos:***

- **SET TRANSACTION ISOLATION LEVEL READ COMMITTED;**
- **Ez az alapértelmezett tranzakcióelkülönítési szint.**
- **Egy tranzakció minden lekérdezése csak a lekérdezés (és nem a tranzakció) elindítása előtt véglegesített adatokat látja.**
- **Piszkos olvasás sohasem történik.**
- **A lekérdezés két végrehajtása között a lekérdezés által olvasott adatokat más tranzakciók megváltoztathatják, ezért előfordulhat nem ismételhető olvasás és fantomok olvasása is.**
- **Olyan környezetekben célszerű ezt a szintet választani, amelyekben várhatóan kevés tranzakció kerül egymással konfliktusba.**

Az Oracle tranzakcióelkülönítési szintjei

2. *Sorbarendezhető:*

- **SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;**
- **A sorba rendezhető tranzakciók csak a tranzakció elindítása előtt véglegesített változásokat látják, valamint azokat, amelyeket maga a tranzakció hajtott végre INSERT, UPDATE és DELETE utasítások segítségével.**
- **A sorba rendezhető tranzakciók nem hajtanak végre nem ismételhető olvasásokat, és nem olvasnak fantomokat.**
- **Ezt a szintet olyan környezetekben célszerű használni, amelyekben nagy adatbázisok vannak, és rövidek a tranzakciók, amelyek csak kevés sort módosítanak, valamint ha kicsi az esélye annak, hogy két konkurens tranzakció ugyanazokat a sorokat módosítja, illetve ahol a hosszú (sokáig futó) tranzakciók elsősorban csak olvasási tranzakciók.**
- **Az Oracle csak akkor engedi egy sor módosítását egy sorbarendezhető tranzakciónak, ha el tudja dönteni, hogy az adott sor korábbi változásait olyan tranzakciók hajtották végre, amelyek még a sorbarendezhető tranzakció elindítása előtt véglegesítődtek. Ennek eldöntésére az Oracle a blokkokban tárolt vezérlőinformációkat használja, amelyek megmondják, hogy az adott blokkban az egyes sorokat mely tranzakciók módosították, és hogy ezek a módosítások véglegesítettek-e. (SCN – írási időbélyegző és commit bit)**
- **Amennyiben egy sorbarendezhető tranzakció megpróbál módosítani vagy törölni egy sort, amelyet egy olyan tranzakció változtatott meg, amely a sorba rendezhető tranzakció indításakor még nem véglegesítődött, az Oracle hibaüzenetet ad („Cannot serialize access for this transaction”).**

Az Oracle tranzakcióelkülönítési szintjei

3. Csak olvasás:

- **SET TRANSACTION READ ONLY;**
- **A csak olvasást végző tranzakciók csak a tranzakció elindítása előtt véglegesített változásokat látják, és nem engednek meg INSERT, UPDATE és DELETE utasításokat.**

A zárolási rendszer

- Bármelyik elkülönítési szintű tranzakció használja a sor szintű zárolást, ezáltal egy T tranzakciónak várnia kell, ha olyan sort próbál írni, amelyet egy még nem véglegesített konkurens tranzakció módosított.

- T megvárja, míg a másik tranzakció véglegesítődik vagy abortál, és felszabadítja a zárat.
  - Ha abortál, akkor T végrehajthatja a sor módosítását, függetlenül az elkülönítési szintjétől.
  - Ha a másik tranzakció véglegesítődik,
    - akkor T csak akkor hajthatja végre a módosítást, ha az elkülönítési szintje az olvasásbiztos.
    - Egy sorbarendezhető tranzakció ilyenkor abortál, és „Cannot serialize access” hibaüzenetet ad.

- A zárakat az Oracle automatikusan kezeli, amikor SQL-utasításokat hajt végre. Mindig a legkevésbé szigorú zármódot alkalmazza, így biztosítja a legmagasabb fokú konkurenciát. Lehetőség van arra is, hogy a felhasználó kérjen zárat.

A zárolási rendszer

- **Egy tranzakcióban szereplő SQL-utasításnak adott zár a tranzakció befejeződéséig fennmarad (kétfázisú zárolás). Ezáltal a tranzakció egy utasítása által végrehajtott változtatások csak azon tranzakciók számára láthatók, amelyek azután indultak el, miután az első tranzakció véglegesítődött.**

- **Az Oracle akkor szabadítja fel a zárakat,**
  - **amikor a tranzakció véglegesítődik**
  - **vagy abortál,**
  - **illetve ha visszagörgetjük a tranzakciót egy mentési pontig (ekkor a mentési pont után kapott zárak szabadulnak fel).**

Zármódok

- **Az Oracle a zárakat a következő általános kategóriákba sorolja:**

1. **DML-zárak (adatzárak): az adatok védelmére szolgálnak;**
2. **DDL-zárak (szótárzárak): a sémaobjektumok (pl. táblák) szerkezetének a védelmére valók;**
3. **belső zárak: a belső adatszerkezetek, adatfájlok védelmére szolgálnak, kezelésük teljesen automatikus.**

- **DML-zárakat két szinten kaphatnak a tranzakciók:**
  - **sorok szintjén**
  - **és teljes táblák szintjén.**

- **Egy tranzakció tetszőleges számú sor szintű zárat fenntarthat.**

- **Sorok szintjén csak egyféle zármód létezik,**
  - **a kizárólagos (írási – X).**

Zármódok

- A többváltozatú időbélyegzés és a sor szintű zárolás kombinációja azt eredményezi, hogy a tranzakciók csak akkor versengenek az adatokért, ha ugyanazokat a sorokat próbálják meg írni. Részletesebben:
  - Adott sorok olvasója nem vár ugyanazon sorok írójára.
  - Adott sorok írója nem vár ugyanazon sorok olvasójára, hacsak az olvasó nem a SELECT … FOR UPDATE utasítást használja, amely zárolja is a beolvasott sorokat.
  - Adott sorok írója csak akkor vár egy másik tranzakcióra, ha az is ugyanazon sorokat próbálja meg írni ugyanabban az időben.

- Egy tranzakció kizárólagos DML-zárat kap minden egyes sorra, amelyet az alábbi utasítások módosítanak:
  - INSERT,
  - UPDATE,
  - DELETE
  - és SELECT … FOR UPDATE.

Zármódok

- **Ha egy tranzakció**
  - **egy tábla egy sorára zárat kap,**
  - **akkor a teljes táblára is zárat kap,**

  hogy elkerüljük az olyan DDL-utasításokat, amelyek felülírnák a tranzakció változtatásait, illetve hogy fenntartsuk a tranzakciónak a táblához való hozzáférés lehetőségét.

- **Egy tranzakció tábla szintű zárat kap, ha a táblát az alábbi utasítások módosítják:**
  - **INSERT,**
  - **UPDATE,**
  - **DELETE,**
  - **SELECT … FOR UPDATE**
  - **és LOCK TABLE.**

- **Táblák szintjén ötféle zármódot különböztetünk meg:**
  1. ***row share* (RS) vagy *subshare* (SS),**
  2. ***row exclusive* (RX) vagy *subexclusive* (SX),**
  3. ***share* (S),**
  4. ***share row exclusive* (SRX) vagy *share-subexclusive* (SSX)**
  5. **és *exclusive* (X).**

- **Ezek a módok a felsorolás sorrendjében egyre erősebbek.**

Zármódok

- **A következő táblázat összefoglalja, hogy az egyes utasítások milyen zármódot vonnak maguk után, és hogy milyen zármódokkal kompatibilisek:**

SQL-utasítás	Zármód	RS	RX	S	SRX	X
SELECT ... FROM tábla	-	I	I	I	I	I
INSERT INTO tábla	RX	I	I	N	N	N
UPDATE tábla	RX	I*	I*	N	N	N
DELETE FROM tábla	RX	I*	I*	N	N	N
SELECT ... FROM tábla ... FOR UPDATE	RS	I*	I*	I*	I*	N
LOCK TABLE tábla IN ROW SHARE MODE	RS	I	I	I	I	N
LOCK TABLE tábla IN ROW EXCLUSIVE MODE	RX	I	I	N	N	N
LOCK TABLE tábla IN SHARE MODE	S	I	N	I	N	N
LOCK TABLE tábla IN SHARE ROW EXCLUSIVE MODE	SRX	I	N	N	N	N
LOCK TABLE tábla IN EXCLUSIVE MODE	X	N	N	N	N	N

* Igen, ha egy másik tranzakció nem tart fenn konfliktusos sor szintű zárat, különben várakozik.

1. **Az RS zár azt jelzi, hogy a zárat fenntartó tranzakció sorokat zárolt a táblában, és később módosítani kívánja őket.**
2. **Az RX zár általában azt jelzi, hogy a zárat fenntartó tranzakció egy vagy több módosítást hajtott végre a táblában.**

Zármódok

- **A következő táblázat összefoglalja, hogy az egyes utasítások milyen zármódot vonnak maguk után, és hogy milyen zármódokkal kompatibilisek:**

SQL-utasítás	Zármód	RS	RX	S	SRX	X
SELECT ... FROM tábla	-	I	I	I	I	I
INSERT INTO tábla	RX	I	I	N	N	N
UPDATE tábla	RX	I*	I*	N	N	N
DELETE FROM tábla	RX	I*	I*	N	N	N
SELECT ... FROM tábla ... FOR UPDATE	RS	I*	I*	I*	I*	N
LOCK TABLE tábla IN ROW SHARE MODE	RS	I	I	I	I	N
LOCK TABLE tábla IN ROW EXCLUSIVE MODE	RX	I	I	N	N	N
LOCK TABLE tábla IN SHARE MODE	S	I	N	I	N	N
LOCK TABLE tábla IN SHARE ROW EXCLUSIVE MODE	SRX	I	N	N	N	N
LOCK TABLE tábla IN EXCLUSIVE MODE	X	N	N	N	N	N

* Igen, ha egy másik tranzakció nem tart fenn konfliktusos sor szintű zárat, különben várakozik.

3. **Az S zárat csak a LOCK TABLE utasítással lehet kérni. Más tranzakció nem módosíthatja a táblát. Ha több tranzakció egyidejűleg S zárat tart fenn ugyanazon a táblán, akkor egyikük sem módosíthatja a táblát (még akkor sem, ha az egyik a SELECT ... FOR UPDATE utasítás hatására sor szintű zárakat tart fenn). Más szóval egy S zárat fenntartó tranzakció csak akkor módosíthatja a táblát, ha nincs másik olyan tranzakció, amely szintén S zárral rendelkezik ugyanezen a táblán.**

Zármódok

- **A következő táblázat összefoglalja, hogy az egyes utasítások milyen zármódot vonnak maguk után, és hogy milyen zármódokkal kompatibilisek:**

SQL-utasítás	Zármód	RS	RX	S	SRX	X
SELECT ... FROM tábla	-	I	I	I	I	I
INSERT INTO tábla	RX	I	I	N	N	N
UPDATE tábla	RX	I*	I*	N	N	N
DELETE FROM tábla	RX	I*	I*	N	N	N
SELECT ... FROM tábla ... FOR UPDATE	RS	I*	I*	I*	I*	N
LOCK TABLE tábla IN ROW SHARE MODE	RS	I	I	I	I	N
LOCK TABLE tábla IN ROW EXCLUSIVE MODE	RX	I	I	N	N	N
LOCK TABLE tábla IN SHARE MODE	S	I	N	I	N	N
LOCK TABLE tábla IN SHARE ROW EXCLUSIVE MODE	SRX	I	N	N	N	N
LOCK TABLE tábla IN EXCLUSIVE MODE	X	N	N	N	N	N

* Igen, ha egy másik tranzakció nem tart fenn konfliktusos sor szintű zárat, különben várakozik.

4. **Az SRX zárat csak a LOCK TABLE utasítással lehet kérni. Egy adott táblán egy időpillanatban csak egy tranzakció tarthat fenn SRX zárat. Más tekintetben megegyezik az S zárral.**

5. **Az X zárat csak a LOCK TABLE utasítással lehet kérni. Egy adott táblán egy időpillanatban csak egy tranzakció tarthat fenn X zárat, és ennek joga van a táblát kizárólagosan írni. Más tranzakciók ilyenkor csak olvashatják a táblát, de nem módosíthatják, és nem helyezhetnek el rajta zárakat.**

Zármódok

- A következő táblázat összefoglalja, hogy az egyes utasítások milyen zármódot vonnak maguk után, és hogy milyen zármódokkal kompatibilisek:

SQL-utasítás	Zármód	RS	RX	S	SRX	X
SELECT ... FROM tábla	-	I	I	I	I	I
INSERT INTO tábla	RX	I	I	N	N	N
UPDATE tábla	RX	I*	I*	N	N	N
DELETE FROM tábla	RX	I*	I*	N	N	N
SELECT ... FROM tábla ... FOR UPDATE	RS	I*	I*	I*	I*	N
LOCK TABLE tábla IN ROW SHARE MODE	RS	I	I	I	I	N
LOCK TABLE tábla IN ROW EXCLUSIVE MODE	RX	I	I	N	N	N
LOCK TABLE tábla IN SHARE MODE	S	I	N	I	N	N
LOCK TABLE tábla IN SHARE ROW EXCLUSIVE MODE	SRX	I	N	N	N	N
LOCK TABLE tábla IN EXCLUSIVE MODE	X	N	N	N	N	N

* Igen, ha egy másik tranzakció nem tart fenn konfliktusos sor szintű zárat, különben várakozik.

- A lekérdezések tehát sohasem járnak zárolásokkal, így más tranzakciók is lekérdezhetik vagy akár módosíthatják a lekérdezett táblát, akár a kérdéses sorokat is.
- Az Oracle ezért gyakran hívja a lekérdezéseket *nemblokkoló lekérdezéseknek*. Másrészt a lekérdezések sohasem várnak zárfeloldásra, mindig végrehajtódhatnak.

Zármódok

- A módosító DML-utasítások és a SELECT … FOR UPDATE utasítás az érintett sorokra kizárólagos sor szintű zárakat helyeznek, így más tranzakciók nem módosíthatják vagy törölhetik a zárolt sorokat, amíg a zárakat elhelyező tranzakció nem véglegesítődik vagy abortál.
- Ha az utasítás alkérdést tartalmaz, az nem jár sor szintű zárolással.
- Az alkérdések garantáltan konzisztensek a lekérdezés kezdetekor fennálló adatbázis-állapottal, és nem látják a tartalmazó módosító utasítás által véghezvitt változtatásokat.
- Egy tranzakcióban lévő lekérdezés látja a tranzakció egy korábbi módosító utasítása által végrehajtott változtatásokat, de nem látja a tartalmazó tranzakciónál később elindult tranzakciók módosításait.

Zárak felminősítése és kiterjesztése

- A módosító utasítás a sor szintű zárakon kívül a módosított sorokat tartalmazó táblákra is elhelyez egy-egy RX zárat.

  Ha a tartalmazó tranzakció már fenntart egy S, SRX vagy X zárat a kérdéses táblán,

  akkor nem kap külön RX zárat is,

  Ha pedig RS zárat tartott fenn,

  akkor az felminősül RX zárrá.

- Mivel sorok szintjén csak egyfajta zármód létezik (kizárólagos), nincs szükség felminősítésre.

- Táblák szintjén az Oracle automatikusan felminősít egy zárat erősebb módúvá, amikor szükséges. Például egy SELECT ... FOR UPDATE utasítás RS módban zárolja a táblát. Ha a tranzakció később módosít a zárolt sorok közül néhányat, az RS mód automatikusan felminősül RX módra.

Zárak felminősítése és kiterjesztése

- **Zárak kiterjesztésének (escalation) nevezzük azt a folyamatot, amikor a szemcsézettség egy szintjén (pl. sorok szintjén) lévő zárakat az adatbázis-kezelő rendszer a szemcsézettség egy magasabb szintjére (pl. a tábla szintjére) emeli.**
- **Például ha a felhasználó sok sort zárol egy táblában, egyes rendszerek ezeket automatikusan kiterjesztik a teljes táblára.**
- **Ezáltal csökken a zárak száma, viszont nő a zárolt elemek zármódjának erőssége.**
- **Az Oracle nem alkalmazza a zárkiterjesztést, mivel az megnöveli a holtpontok kialakulásának kockázatát.**

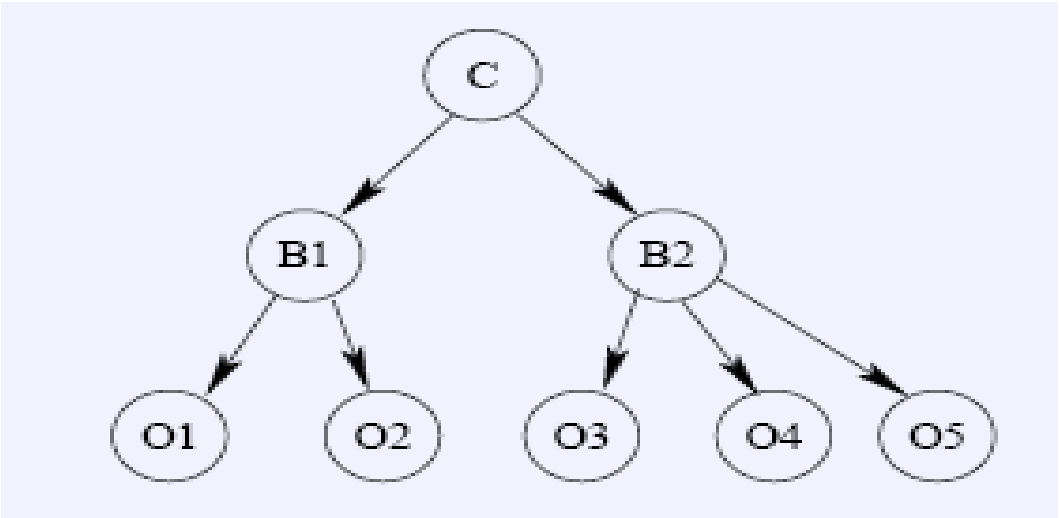
Feladatok

- Vegyünk egy objektum orientált adatbázist. A C osztály objektumait két blokkban tároljuk a B_1-ben és a B_2-ben.

  A B_1 tartalmazza az O_1 és O_2 objektumokat, míg a B_2 az O_3; O_4; O_5 objektumokat.

  Adjuk meg a zárolási kérések sorozatát és a figyelmeztető protokoll alapú ütemező feladatát az alábbi kérési sorozatokhoz. Feltehetjük, hogy minden kérés éppen azelőtt fordul elő, mint amikor éppen szükség van rá, és minden zárfeloldás a tranzakció befejeztével történik. Használjuk az S/X modellt.

$$r_1(O_1); w_2(O_2); r_2(O_3); w_1(O_4)$$

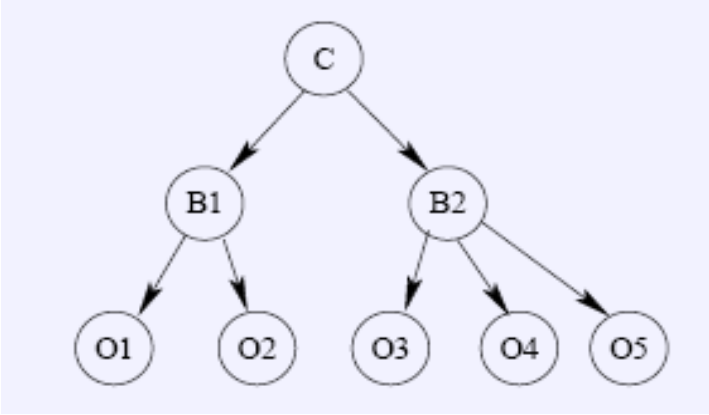


Feladatok

$r_1(O_1); w_2(O_2); r_2(O_3); w_1(O_4)$

- Először az O_1-re kell majd zárat tenni: $IS_1(C)$; $IS_1(B1)$; $S_1(O1)$.
- Utána O_2-re kell majd zárat tenni: $IX_2(C)$;$IX_2(B_1)$;$X_2(O_2)$.
- A harmadik sorozatban O_3-ra kell majd zárat tenni:
  $IS_2(C)$; $IS_2(B2)$; $S_2(O3)$.
- Ezután $T2$ felengedi a zárakat és a figyelmeztetéseit:
  $UNLOCK_2(O_3)$; $UNLOCK_2(B_2)$; $UNLOCK_2(O_2)$; $UNLOCK_2(B_1)$; $UNLOCK_2(C)$
- Majd O_4-re kell majd zárat tenni:
  $IX_1(C)$;$IX_1(B_2)$;$X_1(O_4)$.
- Végül T_1 felengedi a zárakat és figyelmeztetéseit:

$UNLOCK_1(O_4)$; $UNLOCK_1(B_4)$; $UNLOCK_1(O_1)$; $UNLOCK_1(B_1)$; $UNLOCK_1(C)$



Feladatok

Az alábbi legális ütemezés két olyan tranzakció utasításait tartalmazza, melyek betartják a figyelmeztető protokollt. Hogy nézhet ki az ütemezésben szereplő adategységek egymásba ágyazottságát reprezentáló fa, ha tudjuk, hogy a gyökérnek legfeljebb 3 gyereke van? (Ha több lehetséges eset van, akkor mindet add meg).

$IX_1(E)$, $IX_1(H)$, $IX_2(E)$, $X_1(A)$, $X_1(C)$, $UNLOCK_1(A)$, $X_2(F)$,

$UNLOCK_1(H)$, $UNLOCK_2(F)$, $UNLOCK_1(C)$, $UNLOCK_2(E)$, $UNLOCK_1(E)$

Feladatok

Az alábbi legális ütemezés két olyan tranzakció utasításait tartalmazza, melyek betartják a figyelmeztető protokollt. Hogy nézhet ki az ütemezésben szereplő adategységek egymásba ágyazottságát reprezentáló fa, ha tudjuk, hogy a gyökérnek legfeljebb 3 gyereke van? (Ha több lehetséges eset van, akkor mindet add meg).

$IX_1(E)$, $IX_1(H)$, $IX_2(E)$, $X_1(A)$, $X_1(C)$, $UNLOCK_1(A)$, $X_2(F)$,

$UNLOCK_1(H)$, $UNLOCK_2(F)$, $UNLOCK_1(C)$, $UNLOCK_2(E)$, $UNLOCK_1(E)$

Megoldás:

T_2 miatt biztos, hogy *E* a gyökér és *F* ennek a fia.

T_1 miatt biztos, hogy *H* is *E*-nek a fia.

***A* és *C* helyzete a kérdéses még. Két eset lehetséges:**

Az *A* csúcs a *H* gyereke: *C* nem lehet se *A*, se *H* gyereke, mert később oldjuk fel *C*-n a zárat, mint *A*-n és *H*-n, így ekkor *C* csak *E* gyereke lehet és ez összhangban is van a zárolással. Ez egy lehetséges megoldás.

Az *A* csúcs az *E* gyereke: *C* nem lehet se *A*, se *H* gyereke a zárfeloldások miatt, de *E*-jé sem lehet, mert a gyökérnek csak három gyereke lehet. Így ezen az ágon nem kapunk megoldást.

Feladatok

Az alább megadott tranzakciók mindegyikénél tételezzük fel, hogy beszúrjuk a LOCK és UNLOCK műveletet miden egyes adatbáziselemhez, amihez hozzáférünk:

$$r1(A); w1(B).$$

Adjuk meg, hogy a zárolási, feloldási, olvasási és írási műveleteknek hány olyan sorrendje lehet, ha a zárolások megfelelőek és a zárolás i) kétfázisú, ii) nem kétfázisú.

Feladatok

Az alább megadott tranzakciók mindegyikénél tételezzük fel, hogy beszúrjuk a LOCK és UNLOCK műveletet miden egyes adatbáziselemhez, amihez hozzáférünk:

$r1(A); w1(B).$

Adjuk meg, hogy a zárolási, feloldási, olvasási és írási műveleteknek hány olyan sorrendje lehet, ha a zárolások megfelelőek és a zárolás i) kétfázisú, ii) nem kétfázisú.

Megoldás:

Ha a zárolás megfelelő, akkor csak ennek összefésülései jönnek szóba:

a) $l_1(A); r_1(A); u_1(A)$

b) $l_1(B); w_1(B); u_1(B)$

ii) csak akkor nem kétfázisú, ha az egyik megelőzi a másikat: kétféle ilyen van.

i) Hány összefésülés van összesen? Ismétléses kombináció, vagyis a 6 pozícióból melyik 3 lesz az elsőből: 6 alatt a 3, azaz 6/(3!3!)=20 és ebből 2 kétfázisú, azaz 18 a válasz.

Feladatok

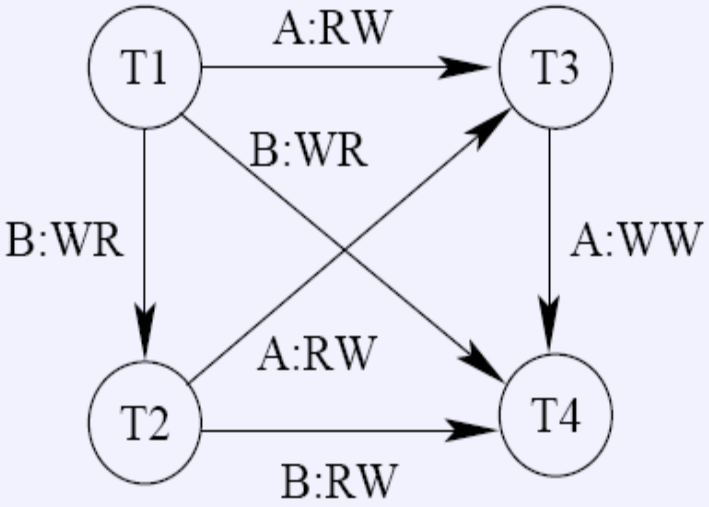
(0)	T_1	T_2	T_3	T_4
(1)		RLOCK A		
(2)	RLOCK A			
(3)	WLOCK C			
(4)	UNLOCK C			
(5)			RLOCK C	
(6)	WLOCK B			
(7)	UNLOCK B			
(8)				RLOCK B
(9)	UNLOCK A			
(10)		UNLOCK A		
(11)			WLOCK A	
(12)				RLOCK C
(13)		WLOCK D		
(14)				UNLOCK B
(15)			UNLOCK C	
(16)		RLOCK B		
(17)			UNLOCK A	
(18)				WLOCK A
(19)		UNLOCK B		
(20)				WLOCK B
(21)				UNLOCK B
(22)		UNLOCK D		
(23)				UNLOCK C
(24)				UNLOCK A

Az alábbi legális ütemezés négy tranzakció zárjait tartalmazza az RLOCK/WLOCK (S/X) modellben.

Rajzoljuk fel a megelőzési gráfot! Sorbarendezhető-e az ütemezés? Ha igen, milyen soros ütemezések ekvivalensek az eredeti ütemezéssel?

Feladatok

(0)	T_1	T_2	T_3	T_4
(1)		RLOCK A		
(2)	RLOCK A			
(3)	WLOCK C			
(4)	UNLOCK C			
(5)			RLOCK C	
(6)	WLOCK B			
(7)	UNLOCK B			
(8)				RLOCK B
(9)	UNLOCK A			
(10)		UNLOCK A		
(11)			WLOCK A	
(12)				RLOCK C
(13)		WLOCK D		
(14)				UNLOCK B
(15)			UNLOCK C	
(16)		RLOCK B		
(17)			UNLOCK A	
(18)				WLOCK A
(19)		UNLOCK B		
(20)				WLOCK B
(21)				UNLOCK B
(22)		UNLOCK D		
(23)				UNLOCK C
(24)				UNLOCK A



Nincs kör benne, tehát sorbarendezhető: egy soros ütemezése van: T_1, T_2, T_3, T_4

Feladatok

1. Tegyük fel, hogy az alábbi műveletsorozatban minden egyes olvasás- és írásműveletet közvetlenül megelőzi az RLOCK ill. a WLOCK igénylése. Tegyük továbbá fel, hogy a zárak feloldása a tranzakció utolsó művelete után történik meg. Adjuk meg azokat a műveleteket, melyek végrehajtását az ütemező megtagadja, és mondjuk meg, hogy létrejön-e holtpont. Hogyan alakul a műveletek végrehajtása során a várakozási gráf? Ha létrejön holtpont, ABORT-áljuk az egyik tranzakciót, és mutassuk meg, hogyan folytatódik a műveletsorozat!

$$r_1(A), r_2(B), w_1(C), r_3(D), r_4(E), w_3(B), w_2(C), w_4(A), w_1(D)$$

1. Tegyük fel, hogy az alábbi műveletsorozatban minden egyes olvasás- és írásműveletet közvetlenül megelőzi az RLOCK ill. a WLOCK igénylése. Tegyük továbbá fel, hogy a zárak feloldása a tranzakció utolsó művelete után történik meg. Adjuk meg azokat a műveleteket, melyek végrehajtását az ütemező megtagadja, és mondjuk meg, hogy létrejön-e holtpont. Hogyan alakul a műveletek végrehajtása során a várakozási gráf? Ha létrejön holtpont, ABORT-áljuk az egyik tranzakciót, és mutassuk meg, hogyan folytatódik a műveletsorozat!

$$r_1(A), r_2(B), w_1(C), r_3(D), r_4(E), w_3(B), w_2(C), w_4(A), w_1(D)$$

Megoldás:

Először sorban kérünk zárakat A, B, C, D, E-re.

$$rl_1(A), r_1(A), rl_2(B), r_2(B), wl_1(C), w_1(C), rl_3(D), r_3(D), rl_4(E)$$

Az első probléma a $wl_3(B)$, hiszen ekkor van zár még $lr_2(B)$. $wl_3(B)$ megtagadva, várakozási gráfba (T_3, T_2) él, T_3 vár

$wl_2(C)$ megtagadva, várakozási gráfba (T_2, T_1) él, T_2 vár

$wl_4(A)$ megtagadva, várakozási gráfba (T_4, T_1) él, T_4 vár

$wl_1(D)$ megtagadva, várakozási gráfba (T_1, T_3) él, kört kapunk, holtpont alakul ki.

ABORT T_1, ekkor eltűnik (T_2, T_1) él és a (T_4, T_1) él a várakozási gráfból, ami így DAG lesz.

Ezért pl. T_2, T_3, T_4 sorrendben lefuthat a többi tranzakció:

$$ul_1(A), ul_1(C), wl_2(C), w_2(C), ul_2(C), ul_2(B), wl_3(B), w_3(B), ul_3(B), wl_1(A), w_4(A), ul_4(A), ul_4(E)$$

Feladatok

Tekintsük az alábbi (csak olvasásokból és írásokból álló) ütemezést:

$$r_2(A),\quad w_3(B),\quad r_1(A),\quad w_2(B),\quad w_1(C)$$

(Itt $r_2(A)$ jelentése: a második tranzakció olvassa A-t, $w_3(B)$ jelentése: a harmadik tranzakció írja B-t.)

Az egyszerű tranzakciómodellt használva illessz be zárkéréseket a fenti ütemezésbe oly módon, hogy legális zárolást kapjunk és

(a) ne kövesse mindegyik tranzakció a 2PL-t, de (a zárkérések alapján döntve) az ütemezés sorosítható legyen,

Feladatok

Tekintsük az alábbi (csak olvasásokból és írásokból álló) ütemezést:

$$r_2(A),\quad w_3(B),\quad r_1(A),\quad w_2(B),\quad w_1(C)$$

(Itt $r_2(A)$ jelentése: a második tranzakció olvassa A-t, $w_3(B)$ jelentése: a harmadik tranzakció írja B-t.)

Az egyszerű tranzakciómodellt használva illessz be zárkéréseket a fenti ütemezésbe oly módon, hogy legális zárolást kapjunk és

(a) ne kövesse mindegyik tranzakció a 2PL-t, de (a zárkérések alapján döntve) az ütemezés sorosítható legyen,

Megoldás:

$$l_2(A),\ r_2(A),\ u_2(A),$$
$$l_3(B),\ w_3(B),\ u_3(B),$$
$$l_1(A),\ r_1(A),\ u_1(A),$$
$$l_2(B),\ w_2(B),\ u_2(B),$$
$$l_1(C),\ w_1(C),\ u_1(C)$$

Ha felrajzoljuk a sorosítási gráfot: $T_3 \rightarrow T_2 \rightarrow T_1$, tehát sorosítható.

Feladatok

Tekintsük az alábbi (csak olvasásokból és írásokból álló) ütemezést:

$$r_2(A),\quad w_3(B),\quad r_1(A),\quad w_2(B),\quad w_1(C)$$

(Itt $r_2(A)$ jelentése: a második tranzakció olvassa A-t, $w_3(B)$ jelentése: a harmadik tranzakció írja B-t.)

Az egyszerű tranzakciómodellt használva illessz be zárkéréseket a fenti ütemezésbe oly módon, hogy legális zárolást kapjunk és

(a) ne kövesse mindegyik tranzakció a 2PL-t, de (a zárkérések alapján döntve) az ütemezés sorosítható legyen,

(b) mindegyik tranzakció kövesse a 2PL-t, de (a zárkérések alapján döntve) ne legyen sorosítható az ütemezés,

Feladatok

Tekintsük az alábbi (csak olvasásokból és írásokból álló) ütemezést:

$$r_2(A),\quad w_3(B),\quad r_1(A),\quad w_2(B),\quad w_1(C)$$

(Itt $r_2(A)$ jelentése: a második tranzakció olvassa A-t, $w_3(B)$ jelentése: a harmadik tranzakció írja B-t.)

Az egyszerű tranzakciómodellt használva illessz be zárkéréseket a fenti ütemezésbe oly módon, hogy legális zárolást kapjunk és

(a) ne kövesse mindegyik tranzakció a 2PL-t, de (a zárkérések alapján döntve) az ütemezés sorosítható legyen,

(b) mindegyik tranzakció kövesse a 2PL-t, de (a zárkérések alapján döntve) ne legyen sorosítható az ütemezés,

Ilyet nem lehet adni, mert tanultuk azt a tételt, hogy ha minden tranzakció követi a 2PL-t, akkor sorosítható lesz az ütemezés.

Feladatok

Tekintsük az alábbi (csak olvasásokból és írásokból álló) ütemezést:

$$r_2(A), \quad w_3(B), \quad r_1(A), \quad w_2(B), \quad w_1(C)$$

(Itt $r_2(A)$ jelentése: a második tranzakció olvassa A-t, $w_3(B)$ jelentése: a harmadik tranzakció írja B-t.)

Az egyszerű tranzakciómodellt használva illessz be zárkéréseket a fenti ütemezésbe oly módon, hogy legális zárolást kapjunk és

(a) ne kövesse mindegyik tranzakció a 2PL-t, de (a zárkérések alapján döntve) az ütemezés sorosítható legyen,

(b) mindegyik tranzakció kövesse a 2PL-t, de (a zárkérések alapján döntve) ne legyen sorosítható az ütemezés,

(c) mindegyik tranzakció kövesse a 2PL-t, és (a zárkérések alapján döntve) legyen sorosítható az ütemezés.

Megoldás:

Feladatok

Tekintsük az alábbi (csak olvasásokból és írásokból álló) ütemezést:

$$r_2(A),\quad w_3(B),\quad r_1(A),\quad w_2(B),\quad w_1(C)$$

(Itt $r_2(A)$ jelentése: a második tranzakció olvassa A-t, $w_3(B)$ jelentése: a harmadik tranzakció írja B-t.)

Az egyszerű tranzakciómodellt használva illessz be zárkéréseket a fenti ütemezésbe oly módon, hogy legális zárolást kapjunk és

(c) mindegyik tranzakció kövesse a 2PL-t, és (a zárkérések alapján döntve) legyen sorosítható az ütemezés.

Megoldás:

Az ötlet az, hogy az $l_2(B)$-t előbbre lehet hozni és így előbb fel lehet oldani a zárat A-n.

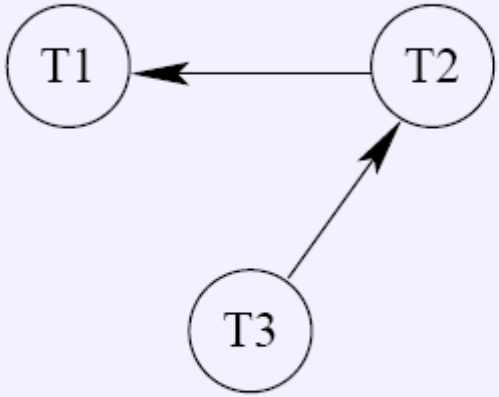
$l_2(A),\ r_2(A),$

$l_3(B),\ w_3(B),\ u_3(B),$

$l_2(B),\ u_2(A),\ l_1(A),\ r_1(A),$

$w_2(B),\ u_2(B),$

$l_1(C),\ w_1(C),\ u_1(A),\ u_1(C)$



(Mivel ez 2PL, a tétel szerint sorosítható.)