
ELTE | IK
INFORMATIKAI KAR

# Adatbázisok 2

## Tranzakciókezelés

# Tranzakció

- Kezdjük egy kissé ködös definícióval, amit később pontosítunk majd:
  - Az adatbázisban tranzakciónak tekintjük egy logikai egysége tartozó utasítások sorozatát.

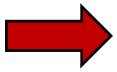
# Motiváció #1

- Az adatbázisban tároljuk egy személy bankszámláján lévő összeget (A).
- A tranzakció során a személy **vásárol** egy kávét 1500 forintért egy kávézóból (B).
- **Kezdetben** a vásárló bankszámláján 5000 forint van.

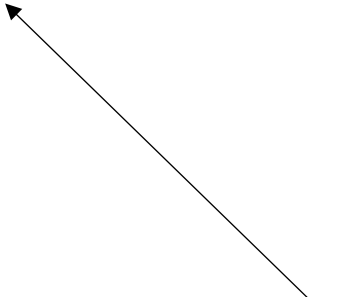
| # | **Tranzakció:** |
|---|---|
| 1. | Read(A); |
| 2. | Read(B); |
| 3. | A := A - 1500; |
| 4. | B := B + 1500; |
| 5. | Write(A); |
| 6. | Write(B); |

# Motiváció #1

- Az adatbázisban tároljuk egy személy bankszámláján lévő összeget.
- A tranzakció során a személy **vásárol** egy kávét 1500 forintért egy kávézóból.
- **Kezdetben** a vásárló bankszámláján 5000 forint van.



| # | Tranzakció: |
|---|---|
| 1. | Read(A); |
| 2. | Read(B); |
| 3. | A := A - 1500; |
| 4. | B := B + 1500; |
| 5. | Write(A); |
| 6. | Write(B); |



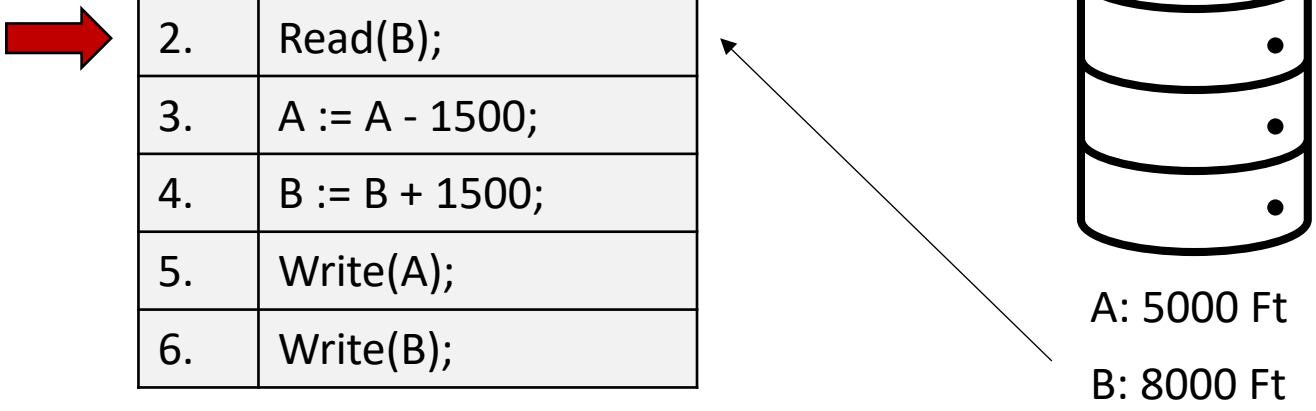
**Adatbázis**

A: 5000 Ft

B: 8000 Ft

# Motiváció #1

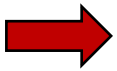
- Az adatbázisban tároljuk egy személy bankszámláján lévő összeget.
- A tranzakció során a személy **vásárol** egy kávét 1500 forintért egy kávézóból.
- **Kezdetben** a vásárló bankszámláján 5000 forint van.



| # | Tranzakció: |
|---|---|
| 1. | Read(A); |
| 2. | Read(B); |
| 3. | A := A - 1500; |
| 4. | B := B + 1500; |
| 5. | Write(A); |
| 6. | Write(B); |

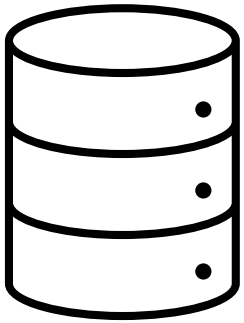
# Motiváció #1

- Az adatbázisban tároljuk egy személy bankszámláján lévő összeget.
- A tranzakció során a személy **vásárol** egy kávét 1500 forintért egy kávézóból.
- **Kezdetben** a vásárló bankszámláján 5000 forint van.



| # | Tranzakció: |
|---|---|
| 1. | Read(A); |
| 2. | Read(B); |
| 3. | A := A - 1500; |
| 4. | B := B + 1500; |
| 5. | Write(A); |
| 6. | Write(B); |

**Adatbázis**

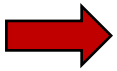


A: 5000 Ft

B: 8000 Ft

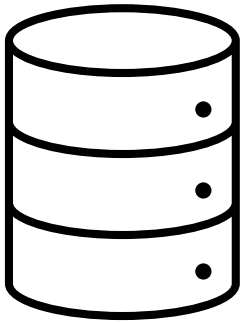
# Motiváció #1

- Az adatbázisban tároljuk egy személy bankszámláján lévő összeget.
- A tranzakció során a személy **vásárol** egy kávét 1500 forintért egy kávézóból.
- **Kezdetben** a vásárló bankszámláján 5000 forint van.



| # | Tranzakció: |
|---|---|
| 1. | Read(A); |
| 2. | Read(B); |
| 3. | A := A - 1500; |
| 4. | B := B + 1500; |
| 5. | Write(A); |
| 6. | Write(B); |

**Adatbázis**



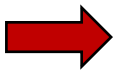
A: 5000 Ft

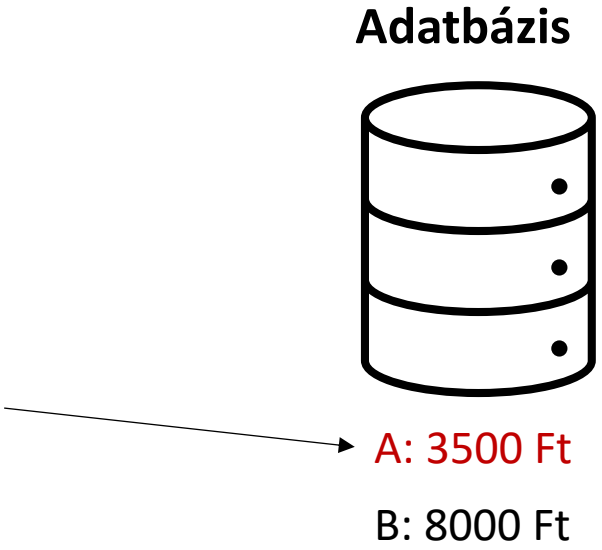
B: 8000 Ft

# Motiváció #1

- Az adatbázisban tároljuk egy személy bankszámláján lévő összeget.
- A tranzakció során a személy **vásárol** egy kávét 1500 forintért egy kávézóból.
- **Kezdetben** a vásárló bankszámláján 5000 forint van.

| # | Tranzakció: |
|---|---|
| 1. | Read(A); |
| 2. | Read(B); |
| 3. | A := A - 1500; |
| 4. | B := B + 1500; |
| 5. | Write(A); |
| 6. | Write(B); |



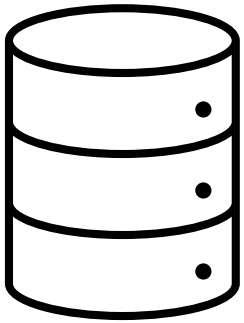


# Motiváció #1

- Az adatbázisban tároljuk egy személy bankszámláján lévő összeget.
- A tranzakció során a személy **vásárol** egy kávét 1500 forintért egy kávézóból.
- **Kezdetben** a vásárló bankszámláján 5000 forint van.

| # | Tranzakció: |
|---|---|
| 1. | Read(A); |
| 2. | Read(B); |
| 3. | A := A - 1500; |
| 4. | B := B + 1500; |
| 5. | Write(A); |
| 6. | Write(B); |

**Adatbázis**



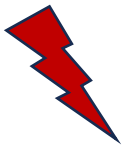
A: 4500 Ft

B: 9500 Ft

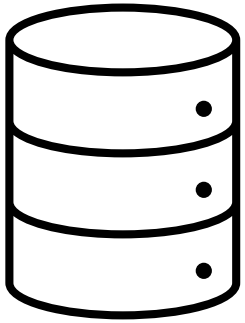
# Motiváció #1

- Mi történik ha egy váratlan esemény (hiba) lép fel az 5.-ik lépés után!
  - Az eladó nem kapja meg a pénzt!

| # | Tranzakció: |
|---|---|
| 1. | Read(A); |
| 2. | Read(B); |
| 3. | A := A - 1500; |
| 4. | B := B + 1500; |
| 5. | Write(A); |
| 6. | Write(B); |



**Adatbázis**



A: 4500 Ft

B: 8000 Ft

# Motiváció #2

- Az adatbázisban tároljuk egy személy bankszámláján lévő összeget.
- A tranzakció során a személy **vásárol** egy kávét 1500 forintért egy kávézóból.
- Ezzel egy időben épp **levonásra kerül** a számlájáról a streaming előfizetése is.

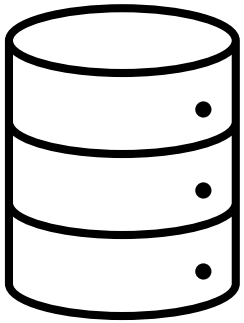
Kávé vásárlás:

| # | **Tranzakció:** |
|---|---|
| 1. | Read(A); |
| 2. | A := A - 1500; |
| 3. | Write(A); |

Streaming előfizetés:

| # | **Tranzakció:** |
|---|---|
| 1. | Read(A); |
| 2. | A := A - 2500; |
| 3. | Write(A); |

**Adatbázis**



A: 5000 Ft

# Motiváció #2

- Ha minden jól megy (pl. **egymás után hajtjuk végre** a két tranzakciót), akkor előbb levonjuk a kávé árát, aztán a streaming árát.

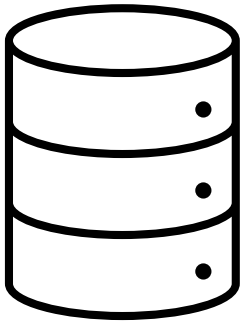
Kávé vásárlás:

| # | **Tranzakció:** |
|---|---|
| 1. | Read(A); |
| 2. | A := A - 1500; |
| 3. | Write(A); |

Streaming előfizetés:

| # | **Tranzakció:** |
|---|---|
| 1. | Read(A); |
| 2. | A := A - 2500; |
| 3. | Write(A); |

**Adatbázis**



A: 3500 Ft

# Motiváció #2

- Ha minden jól megy (pl. **egymás után hajtjuk végre** a két tranzakciót), akkor előbb levonjuk a kávé árát, aztán a streaming árát.

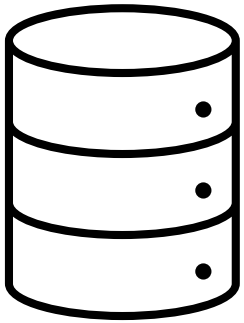
Kávé vásárlás:

| # | **Tranzakció:** |
|---|---|
| 1. | Read(A); |
| 2. | A := A - 1500; |
| 3. | Write(A); |

Streaming előfizetés:

| # | **Tranzakció:** |
|---|---|
| 1. | Read(A); |
| 2. | A := A - 2500; |
| 3. | Write(A); |

**Adatbázis**



A: 1000 Ft

# Motiváció #2

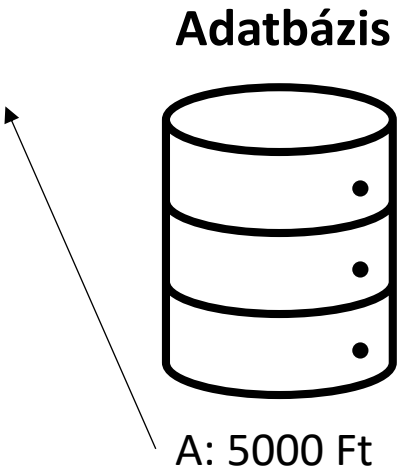
- Mi történik ha **párhuzamosan** futtatjuk?

Kávé vásárlás:

| # | **Tranzakció:** |
|---|---|
| 1. | Read(A); |
| 2. | A := A - 1500; |
| 3. | Write(A); |

Streaming előfizetés:

| # | **Tranzakció:** |
|---|---|
| 1. | Read(A); |
| 2. | A := A - 2500; |
| 3. | Write(A); |



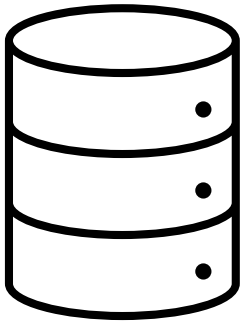
# Motiváció #2

- Mi történik ha **párhuzamosan** futtatjuk?

Streaming előfizetés:

| # | Tranzakció: |
|---|---|
| 1. | Read(A); |
| 2. | A := A - 2500; |
| 3. | Write(A); |

**Adatbázis**



Kávé vásárlás:

| # | Tranzakció: |
|---|---|
| 1. | Read(A); |
| 2. | A := A - 1500; |
| 3. | Write(A); |

A: 5000 Ft

# Motiváció #2

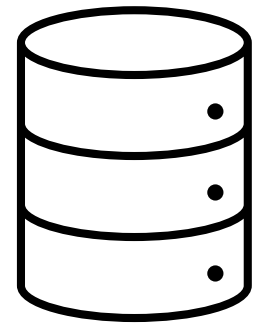
- Mi történik ha **párhuzamosan** futtatjuk?

Kávé vásárlás:

| # | **Tranzakció:** |
|---|---|
| 1. | Read(A); |
| 2. | A := A - 1500; |
| 3. | Write(A); |

Streaming előfizetés:

| # | **Tranzakció:** |
|---|---|
| 1. | Read(A); |
| 2. | A := A - 2500; |
| 3. | Write(A); |




A: 2500 Ft

# Motiváció #2

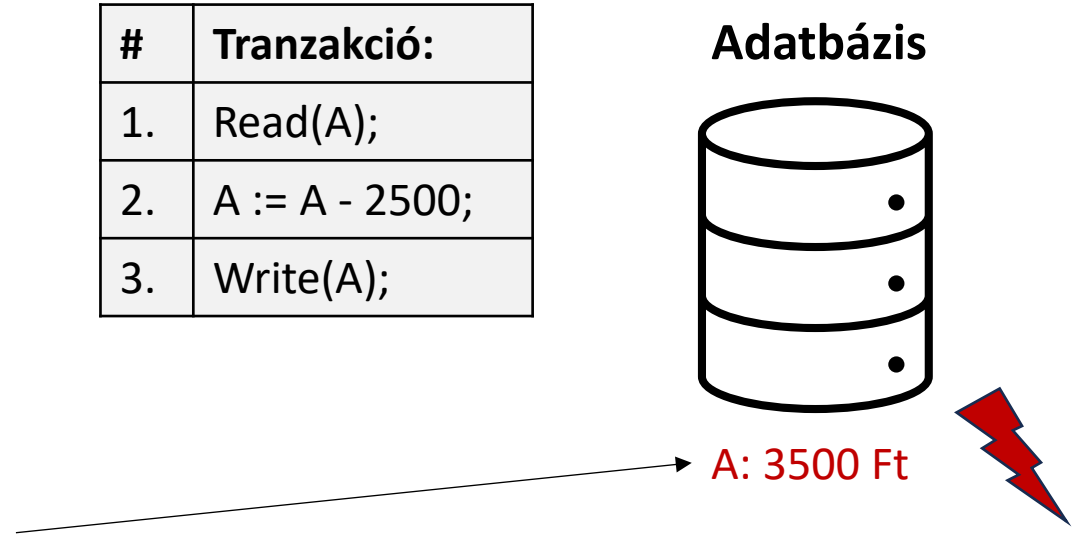
- Mi történik ha **párhuzamosan** futtatjuk?
  - Több pénz marad a számlán, mint amennyinek kellene!

Streaming előfizetés:

| # | Tranzakció: |
|---|---|
| 1. | Read(A); |
| 2. | A := A - 2500; |
| 3. | Write(A); |

Kávé vásárlás:

| # | Tranzakció: |
|---|---|
| 1. | Read(A); |
| 2. | A := A - 1500; |
| 3. | Write(A); |



# Tranzakció definíciója

- **Tranzakciónak** nevezünk egy **olvasás** és **írás** műveletekből álló utasítás sorozatot, amelynek van kezdete és vége.
  - A tranzakció a sorozat első utasításával kezdődik.
  - A tranzakciót egy COMMIT vagy ROLLBACK utasítás zárja le.
  - A tranzakcióban lévő olvasás és írás műveletek adatbázis objektumokra vonatkoznak. Az adatbázis objektumok lehetnek táblák, rekordok, blokkok stb.
- A tranzakcióknak teljesítenie kell az **ACID** helyességi feltételeket.

# ACID tulajdonságok

- **Atomicity** (atomosság) – A tranzakció „mindent vagy semmit” jellegű végrehajtása. Azaz vagy teljesen végrehajtjuk, vagy egyáltalán nem hajtjuk végre.
- **Consistency** (konzisztenica) – A tranzakciónak meg kell őriznie az adatbázis konzisztenciáját. Azaz a tranzakció végrehajtása után is teljesülniük kell az adatbázisban megadott konzisztencia feltételeknek.
- **Isolation** (elkülönítés) – Minden tranzakciónak látszólag úgy kell lefutnia, mintha ez alatt az idő alatt semmilyen másik tranzakciót sem hajtanánk végre.
- **Durability** (tartósság) – Ha egyszer egy tranzakció befejeződött, akkor már soha többé nem veszhet el a tranzakciónak az adatbázisra kifejtett hatása.

# Atomosság

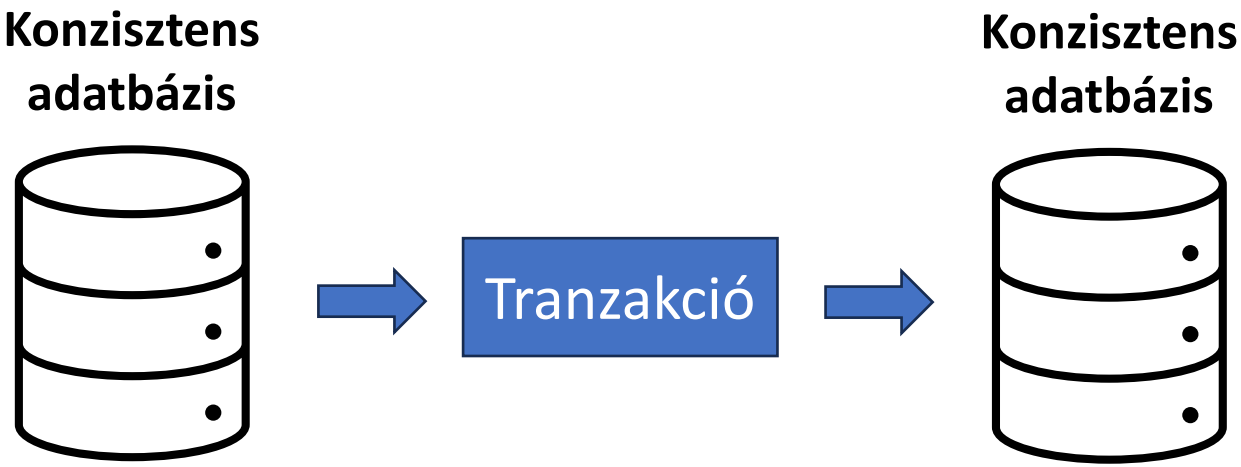
- Egy tranzakció futtatásának **két** lehetséges kimenetele lehet:
  - Sikeresen befejeződik (**COMMIT**)
  - Megszakításra kerül (**ABORT**) a felhasználó vagy a rendszer által.
- **Atomosság**: az adatbázis biztosítja, hogy egy tranzakciót vagy teljesen végrehajtásra kerül, vagy egyáltalán nem kerül végrehajtásra.
- Miért fontos ez? Lásd az első példát a motivációs részből.

# Atomosság

- Hogyan biztosítja ezt a tulajdonságot az adatbázis?
  - Naplózással!
- Naplózzunk minden utasítást, és ha hiba lép fel, akkor a napló segítségével állítsuk vissza a tranzakció előtti állapotot.
- Szinte minden rendszer ezt használja.
- A naplózást a következő előadáson részletesen megnézzük.

# Konzisztencia

- Ha egy tranzakció egy **konzisztens** adatbázis **állapotból indul**, akkor konzisztens állapotban is **kell hagynia** az adatbázist.
- Megjegyzés:
  - Ha csak önmagában a konzisztencia tulajdonságot vizsgáljuk, akkor feltesszük, hogy az adott tranzakció egyedül fut le (így kell teljesítenie a feltételt).



# Mit jelent a konzisztencia?

- Az adatok előre megadott **feltételeket** elégítenek ki.
- Például:
  - Megengedett értékek korlátozása (SQL-ben CHECK).
  - Elsődleges és idegen kulcs megszorítások.
  - Egyediségre vonatkozó megszorítás (UNIQUE).
  - Általánosabb megszorításokat is elképzelhetünk (pl. egyik dolgozó sem keres többet, mint az átlagfizetés kétszerese).
- Azt mondjuk, hogy az **adatbázis konzisztens**, ha konzisztens állapotban van, azaz kielégíti az összes feltételt (megszorítást).

# Helyesség feltétele

1. Ha egy vagy több tranzakció leáll (akár hiba miatt, akár a felhasználó szakítja meg), az adatbázisnak ezután is konzisztensnek kell lennie.
2. Minden egyes tranzakció induláskor konzisztens adatbázist lát.

# Megjegyzések

- A megszorítások hiányossága:
  - Az adatbázis a **valóságot** próbálja reprezentálni, de minden összefüggést (a valóság teljes szemantikáját) lehetetlen megadni.
- A megszorításokat az adatbázis tervezője adja meg.
- **Vegyük észre**, hogy az adatbázis nem lehet minden időpillanatban konzisztens!
  - Pl. egy adatbázisban tároljuk egy banki felhasználó tranzakcióit és a számlája egyenlegét.
  - Ha egy új tranzakciót felviszünk (vásárlás), akkor az egyenlegből le kell vonni a vásárlás összegét (ez két lépés).
  - A két lépés közt nincs meg a konzisztencia.
  - Ezért fontos az atomosság.

# Izoláció

- A tranzakciókat párhuzamosan akarjuk futtatni, de azt szeretnénk ha a hatása megegyezne azzal, mintha egymás után futnának.
- Ezt biztosítja az **ütemező** (konkurenciavezérlés-kezelő).
- Két általános megközelítése:
  - **Pesszimista**: ne engedjük, hogy probléma keletkezzen.
  - **Optimista**: feltételezzük, hogy a problémák ritkák, ezért csak akkor foglalkozzunk velük ha felbukkannak.

# Izoláció

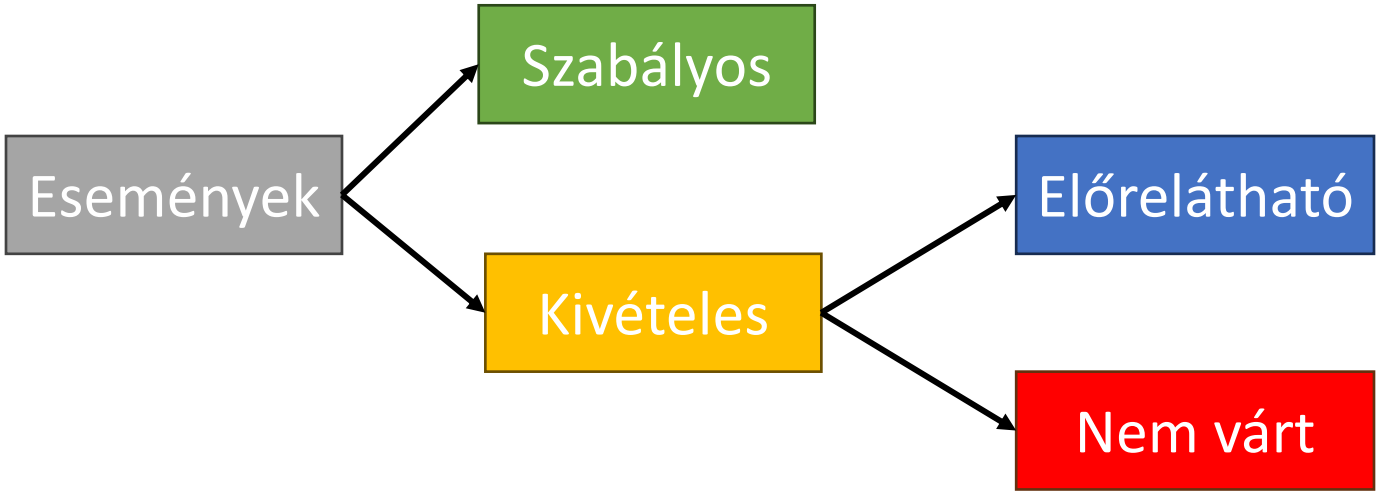
- Az izolációt a rendszerek általában **zárakkal** biztosítják, amelyek megakadályozzák a hibás működést. A zárakat zártáblában tárolja a rendszer.
- A zárak használata esetén is létrejöhetnek **holtpontok**, amelyeket fel kell oldalni.
- Az ütemező működését és a zárolásokat egy későbbi előadáson részletesen megnézzük!

# Tartósság

- Ha egyszer egy tranzakció befejeződött, akkor már soha többé nem veszhet el a tranzakciónak az adatbázisra kifejtett hatása.
- A tartósságot az adatbázisban a **naplózás** biztosítja.
    - A következő előadáson részletesen megnézzük.

# Meghibásodási modell

- Szabályos esemény: normális működés.
- Előrelátható, kivételes esemény: áramszünet, hardver elromlik stb.
- Nem várt, kivételes esemény: minden más.



# Hibák fajtái

- Hibás adatbevitel.
  - Pl. tartalmi hiba (telefonszámba elütött számjegy), formai hiba (kihagyott számjegy).
  - Megoldás: megszorítások, beviteli mezők ellenőrzése.
- Készülékhibák
  - Pl. megsérül néhány bit (kis hiba) vagy akár az egész lemez.
  - Megoldások: RAID, archiválás, osztott másolat.
- Katasztrofális hibák
  - Pl. teljesen tönkremegy az eszköz (tűz, vírusok stb.)
  - Megoldás: archiválás, osztott másolat.
- Rendszerhibák
  - Pl. áramkimaradás, szoftverhibák (hibás driver miatti leállás).
  - Megoldás: naplózás.

# ACID tulajdonságok mégegyszer

- **Atomicity** (atomosság) – A tranzakció „mindent vagy semmit” jellegű végrehajtása. Azaz vagy teljesen végrehajtjuk, vagy egyáltalán nem hajtjuk végre.
- **Consistency** (konzisztenica) – A tranzakciónak meg kell őriznie az adatbázis konzisztenciáját. Azaz a tranzakció végrehajtása után is teljesülniük kell az adatbázisban megadott konzisztencia feltételeknek.
- **Isolation** (elkülönítés) – Minden tranzakciónak látszólag úgy kell lefutnia, mintha ez alatt az idő alatt semmilyen másik tranzakciót sem hajtanánk végre.
- **Durability** (tartósság) – Ha egyszer egy tranzakció befejeződött, akkor már soha többé nem veszhet el a tranzakciónak az adatbázisra kifejtett hatása.

# Tankönyv fejezetek

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

  **Adatbázisrendszerek megvalósítása**

  - 8.1. fejezet: A rendszerhibák kezelése
- Silberschatz, Korth, & Sudarshan: **Database System Concepts**
  - Chapter 17. Transactions