# TRANZAKCIÓKEZELÉS
## ADATBÁZISOK SEPSISZENTGYÖRGY

*Dóka - Molnár Andrea*
*andrea.molnar@ubbcluj.ro*

**BABEŞ-BOLYAI TUDOMÁNYEGYETEM**
Matematika és Informatika Kar

- Eddig feltételeztük:
  - egy felhasználó van csak;
  - a lekérdezések/módosítások hiba nélkül lefutnak.
- A valóságban problémákat okozhat:
  - műveletek párhuzamos végrehajtása, többfelhasználós működés (pl. banki rendszerek, helyfoglalás)
  - rendszerhibák fellépése

# Megoldandó problémák

- Többfelhasználós működés

1. felhasználó: $u1; u2; : : : ; u10$
2. felhasználó: $v1; v2; : : : ; v103$

Ez a két utasítássorozat nem elkülönülve jön, hanem összefésülődnek:

$u1; v1; v2; u2; u3; v3; : : : ; v103; u10$

Ebből viszont problémák származhatnak:

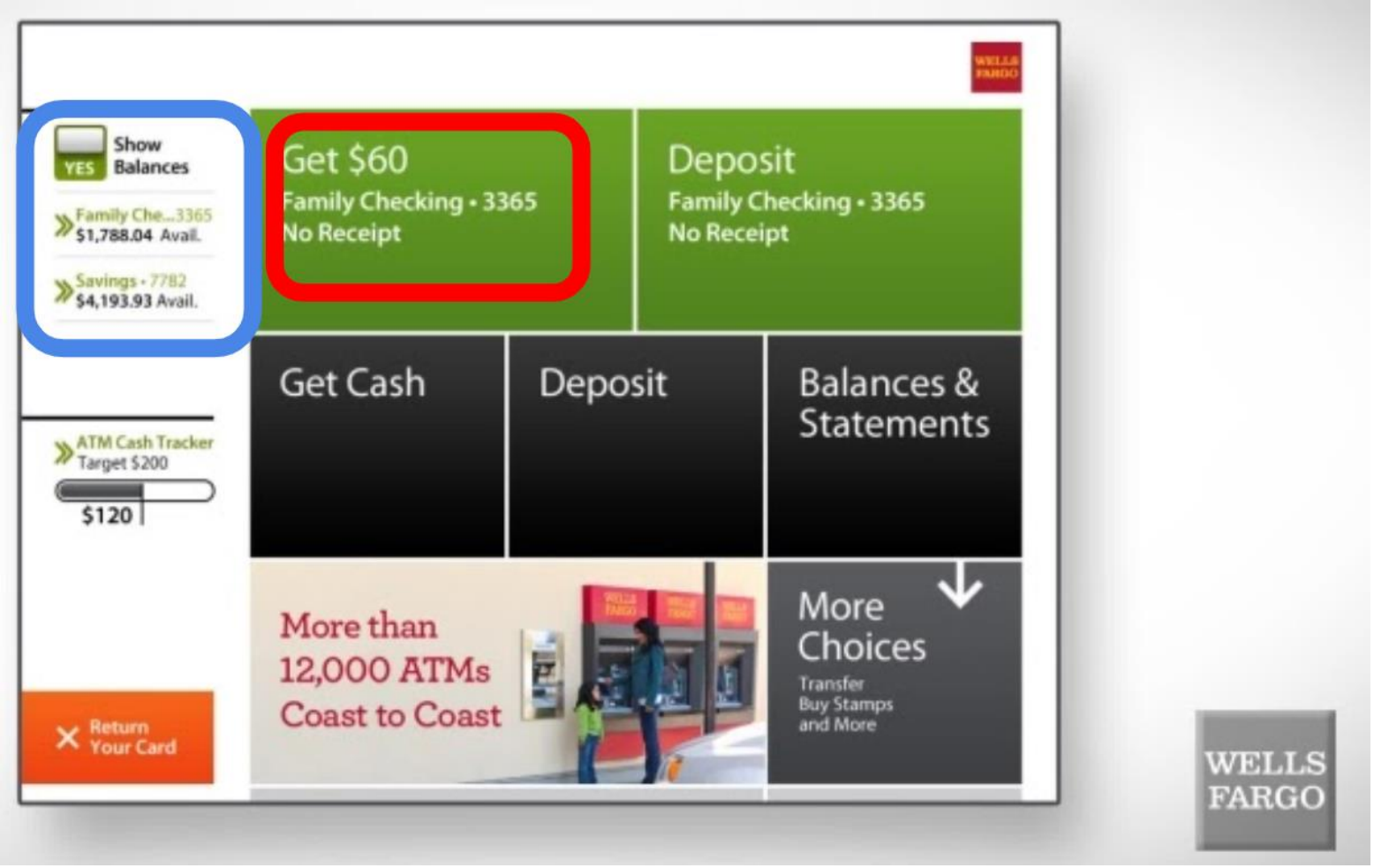
1. felhasználó: READ $A$, $A$ + +, WRITE $A$
2. felhasználó: READ $A$, $A$ + +, WRITE $A$

Ha ezek úgy fésül ődnek össze, hogy:

(READ $A$)1, (READ $A$)2, ($A$ + +)1,($A$ + +)2, (WRITE $A$)1, (WRITE $A$)2

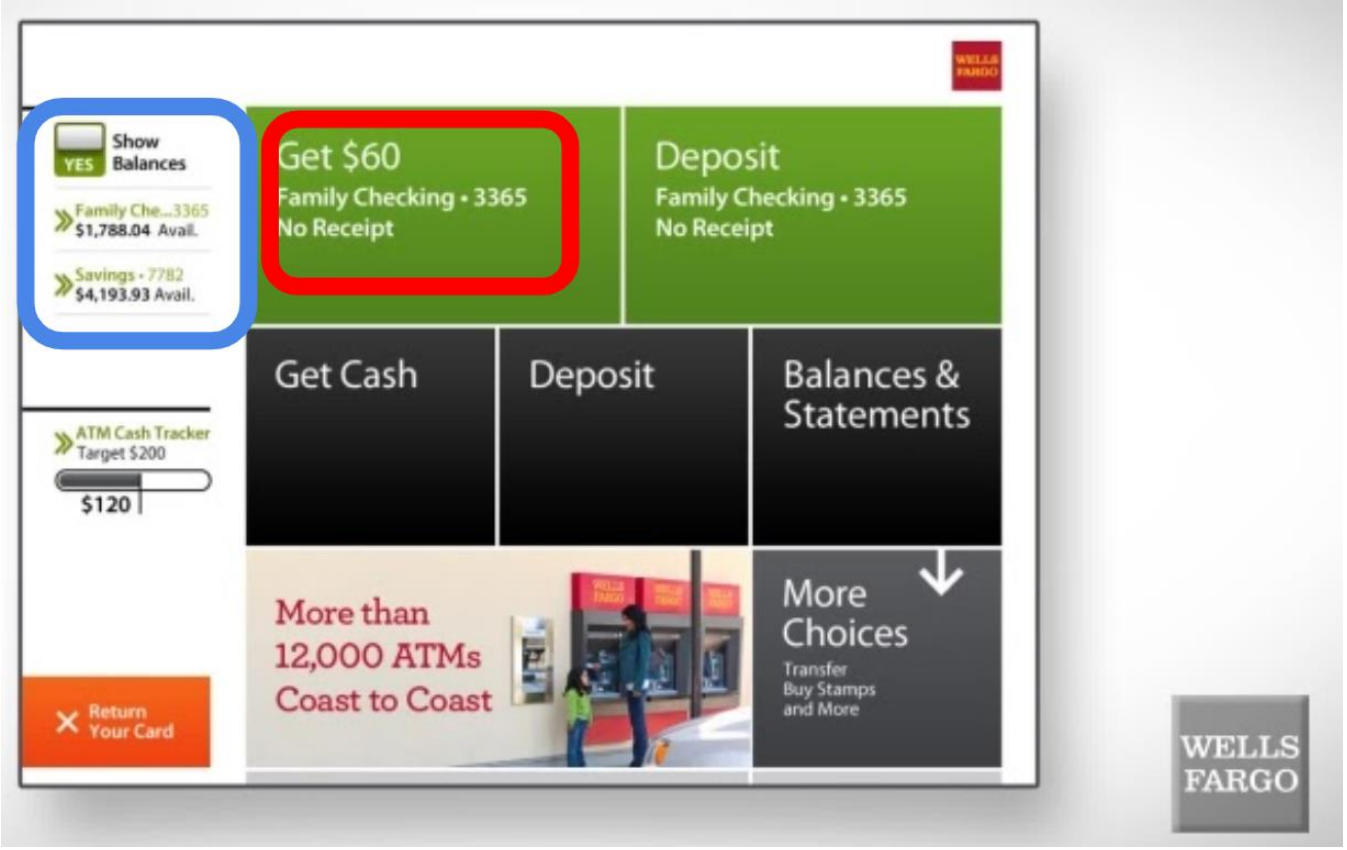
akkor a végén csak eggyel nő A értéke, holott kettővel kellett volna.

# Bevezető példa
## ATM adatbázisa




Forrás: https://bit.ly/2HD4f40

# Bevezető példa
## ATM adatbázisa




Forrás: https://bit.ly/2HD4f40

- Műveletek:
  1. Egyenleg lekérése
  2. Pénz kiadása
  3. Egyenleg módosítása

*Mi történik, ha valamelyik lépésnél hiba lép fel?*
*Definiálhatunk-e más sorrendet*
*a műveletek esetén?*

# Megoldandó problémák

- Rendszerhibák
  - Példa: átutalunk egyik helyről a másik helyre pénzt:

    A := A -50 B := B + 50
  - Ha az a közepén meghal: hibás állapot jön létre.

# Tranzakciókezelés

- A valóságban problémákat okozhat:
  - műveletek párhuzamos végrehajtása, többfelhasználós működés (pl. banki rendszerek, helyfoglalás)
  - rendszerhibák fellépése

Problémák megoldása ↔ tranzakciókezelő dolga:

- párhuzamosság biztosítása többfelhasználós működés esetén is
- rendszerhibák utáni helyreállítás (az adatbázist inkonzisztens állapotból konzisztens állapotba kell hozni)

Megoldás: **tranzakció**

# Tranzakciókezelés

- Tranzakció = az adatbázison végzett műveletek (parancsok) sorozata, mely az ABKR szempontjából **egy egységet** alkot:
  - Ha lehetséges: az egész tranzakció végrehajtása.
  - Ha nem lehetséges: semmi változtatást ne végezzen az adatbázison a tranzakció.

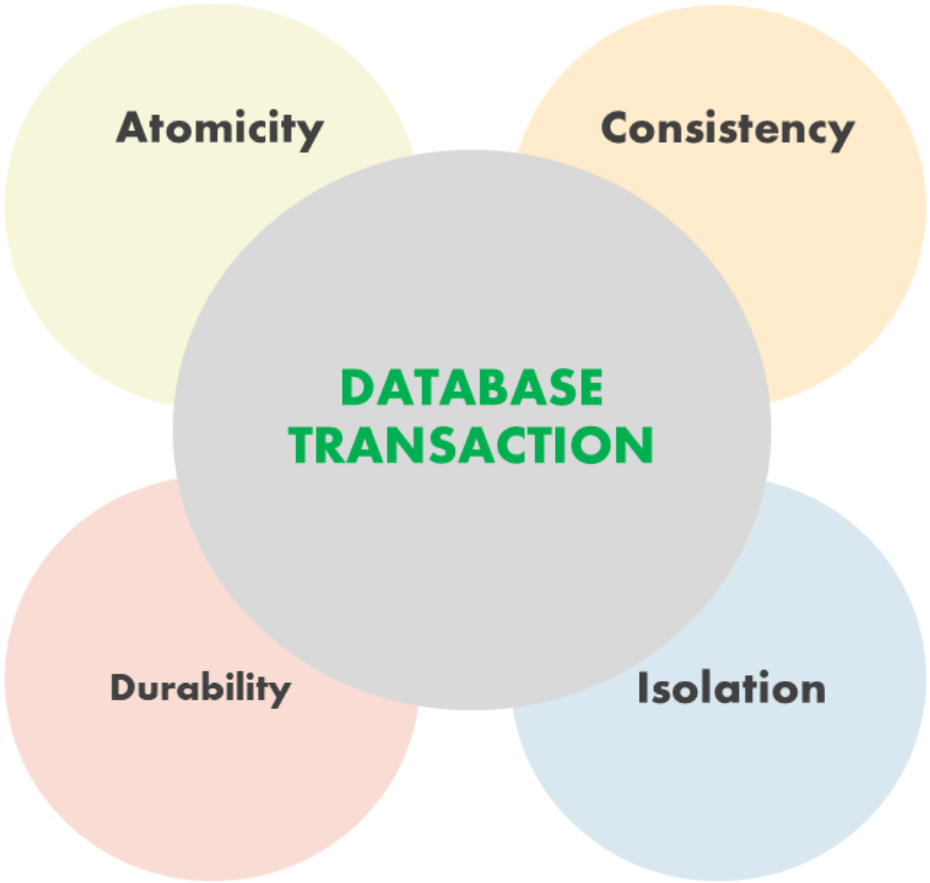
  ⇒ **A (atomicity) - atomiság**

# További elvárások

- **C (consistency) – konzisztencia:** a tranzakció a helyesség (konzisztencia) egysége, az adatbázist egy helyes állapotból egy másik helyes állapotba alakítja.
- **I (isolation) – izoláció:** különböző tranzakciók egymástól elszigetelten futnak, mintha egymás után hajtódnának végre; *valójában*: egyidejűleg versengenek az adatbázis elemekért.
- **D (durability) – tartósság:** ha a tranzakció elért a végpontjához (COMMIT), az általa végzett adatbázis-módosítások véglegesek, még ha közben esetleg hiba is lép fel.

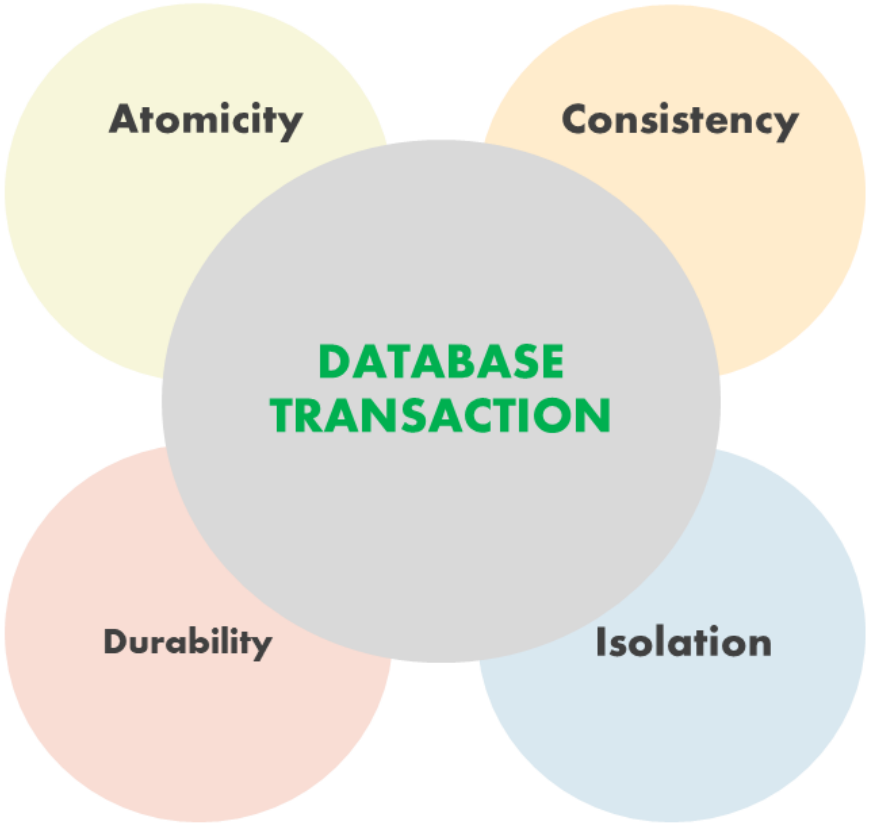
**+ A (atomicity) – atomiság** → tranzakciók **ACID tulajdonságai** ⇔ A tranzakció a ***konkurencia***, a ***helyesség*** és a ***visszaállítás*** egysége.

# További elvárások




⇒ tranzakciók **ACID tulajdonságai** ⇔ A tranzakció a ***konkurencia***, a ***helyesség*** és a ***visszaállítás*** egysége.

# További elvárások




Konzisztencia – mindig adottnak tekintjük.

Másik három tulajdonság biztosítása: ABKR feladata.

# Többfelhasználós működés

Konzisztencia – mindig adottnak tekintjük.

⇔ CÉL: párhuzamos hozzáférés biztosítása oly módon, hogy a konzisztencia megmaradjon.

- Ettől kezdve feltételezzük:

  Ha **T** tranzakció indulásakor *DB* adatbázis konzisztens állapotban van

  \+ **T** tranzakció egyedül fut le (többitől elkülönítve)

  ⇒ **T** tranzakció befejezésekor *DB'* adatbázist konzisztens állapotban hagyja.

# Többfelhasználós működés

Konzisztencia – mindig adottnak tekintjük.

⇔ CÉL: párhuzamos hozzáférés biztosítása oly módon, hogy a konzisztencia megmaradjon.

⇒ tranzakció = konzisztenciát megtartó adatmódosító (adatkezelő) műveletek sorozata

# Implicit vs. Explicit tranzakció

```
INSERT INTO Alkalmazottak
  VALUES (123454, 'Kovács
  Lehel', 2, 500)

UPDATE Alkalmazottak
SET Fizetés=Fizetés * 1.2

CREATE TABLE Activities(
   ActivID int IDENTITY
   (1, 1) NOT NULL,
   TaskID int NOT NULL,
   CPersonID int NULL,
   PlannedStartDate date,
   PlannedEndDate date,
   RealStartDate date,
   … )
```

- Tranzakció kezdete:

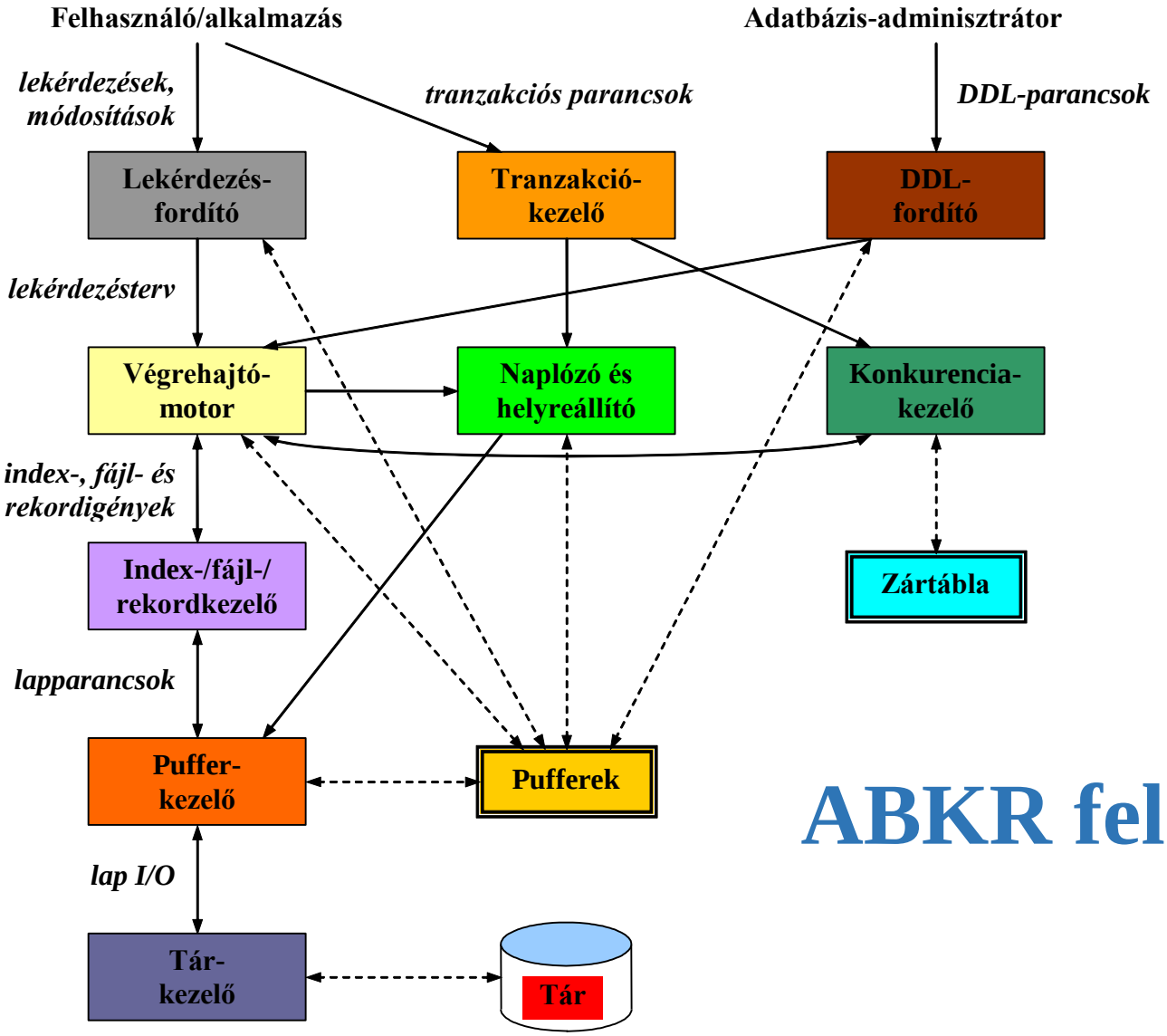
**BEGIN TRANSACTION**

- Sikeres befejeződése:

**COMMIT [TRANSACTION]**

- Sikertelen befejezése:

**ROLLBACK [TRANSACTION]**




# ABKR felépítése

folytonos vonal - vezérlésátadás, adatáramlással

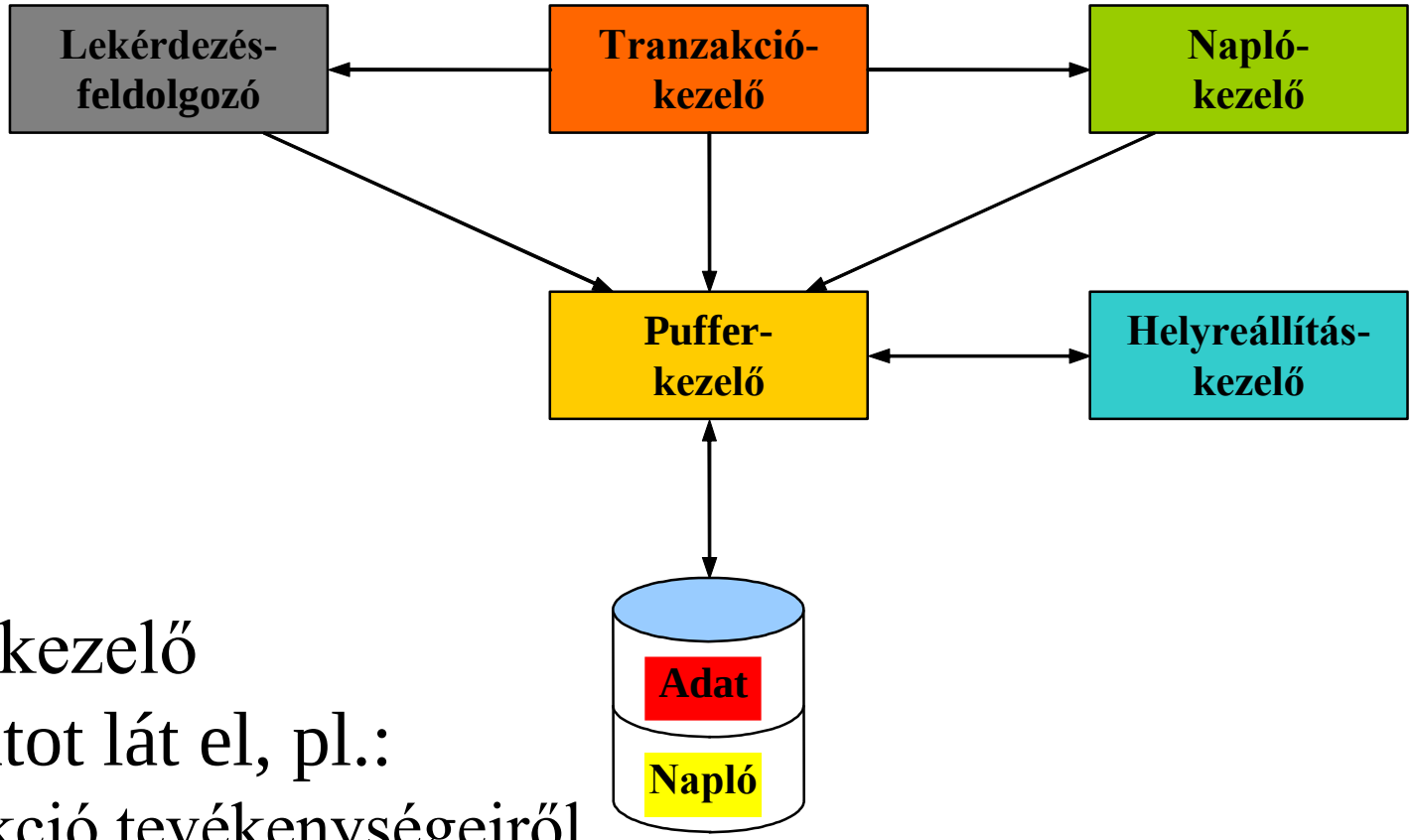
szaggatott vonal - csak adatátvitel

dupla doboz – memóriabeli adatszerkeszetek

# Tranzakciókezelő

- Külön modul az ABKR-en belül.
- Feladata = a tranzakciók helyes végrehajtásának biztosítása.
- A **tranzakciókezelőt** két fő részre osztjuk:
  - ***Konkurenciavezérlés-kezelő*** vagy ***ütemező* (scheduler)**: a tranzakciók elkülönítésének és atomosságának biztosításáért felelős.
  - ***Naplózás- és helyreállítás-kezelő*:** a tranzakciók atomosságáért és tartósságáért felelős.

# Tranzakciókezelő és az ABKR más komponenseinek kapcsolata

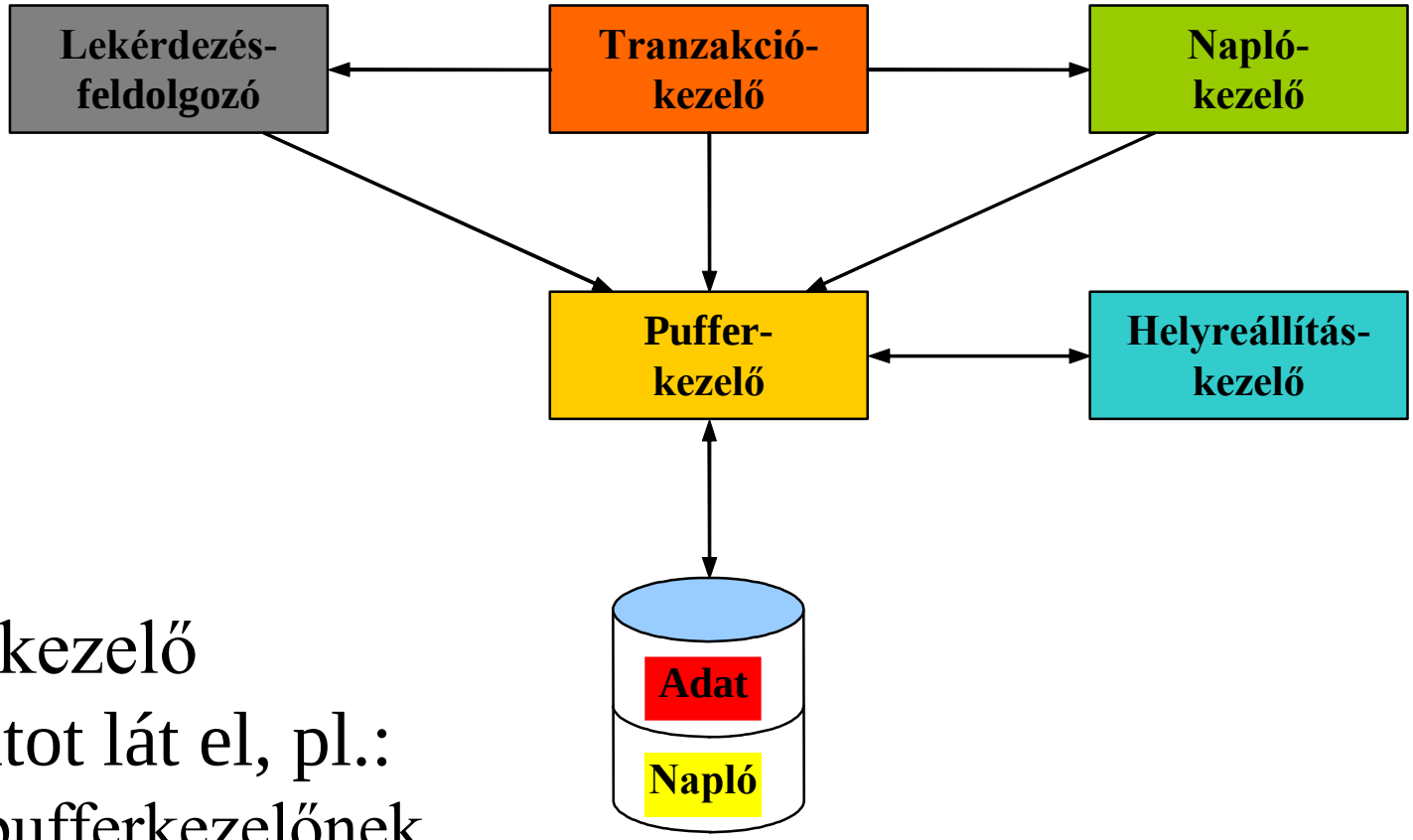



- A tranzakciókezelő

egy sor feladatot lát el, pl.:

- a tranzakció tevékenységeiről üzeneteket küld a naplókezelőnek

# Tranzakciókezelő és az ABKR más komponenseinek kapcsolata



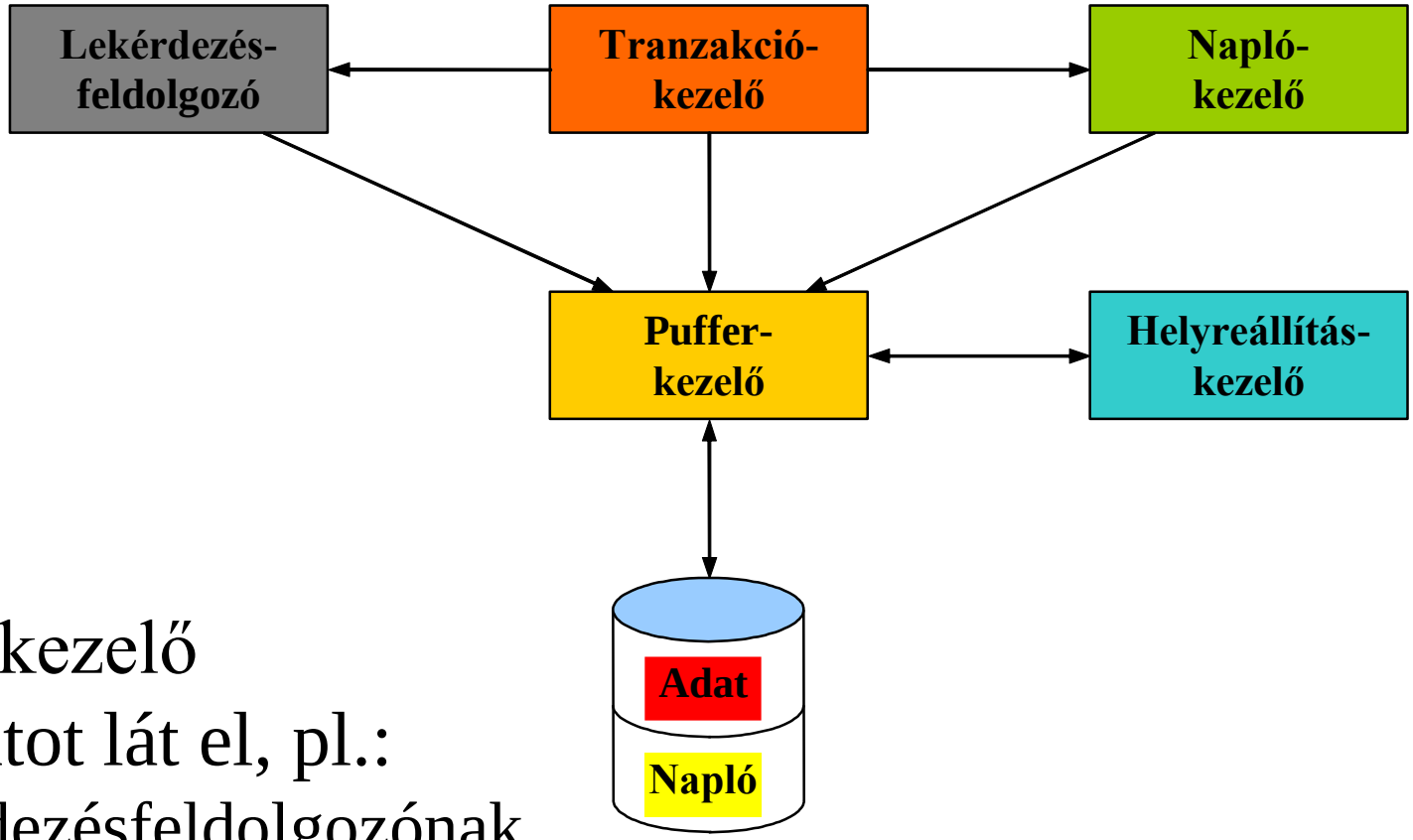

- A tranzakciókezelő

egy sor feladatot lát el, pl.:

- üzen a pufferkezelőnek

arra vonatkozóan, hogy a pufferek tartalmát szabad-e vagy kell-e lemezre másolni,

# Tranzakciókezelő és az ABKR más komponenseinek kapcsolata

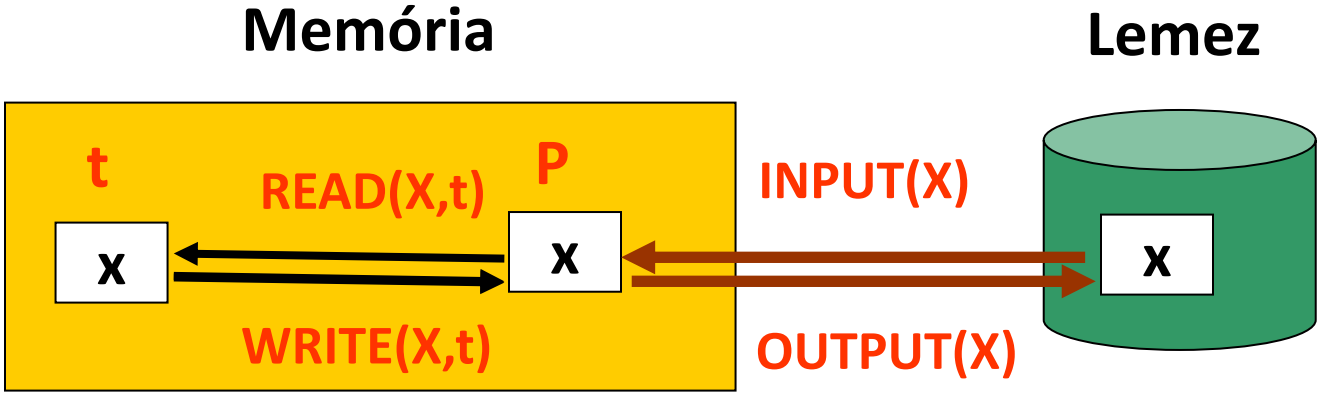



- A tranzakciókezelő

egy sor feladatot lát el, pl.:

  - üzen a lekérdezésfeldolgozónak arról, hogy a tranzakcióban előírt lekérdezéseket vagy más adatbázis-műveleteket kell végrehajtania.

# Tranzakciók alaptevékenységei




## Adatmózgások alapműveletei:

- INPUT (X)
- READ (X, *t*)
- WRITE (X, *t*)
- OUTPUT (X)

# Adatmozgások alapműveletei

Konkurencia tanulmányozása esetén – olvasási-/írási műveletek helyszíne: központi memória pufferei (NEM a lemez).

- INPUT (X): Az X adatbáziselemet tartalmazó lemezblokk másolása a pufferbe.
- READ (X, *t*): Az X adatbáziselem bemásolása a tranzakció *t* lokális változójába. Ha az X adatbáziselemet tartalmazó blokk nincs a pufferben, akkor előbb végrehajtódik INPUT (X), azután kapja meg *t* a változó X értékét.
- WRITE (X, *t*): A *t* lokális változó tartalma az X adatbáziselem pufferbeli tartalmába másolódik. Ha az X adatbáziselemet tartalmazó blokk nincs a pufferben, akkor előbb végrehajtódik az INPUT (X), azután másolódik át a *t* lokális változó értéke a pufferbeli X-be.
- OUTPUT (X): Az X adatbáziselemet tartalmazó puffer kimásolása lemezre.

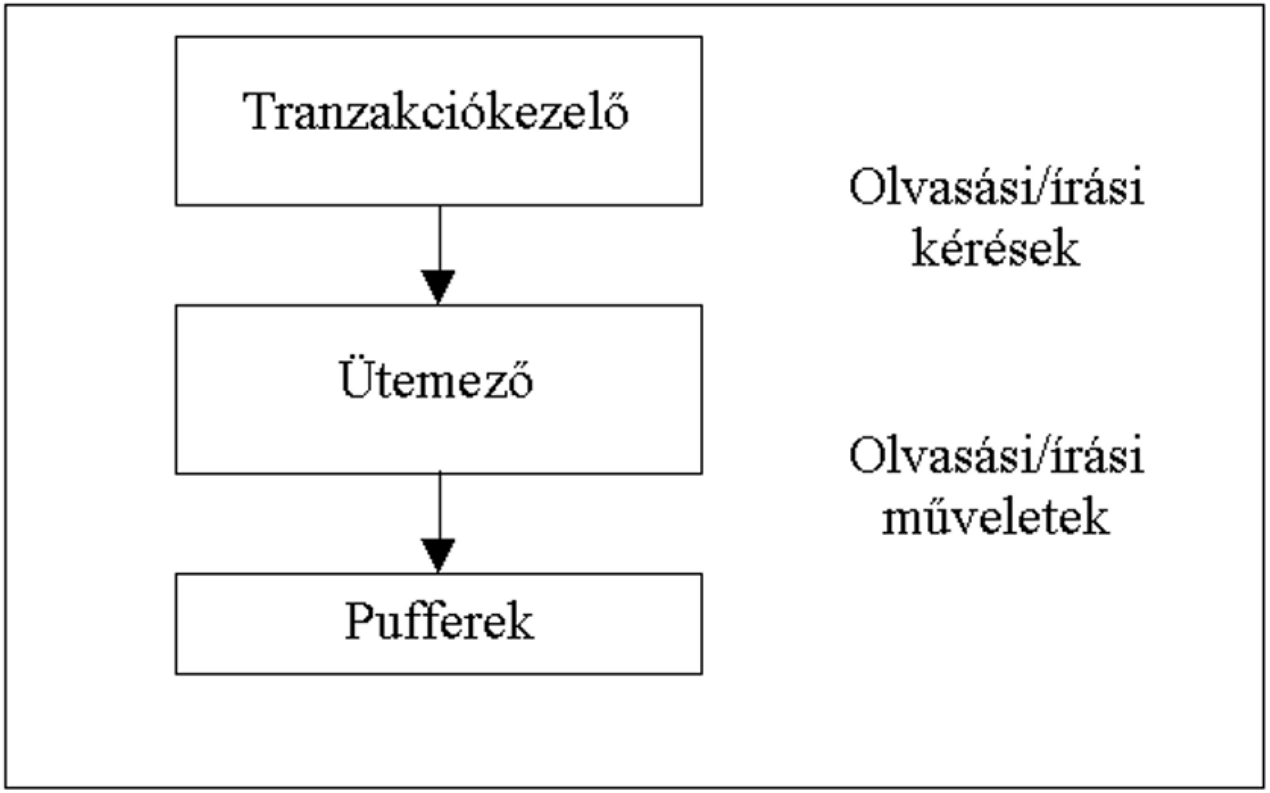
# Többfelhasználós működés

- Többfelhasználós működés ↔ Csak azokat az összefésülődéseit akarjuk megengedni a tranzakcióknak, amelyeknek a hatása ekvivalens valamelyik izolálttal.

# Többfelhasználós működés

- Többfelhasználós működés ↔ Csak azokat az összefésülődéseit akarjuk megengedni a tranzakcióknak, amelyeknek a hatása ekvivalens valamelyik izolálttal.

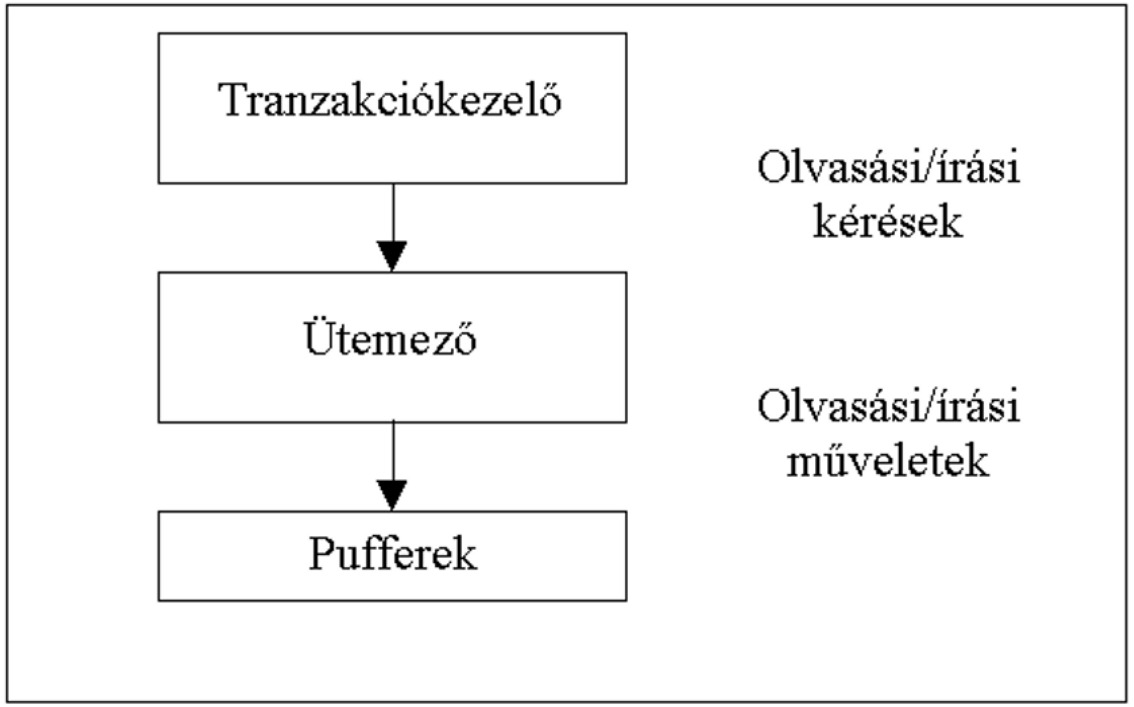
↔ **Ütemező (scheduler)** feladata: tranzakciós lépések szabályozása



# Többfelhasználós működés

- Többfelhasználós működés ↔ Csak azokat az összefésülődéseit akarjuk megengedni a tranzakcióknak, amelyeknek a hatása ekvivalens valamelyik izolálttal.

↔ **Ütemező (scheduler)** feladata: tranzakciós lépések szabályozása




**Konkurenciavezérlés** = az az általános folyamat, amely biztosítja, hogy a tranzakciók egyidejű végrehajtása során megőrizzék az adatbázis konzisztenciáját.

# Konkurenciavezérlés

- **tranzakció, mely nem véglegesített adatokat olvas „piszkos adat olvasása”(uncommited dependency)**
- **az elveszett módosítás (lost update) problémája**
- **helytelen analízis (inconsistent analysis)**

# Elveszett módosítás problémája

| A tranzakció | Idő | B tranzakció |
|---|---|---|
| READ (P, $t$) | $t_1$ | - |
| - | \| | - |
| - | $t_2$ | READ (P, $s$) |
| - | \| | - |
| $t := f(t)$ | $t_3$ | - |
| WRITE (P, $t$) | \| | - |
| - | \| | - |
| - | $t_4$ | $s := g(s)$ |
| - | \| | WRITE (P, $s$) |

# Elveszett módosítás problémája – példa 1

| A tranzakció | Idő | B tranzakció |
|---|---|---|
| READ (X, $t$) | $t_1$ | - |
| ($t = 5000$) | \| | - |
| - | $t_2$ | READ (X, $s$) |
| - | \| | ($s = 5000$) |
| $t := t + 500$ | $t_3$ | - |
| ($t = 5500$) | \| | - |
| WRITE (X, $t$) | $t_4$ | $s := s - 1000$ |
| - | \| | ($s = 4000$) |
| - | \| | WRITE (X, $s$) |

# Elveszett módosítás problémája – példa 2

| *A tranzakció* | *Idő* | *B tranzakció* |
|---|---|---|
| READ (Y, $v$) | $t_1$ | - |
| ($v = 1$) | \| | - |
| - | $t_2$ | READ (Y, $s$) |
| - | \| | ($s = 1$) |
| $v := v - 1$ | $t_3$ | - |
| ($v = 0$) | \| | - |
| WRITE (Y, $v$) | \| | - |
| - | $t_4$ | $s := s - 1$ |
| - | \| | ($s = 0$) |
| | \| | WRITE (Y, $s$) |

# „Piszkos adat” olvasása

**(Tranzakció, mely nem véglegesített adatokat olvas)**

| A tranzakció | Idő | B tranzakció |
|---|---|---|
| - | $t_1$ | READ (P, $s$) |
| - | \| | $s := g(s)$ |
| - | \| | WRITE (P, $s$) |
| - | \| | - |
| - | $t_2$ | - |
| READ (P, $t$) | \| | - |
| $t := f(t)$ | \| | - |
| WRITE (P, $t$) | \| | - |
| | $t_3$ | ROLLBACK |

| *A tranzakció* | *Idő* | *B tranzakció* |
|---|---|---|
| - | \| | - |
| *sum*: = 0 | $t_1$ | - |
| READ (BankSz1, *v*) | \| | - |
| *sum*: = *sum* + *v* | \| | - |
| (*sum* = 40) | \| | - |
| - | \| | - |
| READ (BankSz2, *v*) | $t_2$ | - |
| *sum*: = *sum* + *v* | \| | - |
| (*sum* = 90) | \| | - |
| - | \| | - |
| - | $t_3$ | READ (BankSz3, *s*) |
| - | \| | *s*: = *s* –10 |
| - | \| | (s = 20) |
| - | \| | WRITE (BankSz3, *s*) |
| - | \| | - |
| - | $t_4$ | READ (BankSz1, *s*) |
| - | \| | *s*: = *s* +10 |
| - | \| | (s = 50) |
| - | \| | WRITE (BankSz1, *s*) |
| - | \| | - |
| - | $t_5$ | COMMIT |
| - | \| | - |
| READ (BankSz3, *v*) | $t_6$ | - |
| *sum*: = *sum* + *v* | \| | - |
| (*sum* = 110) | \| | - |

# Inkonzisztens analízis problémája

BankSz1 = 40
BankSz2 = 50
BankSz3 = 30

- ***A*** tranzakció összegzi 3 különböző bankszámlán levő összegeket;
- ***B*** tranzakció áttesz 10 egységet a 3-ik számláról az első bankszámlára.

# Többfelhasználós működés, alapfogalmak

- **ütemezés:** egy vagy több tranzakció műveleteinek valamilyen sorozata (fontos, hogy a tranzakciókon belüli sorrend megmarad)
- **soros ütemezés:** olyan ütemezés, amikor a különböző tranzakciók utasításai nem keverednek, először lefut az egyik összes utasítása, aztán a másiké, aztán a harmadiké, . . .

# Soros ütemezés

Legyen $T_1$ a következő tranzakció:

BEGIN TRANSACTION

$X := X - 10$;

$Y := Y + 10$;

COMMIT TRANSACTION

$T_2$ pedig:

BEGIN TRANSACTION

$X := X * 2$;

$Y := Y * 2$;

COMMIT TRANSACTION

# Példa soros ütemezésre (1)

| $T_1$ | $T_2$ | t | Mem-$X$ | Mem-$Y$ |
|---|---|---|---|---|
| READ($X$, t) | | 50 | 50 | |
| t: = t – 10 | | 40 | 50 | |
| WRITE($X$,t) | | 40 | 40 | |
| READ($Y$,t) | | 20 | 40 | 20 |
| t:= t + 10 | | 30 | 40 | 20 |
| WRITE($Y$, t) | | 30 | 40 | 30 |
| | READ($X$, t) | 40 | 40 | 30 |
| | t: = t * 2 | 80 | 40 | 30 |
| | WRITE($X$,t) | 80 | 80 | 30 |
| | READ( $Y$,t) | 30 | 80 | 30 |
| | t:= t * 2 | 60 | 80 | 30 |
| | WRITE ($Y$, t) | 60 | 80 | 60 |

$T_1$ tranzakció megelőzi $T_2$-t; **jel.: ($T_1$, $T_2$)**

# Példa soros ütemezésre (2)

| $T_1$ | $T_2$ | t | Mem-$X$ | Mem-$Y$ |
|---|---|---|---|---|
| | READ($X$, t) | 50 | 50 | |
| | t: = t * 2 | 100 | 50 | |
| | WRITE($X$,t) | 100 | 100 | |
| | READ( $Y$,t) | 20 | 100 | 20 |
| | t:= t * 2 | 40 | 100 | 20 |
| | WRITE ($Y$, t) | 40 | 100 | 40 |
| READ($X$, t) | | 100 | 100 | 40 |
| t: = t - 10 | | 90 | 100 | 40 |
| WRITE($X$,t) | | 90 | 90 | 40 |
| READ( $Y$,t) | | 40 | 90 | 40 |
| t:= t + 10 | | 50 | 90 | 40 |
| WRITE ($Y$, t) | | 50 | 90 | 50 |

$T_2$ tranzakció megelőzi $T_1$-t; **jel.: ($T_2$, $T_1$)**

# Sorba rendezhető ütemezések

**Ütemező (scheduler)** feladata: *olyan ütemezések biztosítása, melyek megőrzik az adatbázis konzisztenciáját.*

- Egy tranzakció megőrzi az adatbázis helyességét.
  - Egymás után, soros ütemezéssel végrehajtott tranzakciók is megőrzik az adatbázis helyességét.
- **Sorba rendezhető (sorosítható) ütemezés:** olyan ütemezés, amelynek hatása azonos a résztvevő tranzakciók valamely(!!!) soros ütemezésének hatásával (azaz a végén minden érintett adatelem pont úgy néz ki, mint a soros ütemezés után).

# Sorba rendezhető ütemezések

- **Sorba rendezhető (sorosítható) ütemezés:** olyan ütemezés, amelynek hatása azonos a résztvevő tranzakciók valamely soros ütemezésének hatásával (azaz a végén minden érintett adatelem pont úgy néz ki, mint a soros ütemezés után).
- A sorba rendezhető (serializable schedule) ütemezés is biztosítja az adatbázis konzisztenciájának megmaradását.

# Példa sorba rendezhető ütemezésre

| $T_1$ | $T_2$ | t | Mem-$X$ | Mem-$Y$ |
|---|---|---|---|---|
| READ($X$, t) | | 50 | 50 | |
| t: = t – 10 | | 40 | 50 | |
| WRITE($X$,t) | | 40 | 40 | |
| | READ($X$, t) | 40 | 40 | |
| | t: = t * 2 | 80 | 40 | |
| | WRITE($X$,t) | 80 | 80 | |
| READ( $Y$,t) | | 20 | 80 | 20 |
| t:= t + 10 | | 30 | 80 | 20 |
| WRITE ($Y$, t) | | 30 | 80 | 30 |
| | READ( $Y$,t) | 30 | 80 | 30 |
| | t:= t * 2 | 60 | 80 | 30 |
| | WRITE ($Y$, t) | 60 | 80 | 60 |

A ($T_1$, $T_2$) soros ütemezésnek, egy sorba rendezhető ütemezése ↔ Hatása megegyezik a ($T_1$, $T_2$) soros ütemezés hatásával.

# Példa NEM sorba rendezhető ütemezésre

| $T_1$ | $T_2$ | t | Mem-$X$ | Mem-$Y$ |
|---|---|---|---|---|
| READ($X$, t) | | 50 | 50 | |
| t: = t – 10 | | 40 | 50 | |
| WRITE($X$,t) | | 40 | 40 | |
| | READ($X$, t) | 40 | 40 | |
| | t: = t * 2 | 80 | 40 | |
| | WRITE($X$,t) | 80 | 80 | |
| | READ( $Y$,t) | 20 | 80 | 20 |
| | t:= t * 2 | 40 | 80 | 20 |
| | WRITE ($Y$, t) | 40 | 80 | 40 |
| READ( $Y$,t) | | 40 | 80 | 40 |
| t:= t + 10 | | 50 | 80 | 40 |
| WRITE ($Y$, t) | | 50 | 80 | 50 |

A ($T_1$, $T_2$) soros ütemezésnek, egy NEM sorba rendezhető ütemezése

Hatása nem lehet a soros ütemezéssel megegyező.

# Sorba rendezhető ütemezések

- Egy tranzakció megőrzi az adatbázis helyességét.
  - Egymás után, soros ütemezéssel végrehajtott tranzakciók is megőrzik az adatbázis helyességét.
  - A sorba rendezhető (serializable schedule) ütemezés biztosítja még az adatbázis konzisztenciájának a megmaradását.
- **Sorba rendezhető (sorosítható) ütemezés:** olyan ütemezés, amelynek hatása azonos a résztvevő tranzakciók valamely soros ütemezésének hatásával (azaz a végén minden érintett adatelem pont úgy néz ki, mint a soros ütemezés után).

# Konfliktusok

- **Konfliktus:** olyan egymást követő műveletpár az ütemezésben, amelynek ha a sorrendjét felcseréljük, akkor legalább az egyik tranzakció viselkedése megváltozhat.

# Konfliktusok

- Ugyanannak a tranzakciónak két művelete konfliktus.
  - pl: $r_i(X)$; $w_i(Y)$.
  - *Egy tranzación belül a műveletek sorrendje rögzített, az ABKR ezt a sorrendet nem rendezheti át.*
- Különböző tranzakciók ugyanarra az adatbáziselemre vonatkozó írása konfliktus.
  - Pl. $w_i(X)$; $w_j(X)$ .
  - Ebben a sorrendben $X$ értéke az marad, melyet $T_j$ ír, fordított sorrendben pedig az marad, melyet $T_i$ ír. Ezek az értékek pedig nagy valószínűséggel különbözőek.

# Konfliktusok

- Különböző tranzacióknak ugyanabból az adatbáziselemből való olvasása és írása konfliktus.
  - Pl. $r_i(X)$; $w_j(X)$ konfliktus, mivel ha felcseréljük a sorrendet, akkor a $T_i$ által olvasott $X$-beli érték az lesz, melyet a $T_j$ ír és az nagy valószínűséggel nem egyezik meg az $X$ korábbi értékével.
  - Hasonlóan a $w_i(X)$; $r_j(X)$ is konfliktus.

# Az ütemező eszközei a sorosíthatóság elérésére

- Az ütemező lehetőségei a sorosítható ütemezések kikényszerítésére:
  - **zárak** (ezen belül is még: protokoll elemek, pl. 2PL)
  - **időbélyegek** (time stamp)
  - érvényesítés
- Irányelv: *Inkább legyen szigorú és ne hagyjon lefutni egy sorosítható ütemezést, mintsem hagyjon lefutni egy nem sorosíthatót.*

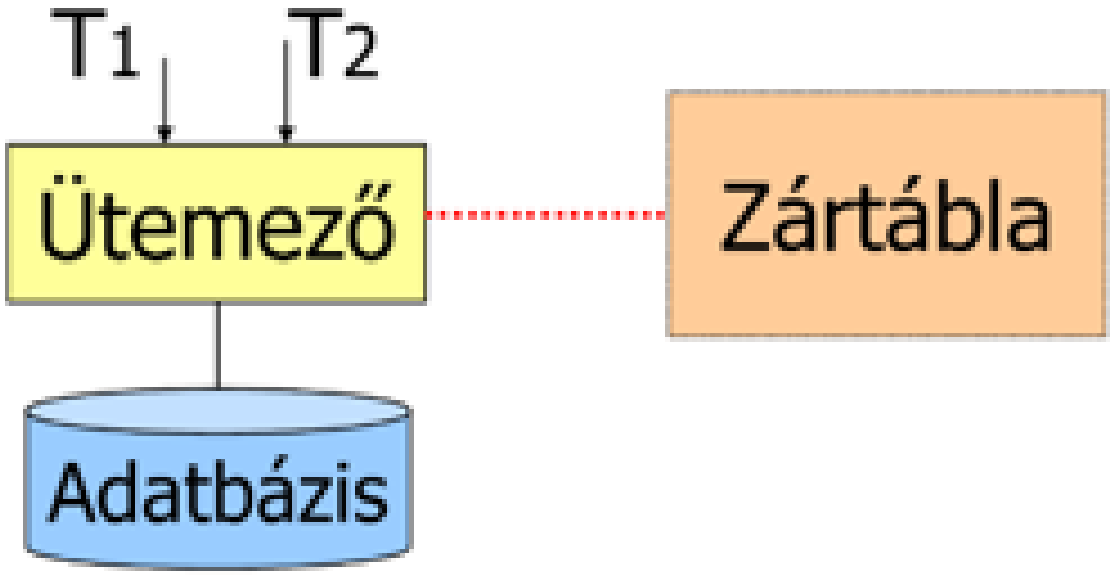
# Zárolási módszerek

- A tranzakció használat idejére lefoglalja az objektumot
- A többi tranzakció korlátozva van az objektum elérésében ↔ várakozás
- Várakozás feloldása:
  - az objektum elérhetővé vált;
  - kiderül, hogy nem érdemes várni.
- Nyilván kell tartani objektumonként kiegészítő információkat:
  - szabad-e;
  - ki foglalja (felszabadításnál tudni kell).
- Zárolás ↔ helytöbblettel jár

# Sorbarendezhetőség biztosítása zárakkal

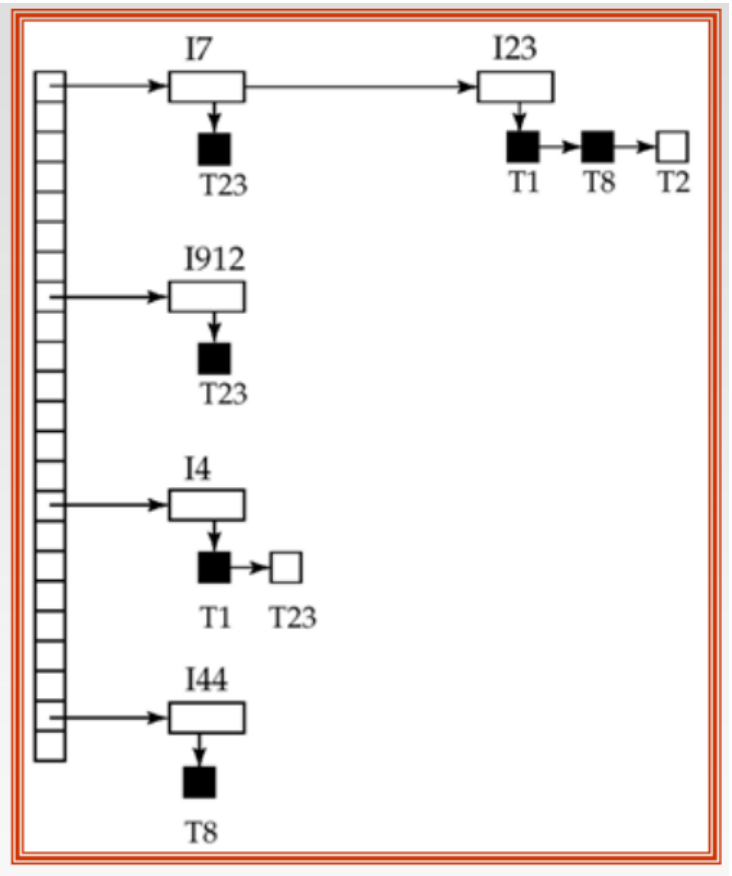
A tranzakciók kérései

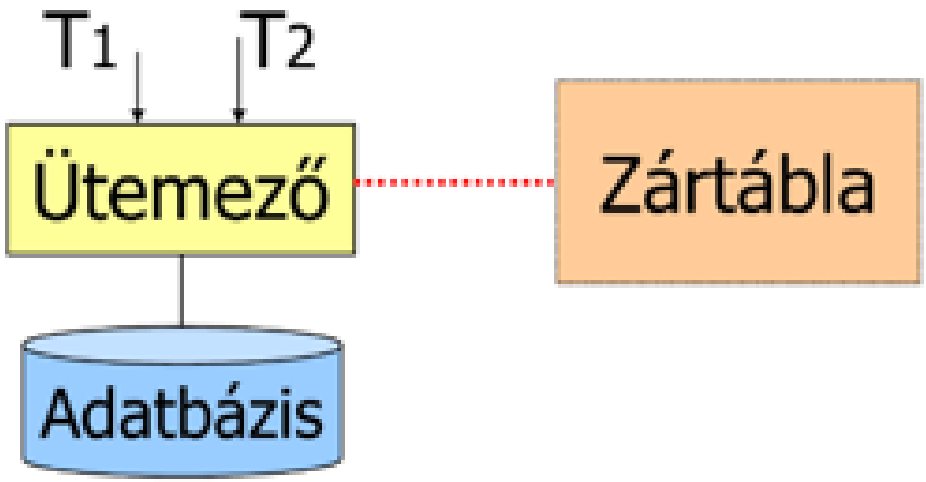
A műveletek sorba rendezhető ütemezése




- A tranzakciók zárolni fogják azokat az adatbáziselemeket, amelyekhez hozzáférnek. ↔ nem sorbarendezhetőség kockázatának kikerülése (más tranzakciók hozzáférésének megakadályozása ezen adatbáziselemekhez).

# Sorbarendezhetőség biztosítása zárakkal




Példa zártáblára

# Jelölések

$r_i(X)$ - a $T_i$ tranzakció olvassa az $X$ adatbáziselemet (r - read)

$w_i(X)$ - a $T_i$ tranzakció írja az $X$ adatbáziselemet (w -write)

$sl_i(X)$ - a $T_i$ tranzakció osztott zárat kér az $X$-re

$xl_i(X)$ – a $T_i$ tranzakció kizárólagos zárat kér az $X$-re

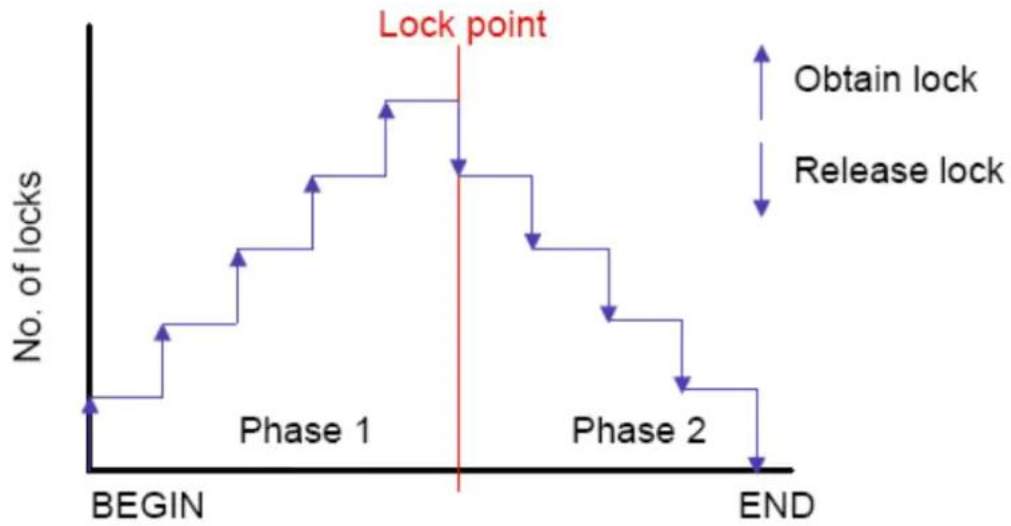
$u_i(X)$ - a $T_i$ tranzakció feloldja (unlock) $X$ adatbáziselemen tartott zárát.

# 2PL

- **Kétfázisú lezárás tétel**: Ha minden tranzakció betartja a kétfázisú lezárási protokollt, az összes lehetséges ütemezés sorbarendezhető.
- **Kétfázisú lezárási protokoll**: A zárolásoknak meg kell előzniük a zárak feloldását:

  **1.fázis** („növekedési fázis”): zárolások lefoglalása

  **2.fázis** („csökkenő fázis”): 1. fázisbeli zárolások felengedése



# Sorbarendezhetőség biztosítása zárakkal

- A tranzakciók kérhetnek különböző zárakat.
  - Ha az ütemező nem tudja megadni a kért zárat, a tranzakciók egy *első-beérkezett-első-kiszolgálása* (first-come-first-served) várási listába kerülnek, míg a szükséges adat felszabadul.

# Sorbarendezhetőség biztosítása zárakkal

Követelmények minden $T_i$ tranzakció esetén (a tranzakciók konzisztenciájának megőrzése érdekében):

- Az $r_i(X)$ olvasási műveletet **meg kell előzze** egy $sl_i(X)$ vagy $xl_i(X)$ úgy, hogy közben nincs $u_i(X)$
- A $w_i(X)$ olvasási műveletet **meg kell előzze** egy $xl_i(X)$ úgy, hogy közben nincs $u_i(X)$
- 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 $u_i(X)$.
- Ha $sl_i(X)$ szerepel egy ütemezésben, akkor ezután nem következhet $xl_j(X)$ $i$-től különböző $j$-re anélkül, hogy közben ne szerepelne $u_i(X)$.

# Sorbarendezhetőség biztosítása zárakkal

- lezárható az egész adatbázis - annak, akinek sikerült lezárnia, könnyű dolga van a programozás szempontjából, de senki más nem férhet hozzá;
- lezárható egy rekord, általában ez a legkézenfekvőbb megoldás
- lezárható egy rekordcsoport (egy lemezblokkban lévő rekordok vagy fa szerkezetű indexnek megfelelő rekordcsoport).

# Kompatibilitási mátrix

- Kompatibilitási mátrix - ütemező ez által dönti el, hogy egy ütemezés/zárkérés legális-e vagy sem + várakoztatnia kell-e a tranzakciókat vagy sem.

# Kompatibilitási mátrix

- Kompatibilitási mátrix - ütemező ez által dönti el, hogy egy ütemezés/zárkérés legális-e vagy sem + várakoztatnia kell-e a tranzakciókat vagy sem.
- Minden zármódhoz: 1-1 sor + 1-1 oszlop
  - Sorok – egy másik tranzakció által az A adatbáziselemre elhelyezett zárak
  - Oszlopok – egy másik tranzakció által az A adatbáziselemre kért zárak

| Érvényes zár A-n | Megkapható-e az adott típusú zár A-ra? sl | xl |
|---|---|---|
| sl | igen | nem |
| xl | nem | nem |

Osztott (sl) és kizárólagos (xl) zárak kompatibilitási mátrixa

# Zárak felminősítése

- L2 zár erősebb L1-nél, ha: a kompatibilitási mátrixban L2 sorában/oszlopában minden olyan pozícióban „nem” szerepel, ahol L1 sorában/oszlopában is „nem” áll.
  - Pl. osztott és kizárólagos zárak használata esetén xl erősebb sl-nél.
- T tranzakció felminősíti az L1 zárját egy erősebb L2 zárra az A adatbáziselemen, ha:
  - L1-et még birtokolja A-n (még nem oldotta fel);
  - L2 zárat kéri és meg is kapja A-ra.
  - Pl. Ha egy tranzakciónak már van osztott zára egy adatbáziselemen, melyet módosítani akar, felminősítheti a zárat kizárólagossá.

| | sl | xl |
|---|---|---|
| sl | igen | nem |
| xl | nem | nem |

# Zárolások gyenge pontjai

- Zárolások biztos megoldást nyújtanak a helyes ütemezés megvalósítására.
- Gyenge pont:
  - tranzakciók várakozásra kényszerítése
  - többen várakoznak - körbevárakozás alakulhat ki
    - végtelen várakozás

# Zárolások gyenge pontjai

- Zárolások biztos megoldást nyújtanak a helyes ütemezés megvalósítására.
- Gyenge pont:
  - tranzakciók várakozásra kényszerítése
  - többen várakoznak - körbevárakozás alakulhat ki
    - végtelen várakozás ↔ **holtpont**

# Holtpont

- **Holtpont (deadlock) -** olyan állapot, mikor két vagy több tranzakció várási állapotban van, mindegyik vár a másik által lezárt objektumra *(vár egy olyan zár elengedésére, amit egy másik, ebbe a részhalmazba tartozó, tranzakció tart).*

*Megj.* Általában nem jelenik meg kettőnél több tranzakció holtpontban.

# Holtpont

- **Holtpont (deadlock) -** olyan állapot, mikor két vagy több tranzakció várási állapotban van, mindegyik vár a másik által lezárt objektumra *(vár egy olyan zár elengedésére, amit egy másik, ebbe a részhalmazba tartozó, tranzakció tart).*

*Megj.* Általában nem jelenik meg kettőnél több tranzakció holtpontban.

- **Példa:** Legyenek $T_1$, $T_2$, $T_3$ tranzakciók.

Jel. $l_i(U)$-$T_i$ tranzakció zárolta U-t; $i \in \{1,2,3\}$; $U \in \{X,Y,Z\}$

$l_1(X)$; $l_2(Y)$; $l_3(Z)$; $l_1(Z)$; $l_2(X)$; $l_3(Y)$

*Ezen sorrendű zárkérések esetén egyik tranzakció sem tud tovább futni.*

# Elveszett módosítás problémája

| Tranzakció 1 | Idő | Tranzakció 2 |
|---|---|---|
| $sl_1(P)$; $r_1(P)$ | $t_1$ | - |
| - | \| | - |
| - | $t_2$ | $sl_2(P)$; $r_2(P)$ |
| - | \| | - |
| $xl_1(P)$ **vár** | $t_3$ | - |
| - | \| | - |
| - | $t_4$ | $xl_2(P)$ **vár** |
| - holtpont | \| | - holtpont |



Probléma megoldási kísérlete osztott és kizárólagos zárakkal

# Tranzakció, mely nem véglegesített adatokat olvas

| Tranzakció 1 | Idő | Tranzakció 2 |
|---|---|---|
| - | $t_1$ | $xl_2(P)$; $r_2(P)$; $w_2(P)$; |
| - | \| | $u_2(P)$ |
| - | \| | - |
| $xl_1(P)$; $r_1(P)$; $w_1(P)$; | $t_2$ | - |
| $u_1(P)$ | \| | - |
| | \| | - |
| | \| | - |
| | $t_3$ | ROLLBACK |



Probléma megoldási kísérlete osztott és kizárólagos zárakkal

# Az inkonzisztens analízis problémája

| Tranzakció 1 | Idő | Tranzakció 2 |
|---|---|---|
| - | \| | - |
| $sl_1$(BankSz1); $r_1$(BankSz1); | $t_1$ | - |
| - | \| | - |
| $sl_1$(BankSz2); $r_1$(BankSz2); | $t_2$ | - |
| - | \| | - |
| - | $t_3$ | $xl_2$(BankSz3); $r_2$(BankSz3);$w_2$(BankSz3) |
| - | \| | - |
| - | $t_4$ | $xl_2$(BankSz1); **vár** |
| - | \| | - |
| $sl_1$ (BankSz3); **vár** | $t_5$ | - |
| holtpont | \| | holtpont |



Probléma megoldási kísérlete osztott és kizárólagos zárakkal

# Holtpont feloldása

- **Holtpont feloldása -** tranzakciókezelő feladata: közbeavatkozás egy (vagy több) tranzakció abortálásával úgy, hogy a többi tranzakciót folytatni lehessen.

- **Holtpont feloldásának lehetőségei:**
  - módosítási zárak
  - várakozási gráf (Wait-For-Graph)
  - időkorlát mehanizmus

# Módosítási zárak

- **$ul_i(X)$ - módosítási zár** (update lock)
  - jog X adatbáziselem olvasására $T_i$ tranzakció által, de írására NEM
  - Kizárólagos zárrá felminősíthető zár(ak): módosítási zár, osztott zár NEM

    => Ha $T_i$ szándéka valamikor (!) a végrehajtása során módosítani X-et: olvasáskor $ul_i(X)$ zár kérése

# Módosítási zárak

- **$ul_i(X)$ - módosítási zár** (update lock)
  - a $T_i$ tranzakciónak csak X adatbáziselem olvasására ad jogot, az X írására nem
  - csak a módosítási zárat lehet később felminősíteni, az olvasásit nem. => Ha egy tranzakciónak szándékában áll módosítani az X adatbáziselemet, akkor módosítási zárat kér rá.

Megkapható-e az adott típusú zár A-ra?

| Érvényes zár A-n | sl | xl | ul |
|---|---|---|---|
| sl | igen | nem | igen |
| xl | nem | nem | nem |
| ul | nem | nem | nem |

Osztott(*sl*), kizárólagos (*xl*) és módosítási (*ul*) zárak kompatibilitási mátrixa

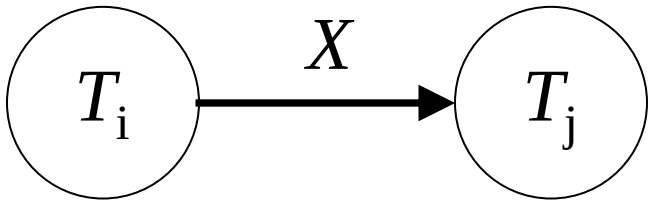
# Elveszett módosítás problémája

| Tranzakció 1 | Idő | Tranzakció 2 |
|---|---|---|
| $ul_1(P)$; $r_1(P)$ | $t_1$ | - |
| - | \| | - |
| - | $t_2$ | $ul_2(P)$ **vár** |
| - | \| | - |
| $xl_1(P)$; $w_1(P)$; $u_1(P)$ | $t_3$ | - |
| - | \| | - |
| - | $t_4$ | $ul_2(P)$; $r_2(P)$ |
| | \| | $xl_2(P)$; $w_2(P)$; $u_2(P)$ |

Probléma megoldása osztott, módosítási és kizárólagos zárak használatával: 2-es tranzakció késleltetve van, míg 1-es felszabadítja a zárat.

# Holtpont felismerése várakozási gráf segítségével

- Várakozási gráf - holtpont felismerésében nyújt segítséget.
- Gráf felépítése:
  - csúcsok – tranzakciók
  - élek:
    - $T_i$ és $T_j$ csúcs között élet rajzolunk, ha:
      - $T_j$ lezárva tartja az $X$ adatbáziselemet;
      - $T_i$ kéri az $X$ adatbáziselemet, hogy zárolhassa azt.
      - $T_i$ csak akkor kapja meg X zárját, ha $T_j$ lemond róla
    - Él irányítása: $T_i$-től $T_j$ felé.
    - Él címkéje: $X$.



# Holtpont felismerése várakozási gráf segítségével

- **Példa (visszacsatolás):** Legyenek $T_1$, $T_2$, $T_3$ tranzakciók.

Jel. $l_i(U)$-$T_i$ tranzakció zárolta U-t; $i \in \{1,2,3\}$; $U \in \{X,Y,Z\}$

$l_1(X)$; $l_2(Y)$; $l_3(Z)$; $l_1(Z)$; $l_2(X)$; $l_3(Y)$

*Ezen sorrendű zárkérések esetén*

*egyik tranzakció sem tud tovább futni.*

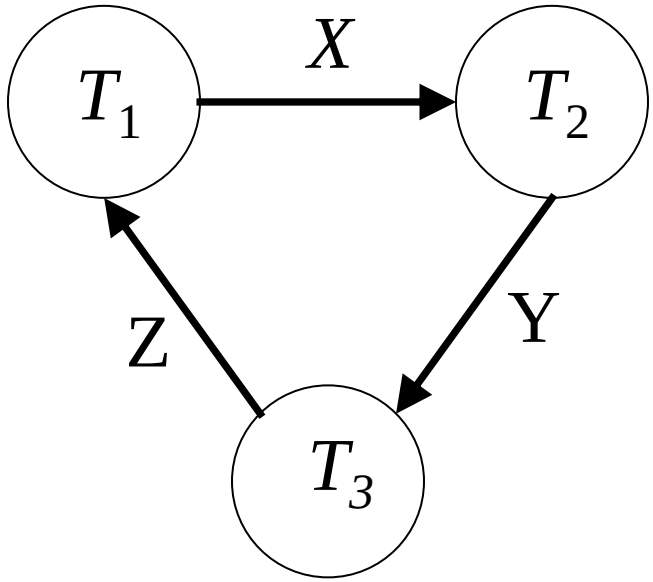
# Holtpont felismerése várakozási gráf segítségével

- **Példa (visszacsatolás):** Legyenek $T_1$, $T_2$, $T_3$ tranzakciók.

Jel. $l_i(U)$-$T_i$ tranzakció zárolta U-t; i $\in\{1,2,3\}$; U $\in\{X,Y,Z\}$

$l_1(X)$; $l_2(Y)$; $l_3(Z)$; $l_1(Z)$; $l_2(X)$; $l_3(Y)$

*Ezen sorrendű zárkérések esetén egyik tranzakció sem tud tovább futni.*




Megfelelő várakozási gráf

# Holtpont felismerése várakozási gráf segítségével

- **Példa (visszacsatolás):** Legyenek $T_1$, $T_2$, $T_3$ tranzakciók.

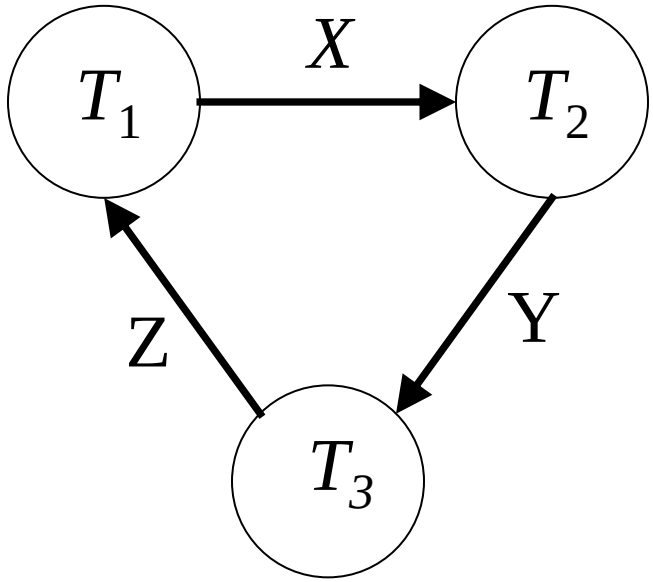
Jel. $l_i(U)$-$T_i$ tranzakció zárolta U-t; i $\in\{1,2,3\}$; U $\in\{X,Y,Z\}$

$l_1(X)$; $l_2(Y)$; $l_3(Z)$; $l_1(Z)$; $l_2(X)$; $l_3(Y)$

*Ezen sorrendű zárkérések esetén egyik tranzakció sem tud tovább futni.*

A gráf tartalmaz **ciklust**

→ van holtpont.




Megfelelő várakozási gráf

# További példák várakozási gráfra

Ütemezés:

w1(A), w2(B), w3(C), w4(C), w1(B), w2(A), w3(B)

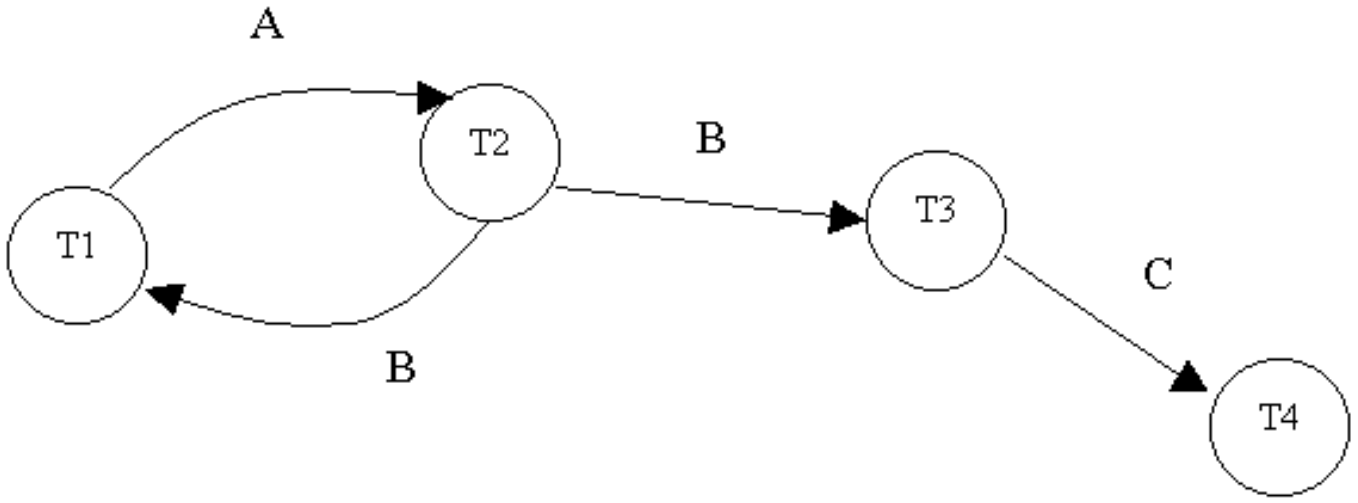
| Idő | $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|---|
| $t_1$ | $xl_1(A)$; $w_1(A)$; | | | |
| $t_2$ | | $xl_2(B)$; $w_2(B)$; | | |
| $t_3$ | | | $xl_3(C)$; $w_3(C)$; | |
| $t_4$ | | | | $xl_4(C)$; **vár** |
| $t_5$ | $xl_1(B)$; **vár** | | | |
| $t_6$ | | $xl_2(A)$; **vár** | | |
| $t_7$ | | | $xl_3(B)$; **vár** | |

# További példák várakozási gráfra

Ütemezés:

w1(A), w2(B), w3(C), w4(C), w1(B), w2(A), w3(B)

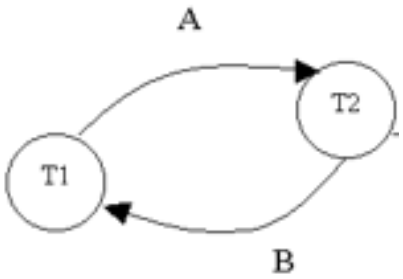
Ütemezésnek megfelelő várakozási gráf:




↔ holtpont alakult ki

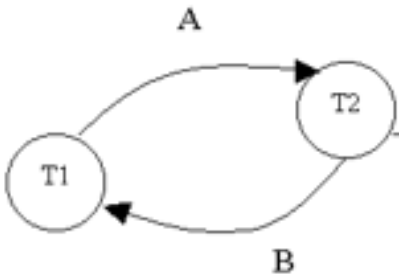
# Megoldások holtpont ellen

- Várakozási gráf folyamatos ábrázolása ABKR által.
  - A gráf tartalmaz ciklust → van holtpont.
- Körben szereplő tranzakciók valamelyikének megszakítása és visszapörgetése.
  - A rendszer ki kell derítse, hogy holtpont-probléma áll fenn.



# Megoldások holtpont ellen

- Várakozási gráf folyamatos ábrázolása ABKR által.
  - A gráf tartalmaz ciklust → van holtpont.
- Körben szereplő tranzakciók valamelyikének megszakítása és visszapörgetése.



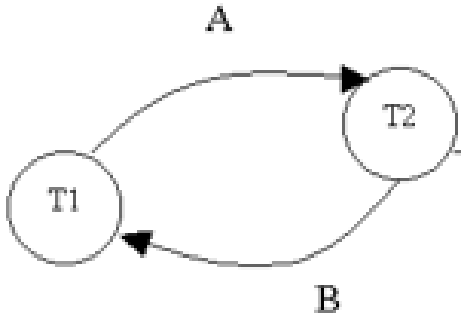

  - A rendszer ki kell derítse, hogy holtpont-probléma áll fenn.

- Tranzakció megszakítása:
  - holtpontban szereplő tranzakciók egyikének kiválasztása ↔ áldozat tranzakció (victim)
  - áldozat tranzakció visszapörgetése

  ⇒Tranzakció által kiadott zárak felszabadulása

  ⇒ Többi tranzakció folytatásának lehetősége

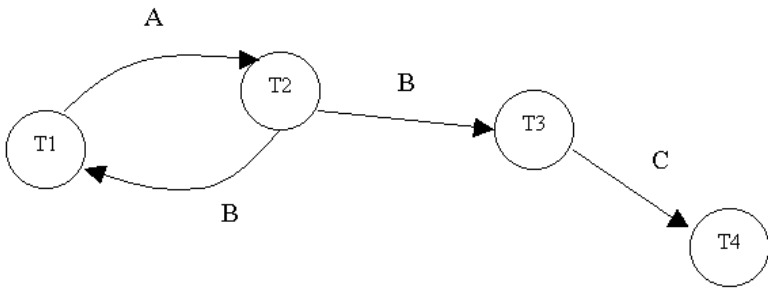
# Megoldások holtpont ellen



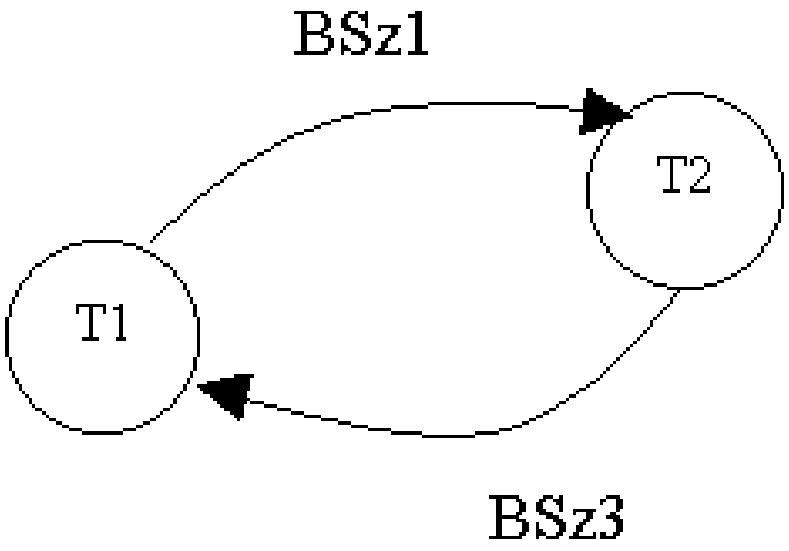

- Hogyan választjuk ki, hogy mely tranzakciót pörgessük vissza? Figyelembe vesszük:
  - tranzakció mennyi ideje fut már;
  - hány adatbáziselemet módosított eddig;
  - hány adatbáziselemet módosítana ezután.
- Az áldozat kiválasztásakor figyelembe kell venni, hogy ne mindig ugyanazt a tranzakciót válasszuk deadlock victim-nek.

# Példa - holtpont megoldása

| Idő | $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|---|
| $t_1$ | $xl_1(A)$; $w_1(A)$; | | | |
| $t_2$ | | $xl_2(B)$; $w_2(B)$; | | |
| $t_3$ | | | $xl_3(C)$; $w_3(C)$; | |
| $t_4$ | | | | $xl_4(C)$; **vár** |
| $t_5$ | $xl_1(B)$; **vár** | | | |
| $t_6$ | | $xl_2(A)$; **vár** | | |
| $t_7$ | | | $xl_3(B)$; **vár** | |
| $t_8$ | ABORT $T_1$<br>$u_1(A)$ | | | |
| $t_9$ | | $xl_2(A)$; $w_2(A)$;<br>$u_2(B)$; $u_2(A)$ | | |
| $t_{10}$ | | | $xl_3(B)$; $w_3(B)$;<br>$u_3(C)$; $u_3(B)$ | |
| $t_{11}$ | | | | $xl_4(C)$; $w_4(C)$;<br>$u_4(C)$; |



# Inkonzisztens analízis problémája




Ábra: Inkonzisztens analízisnek megfelelő várási gráf

# Inkonzisztens analízis problémájának megoldása

| Tranzakció 1 | Idő | Tranzakció 2 |
|---|---|---|
| - | \| | - |
| $sl_1$(BankSz1); $r_1$(BankSz1); | $t_1$ | - |
| - | \| | - |
| $sl_1$(BankSz2); $r_1$(BankSz2); | $t_2$ | - |
| - | \| | - |
| - | $t_3$ | $xl_2$(BankSz3); $r_2$(BankSz3);$w_2$(BankSz3) |
| - | \| | - |
| - | $t_4$ | $xl_2$(BankSz1); **vár** |
| - | \| | - |
| $sl_1$ (BankSz3); **vár** | $t_5$ | - |
| **holtpont** | \| | **holtpont** |
| ABORT $T_1$ | $t_6$ | |
| $u_1$(BankSz1); $u_1$(BankSz2); | \| | |
| | $t_7$ | $xl_2$(BankSz1); $r_2$(BankSz1);$w_2$(BankSz1) |
| | \| | $u_2$(BankSz1); $u_2$(BankSz3); |
| | \| | COMMIT |

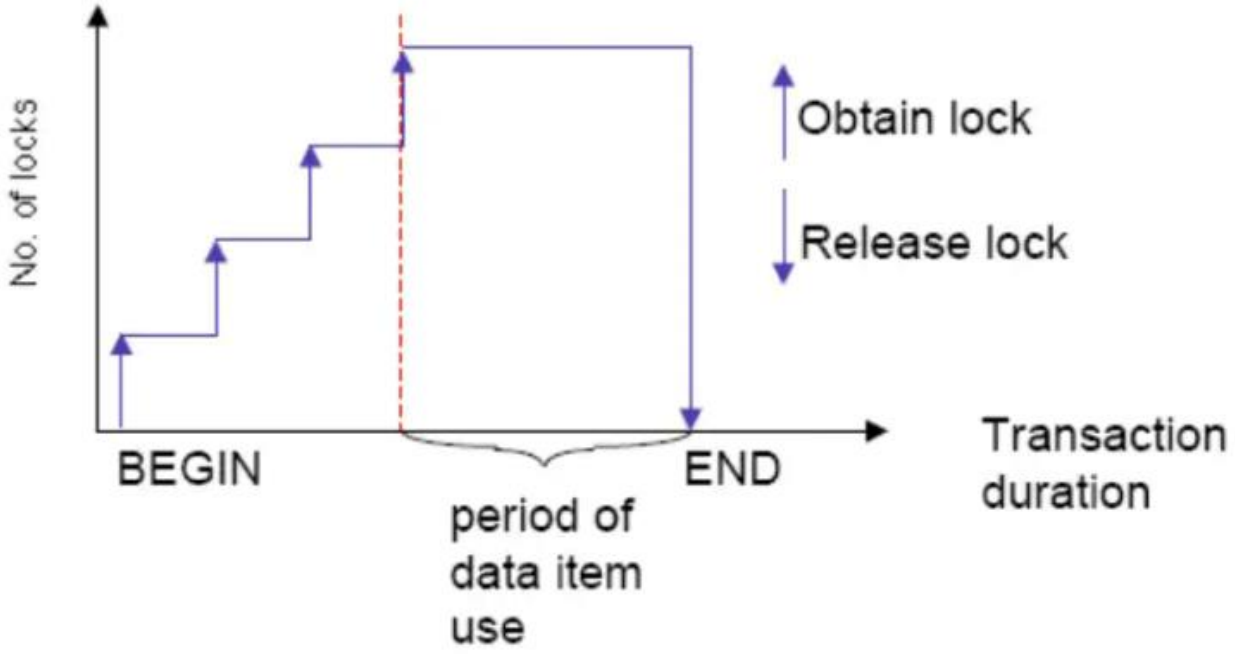
# Időkorlát mehanizmus

- Gyakorlatban: nincs minden rendszernek holtpont-felfedező mehanizmusa, csak *időtúllépés* (timeout) mehanizmusa:
  - Feltételezi, ha egy adott időintervallumban a tranzakció nem dolgozik semmit ⇔ holtpontban van.
  - Áldozatban: nem lesz semmi hiba; újraindítása rendszerfüggő:
    - Néhány rendszer újraindítja a tranzakciót (feltételezve, hogy változtak a feltételek, melyek a holtpontot okozták).
    - Más rendszer: “deadlock victim” hibakód az applikációnak ⇔ tranzakció újraindítása a programozó feladata.
    - Mindkét esetben ajánlatos tudatni a felhasználóval a történteket.

# Szigorú kétfázisú lezárási protokoll (strict 2PL)

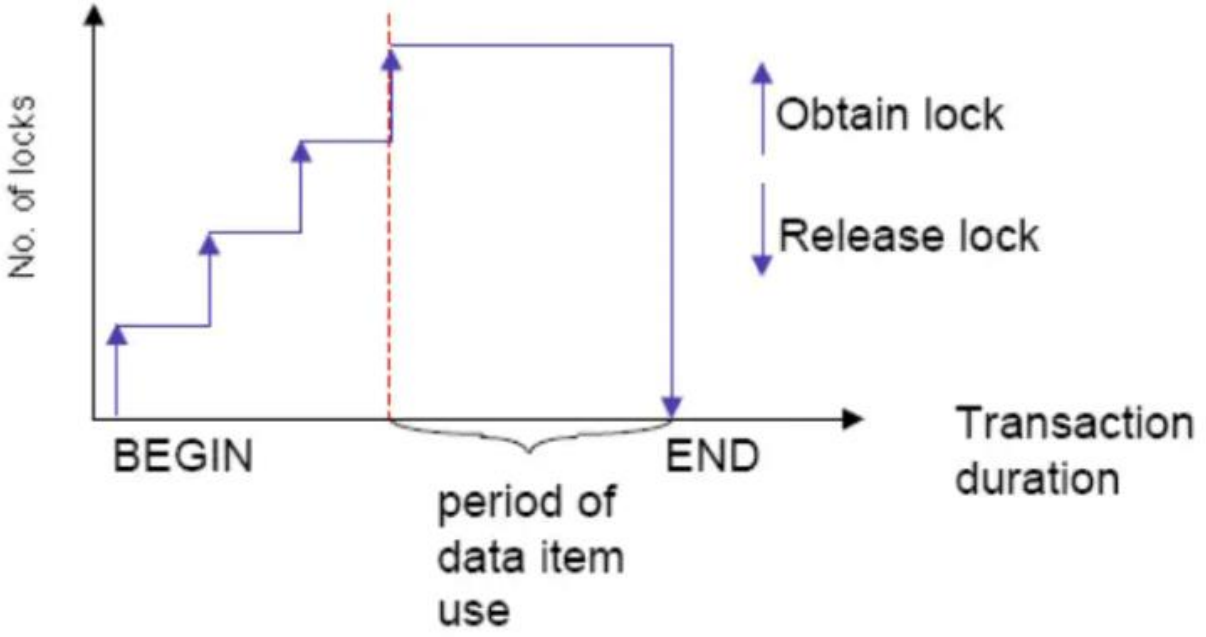
**1.** Egy tranzakció nem írhat az adatbázisba, míg nem éri el a COMMIT pontját.

**2.** Egy tranzakció nem szabadíthatja fel a zárait, míg be nem fejezi az írást az adatbázisba ⇔A zárak felszabadítása is a COMMIT pont után történik.



# Szigorú kétfázisú lezárási protokoll (strict 2PL)

- Egy tranzakció, mely betartja a szigorú kétfázisú lezárási protokollt, a következő sorrendben hajtja végre a munkát:
  - betartja a kétfázisú lezárási protokollt;
  - mikor elérkezik a COMMIT ponthoz, beírja az adatbázisba a módosított adatbáziselemeket;
  - majd felszabadítja a zárakat.



# “piszkos adat olvasása” megoldása

| **Tranzakció 1** | **Idő** | **Tranzakció 2** |
|---|---|---|
| $ul_1(P)$, $r_1(P)$ | $t_1$ | - |
| $xl_1(P)$, $w_1(P)$ | $t_2$ | - |
| - | $t_3$ | $sl_2(P)$, **vár** |
| ROLLBACK | $t_4$ | - |
| $u_1(P)$ | \| | - |
| - | $t_5$ | $sl_2(P)$, $r_2(P)$ |

# Tranzakciók SQL-ben

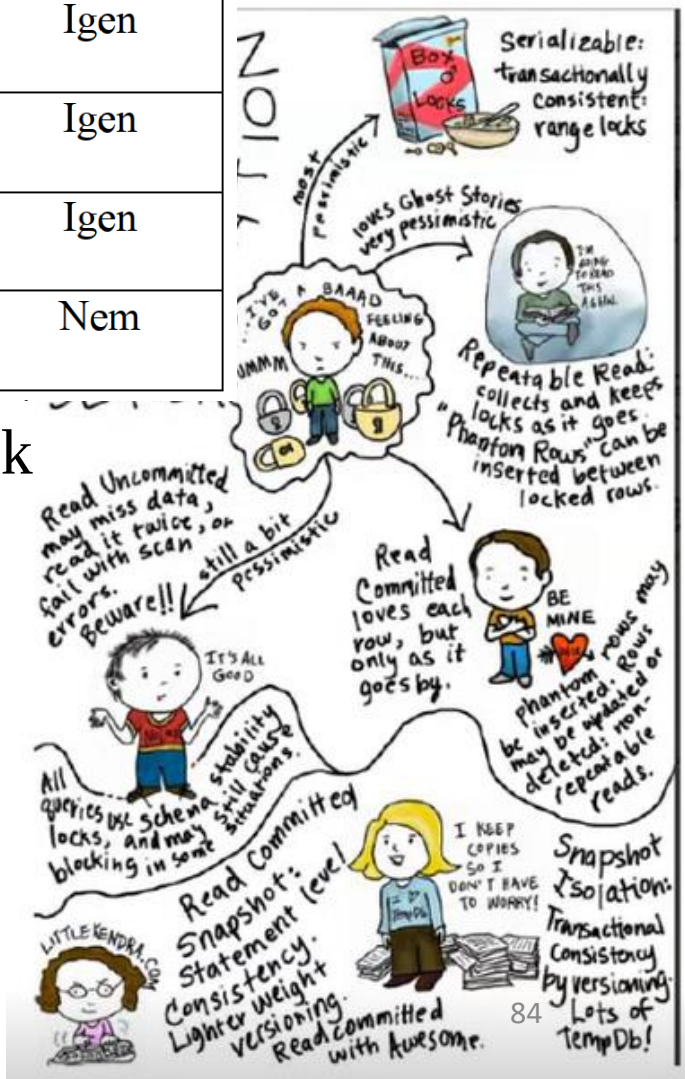
- Gyakorlatban - általában nem lehet megkövetelni, hogy a műveletek egymás után legyenek végrehajtva (túl sok van belőlük) ↔ párhuzamosság által nyújtott lehetőségek kihasználásának fontossága.
- ABKR-ek - biztosítják a sorbarendezhetőséget:
  - a felhasználó úgy látja, mintha a műveletek végrehajtása sorban történt volna *(valójában nem sorban történik)*.
- Tranzakciók kérhetnek különböző zárakat.
  - Ha az ütemező nem tudja megadni a kért zárat ⇒ a tranzakciók egy első-beérkezett-első-kiszolgálása (first-come-first-served) várási listába kerülnek, míg a szükséges adat felszabadul.

# Elkülönítési szintek SQL-ben

# Elkülönítési szintek SQL-ben

| *ISOLATION LEVEL (Elkülönítési szint)* | *Dirty read (Piszkos adat olvasása)* | *Unrepeteable read (Nem ismételhető olvasás)* | *Fantom* |
|---|---|---|---|
| READ UNCOMMITTED (nem olvasásbiztos) | Igen | Igen | Igen |
| READ COMMITTED (olvasásbiztos) | Nem | Igen | Igen |
| REPETEABLE READ (megismételhető olvasás) | Nem | Nem | Igen |
| SERIALIZABLE (sorba rendezhető) | Nem | Nem | Nem |

Elkülönítési szintek és megoldandó problémák



# Nem ismételhető olvasás (nonrepeatable read)

- Előfordulása:
  - ha a tranzakciók párhuzamosan futnak
  - **A** tranzakció beolvas egy adatbáziselemet
  - Időközben **B** tranzakció módosítja ugyanazt az adatbáziselemet
  - Majd **A** tranzakció ismét beolvassa ugyanazt az adatbáziselemet

    $\Rightarrow$ **A** tranzakció „ugyanazt” az adatbáziselemet olvasta kétszer és két különböző értéket látott.

# Nem ismételhető olvasás

| A Tranzakció | Idő | B Tranzakció |
|---|---|---|
| $sl_1(P)$; $r_1(P)$; $u_1(P)$ | $t_1$ | - |
| - | \| | - |
| - | $t_2$ | $sl_2(P)$; $r_2(P)$; |
| - | \| | $xl_2(P)$; $w_2(P)$;$u_2(P)$ |
| - | \| | - |
| $sl_1(P)$; $r_1(P)$; $u_1(P)$ | $t_3$ | - |
| - | \| | - |

$\Rightarrow$ Kiküszöbölése: REPEATABLE READ izolációs szint beállítása

# Elkülönítési szintek SQL-ben

**Fantom:**

- A rendszer csak létező adatbáziselemeket tud zárolni.
  - Nem könnyű olyan elemeket zárolni, melyek nem léteznek, de később beszúrhatók.
- **T1** olvassa azon sorok halmazát, melyek adott feltételnek eleget tesznek.
- **T2** új sort illeszt a táblába, mely kielégíti a feltételt.
- **T1** megismétli a kérést

⇔ **T1** eredménye helytelenné válik.

# Fantom

**T1:**

```
• SELECT AVG(Atlag)
   FROM Diakok
   WHERE CsopKod = '531'
```

• megismétli a kérést

➡ fantom-ot lát

• **T2** beszúr egy új diákot a Diakok táblába

⇒ Kiküszöbölése: SERIALIZABLE izolációs szint beállítása

# Fantom

- Ez valójában nem konkurenciaprobléma, ugyanis a (T1, T2) soros sorrend ekvivalens azzal, ami történt.
- Van egy fantom sor a Diakok táblában, melyet zárolni kellett volna - mivel még nem létezett, nem lehetett zárolni.
  - Megoldás: sorok beszúrását és törlését az egész relációra vonatkozó írásnak kell tekinteni és X zárat kell kérni az egész relációra. Ezt nem kaphatja meg, csak ha a minden más zár fel van szabadítva – lsd. példa esetén a T1 befejezése után.
  - Más megoldás arra, hogy a rendszer megelőzze a fantom megjelenését: le kell zárjnia a hozzáférési utat (Acces Path), mely a feltételnek eleget tevő adathoz vezet.

# Elkülönítési szintek SQL-ben

| *ISOLATION LEVEL (Elkülönítési szint)* | *Dirty read (Piszkos adat olvasása)* | *Unrepeteable read (Nem ismételhető olvasás)* | *Fantom* |
|---|---|---|---|
| READ UNCOMMITTED (nem olvasásbiztos) | Igen | Igen | Igen |
| READ COMMITTED (olvasásbiztos) | Nem | Igen | Igen |
| REPETEABLE READ (megismételhető olvasás) | Nem | Nem | Igen |
| SERIALIZABLE (sorba rendezhető) | Nem | Nem | Nem |

Elkülönítési szintek és megoldandó problémák

# Elkülönítési szintek SQL-ben

- Az izolálási szintet a tranzakciók esetén a felhasználó beállíthatja:

  **SET TRANSACTION ISOLATION LEVEL <szint>**

- Elkülönítési szintek közötti relációk:
  - Ha > erősebb feltételt jelent, akkor az elkülönítés szintjei között fennállnak a következő relációk:

  SERIALIZABLE > REPEATEBLE READ > READ COMMITED > READ UNCOMMITED

- Ha minden tranzakciónak SERIALIZABLE az elkülönítési szintje, akkor több tranzakció párhuzamos végrehajtása esetén a rendszer garantálja, hogy az ütemezés sorbarendezhető.
- Ha egy ennél kisebb elszigeteltségi szinten fut egy tranzakció, a sorbarendezhetőség meg van sértve.

# Elkülönítési szintek SQL-ben

- READ UNCOMMITED
  - piszkos, véglegesítés előtti adatokat is olvashatnak
- READ COMMITED
  - csak véglegesített, tiszta adatok olvashatók
- REPEATABLE READ
  - teljesül az ismételhető olvasás
  - két olvasás között bővülhet a tábla
- SERIALIZABLE
  - a teljesen soros végrehajtást kérvényezi
  - mindennemű módosítás tiltott

# Elkülönítési szintek SQL-ben

- SERIALIZABLE
  - A tranzakció betartja a szigorú kétfázisú lezárási protokollt: lezárást alkalmaz, írás és olvasás előtt; tranzakció végéig tartja - az objektumhalmazon (indexen) is (fantom probléma elkerülése).
- REPEATABLE READ
  - Serializable –től abban különbözik, hogy indexet nem zárol – csak egyedi objektumokat, nem objektumhalmazokat is.
  - Olvasás előtt SLOCK, írás előtt XLOCK, tranzakció végéig tartja.

# Elkülönítési szintek SQL-ben

- READ COMMITED
  - Implicit – nem enged meg az adatbázisból visszatéríteni olyan adatot, mely nincs véglegesítve (uncommited) (dirty read nem fordulhat elő).
  - Share lock-ot kér olvasás előtt a tranzakcióban szereplő objektumokra, utána rögtön felengedi – XLOCK írás előtt, tartja tranzakció végéig.
- READ UNCOMMITED
  - share lock-ot nem kér olvasás előtt, sem XLOCK-ot írás esetén
  - elolvashatja egy futó tranzakció által végzett változtatást, mely még nem volt véglegesítve
  - ha valaki más közben kitörli az adatot, melyet olvasott, hibát sem jelez, vagy törli az egész táblát
  - nem ajánlott egyetlen applikációnak sem
  - esetleg, olyan statisztikai kimutatások esetén, ahol egy-két változtatás nem lényeges.

# Csak olvasó tranzakciók

- Ajánlott használni: ha a tranzakció csak olvas és nem módosítja az adatbázis tartamát.
- Az utasítás, amivel ezt közölhetjük az ABKR-el:

  SET TRANSACTION READ ONLY;

⇒ optimálisabb ütemezés megvalósítása

# Források

- Jánosi-Rancz Katalin Tünde slide-jai *(Sapientia, Marosvásárhely)*
- Varga Ibolya slide-jai *(BBTE, Kolozsvár)*
- Katona Gyula Y. slide-jai *(BME, Budapest)*