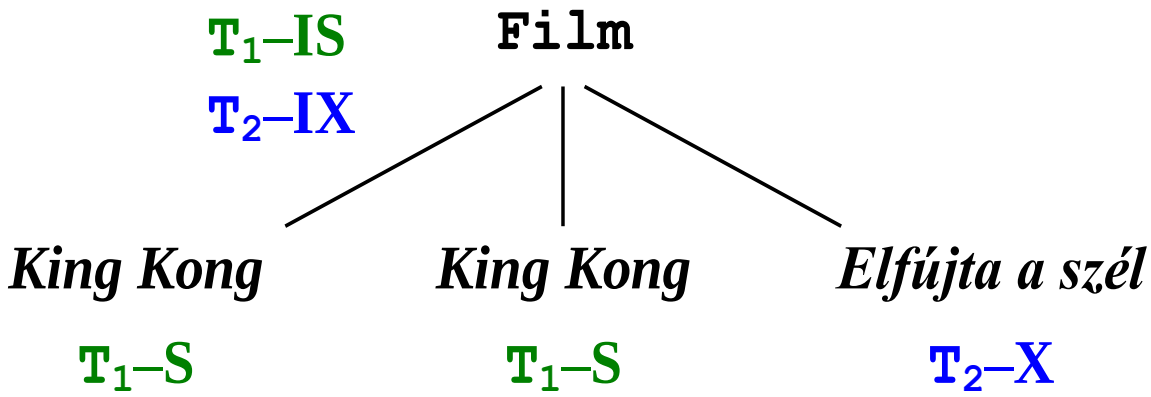
Figyelmeztető zárak

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 sora és oszlopa 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.**

Csoportos mód a szándékzárolásokhoz

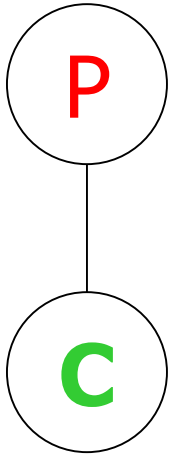
Zárolás \ Kérés	IS	IX	S	SIX	X
IS	T	T	T	T	F
IX	T	T	F	F	F
S	T	F	T	F	F
SIX	T	F	F	F	F
X	F	F	F	F	F

T="igen"

F="nem"

Csoportos mód a szándékzárolásokhoz

P szülőn ilyen a zár	A C gyerek ilyen zárat kaphat
IS	IS, S
IX	IS, S, IX, X, SIX
S	S, IS
SIX	X, IX, SIX
X	semmit



Csoportos mód a szándékzárolásokhoz

Szabályok:

(1) A kompatibilitási mátrixnak megfelelően helyezhetjük el a zárakat.

(2) Először a gyökeret zároljuk valamilyen módban.

(3) Egy Q csúcsot egy T_i csak akkor zárolhat S vagy IS módban, ha a szülő(Q)-t T_i IX vagy IS módban már zárolta.

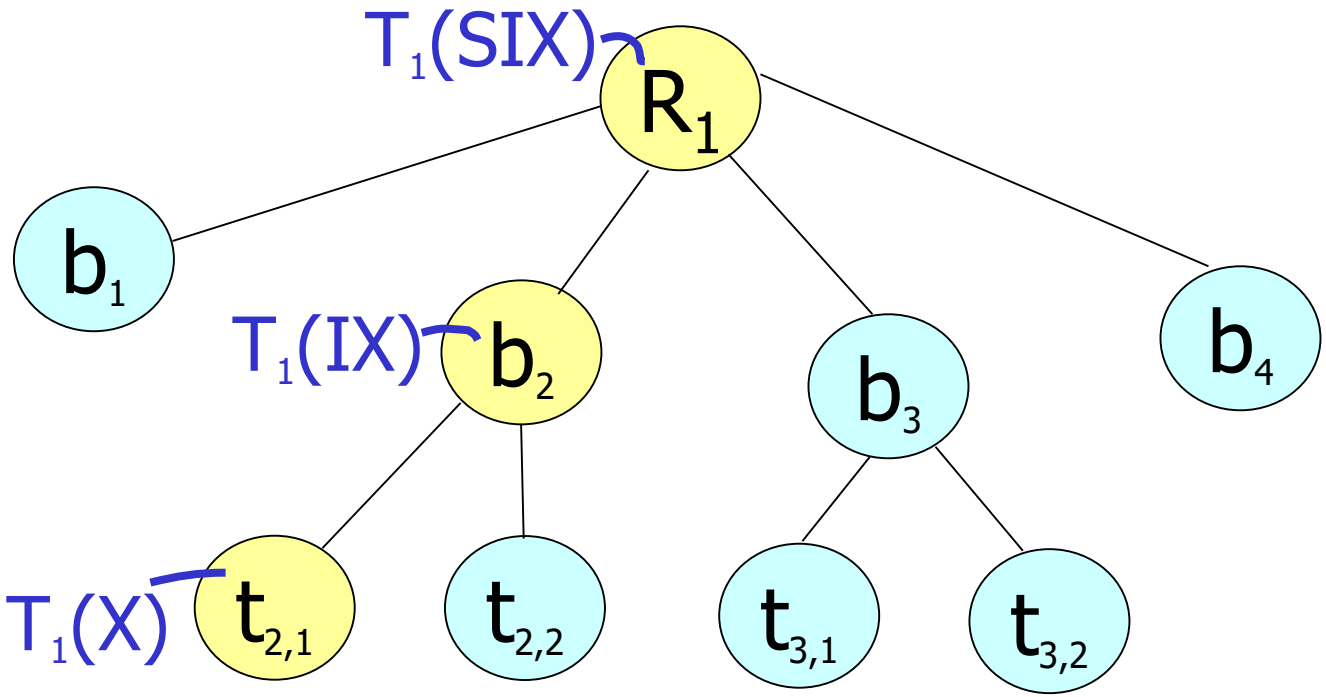
(4) Egy Q csúcsot egy T_i csak akkor zárolhat X,SIX,IX módban, ha a szülő(Q)-t T_i IX,SIX módban már zárolta.

(5) T_i a két fázisú zárolási protokollnak tesz eleget.

(6) T_i csak akkor engedhet el egy zárat a Q csúcson, ha Q egyik gyerekét sem zárolja a T_i .

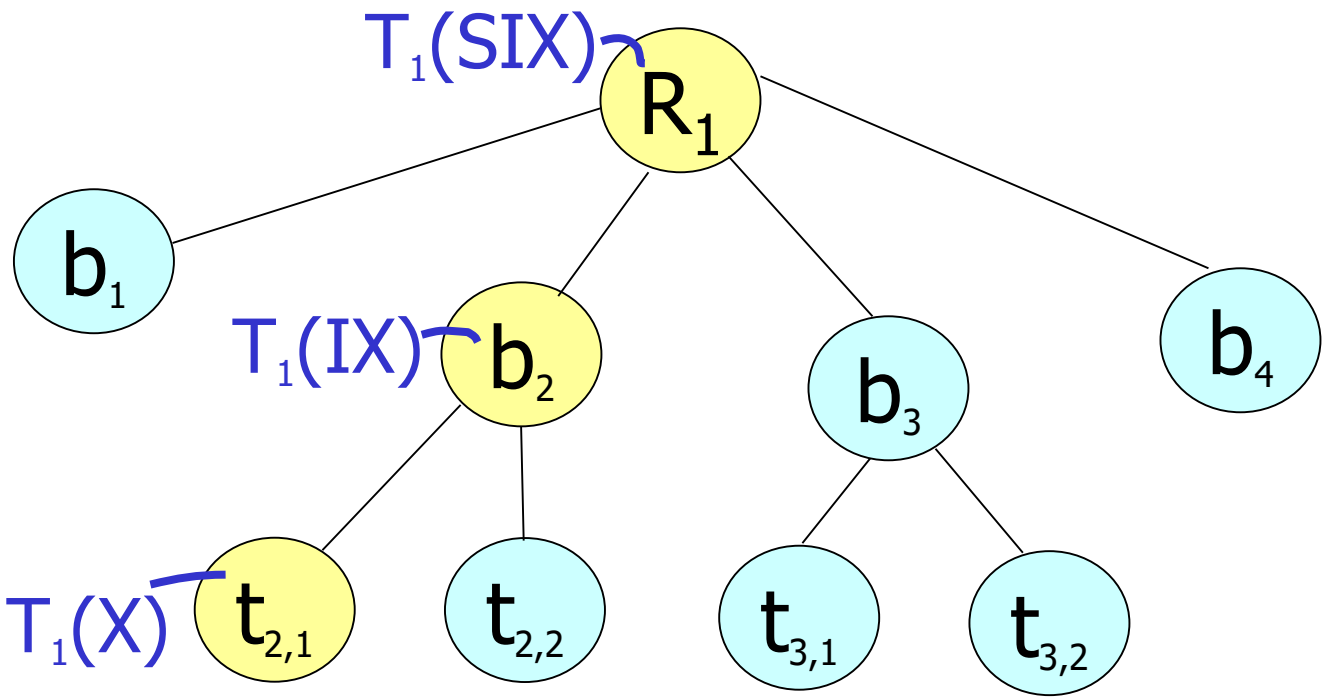
Csoportos mód a szándékzárolásokhoz

A T_2 kaphat-e $t_{2,2}$ sorra **S** zárat?



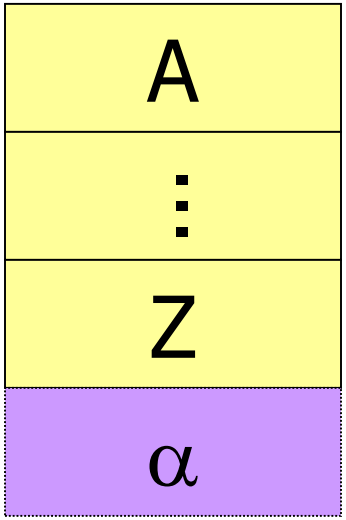
Csoportos mód a szándékzárolásokhoz

A T_2 kaphat-e $t_{2,2}$ sorra **X** zárat?



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.**

Megszorítások is sérülhetnek

Példa: R reláció (ID#,név,...)

megszorítás: ID# kulcs

sorszintű zárolás

R	ID#	Név	....
o1	55	Smith	
o2	75	Jones	

T_1: Insert <04,Kerry,…> into R
T_2: Insert <04,Bush,…> into R

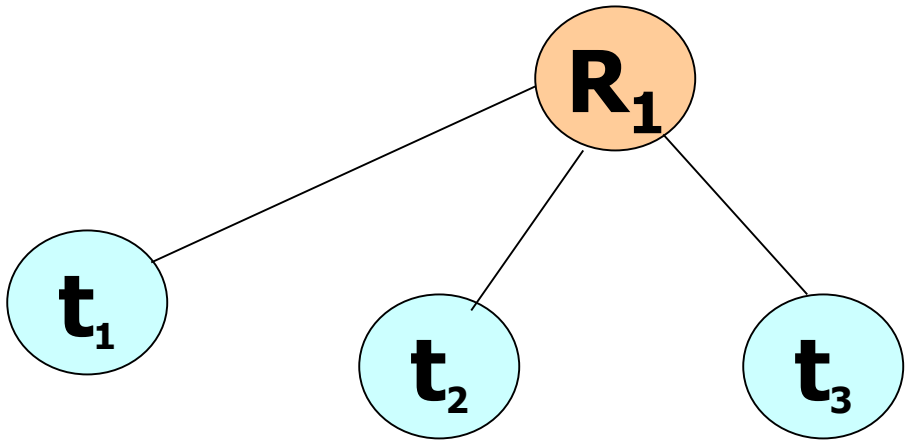
Előtte olvasási S zárak elhelyezése a többi objektumra:

T_1	T_2
$S_1(o_1)$	$S_2(o_1)$
$S_1(o_2)$	$S_2(o_2)$
Megszorítás ellenőrzése	Megszorítás ellenőrzés
⋮	⋮
Insert o_3[04,Kerry,..]	
	Insert o_4[04,Bush,..]

Megsérül a megszorítás!

Megoldás

- **Mielőtt egy Q csúcsot beszúrunk, zároljuk a szülő(Q) csúcsot X módban!**

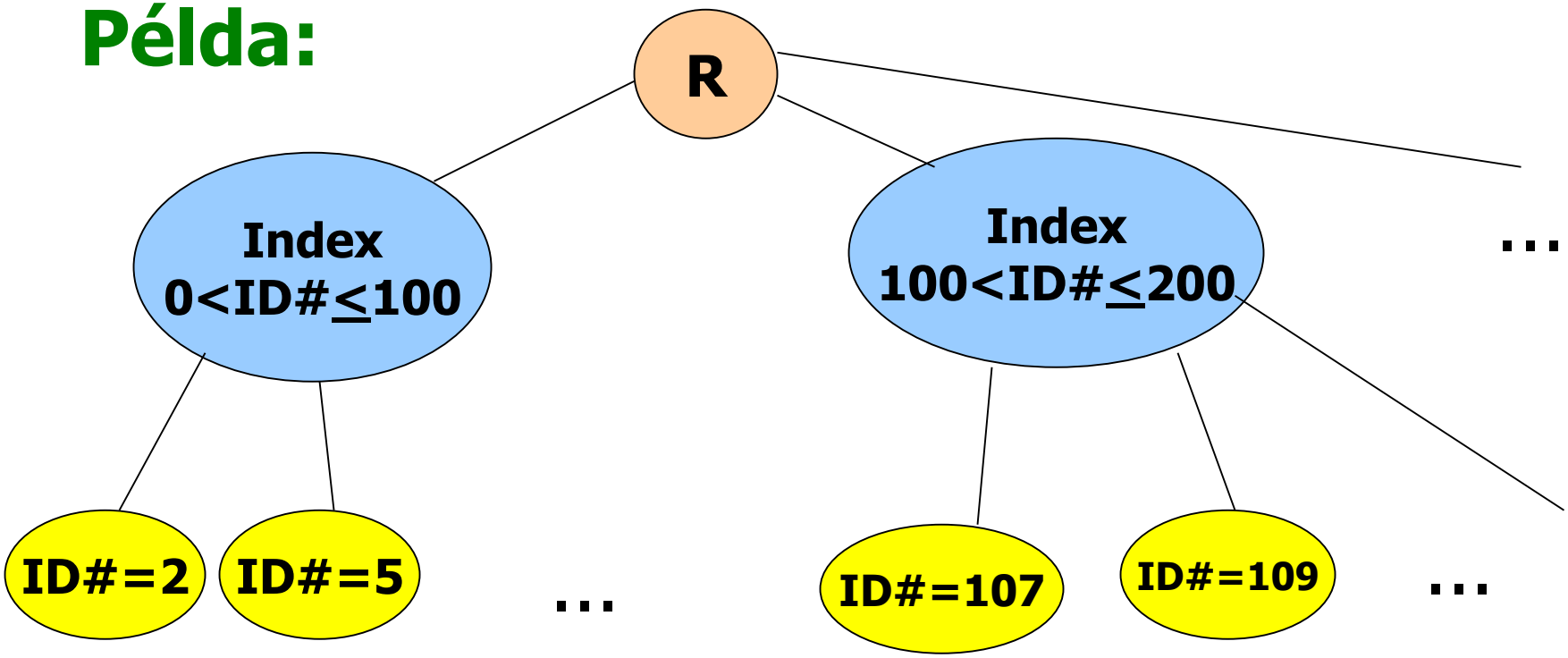


Példa

T_1: Insert<04,Kerry>	T_2: Insert<04,Bush>
T_1	T_2
$X_1(R)$	
	$X_2(R)$ *várakozik*
Megszorítás ellenőrzése	
Insert<04,Kerry>	
U(R)	
	$X_2(R)$
	Megszorítás ellenőrzése
	Hoppá! ID# = 04 már létezik!

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).

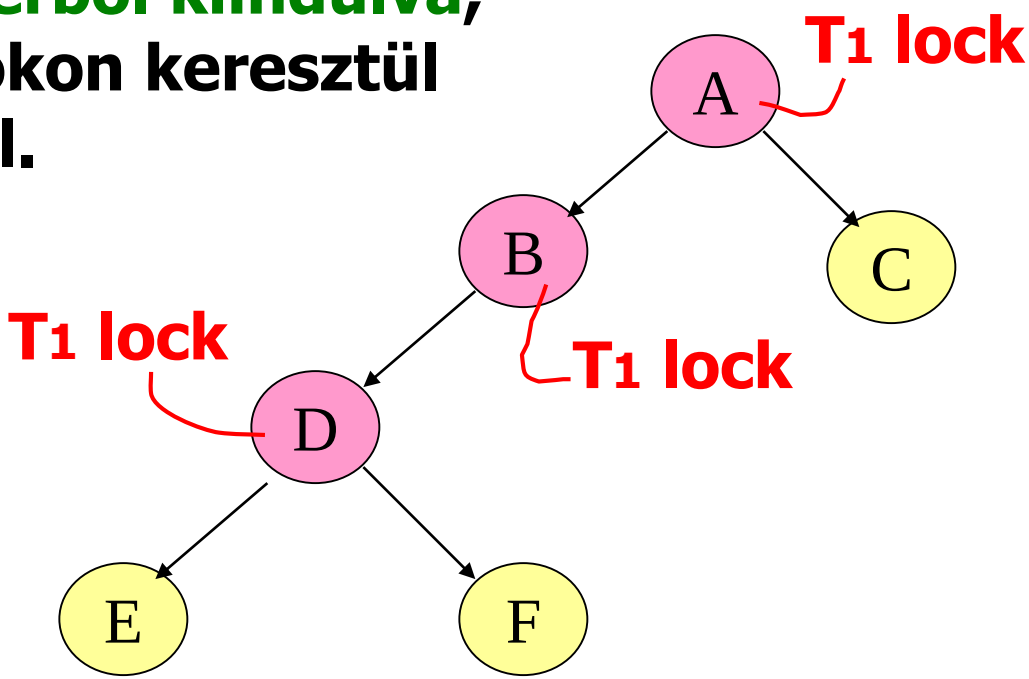
- Ilyenkor a szokásos technikák mennek ugyan, de nagyon előnytelenek lehetnek. Például a 2PL esetén egész addig kell tartani a zárat a gyökéren, amíg le nem értünk a levélhez, ami indokolatlanul sok várakozáshoz vezet.

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.**

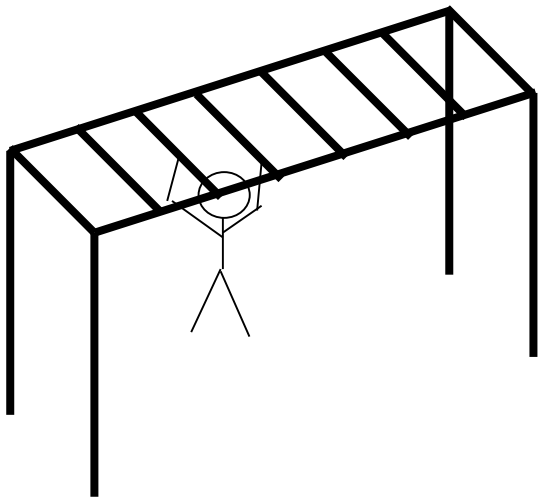
Példa

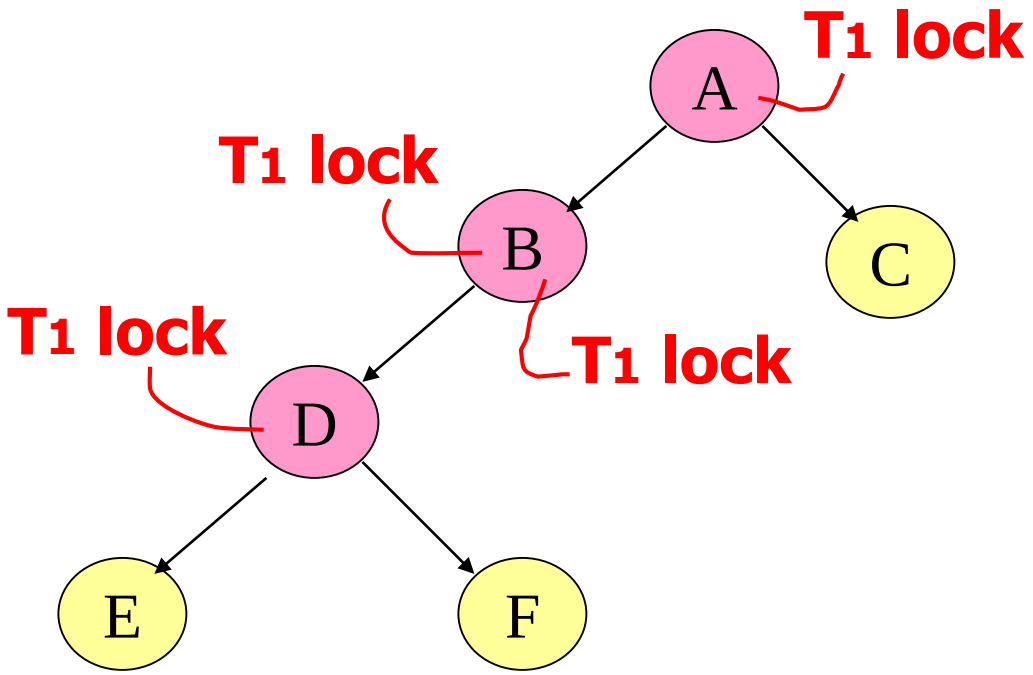
- **Az összes objektumot a gyökérből kiindulva, mutatókon keresztül érjük el.**



☛ **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

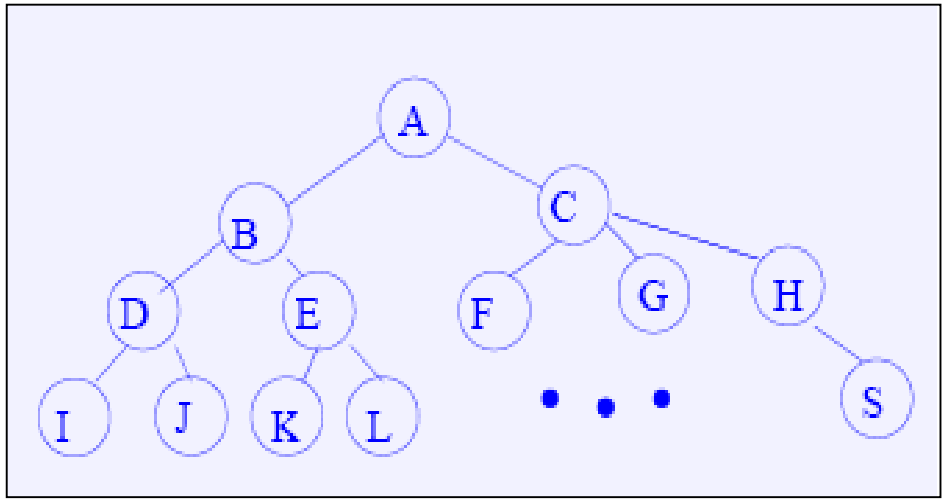
Egyszerű tranzakciómodellben vagyunk (de lehetne (S/X) modellre kibővíteni), azaz

- 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.

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

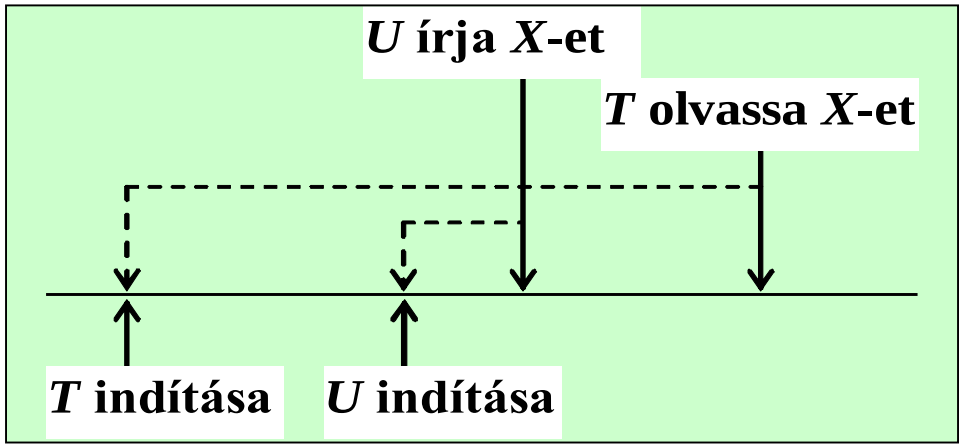
- **Az ütemező olvasáskor és íráskor ellenőrzi, hogy ez abban a sorrendben történik-e, mintha a tranzakciókat az időbélyegzőjük szerinti növekvő, soros ütemezésben hajtottunk volna végre.**

- **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.

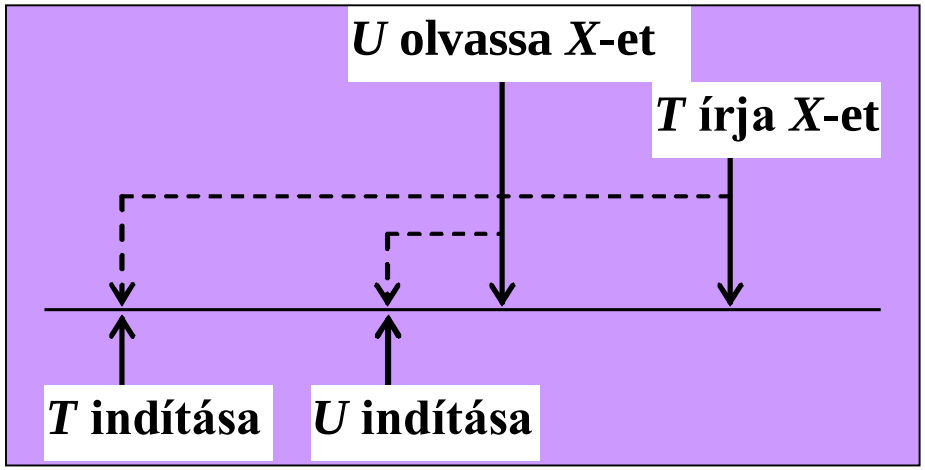
2. Túl késői írás

- **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).

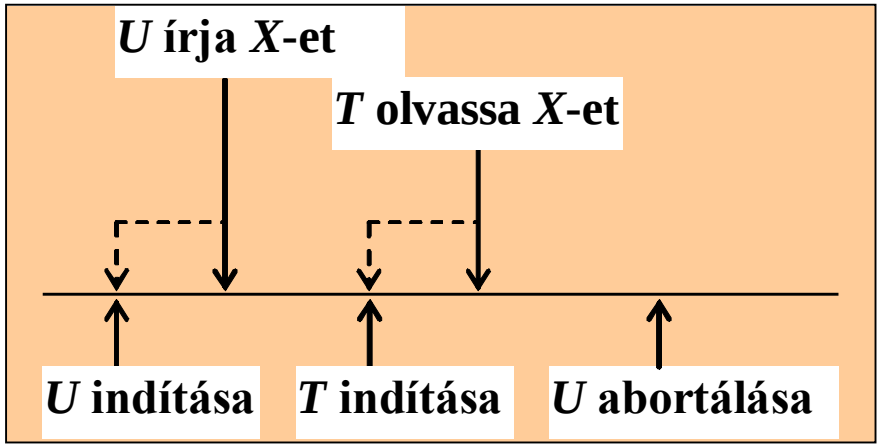
Semelyik más tranzakció sem írta X-et, amellyel felülírta volna a T által írt értéket, és ezzel érvénytelenítette volna T hatását.



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

A piszkos adatok problémái

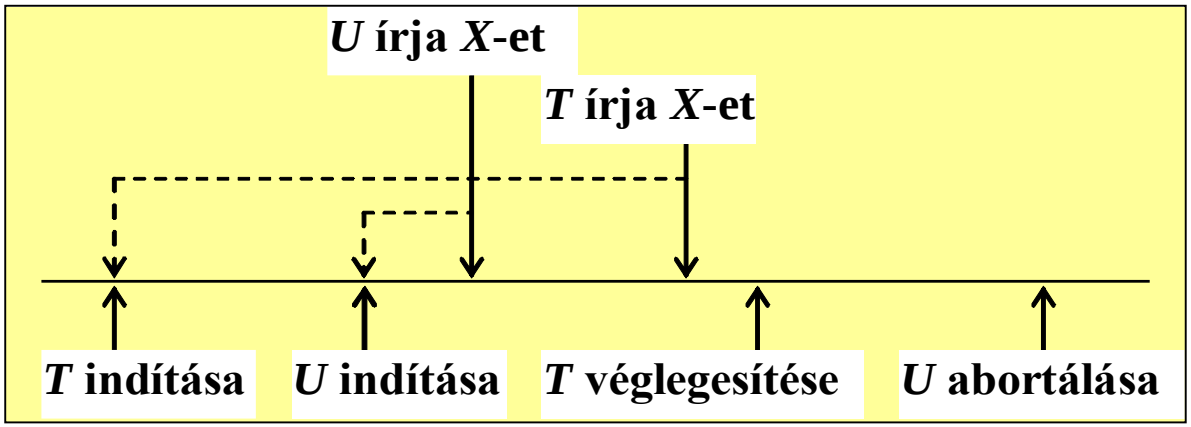
- **Bár nincs fizikailag nem megvalósítható abban, hogy T olvassa X-et, mégis jobb a T általi olvasást azutánra elhalasztani, hogy U véglegesítését vagy abortálását már elvégeztük, különben az ütemezésünk nem lesz konfliktus-sorbarendezhető. Azt, hogy U még nincs véglegesítve, onnan tudjuk, hogy a C(X)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

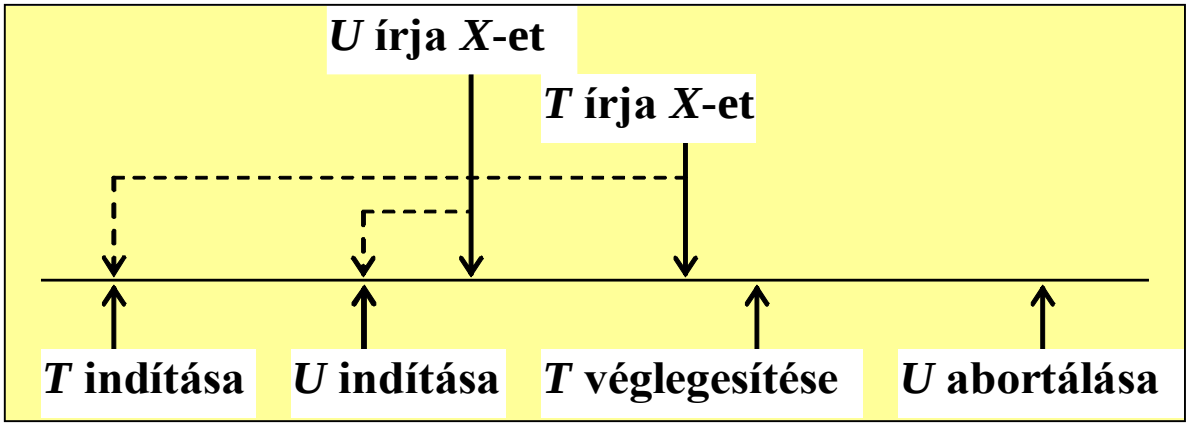
Egy másik probléma (Thomas-féle írás)

- **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)

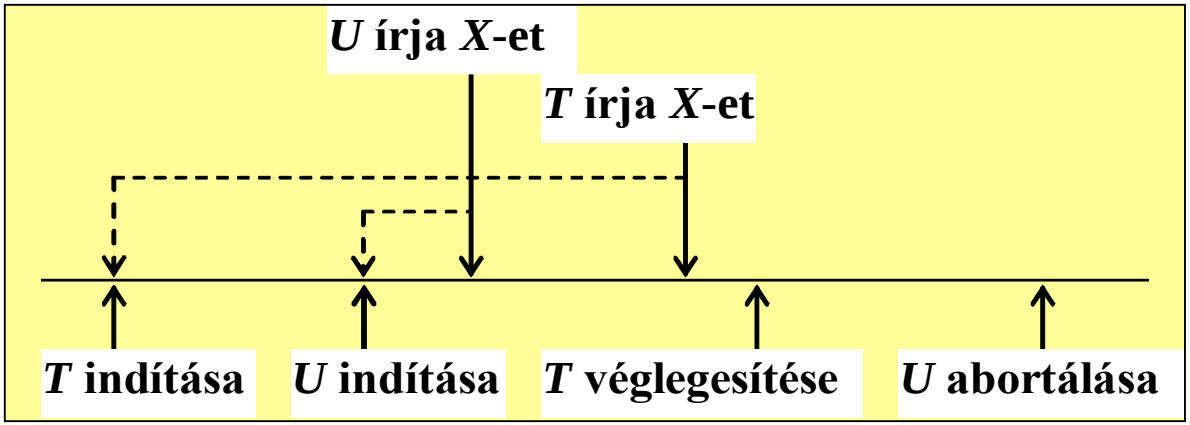


1. MEGOLDÁS: Amikor a T tranzakció írja az X adatbáziselemet, és azt látjuk, hogy X írási ideje nagyobb T időbélyegzőjénél (azaz TS(T) < WT(X)), valamint hogy az X-et író tranzakció még nincs véglegesítve (azaz C(X) hamis), akkor T-t késleltetjük mindaddig, amíg C(X) igazzá nem válik.

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.**

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 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

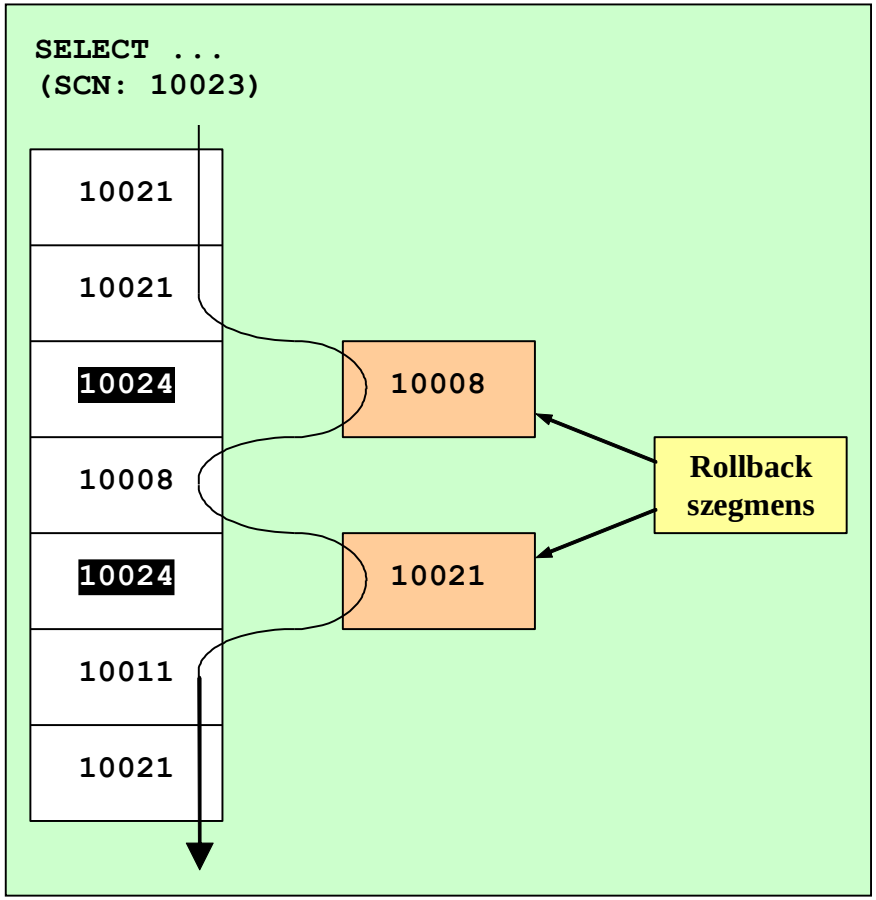
- **A kétféle olvasási konzisztencia eléréséhez az Oracle a rollback szegmensekben található információkat használja fel.**

- **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.**

- **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.**

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.**

Összefoglalás

- Figyelmeztető zárak csoportos módja
- Nem ismételhető olvasás, fantomok, megszorítások
- Indexek zárolása, mászóka-elv, Fa protokoll
- Zárakat nem használó ütemezők (időbélyegzéses, érvényesítéses)
- Az időbélyegzéses ütemező optimista működése, túl késői olvasás, túl késői írás, piszkos adatok olvasása, Thomas-féle írás
- Oracle konkurenciakezelése (két szintű olvasási konzisztencia, két szintű zárolása, figyelmeztető csoportos módú zárak a táblákra, felminősítés, többváltozatú időbélyegzés)
- A tranzakciók 4 féle ISO szabvány szerinti elkülönítési szintje