ELTE | IK
INFORMATIKAI KAR

Adatbázisok 2

Zárak

Eszközök sorbarendezhetőség elérésére

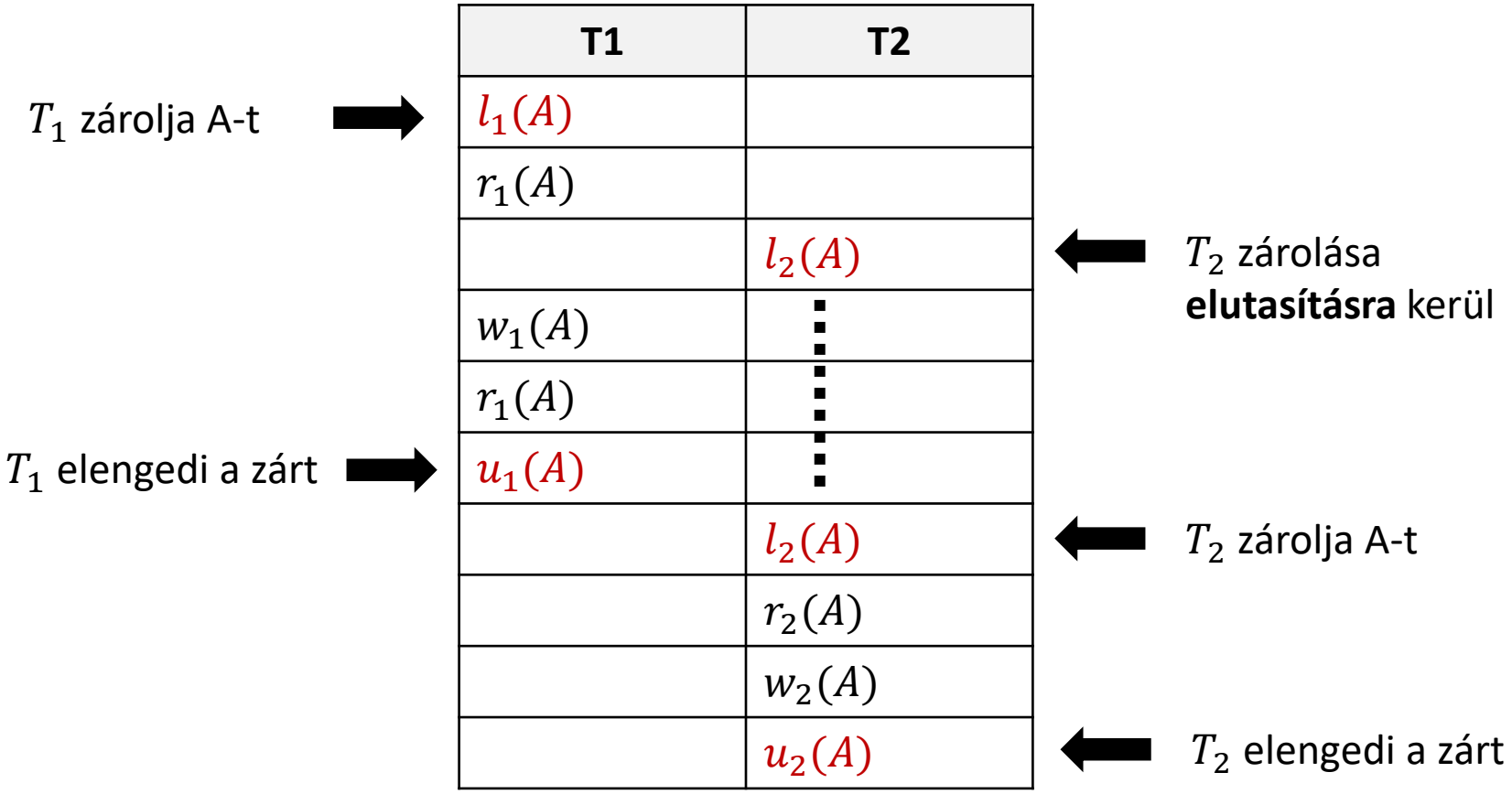
- Az ütemezőnek különböző lehetőségei vannak arra, hogy kikényszerítse a sorbarendezhető ütemezéseket:
  - **Zárak**
  - Időbélyegek
  - Érvényesítés
- **Fő elv**: inkább legyen szigorúbb és ne hagyjon lefutni egy olyan ütemezést, ami sorbarendezhető, mint hogy fusson egy olyan, ami nem az.

Új műveletek

- 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).

- 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).
- Ez a fajta zárolást **kizárólagos** zárnak fogjuk nevezni.

Példa zárolással



Tranzakció konzisztenciája

- **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 $u_i(X)$.
- Azaz: ha egy tranzakció olvasni vagy írni akar egy adatbáziselemet, előtte zárolnia kell, később pedig el kell engednie a zárat.

$$T_i: \ldots l_i(X) \ldots r_i\ (X)/w_i\ (X)\ \ldots u_i(X)$$

Ütemezés jogszerűsége

- **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.
- Azaz: ha egy tranzakció zárolni akar egy olyan adatbáziselemet, amelyet egy másik tranzakció korábban zárolt, csak akkor zárolhatja, ha az előző tranzakció feloldja az adatbáziselem zárolását (egyszerre két tranzakció nem zárolhatja ugyanazt az elemet):

$$T_i: \dots l_i(X) \dots \dots u_i(X) \dots$$

$l_j(X)$ csak ez után jöhet

Zárolási ütemező

- A zároláson alapuló ütemező feladata, hogy csak akkor engedélyezze a kérések végrehajtását, ha azok jogszerű ütemezéseket eredményeznek.
- Ezt a döntést segíti a **zártábla**, amely minden adatbáziselemhez megadja azt a tranzakciót, amelyik pillanatnyilag zárolja az adott elemet – ha van ilyen.
- A zártábla szerkezete (egyféle zárolás esetén):
  - **Zárolások(X, T)** - ahol a T tranzakció zárolja az X adatbáziselemet.

Elég ez a két tulajdonság ahhoz, hogy sorbarendezhető ütemezéseket kapjunk?

Ellenpélda

- A bankos példa korábbi előadásról.
- A átutal B-nek 100 forintot és közben a bank 6%-os kamatot ír jóvá.
- Az eredmény nem lesz azonos egyik soros ütemezéssel sem.
- Tehát **nem sorbarendezhető**!
- Pedig mindkét tranzakció **konzisztens**, az ütemezés pedig **jogszerű**.

T1	T2
$l_1(A)$; $r_1(A)$	
A := A - 100	
$w_1(A)$; $u_1(A)$	
	$l_2(A)$; $r_2(A)$
	A := A * 1.06
	$w_2(A)$; $u_2(A)$
	$l_2(B)$; $r_2(B)$
	B := B * 1.06
	$w_2(B)$; $u_2(B)$
$l_1(B)$; $r_1(B)$	
B := B + 100	
$w_1(B)$; $u_1(B)$	

Megoldás

- Hozzuk a T_1-ben a B zárolását korábbra, az A adatbáziselem zárolásának a feloldása elé.
- Miért segít ez?

T_2 zárolni próbálja B-t, de elutasításra kerül →

T_2 zárolni próbálja B-t, most megkapja a zárat →

T1	T2
$l_1(A)$; $r_1(A)$	
A := A - 100	
$w_1(A)$; $l_1(B)$; $u_1(A)$	
	$l_2(A)$; $r_2(A)$
	A := A * 1.06
	$w_2(A)$
	$l_2(B)$
$r_1(B)$	⋮
B := B + 100	⋮
$w_1(B)$; $u_1(B)$	⋮
	$l_2(B)$; $u_2(A)$
	$r_2(B)$
	B := B * 1.06
	$w_2(B)$; $u_2(B)$

Kétfázisú zárolás (two-phase locking, 2PL)

- Általánosítsuk az előző példában adott megoldást.
- Ha csak a zárak kiadását és feloldását nézzük akkor azt fogjuk látni, hogy egy tranzakcióban minden zárolási művelet megelőzi az összes zárfeloldási műveletet – ezt nevezzük kétfázisú zárolásnak (vagy röviden: 2PL). Első fázis (növekedési fázis): zárak kiadása.
- Második fázis (csökkenő fázis): zárak feloldása.

Korábbi nem sorbarendezhető példa

- Mindkét tranzakció **konzisztens**, az ütemezés pedig **jogszerű**.
- De **nem** sorbarendezhető.
- **Nem kétfázisú!**

T1	T2
$l_1(A); r_1(A)$	
A := A - 100	
$w_1(A); u_1(A)$	
	$l_2(A); r_2(A)$
	A := A * 1.06
	$w_2(A); u_2(A)$
	$l_2(B); r_2(B)$
	B := B * 1.06
	$w_2(B); u_2(B)$
$l_1(B); r_1(B)$	
B := B + 100	
$w_1(B); u_1(B)$	

A kétfázisú zárolás tétele

- **Tétel**: Konzisztens, kétfázisú zárolású tranzakciók bármely 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 konfliktus-sorbarendezhető.
- Indukció: Legyen S a $T_1, \dots, T_n$ konzisztens, kétfázisú zárolású tranzakciókból álló jogszerű ütemezés, és legyen T_i az a tranzakció, amely a legelső zárfeloldási műveletet végzi ($u_i(X)$).
    - **Azt állítjuk**, hogy T_i összes olvasási és írási műveletét az ütemezés legelejére tudjuk vinni anélkül, hogy konfliktusműveleteket kellene áthaladnunk.

Bizonyítás folytatása

- Vegyünk egy konfliktusos párt: $\boldsymbol{w_i(X)}$ **és** $\boldsymbol{w_j(X)}$.
- Megelőzheti a $\boldsymbol{w_j(X)}$ a $\boldsymbol{w_i(X)}$ műveletet?
- Ha így lenne, akkor az ütemezésben az $u_j(X)$ és az $l_i(X)$ az alábbi módon helyezkednének el:

$$\dots ; w_j(X); \dots ; u_j(X); \dots ; l_i(X); \dots ; w_i(X); \dots$$

Mivel a feltételezésünk szerint T_i az első tranzakció, amely zárat old fel, ezért az $u_i(X)$ megelőzi $u_j(X)$ -t, vagyis az ütemezés a következőképpen néz ki:

$$\dots ; w_j(X); \dots ; u_i(X); \dots ; u_j(X); \dots ; l_i(X); \dots ; w_i(X); \dots$$

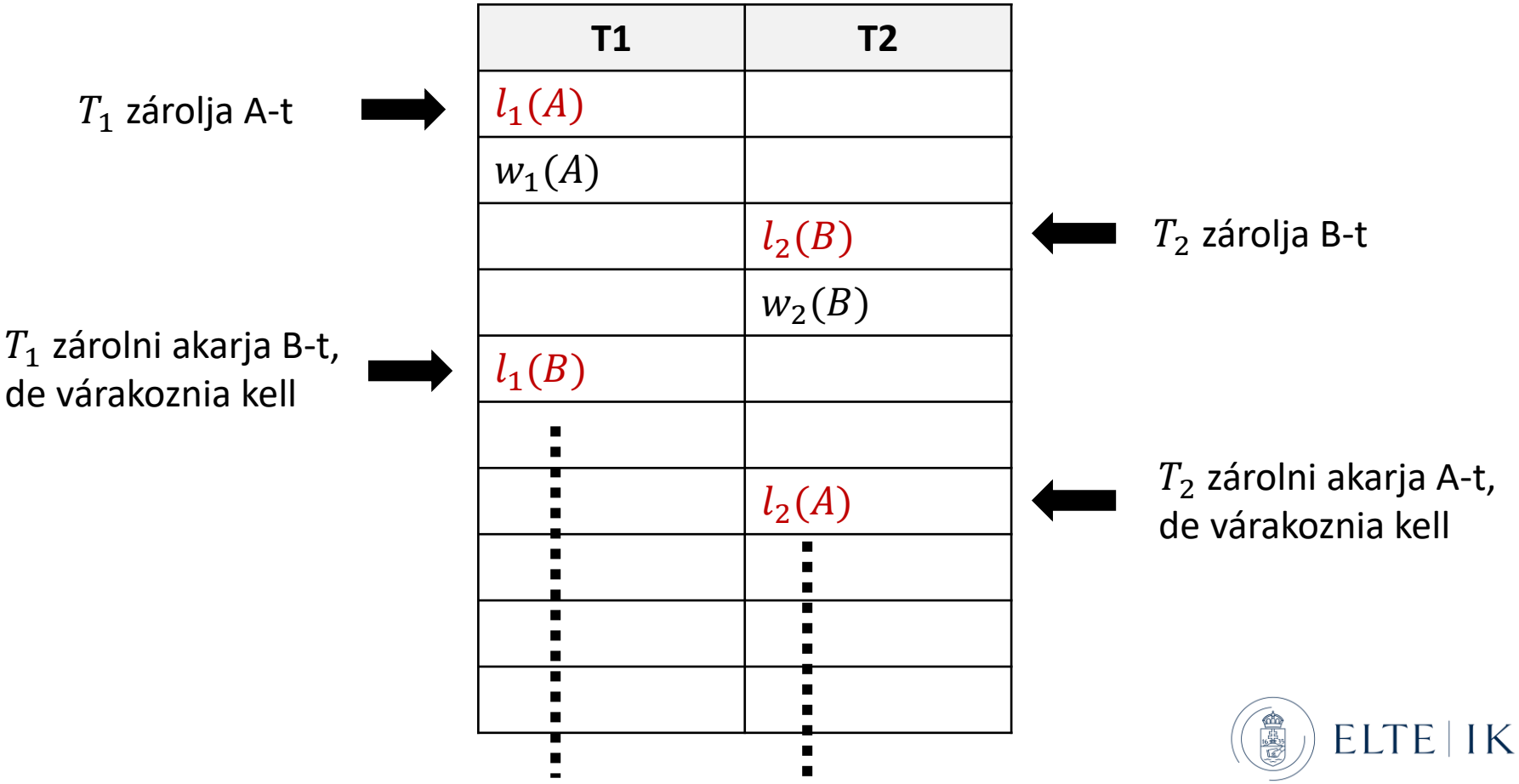
Viszont ekkor az $u_i(X)$ az $l_i(X)$ előtt áll, ami azt jelenti, hogy T_i nem kétfázisú zárolás, amint azt feltételeztük, így ellentmondáshoz jutunk.

Bizonyítás folytatása

- Bebizonyítottuk, hogy az ütemezés **legelejére lehet vinni** T_i összes műveletét.
- Így az ütemezés 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 felvetést**.
- Átalakítjuk a második részt konfliktusekvivalens soros ütemezéssé, így a teljes ütemezés konfliktus-sorbarendezhetővé válik.

A holtpont kockázata

- A zárolás ütemezés – mint láttuk – várakoztathatja a tranzakciókat.



Holtpont (deadlock)

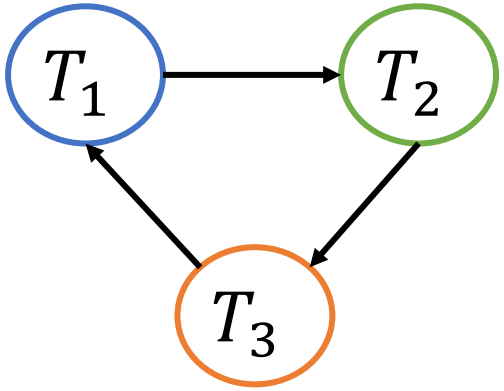
Forrás (2025 november): https://www.vg.no/nyheter/i/3pkoP9/busser-fast-paa-alexander-kiellands-plass-i-oslo

Holtpont (deadlock)

- A **holtpont** egy olyan helyzet, amikor tranzakciók kölcsönösen **várakoznak egymásra**. Legalább két tranzakció kell a holtpont kialakulásához.
- A holtpont kezelésének két módja van:
  - Holtpont megelőzése (pesszimista).
  - Holtpont felismerése és megszüntetése (optimista).

Várakozási gráf

- A **holtpont** felismerésében segít a zárkérések sorozatához tartozó **várakozási gráf** (waits-for graph).
- A 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 folyamatosan változik, ahogy haladunk az ütemezésben.
- Példa ütemezés: $l_1(A); l_2(B); l_3(C); l_1(B); l_2(C); l_3(A)$
  - Az ütemezéshez tartozó 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 a gráfban nincs irányított kör, akkor van topologikus rendezése a tranzakcióknak és ebben a sorrendben le tudnak futni. Vegyük egy topologikus sorrend utolsó tranzakcióját. Ez le tud futni, hiszen nincs belőle kimenő él. Ezután vegyük a sorrend utolsó előtti tranzakcióját (ez is le tud futni), és így tovább. Tehát, ha nincs irányított kör, akkor holtpont sincs.

Holtpont megszüntetése (optimista)

- Folyamatosan rajzoljuk a várakozási gráfot és időközönként ellenőrizzük. Ha holtpont alakult ki, akkor ABORT-áljuk az egyik olyan tranzakciót, amelyik benne van az irányított körben.
- **Miért optimista?** Mert azt feltételezi, hogy alapvetően ritkán van holtpont – olyankor pedig megoldja.
- Az áldozat kiválasztása a következő **tényezőktől** függ (többek között):
  - Mióta fut a tranzakció.
  - Hány műveletet hajtott eddig sikeresen végre.
  - Hány adatbáziselemet zárolt eddig.

Holtpont megelőzése (pesszimista)

- **Miért pesszimista?** Ha engedjük, hogy a tranzakciók mindenféle zárakat kérjenek, akkor abból könnyen baj lehet. Legyünk inkább szigorúbbak.
- Lehetséges működések:
    1. Minden egyes tranzakció előre elkéri az összes zárat, ami neki kelleni fog Ha nem kapja meg az összeset, akkor egyet se kér, el se indul.
    2. Amikor egy tranzakció egy zárat kér, amelyet épp egy másik tranzakció tart, akkor valamelyik tranzakciót ABORT-áljuk, így nem lesznek várakozások.
- Ezek a pesszimista megközelítések **nem igényelnek** várakozási gráfokat.

Éhezés

- Az éhezés egy másik probléma, amely előfordulhat a zárolási ütemezők használatakor.
- **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**: egy tranzakciónak akkor is zárolnia kell egy adatbáziselemet, ha csak olvasni akarja, írni nem.
- **Kérdés**: miért ne olvashatná több tranzakció egyidejűleg egy adatbáziselem értékét mindaddig, amíg egyiknek sincs engedélyezve, hogy írja?

Osztott és kizárólagos zárak

- Két típusú zárat fogunk használni: **osztott** zárak (shared locks) és **kizárólagos** zárakat (exclusive locks).
  - Más néven: olvasás és írási zárak.
- A zárolások **szabályai**:
  - Tetszőleges X adatbáziselemet vagy **egyszer lehet zárolni kizárólagosan** vagy **akárhányszor lehet zárolni osztottan**, ha még nincs kizárólagosan zárolva.
  - Ha egy tranzakció írni akar egy X adatbáziselemet, akkor X-en kizárólagos zárral kell rendelkeznie, de ha csak olvasni akarja, akkor akár osztott, akár kizárólagos zár is megfelel.
- Feltételezzük, hogy ha egy tranzakció csak olvasni akar egy adatbáziselemet, akkor **előnyben részesíti** az osztott zárolást.

Jelölések

- **Osztott zár**: $sl_i(X)$ - a T_i tranzakció osztott zárat kér az X adatbáziselemre.
- **Kizárólagos zár**: $xl_i(X)$ - a T_i tranzakciókizárólagos zárat kér az X adatbáziselemre.
- Továbbra is $u_i(X)$ jelöli, hogy a T_i tranzakció feloldja az X adatbáziselem zárolását, azaz felszabadítja minden zár alól (legyen az osztott vagy kizárólagos).

Tranzakciók konzisztenciája

- **Konzisztencia**: nem írhatunk kizárólagos zár fenntartása nélkül és nem olvashatunk valamilyen zár fenntartása nélkül.
- Pontosabban:
  1. Bármely T_i tranzakcióban az $r_i(X)$ **olvasási** műveletet meg kell, hogy előzze egy $sl_i(X)$ vagy egy $xl_i(X)$ úgy, hogy közben nincs $u_i(X)$.
  2. Bármely T_i tranzakcióban a $w_i(X)$ **írási** műveletet meg kell, hogy előzze egy $xl_i(X)$ úgy, hogy közben nincs $u_i(X)$.
- Minden zárolást követnie kell (nem közvetlenül) egy ugyanannak az elemnek a zárolását **feloldó** műveletnek.

Kétfázisú zárolás (2PL)

- A zárolásoknak meg kell előzniük a zárak feloldását.
- Pontosabban fogalmazva: bármely T_i kétfázisú zárolású tranzakcióban egyetlen $sl_i(X)$ vagy $xl_i(X)$ művelet sem előzhet meg egyetlen $u_i(X)$ művelet sem semmilyen adatbáziselemre.

Ütemezések jogszerűsége

- **Jogszerűség**: egy elemet vagy egyetlen tranzakció zárol kizárólagosan, vagy több is zárolhatja osztottan, de a kettő egyszerre nem lehet.
- Pontosabban fogalmazva:
  1. 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 $u_i(X)$.
  2. 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 $u_i(X)$.

A kétfázisú zárolás tétele (osztott és kizárólagos)

- **Tétel**: Konzisztens kétfázisú zárolású tranzakciók jogszerű ütemezése konfliktus-sorbarendezhető.
- **Bizonyítás**: Ugyanaz, mint korábban volt.

- **Megjegyzés**: Engedélyezett, hogy egy tranzakció ugyanazon elemre kérjen és tartson mind osztott, mind kizárólagos zárat, feltéve, hogy ezzel nem kerül konfliktusba más tranzakciók zárolásaival.

Kompatibilitási mátrix

- 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.
- **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 már van ilyen zár kiadva…

	S	X
S	igen	nem
X	nem	nem

Kompatibilitási mátrixok felhasználása

- Az ütemező a mátrix alapján dönti el, hogy egy zárkérés megadható-e, illetve ez alapján várakoztatja a tranzakciókat. Minél több az „Igen” a mátrixban, annál kevesebb lesz a várakoztatás.
- A mátrix alapján kialakult várakoztatásokhoz elkészíthet a várakozási gráf, amely segítségével kezelhetőek a holtpontok (korábban volt szóra róla).
- Egy zárkezelés-sorozathoz szintén elkészíthető egy **megelőzési gráf**.

Megelőzési gráf zárolási ütemező esetén

- A **megelőzési gráf** csúcsai a tranzakciók és akkor van él T_i-ből T_j-be, ha van olyan A adatbáziselem, amelyre az ütemezés során a T_i tranzakció Z_k zárat kért és kapott, később 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.
- Tehát olyankor lesz él, **ha a két zár nem kompatibilis egymással** (nem mindegy a két művelet sorrendje).
- Pl:

$$T_i: Z_k(A) \dots T_i: UZ_k(A) \dots T_j: Z_l(A)$$

Nem kompatibilis zárak

Sorbarendezhetőség (ismét)

- A sorbarendezhetőséget a **megelőzési gráf segítségével** is el lehet dönteni.
- **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: házi feladat.
- Az ütemező egyik **lehetősége** a sorbarendezhetőség elérésére, hogy folyamatosan figyeli a megelőzési gráfot és ha irányított kör keletkezne, akkor ABORT-álja valamelyik tranzakciót.

Zárak erőssége

- Azt mondjuk, hogy az L_2 zár **erősebb** az L_1 zárnál, ha a kompatibilitási mátrixban L_2 sorában és oszlopában minden olyan pozícióban „Nem” áll, amelyben L_1 sorában és oszlopában „Nem” áll.
- Az osztott és kizárólagos zárak kompatibilitási mátrixa:

… megkaphatjuk-e ezt a típusú zárat?

Ha már van ilyen zár kiadva…

	S	**X**
S	igen	nem
X	nem	nem

- **Kérdés**: melyik az erősebb zár?

Zárak felminősítése

- Azt mondjuk, hogy a T tranzakció **felminősíti** (upgrade) az L_1 zárját az L_1 -nél **erősebb** L_2 zárra az A adatbáziselemen, ha L_2 típusú zárat kér (és kap) A-ra, amelyen már birtokol egy L_1 típusú zárat (azaz még nem oldotta fel L_1-et).
- Például:

$$sl_i(A) \dots xl_i(A)$$

- Nézzük meg miért jó ez (következő dia)!

Zárak felminősítése példa

- Felminősítéssel a T_1 tranzakció a T_2-vel **konkurensen** tudja végrehajtani az írás előtti, esetleg hosszadalmas számításait (felminősítés nélkül nem).

T1	T2
$sl_1(A); r_1(A)$	
	$sl_2(A); r_2(A)$
	$sl_2(B); r_2(B)$
$xl_1(B)$	
	$u_2(A); u_2(B)$
$xl_1(B); r_1(B)$	
$w_1(B)$	
$u_1(A)\ ; u_1(B)$	

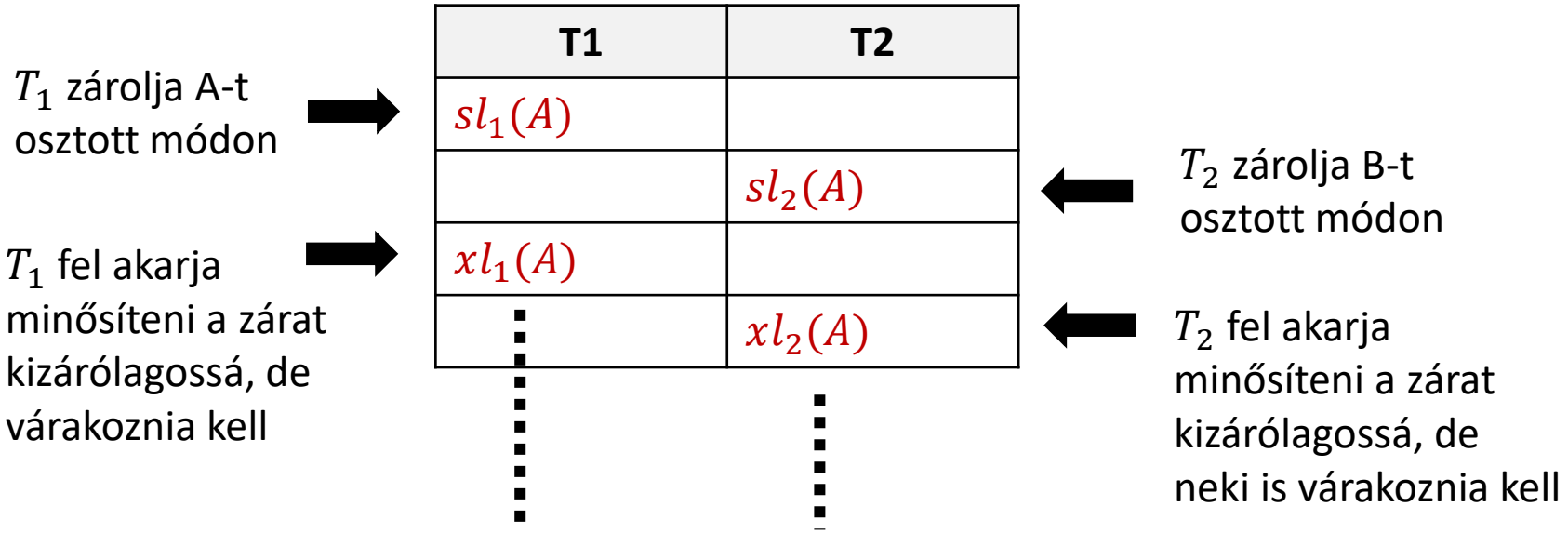
Felminősítés nélkül

T1	T2
$sl_1(A); r_1(A)$	
	$sl_2(A); r_2(A)$
	$sl_2(B); r_2(B)$
$sl_1(B); r_1(B)$	
$xl_1(B)$	
	$u_2(A); u_2(B)$
$xl_1(B)$	
$w_1(B)$	
$u_1(A)\ ; u_1(B)$	

Felminősítéssel

Új típusú holtpontok

- Ha használjuk a felminősítést, akkor új típusú holtpontok jelennek meg.
- Az alábbi ütemezés nem okozna holtpontot, ha nincs felminősítés.



Módosítási zár

- Az előző problémát meg tudjuk oldani egy új típusú zár bevezetésével.
- Az $ul_i(X)$ **módosítási zár** a T_i tranzakciónak csak az X olvasására ad jogot, X írására nem. Később azonban csak a módosítási zárat lehet felminősíteni írásira, az olvasási zárat nem.
- Ha egy adatbáziselemen van módosítási zár, az megakadályozza, hogy bármilyen más zárat adjunk arra az elemre.
- Kompatibilitási mátrix:

Kiadott zár \ Kért zár	S	X	U
S	igen	nem	igen
X	nem	nem	nem
U	nem	nem	nem

Módosítási zár

- Ha egy adatbáziselemen van módosítási zár, az megakadályozza, hogy bármilyen más zárat adjunk arra az elemre.
- Vegyük észre, hogy a kompatibilitási mátrix nem szimmetrikus.
- A módosítási zár (U) úgy néz ki, mintha **osztott zár lenne**, amikor kérjük, és úgy néz ki, mintha **kizárólagos zár lenne**, amikor már megvan.
- Kompatibilitási mátrix:

Kért zár

Kiadott zár	S	X	U
S	igen	nem	igen
X	nem	nem	nem
U	nem	nem	nem

Módosítási zár példa

- Az alábbi példa módosítási zár nélkül (osztott és kizárólagos zárak és felminősítés használatával) holtpontot okozott volna.

T_1 zárolja A-t módosított zárral

Felminősítés

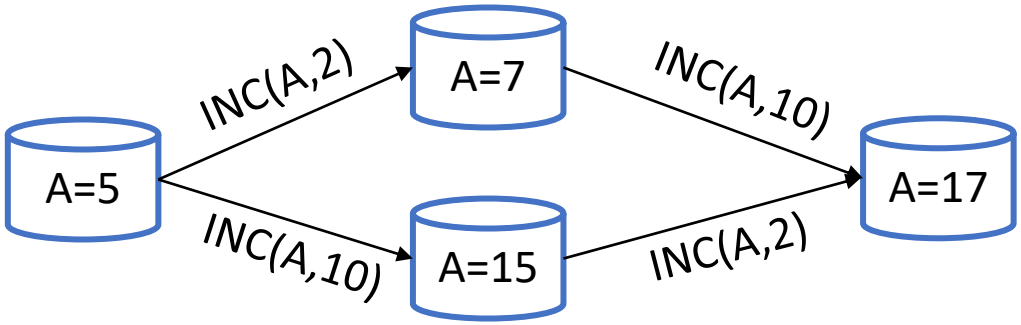
T1	T2
$ul_1(A); r_1(A)$	
	$ul_2(A)$
$xl_1(A)$	⋮
$w_1(A)$	⋮
$u_1(A)$	⋮
	$ul_2(A); r_2(A)$
	$xl_2(A)$
	$w_2(A)$
	$u_2(A)$

Zárkérés elutasítva, várakoznia kell

Most kapja meg a zárat

Növelési művelet

- Egy érdekes (és hasznos) zármód a növelési zár. Sok tranzakció csak növeli vagy csökkenti egy elem értékét. A növelések tetszőleges sorrendben kiszámíthatók.



- $\boldsymbol{INC(A, c)}$ művelet jelentése (ahol **c** konstans):

$$READ(A, t); t := t + c; WRITE(A, t);$$

- Viszont a növelés nem cserélhető fel sem az olvasással, sem az írással.
- Tranzakciós jelöléssel: $inc_i(X)$ – a T_i tranzakció megnöveli az X adatbáziselemet valamely konstanssal (nem számít az értéke)

Növelési zár

- A növelési művelethez tartozik egy új típusú zár, amelyet növelési zárnak (increment lock) nevezünk: $il_i(X)$
- Egy konzisztens tranzakció csak akkor végezheti el X-en a növelési műveletet, ha egyidejűleg növelési zárat tart fenn rajta. A növelési zár nem teszi lehetővé sem az olvasás, sem az írás műveletet.
- Az $inc_i(X)$ művelet konfliktusban áll az $r_j(X)$ és $w_j(X)$ műveletekkel $(i \neq j)$, de nem áll konfliktusban $inc_j(X)$-vel.

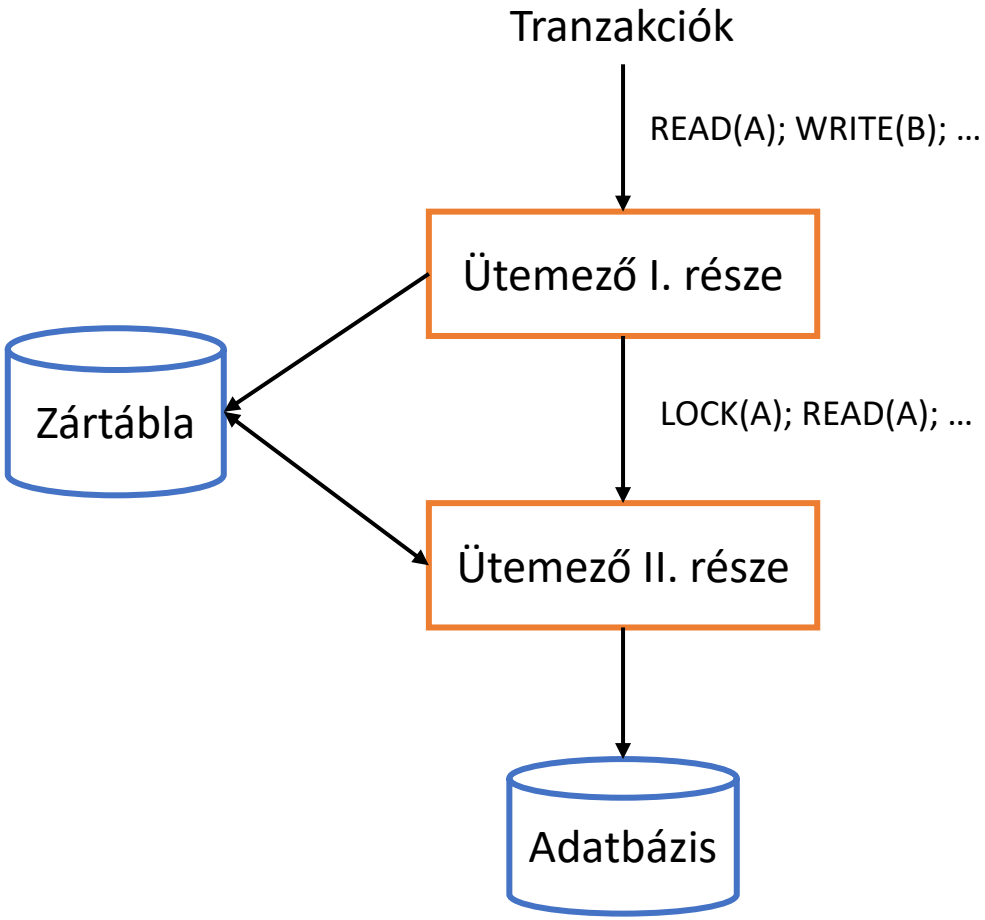
Növelési zár

- 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.
- Kompatibilitási mátrix:

Kiadott zár \ Kért zár	**S**	**X**	**I**
S	igen	nem	nem
X	nem	nem	nem
I	nem	nem	igen

Zárolási ütemező

- Tekintsük meg a zárolási ütemező egy lehetséges modelljét.



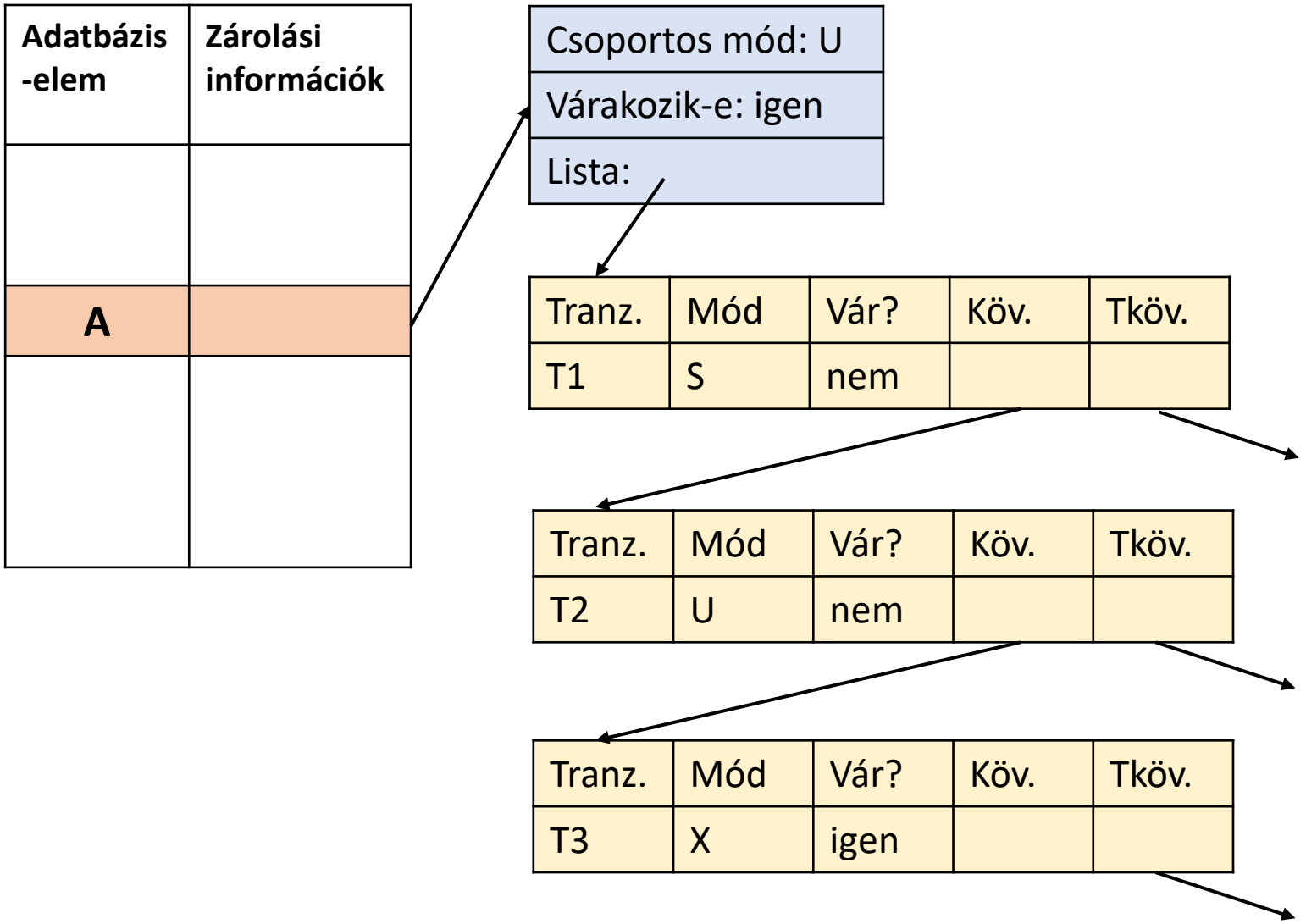
Zárolási ütemező működése

- Az **I. rész** fogadja a tranzakciók által generált kérések sorát és minden adatbázis hozzáférési művelet elé (olvasás, írás, növelés stb.) **beszúrja a megfelelő zárolási műveletet**. Ezeket aztán elküldi a II. résznek.
- A **II. rész** fogadja az I. részen keresztül érkező műveletek sorozatát és mindegyiket **végrehajtja**.
  - Ha a művelet adatbázis-hozzáférés, akkor továbbítja az adatbázishoz és végrehajtja.
  - Ha a művelet zárolás, megvizsgálja a **zártáblát**, hogy engedélyezhető-e a zár.
    - Ha igen, akkor módosítja a zártáblát (megadja a zárat).
    - Ha nem, akkor bejegyzést készít a zártáblába a várakozásról és késlelteti a tranzakciót.

Zárolási ütemező működése

- Amikor egy T tranzakciót **véglegesítünk** vagy **abortálunk**, az I. rész **feloldja** a T tranzakció által fenntartott zárakat. Ha van olyan tranzakció, amely vár ezekre a zárakra, akkor értesíti a II. részt.
- Amikor a II. rész értesül arról, hogy valamelyik adatbáziselem elérhetővé vált, akkor **eldönti**, hogy melyik tranzakció kapja meg a zárat.

Zártábla


Adatbázis -elem
Zárolási információk
A
Csoportos mód: U
Várakozik-e: igen
Lista:
Tranz. Mód Vár? Köv. Tköv.
T1 S nem
Tranz. Mód Vár? Köv. Tköv.
T2 U nem
Tranz. Mód Vár? Köv. Tköv.
T3 X igen

Zártábla felépítése

- A zártábla nem feltétlenül egy reláció (lehet hasító tábla stb.)
- Csak azok az adatbáziselemek szerepelnek benne, amelyekre van kiadva zár.
- Zárolási információk:
  - **Csoportos mód** – az adatelemre kiadott **legerősebb** zár:
    - S – azt jelenti, hogy csak osztott zárak vannak.
    - U – azt jelenti, hogy egy módosítási zár van, és lehet még egy vagy több osztott zár is.
    - X – azt jelenti, hogy csak egy kizárólagos zár van és semmilyen más zár nincs.
  - **Várakozási bit** (waiting bit) – azt adja meg, hogy van-e legalább egy tranzakció, amely az A zárolására várakozik.

Zártábla felépítése

- A **lista** az összes olyan tranzakciót megadja, amelyek vagy **jelenleg zárolják** az adatbáziselemet, vagy a **zárolására várakoznak**. Tartalmazza:
  - Tranz. – a tranzakció azonosítója (neve).
  - Mód – a zár módja.
  - Vár? – fenntartja-e a zárat vagy várakozik.
  - Köv. – a következő tranzakció, amely az elemet zárolja (vagy várakozik).
  - Tköv. – a tranzakció következő zárolási bejegyzése.

Zártábla használata

- Ha egy tranzakció **zárat kér** egy adatbáziselemre, akkor **ellenőrizzük** az elemet a zártáblában.
  - **Ha nincs** az adatbáziselemre **bejegyzés**, akkor biztos, hogy zárak sincsenek. Így létrehozhatjuk a bejegyzést és engedélyezhetjük a kérést.
  - **Ha van bejegyzés**, akkor megnézzük a csoportos módot és ez alapján várakoztatunk (beírjuk a várakozási listába) vagy megengedjük a zárat.
- Ha egy tranzakció **felold** egy zárat, akkor:
  - A tranzakció bejegyzését **töröljük** a listából.
  - Ha kell **megváltoztatjuk** a csoportos módot is.
  - Ha maradnak várakozó tranzakciók, akkor **engedélyeznünk** kell egy vagy több zárat a kért zárak listájáról.

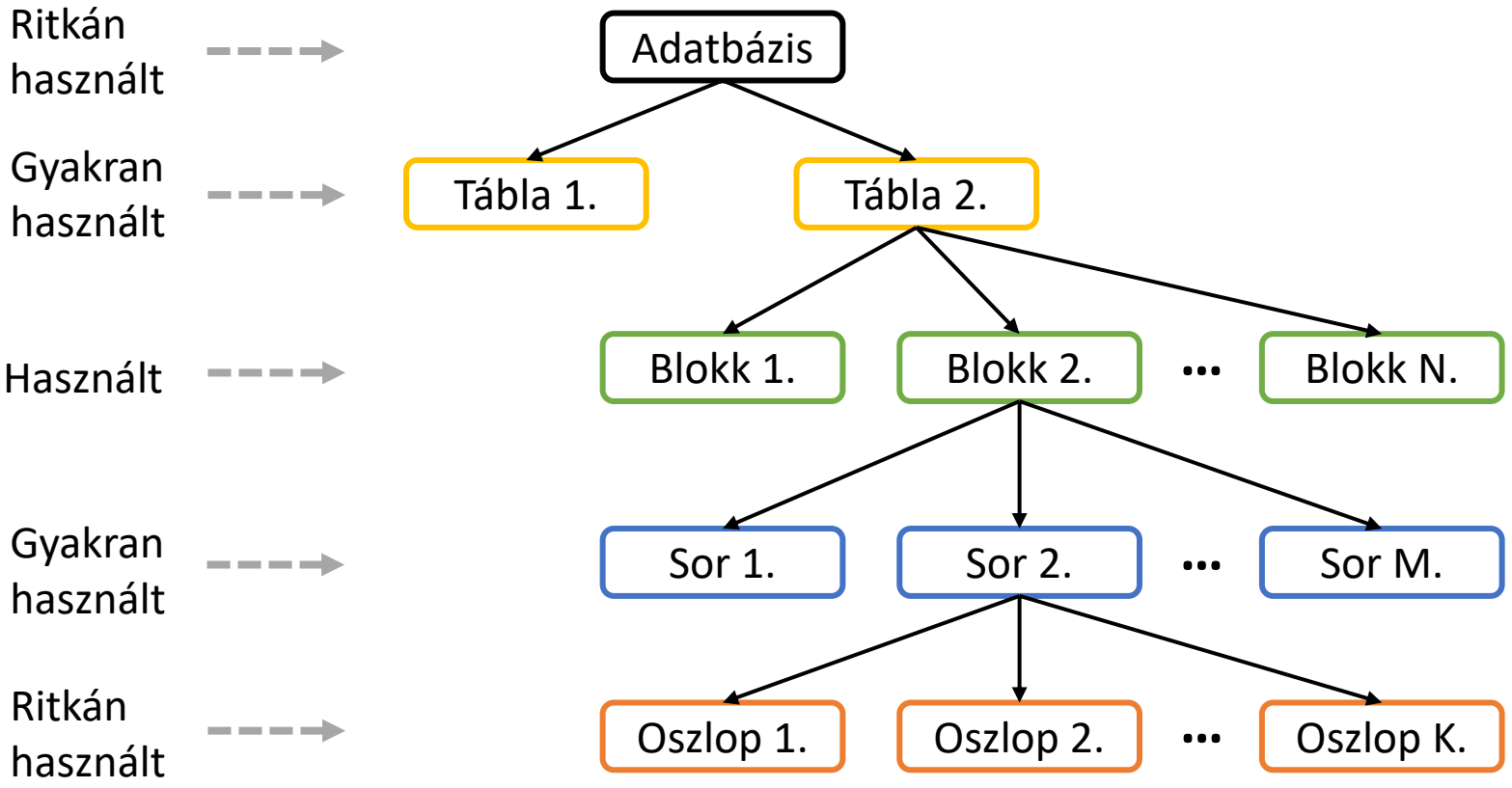
Zárfeloldások kezelése

- Ha maradnak várakozó tranzakciók, akkor **engedélyeznünk** kell egy vagy több zárat a kért zárak listájáról. Melyiket engedélyezzük? Megközelítések:
  - **Első beérkezett kiszolgálása** (first-come-first-served): Azt a zárolási kérést engedéleyzzük, amelyik a legrégebb óta várakozik. Ezzel a stratégiával elkerülhetjük a kiéheztetést.
  - **Elsőbbségadás az osztott zároknak** (priority to shared locks): Először az összes várakozó osztott zárat engedélyezzük. Ezután egy módosítási zárolást – ha várakozik ilyen. Kizárólagos zárolást csak akkor, ha nincs semmilyen más igény.
  - **Elsőbbségadás a felminősítésnek** (priority to upgrading): Ha van olyan U zárral rendelkező tranzakció, amely X zárrá való felminősítésre vár, akkor ezt engedélyezzük előbb.

Hierarchiák kezelése

- **Motiváció**: Gondoljunk egy banki adatbázisra, ahol egy táblában vannak tárolva az egyenlegek. Ilyenkor minden tranzakciónak zárolnia kellene a táblát, akkor is ha csak egyetlen egyenleget módosít.
- **Megoldás**: Vezessünk be különböző szemcsézettségű zárakat (granularity).

Hierarchiák kezelése



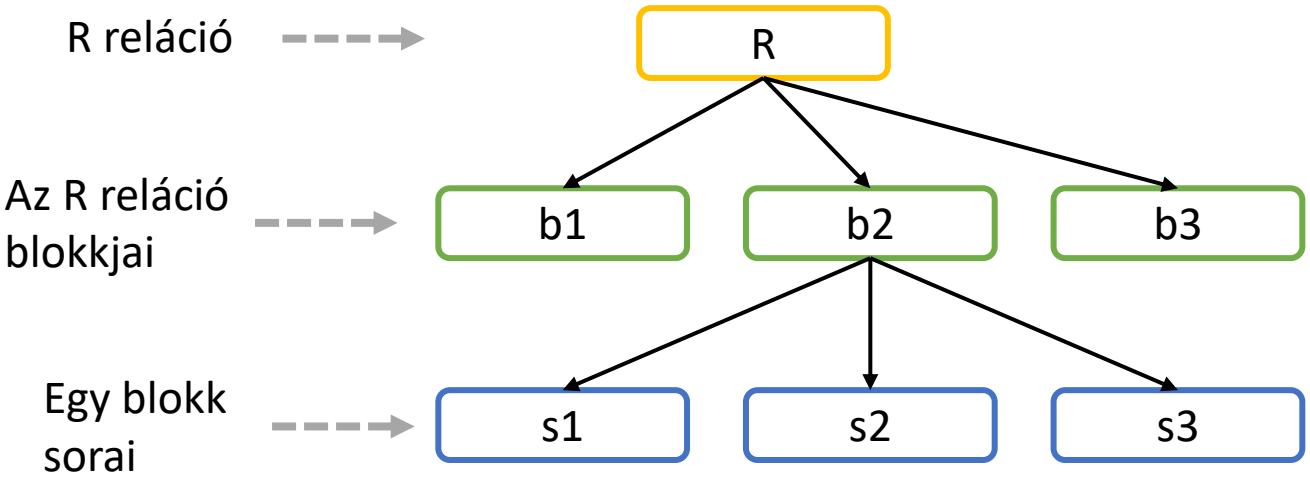
Milyen szemcsézettségű zárakat használjunk?

Hierarchiák kezelése

- **Megoldás**: kis és nagy adatbáziselemeket is zárolhassunk!
- Amikor egy tranzakció kér egy zárat, a rendszernek **el kell döntenie**, hogy milyen szemcsézettségű zárat adjon.
- A cél az, hogy minél kevesebb zárat adjunk egy tranzakciónak.
- Van egy **trade-off** a konkurens működés és a zárkezelés költsége között:
  - Ha nagy szemcsézettségű zárakat használunk -> a tranzakciók nem tudnak párhuzamosan futni.
  - Ha kis szemcsézettségű zárakat használunk -> nagyon sok zár lesz, amelyeket karban kell tartani, használni.
- **Új problémák** keletkeznek:
  - Honnan tudjuk, ha egy tranzakció épp csak egy rekordot ír? Hogyan zárolhatjuk ekkor az egész relációt?

Figyelmeztető protokoll

- Bevezetünk egy új, figyelmeztető protokollt, amely a hagyományos zárak (osztott, kizárólagos) mellett **figyelmeztető** zárakat is használ.
- Három szintű hierarchiát fogunk vizsgálni: relációk, blokkok, sorok.
- A figyelmeztető zárakat a hagyományos zárak elé tett I (intention - szándék) előtaggal jelöljük: $IS_i(X)$ és $IX_i(X)$.

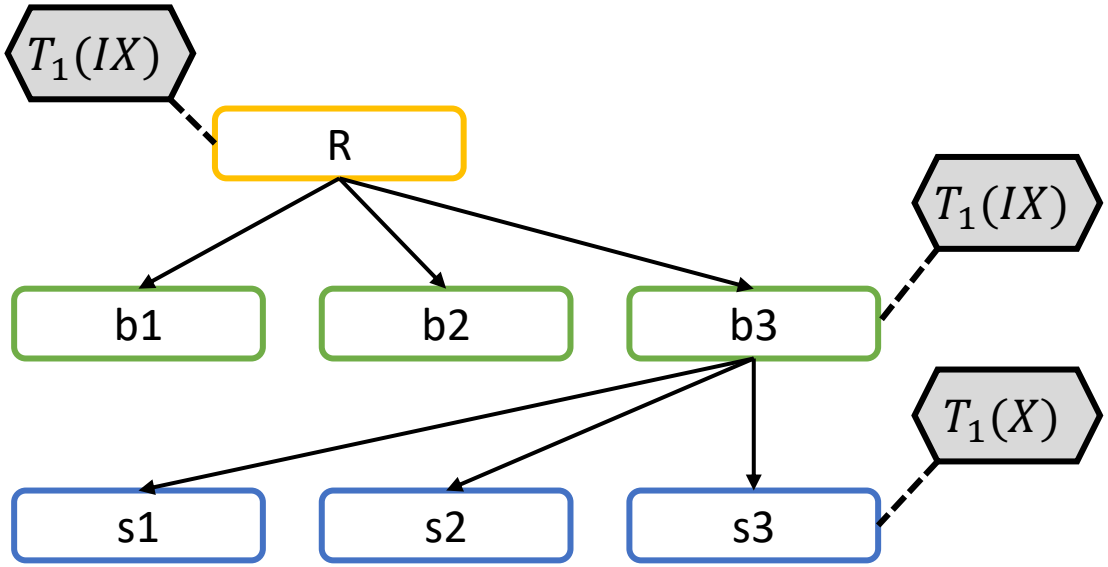


Figyelmeztető zárak használata

- Ahhoz, hogy elhelyezzünk egy hagyományos zárat egy adatbáziselemen, a hierarchia **gyökerénél kell kezdenünk**.
- Ha már annak az elemnél tartunk, amelyet zárolni akarunk, akkor kérjük az X vagy X zárolást arra az elemre.
- Ha az elem amelyet zárolni szeretnénk, lejjebb van a hierarchiában, akkor **elhelyezünk egy figyelmeztetést** ezen a csomóponton (annak megfelelően, hogy milyen zárat szeretnék az elemre kérni: ha S, akkor IS; ha X, akkor IX). Ha megkaptuk a zárat, akkor **haladjunk lejjebb a hierarchiában**.

Figyelmeztető zárak használata (példa)

- Az alábbi példában a T_1 tranzakció zárolta az **R** reláció **b3** blokkjának **s3** sorát.



Kompatibilitási mátrix

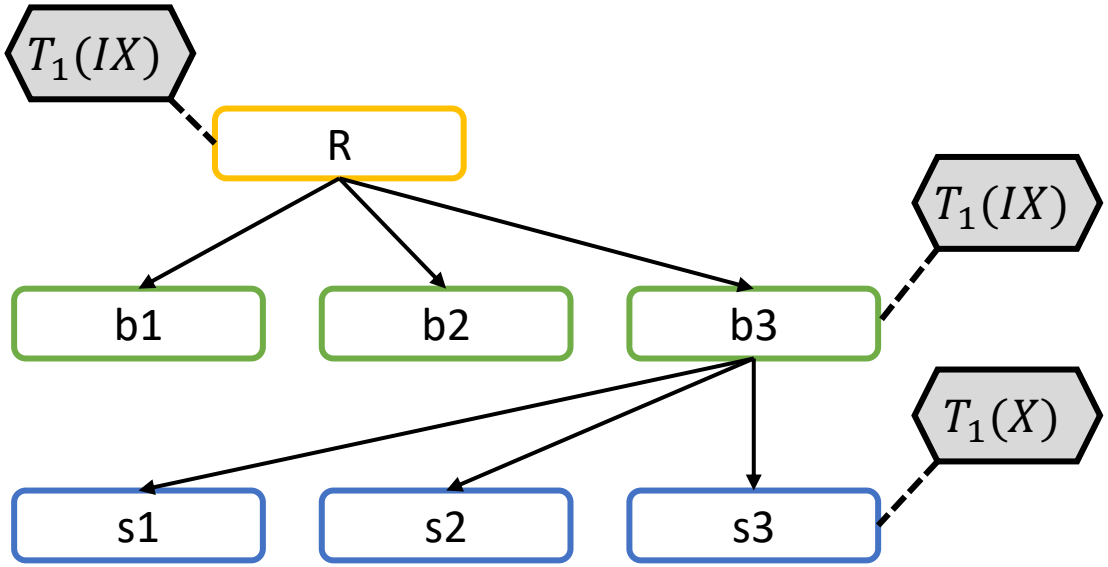
- A figyelmeztető zárak használatával a konfliktusok lejjebb kerülnek a hierarchiában:
  - Egy IX nincs konfliktusban egy más IX-el vagy IS-el.

Kért zár

Kiadott zár	IS	IX	S	X
IS	igen	igen	igen	nem
IX	igen	igen	nem	nem
S	igen	nem	igen	nem
X	nem	nem	nem	nem

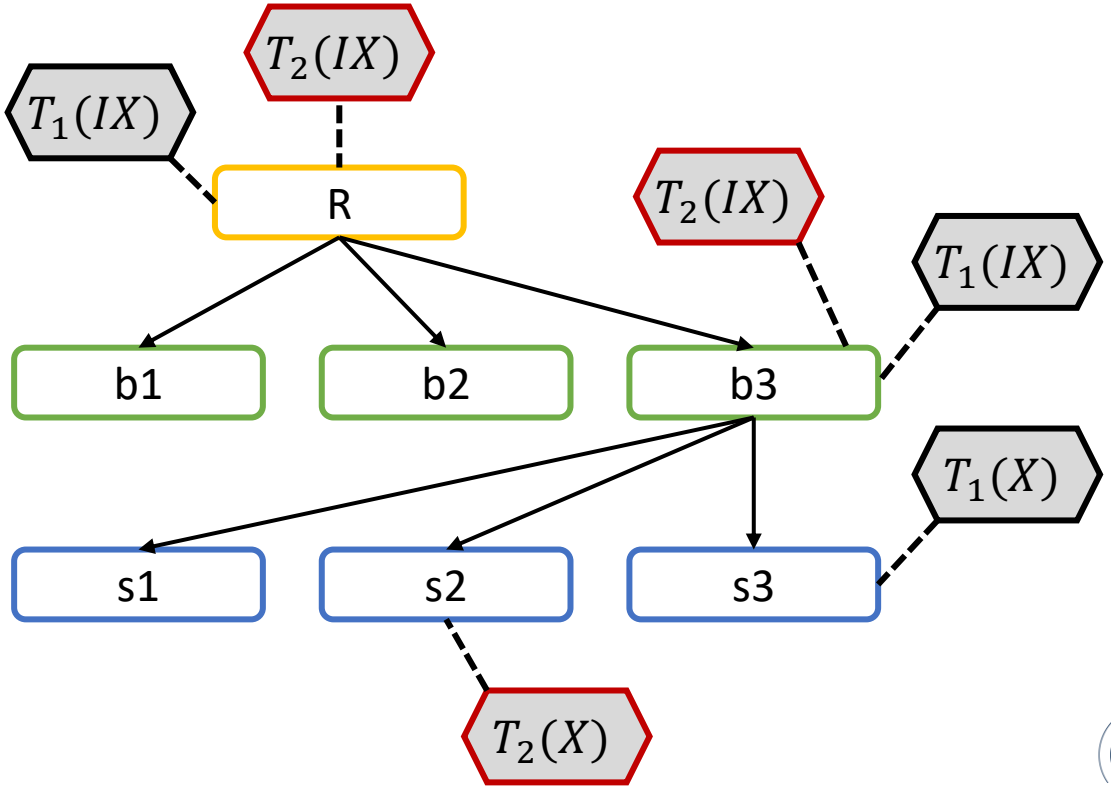
Figyelmeztető zárak használata (példa)

- Az alábbi példában a T_1 tranzakció zárolta az **R** reláció **b3** blokkjának **s3** sorát.
- **Kérdés**: Megkaphatja-e T_2 az X zárat az **s2** sorra?



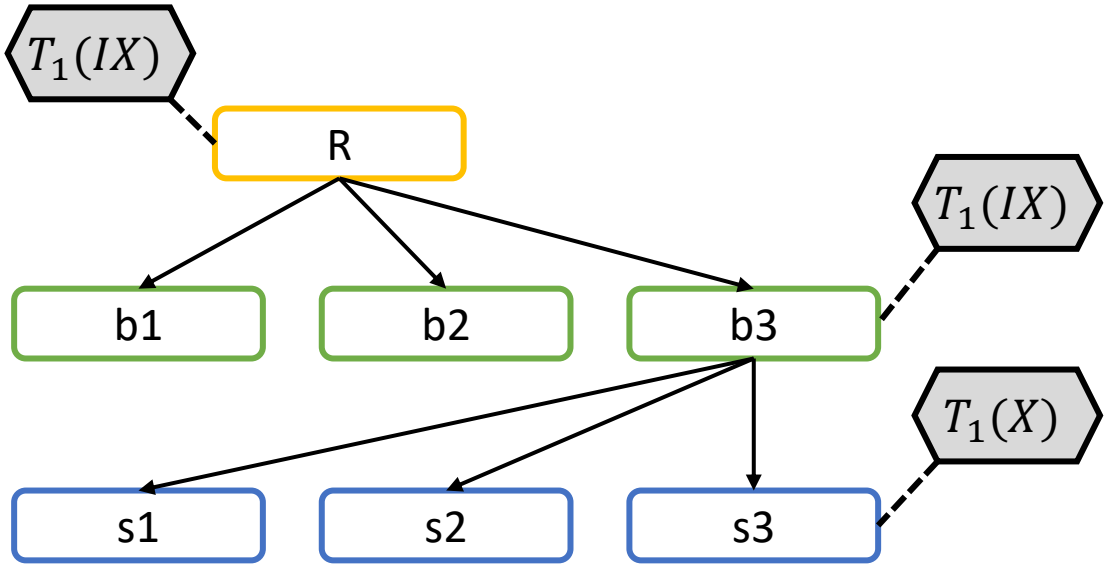
Figyelmeztető zárak használata (példa)

- Az alábbi példában a T_1 tranzakció zárolta az **R** reláció **b3** blokkjának **s3** sorát.
- **Kérdés**: Megkaphatja-e T_2 az X zárat az **s2** sorra? **IGEN**



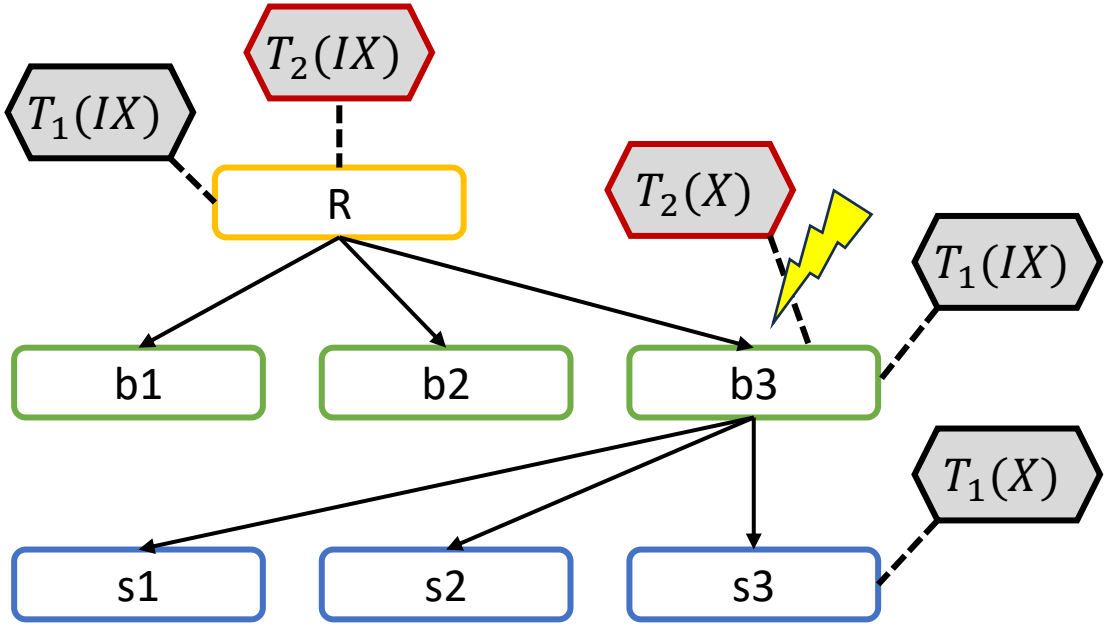
Figyelmeztető zárak használata (példa)

- Az alábbi példában a T_1 tranzakció zárolta az **R** reláció **b3** blokkjának **s3** sorát.
- **Kérdés**: Megkaphatja-e T_2 az X zárat a **b3** blokkra?



Figyelmeztető zárak használata (példa)

- Az alábbi példában a T_1 tranzakció zárolta az **R** reláció **b3** blokkjának **s3** sorát.
- **Kérdés**: Megkaphatja-e T_2 az X zárat a **b3** blokkra? **NEM**



Csoportos mód a szándékzárolásokhoz

- Korábban definiáltuk a zárak **erősségének** a fogalmát (egyik zár erősebb a másiknál).
- A zártábla esetén láttuk a **csoportos mód** hasznát, amely a legerősebb zárat mutatta.
- A figyelmeztető záraknál is jó lenne egy ilyen csoportos mód, ami alapján egy csúcsban értesülhetnénk arról, hogy a hierarchia lentebbi szintjei mire számíthatunk.
- Ehhez minden zárhalmazra **meg kell tudnunk határozni**, hogy melyik a legerősebb zár a halmazban!

Melyik az erősebb zár?

- Zárak erőssége: Azt mondjuk, hogy az L_2 zár **erősebb** az L_1 zárnál, ha a kompatibilitási mátrixban L_2 sorában és oszlopában minden olyan pozícióban „Nem” áll, amelyben L_1 sorában és oszlopában „Nem” áll.
- Melyik erősebb?
  - X vagy S?
  - IS vagy IX?
  - IX vagy S?

Kért zár

Kiadott zár	IS	IX	S	X
IS	igen	igen	igen	nem
IX	igen	igen	nem	nem
S	igen	nem	igen	nem
X	nem	nem	nem	nem

Csoportos mód a szándékzárolásokhoz

- Mivel az S és az IX zárak közül egyik sem erősebb a másiknál, vezessünk be egy **új zárat**!
- Ha egy tranzakciónak S és IX zárolásai is vannak egy elemen akkor ez akár egy zárnak is tekinthető. Legyen ez egy új **SIX** zárolási mód.
- Az SIX mód csoportmódként szolgál, ha van olyan tranzakció, amelynek van S, illetve IX módú, de nincs X módú zárolása!

Kért zár

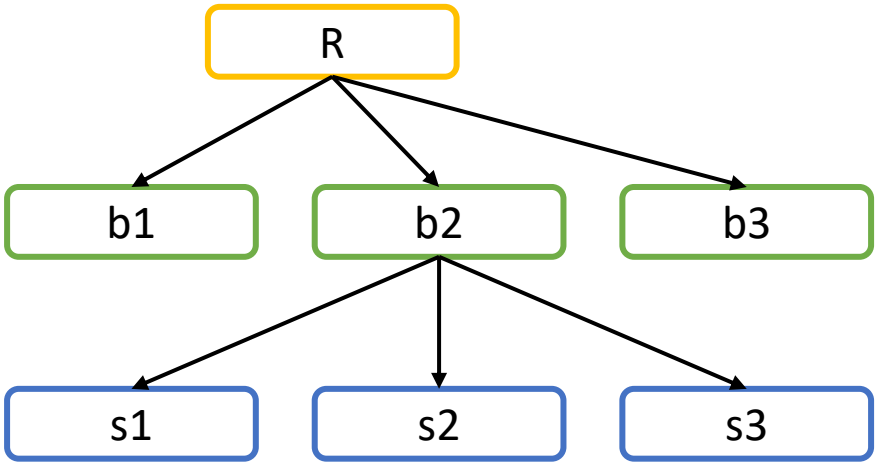
Kiadott zár	IS	IX	S	SIX	X
IS	igen	igen	igen	igen	nem
IX	igen	igen	nem	nem	nem
SIX	igen	nem	nem	nem	nem
S	igen	nem	igen	nem	nem
X	nem	nem	nem	nem	nem

Zárolás csoportos mód használatával

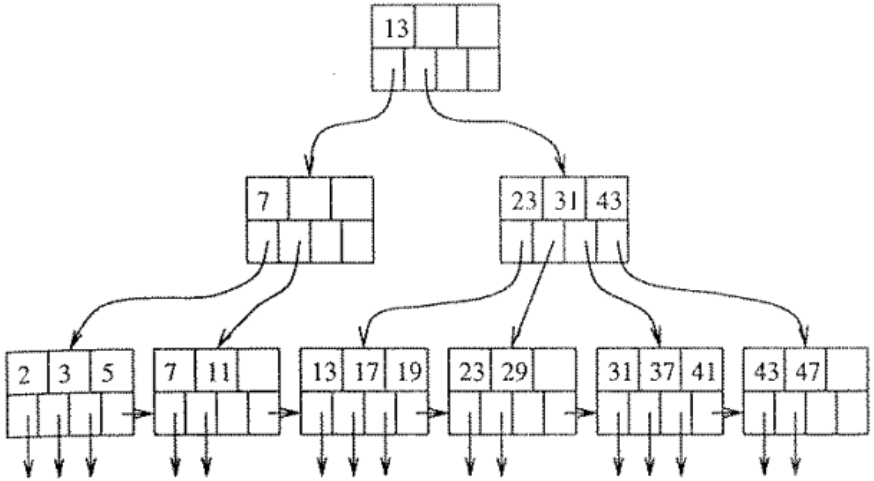
1. A kompatibilitási mátrixnak megfelelően helyezhetjük el a zárakat.
2. Először a **gyökeret zároljuk** valamilyen módban.
3. Egy Q csúcsot egy T_i csak akkor zárolhat **S** vagy **IS** módban, ha a szülő(Q)-t a T_i tranzakció **IX** vagy **IS** módban már zárolta.
4. Egy Q csúcsot egy T_i csak akkor zárolhat **X, SIX, IX** módban, ha a szülő(Q)-t a T_i tranzakció **IX**, vagy **SIX** módban már zárolta.
5. T_i **kétfázisú** zárolási protokollnak eleget tesz.
6. T_i csak akkor engedhet el egy zárat a Q csúcson, ha Q egyik gyerekét sem zárolja a T_i.

Különböző hierarchiák

- Kérdés: mi a **szerkezeti különbség** a most bemutatott hierarchia (relációk, blokkok, rekordok) és egy B+ fa között?



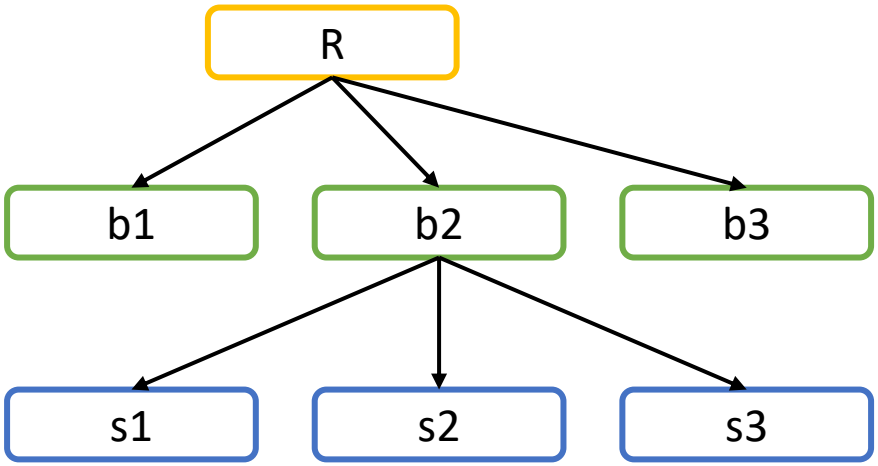
Reláció, blokkok, rekordok



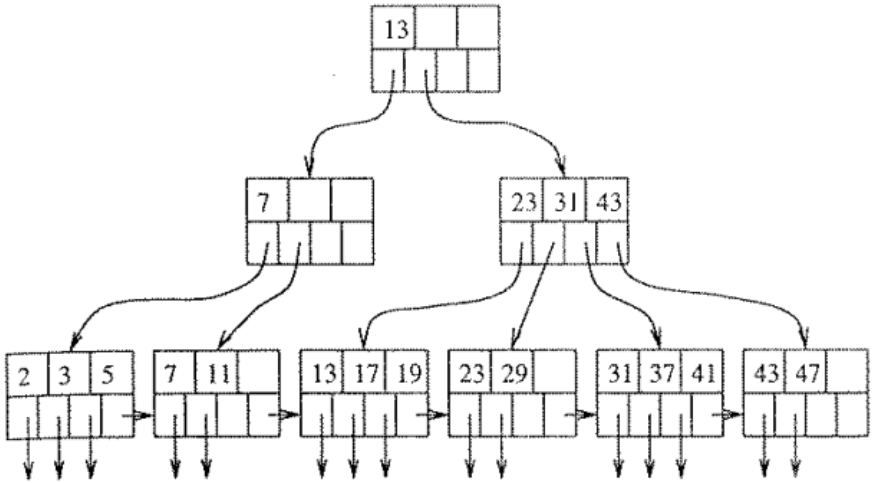
B+ fa index

Különböző hierarchiák

- Kérdés: mi a **szerkezeti különbség** a most bemutatott hierarchia (relációk, blokkok, rekordok) és egy B+ fa között?
- **Válasz**: Az eddigi hierarchiában a gyerekek a szülők részei voltak (pl. egy sor egy blokk része). A B+ fa esetén az egyes csúcsok (blokkok) diszjunkt adatdarabok.



Reláció, blokkok, rekordok



B+ fa index

Miért nem jó a B+ fák esetén a jelenlegi figyelmeztető protokoll?

Zárolás fa struktúrában

- Ha egy új értéket szúrunk be a B+ fába, akkor elméletileg a gyökér csúcs is módosulhat, ezért X zárat kell tennünk minden csúcsra a gyökértől kezdve.
- Emiatt viszont nem lehetne közben olvasni az index egyetlen levél csúcsát sem.
- Sok esetben a gyökér csúcs nem módosul, mi mégis blokkoltuk a többi tranzakciót :\

Kért zár

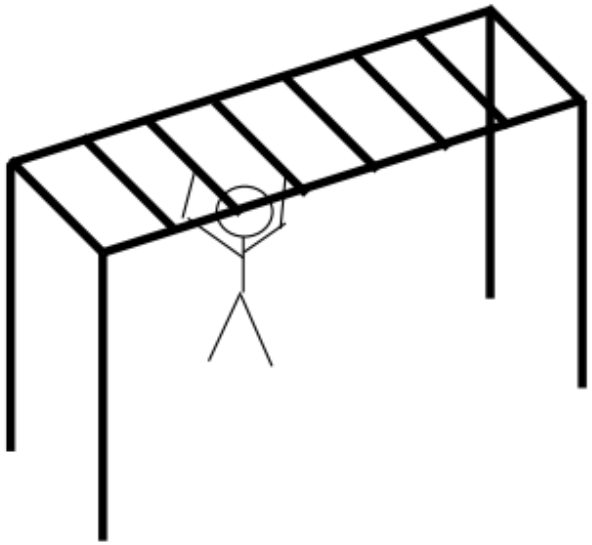
Kiadott zár	IS	IX	S	SIX	X
IS	igen	igen	igen	igen	nem
IX	igen	igen	nem	nem	nem
SIX	igen	nem	nem	nem	nem
S	igen	nem	igen	nem	nem
X	nem	nem	nem	nem	nem

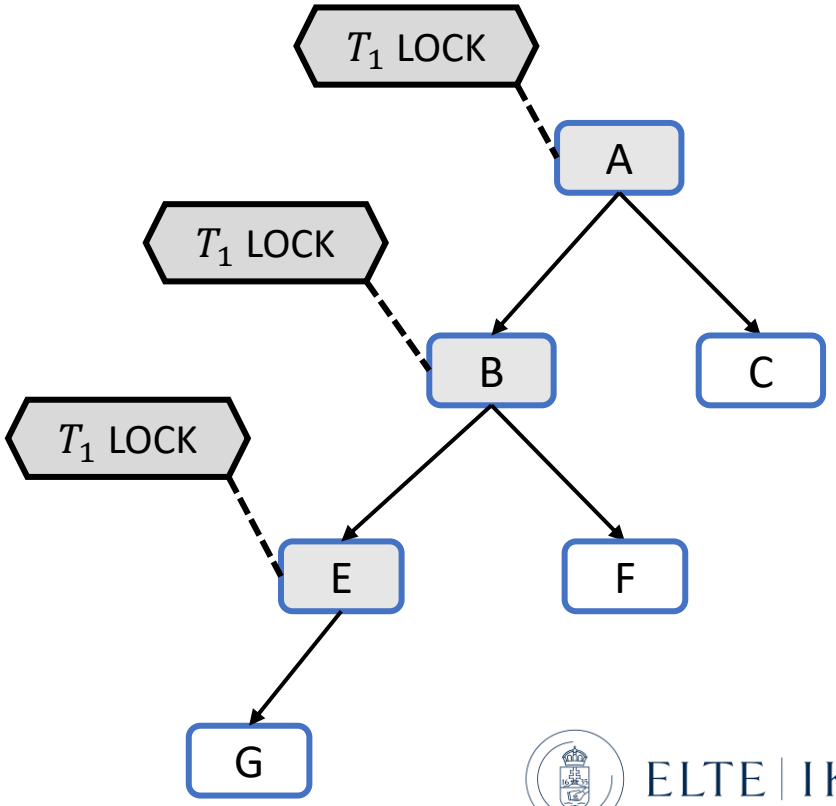
Zárolás fa struktúrában

- A következő megfigyeléseket tehetjük:
  - Ha beszúrunk egy értéket, de a levél csúcs (vagy általánosabban bármelyik csúcs) nincs teljesen tele, akkor tudhatjuk, hogy a beszúrás nem gyűrűzik fel a gyökérig (mivel nincs szükség csúcsosztásra).
  - Hasonló a helyzet a törlésnél is.
- **Ötlet**: Ha a tranzakció látja, hogy a gyökér biztosan nem változik meg, azonnal feloldhatnánk a zárat (ugyanígy más csomópontokra is).
- **Probléma**: A gyökéren lévő zárolás korai feloldása ellentmond a 2PL-nek (pedig erre épült a sorbarendezhetőségünk).
- **Megoldás**: egy speciális protokoll a fa struktúrájú adatokhoz.

Faprotokoll (mászóka analógia)

- Ha egy zárra már nincs szükségünk a hierarchia egy magasabb szintjén, akkor azt elengedhetjük





Faprotokoll szabályai

- Használjunk most csak egy féle zárat (az ötlet kiterjeszthető több zármódra is).
- Egy tranzakció követi a faprotokollt, ha:
    1. Az első zárat bárhová elhelyezheti.
    2. A későbbiekben csak akkor kaphat zárat egy csúcson, ha ekkor zárja van a csúcs szülőjén.
    3. Zárat bármikor fel lehet oldani.
    4. Nem lehet újrazárolni – azaz, ha a tranzakció elengedte egy adategység zárját, akkor később nem kérhet rá újra (még akkor sem, ha annak a szülőjén még megvan a zárja).

Faprotokoll tétele

- **Tétel**: Ha minden tranzakció követi a faprotokollt egy jogszerű ütemezésben, akkor az ütemezés sorbarendezhető lesz, noha nem feltétlenül lesz 2PL.
- Bizonyítás: indukciós (tankönyvben megtalálható)
- **Eredménye**: A fa struktúrát (pl. B+ fa) konkurensebben tudjuk használni, mint más zárolási protokollok esetén.

Eszközök sorbarendezhetőség elérésére

- Az ütemezőnek különböző lehetőségei vannak arra, hogy kikényszerítse a sorbarendezhető ütemezéseket:
  - Zárak
  - **Időbélyegzés**
  - **Érvényesítés**
- Eddig a zárolásokkal foglalkoztunk, amely pesszimista megközelítésnek tekinthető.
  - Pesszimista, mert azt feltételezi, hogy a dolgok rosszra fordulnak, hacsak nem akadályozzuk meg.
  - A zárolási ütemező késlelteti a tranzakciókat és csak akkor abortálja a holtpont alakul ki.

Optimista megközelítések

- Az **időbélyegzés** és az **érvényesítés** optimista megközelítésnek tekinthetők.
  - Az optimista megközelítések abban különböznek a zárolásoktól, hogy egyetlen ellenszerük, amikor valami rosszra fordul, hogy azt a tranzakciót, amely nem sorbarendezhető viselkedést okozna abortálják, majd újraindítják.
- **Mikor érdemes** optimista megközelítést használni? Ha 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** (timestamp).
- Az időbélyegzőket **növekvő sorrendben** kell kiadni abban az időpontban, amikor a tranzakció az elindításáról először értesíti az ütemezőt.
- **Két lehetséges megoldás** időbélyegző generálására:
  - A **rendszeróra** felhasználásával hozzuk létre.
  - Az ütemező karbantart egy **számlálót** (counter). Minden alkalommal, amikor egy új tranzakció elindul, a számláló növekszik egyel és ez lesz az új tranzakció időbélyegzője. Így egy később elindított tranzakció biztosan nagyobb időbélyegzőt kap, mint a korábban elindított tranzakciók.

Adatbáziselemek tartalmaz

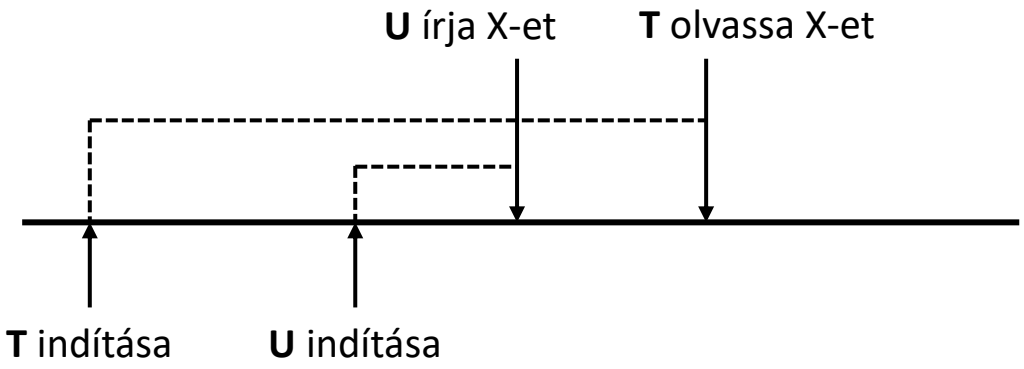
- Minden egyes X adatbáziselemhez hozzá kell rendelnünk **két időbélyegzőt** és egy további **bitet**:
  - **RT(X): X olvasási ideje** (read time), amely a legmagasabb időbélyegző, ami egy olyan tranzakcióhoz tartozik, amely már olvasta X-et.
  - **WT(X): X írási ideje** (write time), amely a legmagasabb időbélyegző, ami egy olyan trnazakcióhoz tartozik, amely már írta X-et.
  - **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 véglegesítés bit célja, hogy elkerüljük a **piszkos olvasást**, amikor egy tranzakció egy másik tranzakció által írt, nem véglegesített adatokat olvas – ez az adatbázis inkonzisztenssé válását okozhatja.

Ellenőrzés időbélyegzőkkel

- Az ütemező olvasáskor és íráskor **ellenőrzi**, hogy ez abban a sorrendben történik-e, mintha a tranzakciókat az időbélyegzőjük szerinti növekvő, **soros ütemezésbe**n hajtottuk volna végre.
- Ha nem, akkor azt mondjuk, hogy a viselkedés fizikailag nem megvalósítható és ilyenkor beavatkozik az ütemező (abortál).
- Kétféle „fizikailag nem megvalósítható” jelenség jöhet elő:
    1. Túl késői olvasás.
    2. Túl késői írás.

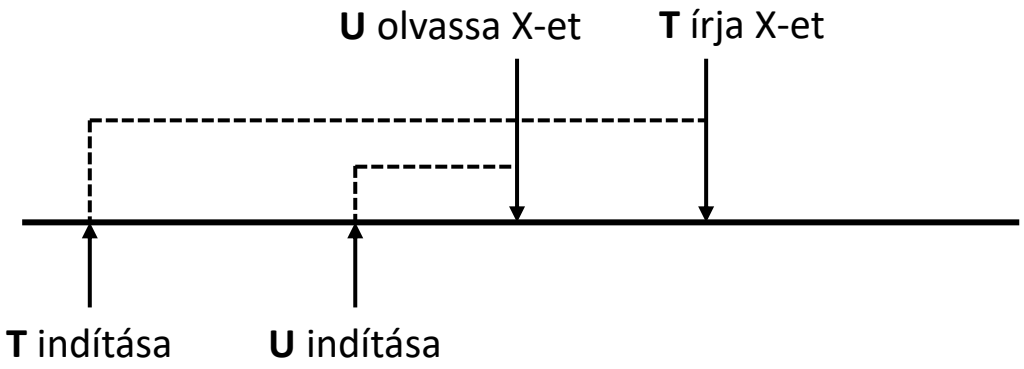
Túl késői olvasás

- A T tranzakció megpróbálja olvasni az X adatbáziselemet, de az X írási ideje azt jelzi, hogy X jelenlegi értékét azután írtuk, miután T-t már elméletileg végrehajtottuk (az elképzelt soros ütemezés szerint).
- Azaz: **TS(T) < WT(X)**
- **Megoldás**: ha ilyen probléma felmerül, akkor a T tranzakciót abortáljuk.



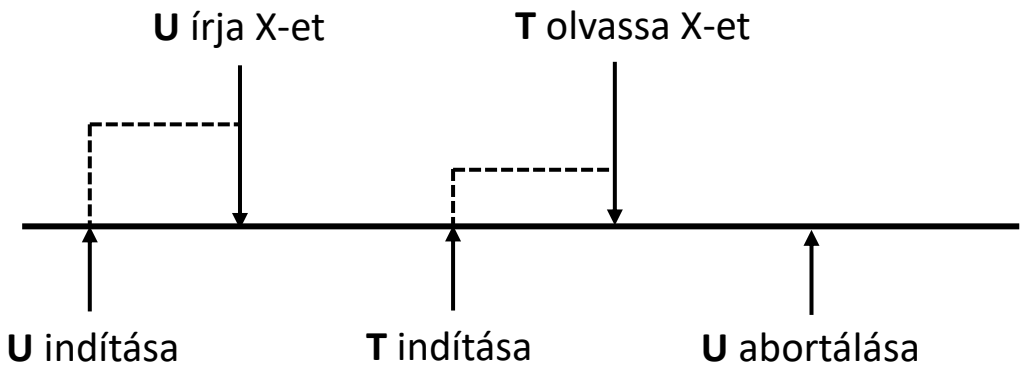
Túl késői írás

- A T tranzakció megpróbálja írni az X adatbáziselemet, de az X olvasási ideje azt jelzi, hogy van egy másik tranzakció is, amelynek a T által beírt értéket kellene olvasnia, ám ehelyett más érétket olvas.
- Azaz: **WT(X) <= TS(T) < RT(X)** vagy **TS(T) < RT(X) < WT(X)**
- **Megoldás**: ha ilyen probléma felmerül, akkor a T tranzakciót abortáljuk.



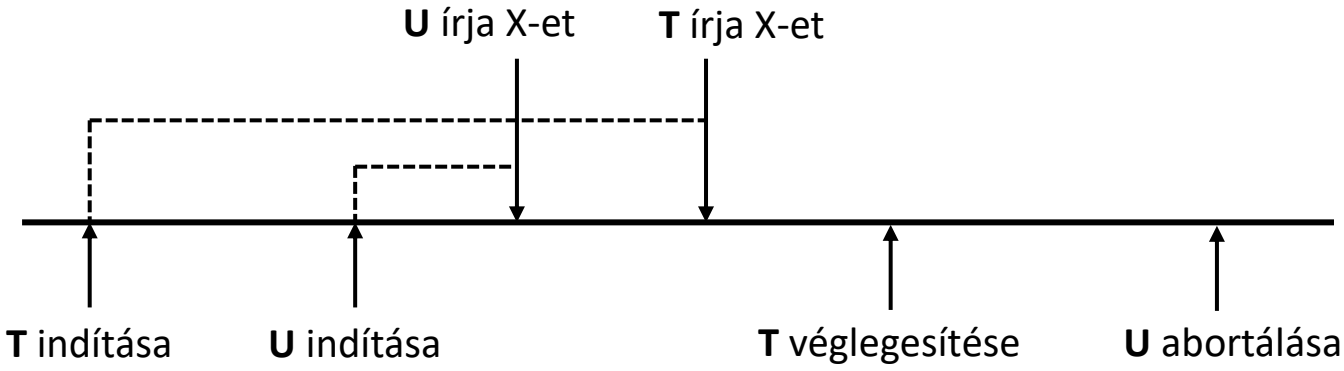
Piszkos adatok problémái

- A lenti esetben T **piszkos adatokat** (nem véglegesített) olvas – ezt szeretnénk elkerülni. Azt, hogy U még nincs véglegesítve onnan tudjuk, hogy a C(X) véglegesítési bit hamis.
- **Megoldás**: a T tranzakció olvasását várakoztatjuk.



Thomas-féle írási szabály

- **Szabály**: Amikor T írja X-et, igazából semmit sem csinálunk (kihagyjuk az írást), mivel X-ben az U által írt értéknek kell lennie, mivel TS(U) > TS(T).
- **Probléma**: Ha az U-t később abortáljuk (mint a lenti példában), akkor az U általi írást semmissé kell tennünk, és a T általi írás eredményét kell az X-ben látnunk. Viszont azt az írást kihagytuk :\



Megoldás a problémára

- A **Thomas-féle írási szabály** problémájára több megoldás is lehetséges, ezek közül egyik:
  - Amikor a T tranzakció írja az X adatbáziselemet, az írás „kísérleti”, és vissza lehet állítani, ha a T-t abortáljuk. A C(X) véglegesítési bitet hamisra állítjuk, egyúttal az ütemező másolatot készít az X régi értékéről, és az előző WT(X)-ről.
- Ezen a megoldáson alapul az ún. **többváltozatú időbélyegzés** (a tankönyvben kifejtésre kerül).

Időbélyegzésen alapuló ütemező

- Az időbélyegzésen alapuló ütemező egy tranzakció kérésére a következő válaszokat adhatja:
    1. Engedélyezi a kérést.
    2. Abortálja a T-t (ha T megsérti a fizikai valóságot) és egy új időbélyegzővel újraindítja.
    3. Késlelteti a T-t, és később dönti el, hogy abortálja vagy engedélyezi a kérést

Konkurenciavezérlés érvényesítéssel

- Az **érvényesítés** (validation) az optimista konkurenciavezérlés másik típusa, amelyben a tranzakcióknak megengedjük, hogy **zárolások nélkül** hozzáférjenek az adatokhoz és a megfelelő időben ellenőrizzék a tranzakció sorbarendezhető viselkedését.
- Az egyes tranzakciókhoz nyilvántartja, hogy milyen adatbáziselemeket **olvasnak és írnak**.
- Mielőtt a tranzakció írni kezdene értékeket, egy **érvényesítési fázison** megy keresztül, amikor az olvasandó és írandó adatbáziselemek halmazát összehasonlítjuk más tranzakciók halmazaival. Ha ütközést tapasztalunk, akkor a tranzakciót abortáljuk és újraindítjuk (**visszagörgetés**).

A módszerek összehasonlítása

- A **zárolás késlelteti** a tranzakciókat, azonban elkerüli a visszagörgetéseket (kivéve a holtpontoknál), még ha erős is a tranzakciók egymásra hatása. Az **időbélyegzés** és **érvényesítés** nem késleltet, de **visszagörgeti** a tranzakciókat.
- Ha gyenge az egymásra hatás, az időbélyegzés és érvényesítés nem okoz sok visszagörgetést és előnyösebb lehet a zárolásnál, mivel ezeknek általában alacsonyabb a költsége (pl. nincs zártábla).
- Amikor szükséges a visszagörgetés az **időbélyegzők hamarabb észreveszik a problémákat**, mint az érvényesítés.

Konkurenciavezérlés Oracle-ben

- Az Oracle alapvetően a zárolás módszerét használja a konkurenciavezérléshez.
- A zárakat az ütemező automatikus helyezi el, de lehetőség van manuális zárolásra is.
- Az Oracle alkalmazza a **kétfázisú zárolást** (2PL), a **figyelmeztető protokollt** és a **többváltozatú időbélyegzőket** is (néhány módosítással).

Elkülönítési szintek

- Az SQL92 ANSI/ISO szabvány a tranzakcióelkülönítés négy szintjét definiálja, amelyek abban különböznek egymástól, hogy milyen jelenségeket engednek meg:
  - **Piszkos olvasás** (dirty read): a tranzakció olyan adatot olvas, amelyet egy másik, még nem véglegesített tranzakció írt.
  - **Nem ismételhető (fuzzy) olvasás (nonrepeatable read)**: a tranzakció újraolvas olyan adatokat, amelyeket már korábban beolvasott és azt találja, hogy egy másik, már véglegesített tranzakció módosította vagy törölte őket.
  - **Fantomok olvasása (phantom read)**: egy tranzakció újra végrehajt egy lekérdezést és azt találja, hogy egy másik, már véglegesített tranzakció további sorokat szúrt be.

Elkülönítési szintek

- Ezek közül az Oracle a **read committed** és a **serializable** elkülönítése szinteket implementálja.
- És még egyet, amely nem része a szabványnak: **csak olvasás** (read-only).

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Lehetséges	Lehetséges	Lehetséges
Read committed	Nem lehetséges	Lehetséges	Lehetséges
Repeatable read	Nem lehetséges	Nem lehetséges	Lehetséges
Serializable	Nem lehetséges	Nem lehetséges	Nem lehetséges

Read committed elkülönítési szint

- **Read committed** (magyarul: olvasásbiztos).
  - Ez az **alapértelmezett** elkülönítési szint.
  - Egy tranzakció minden lekérdezése csak a véglegesített tranzakciók által írt adatokat láthatják.
  - **Piszkos olvasás sohasem történik**, nem ismételhető olvasás és fantom sorok olvasása előfordulhat.
- Olyan környezetben érdemes ezt a szintet használni, ahol várhatóan kevés tranzakció kerül egymással konfliktusba.

Serializable elkülönítése szint

- **Serlializable** (magyarul: sorbarendezhető)
  - Nem alapértelmezett, de **beállítható** (tranzakció és munkamenet szintjén is)
  - Csak a tranzakció elindítása előtt véglegesített változásokat látják.
  - **Semmilyen probléma** nem léphet fel ezen a szinten.
- Olyan rendszerekben érdemes használni, ahol kiemelt fontosságú a megfelelő végrehajtás (pl. banki rendszerek) és általában kicsi az esélye, hogy két tranzakció ugyanazokat a sorokat módosítja.

Read only elkülönítés

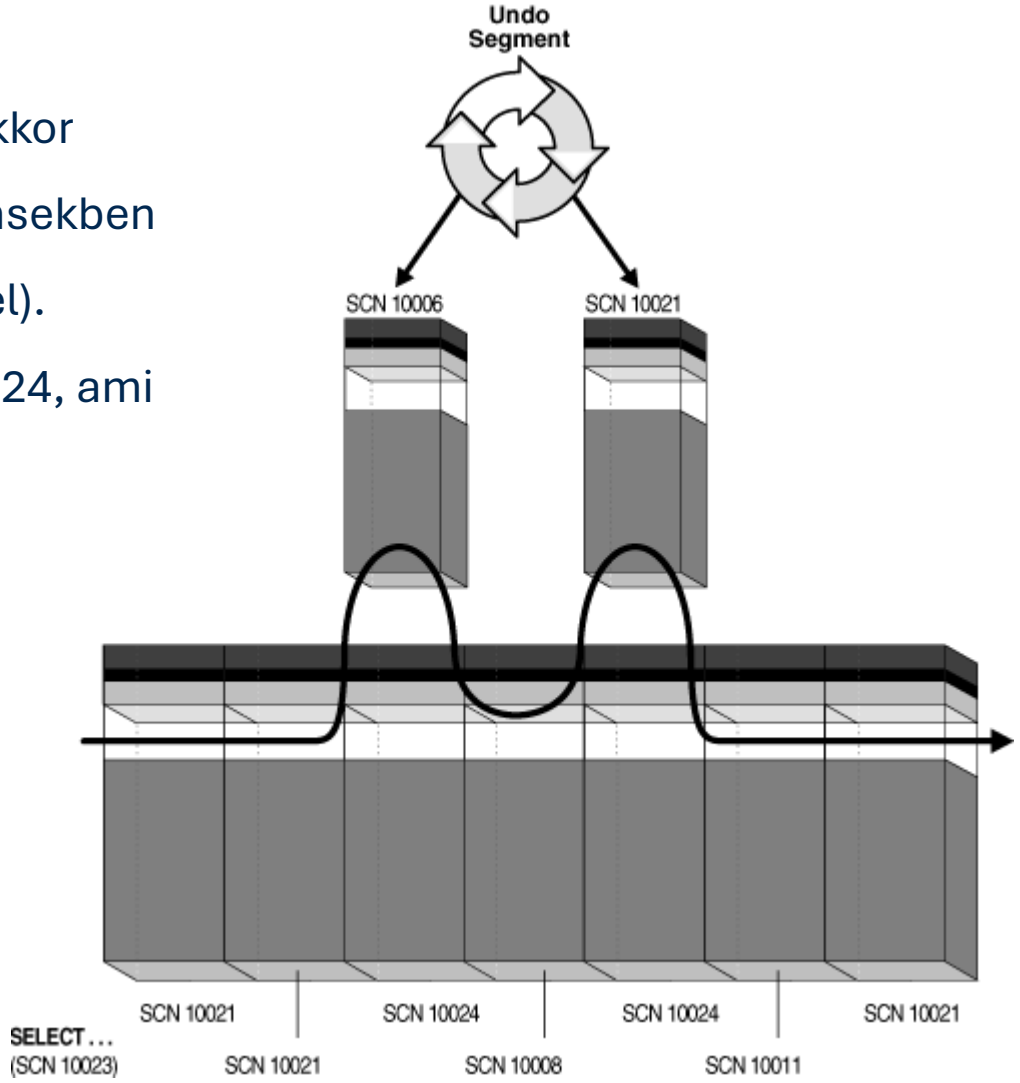
- A csak olvasást végző tranzakciók csak a tranzakció elindítása előtt véglegesített változásokat látják és nem engednek meg INSERT, UPDATE és DELETE utasításokat.

Olvasási konzisztencia elérése

- Az olvasási konzisztencia eléréshez az Oracle a rollback szegmenseket és a bennük található információkat használja fel.
    - Ezek tárolják az adatok régi értékeit, amelyeket még nem véglegesített vagy nem rég véglegesített tranzakciók változtattak meg.
- Amint egy lekérdezés vagy tranzakció megkezdi működését, meghatározódik a **system change number** (**SCN**) aktuális értéke.
- Az SCN a blokkokhoz, mint adatábziselemekhez tartozó **időbélyegzőnek** tekinthető.
- Ezt Multiversion Concurrency Control-nak (**MVCC**) nevezik, ami az időbélyegzés egy implementációjának tekinthető.

SCN használata

- A tranzakció SCN-je: **10023**.
- Ha ettől nagyobb SCN-t lát, akkor megkeresi a rollback szegmensekben a régi értékeket (kisebb SCN-el).
- **Pl.** a 3. és 5. blokk SCN-je 10024, ami nagyobb, mint a 10023.



Zármódok

- Zármódok:
  - **DML-zárak** (adatzárak): az adatok védelmére szolgálnak.
  - **DDL-zárak** (szótárzárak): a sémaobjektumok szerkezetének a védelmére valók.
  - **Belső zárak**: a belső adatszerkezetek, adatfájlok védelmére szolgálnak, kezelésük teljesen automatikus.
- A DML-záraknak két szintje van:
  - **Sor szintű (TX)**
  - **Tábla szintű (TM)**
- Egy tranzakció **tetszőleges számú** sor szintű zárat fenntarthat.
- Sorok szintjén csak egyféle zármód létezik, a **kizárólagos (X)**.

Zármódok

- **Működés lényege**: a tranzakciók csak akkor versengenek az adatokért, ha ugyanazokat a sorokat próbálják meg írni.
- Egy tranzakció kizárólagos DML-zárat kap **minden egyes sorra**, amelyet az alábbi utasítások módosítanak:
  - INSERT
  - UPDATE
  - DELETE
  - SELECT … FOR UPDATE
- Amikor egy tranzakció sor szintű zárat kap bizonyos sorokra, akkor emellett **tábla szintű zárat (TM)** is kap a táblára, amelyben a sorok vannak.
- A következő dián ezek felsorolásra kerülnek (a felsorolás sorrendjében egyre erősebbek a zárak).

Tábla szintű (TM) zárak

Mód	LMODE	Manuális utas.	Jelentése
RS/SS	2	LOCK TABLE <table> IN ROW SHARE MODE;	Egy tranzakció zárolt bizonyos sorokat a táblában és módosítani szándékozik.
RX/SX	3	LOCK TABLE <table> IN ROW EXCLUSIVE MODE;	Egy tranzakció zárolt bizonyos sorokat a táblában és módosította őket.
S	4	LOCK TABLE <table> IN SHARE MODE;	Több tranzakció is birtokolhatja és olvashatja a táblát egy időben, de csak akkor lehet módosítani ha csak egyetlen tranzakció zárolja.
SRX/SSX	5	LOCK TABLE <table> IN SHARE ROW EXCLUSIVE MODE;	Csak egy tranzakció birtokolhatja ezt a zárat. Mások olvashatják, de nem módosíthatják.
X	6	LOCK TABLE <table> IN EXCLUSIVE MODE;	A legszigorúbb zár, csak egy tranzakció birtokolhatja, csak ő módosíthat és olvashat.

Kompatibilitási mátrix

SQL Statement	Row Locks	Table Lock Mode	RS	RX	S	SRX	X
SELECT ... FROM *table*...	—	none	Y	Y	Y	Y	Y
INSERT INTO *table* ...	Yes	SX	Y	Y	N	N	N
UPDATE *table* ...	Yes	SX	Y[Foot 1]	Y[Foot 1]	N	N	N
MERGE INTO *table* ...	Yes	SX	Y	Y	N	N	N
DELETE FROM *table* ...	Yes	SX	Y[Foot 1]	Y[Foot 1]	N	N	N
SELECT ... FROM *table* FOR UPDATE OF ...	Yes	SX	Y[Foot 1]	Y[Foot 1]	N	N	N
LOCK TABLE *table* IN ...	—						
ROW SHARE MODE		SS	Y	Y	Y	Y	N
ROW EXCLUSIVE MODE		SX	Y	Y	N	N	N
SHARE MODE		S	Y	N	Y	N	N
SHARE ROW EXCLUSIVE MODE		SSX	Y	N	N	N	N
EXCLUSIVE MODE		X	N	N	N	N	N

[1] Igen, ha nincsenek ütköző sorzárak egy másik tranzakció által fenntartva. Ellenkező esetben várakozás történik.

Forrás: https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/Automatic-Locks-in-DML-Operations.html#GUID-3D57596F-8B73-4C80-8F4D-79A12F781EFD__BABHAJFD

Zárak felminősítése

- Mivel a sor szintű záraknak csak egy módja van (kizárólagos), ezért ott nem létezik felminősítése.
- A táblák szintjén az Oracle **automatikusan felminősít** egy zárat egy erősebb módú zárrá, **amikor szükséges** (és lehetséges).
    - Pl. Egy SELECT … FOR UPDATE utasítás RS módban zárolja a táblát. Ha a tranzakció később módosít a zárolt sorok közül néhányat, akkor az RS mód automatikusan felminősül RX módra.

Zárak 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. tábla szintjére) emeli.
    - Pl. Ha a felhasználó sok sort zárol egy táblán, 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 a kockázatát (pl. IDMB Db2 és SQL Server használja).

Összefoglalás

- A valós rendszerek a konkurencia vezérléshez használják a zárakat, időbélyegzőket és érvényesítést is.
- A zárolási ütemező jogszerű konzisztens és 2PL zárolásokat készít.
- Emellett a többszintű zárolás (hierarchia) és a felminősítés is jelen van.
- Leggyakrabban a zárolások és az időbélyegzők hibrid implementálása fordul elő.

Tankönyv fejezetek

- Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom:

  Adatbázisrendszerek megvalósítása

  - 9.3 - 9.10 fejezetek
- Silberschatz, Korth, & Sudarshan: **Database System Concepts**
  - Chapter 18. Concurrency Control