ELTE | IK
INFORMATIKAI KAR

Adatbázisok 2

Konkurenciakezelés

Konkurenciakezelés

- Egyszerre **több** tranzakció is használja ugyanazt az adatbázist.
- AC**I**D, **isolation** (elkülönítés): minden tranzakciónak úgy kell lefutnia, mintha abban az időpillanatban az lenne az egyetlen tranzakció az adatbázisban.
- Az adatbázisnak konzisztensnek kell maradnia több tranzakció párhuzamos futtatása során is.
- Azt a folyamatot, amely biztosítja, hogy a tranzakciók egyidejű végrehajtása során megőrizzék a konzisztenciát, **konkurenciavezérlésnek** (concurrency control) nevezzük.

Példa (bankszámla)

- Tegyük fel, hogy A-nak és B-nek is van 1000 forintja.
- A T1 tranzakció elvégez egy átutalást A számlájáról B számlájára (100 forint).
- A T2 tranzakció mindkét számlán lévő összegnek a 6%-át jóváírja kamatként.

T1:

Tranzakció:
Read(A)
A := A - 100
Write(A)
Read(B)
B := B + 100
Write(B)

T2:

Tranzakció:
Read(A)
A := A * 1.06
Write(A)
Read(B)
B := B * 1.06
Write(B)

Példa

- Milyen lehetséges kimenetelei lehetnek a két tranzakció végrehajtásának?
  - Több is van!
- Ha pont egyszerre érkezik be az adatbázishoz a két tranzakció, akkor nem tudjuk, hogy melyik fog korábban lefutni.
- Az biztos, hogy akkor helyes a működés ha: $(\mathbf{2} * \mathbf{1000}) * \mathbf{1.06} = \mathbf{2120}$

T1:

Tranzakció:
Read(A)
A := A - 100
Write(A)
Read(B)
B := B + 100
Write(B)

T2:

Tranzakció:
Read(A)
A := A * 1.06
Write(A)
Read(B)
B := B * 1.06
Write(B)

Ütemezés

- Az **ütemezés** (schedule) egy vagy több tranzakció által végrehajtott lényeges műveletek időrendben vett sorozata.
- Azt mondjuk, hogy egy ütemezés **soros** (serial schedule), ha úgy épül fel a tranzakciós műveletekből, hogy először az egyik tranzakció összes műveletét tartalmazza, majd azután egy másik tranzakció összes műveletét stb., miközben nem cseréli fel a műveleteket.

Soros ütemezések

T1	T2
Read(A)	
A := A - 100	
Write(A)	
Read(B)	
B := B + 100	
Write(B)	
	Read(A)
	A := A * 1.06
	Write(A)
	Read(B)
	B := B * 1.06
	Write(B)

A=954, B=1166, **A+B=2120**

T1	T2
	Read(A)
	A := A * 1.06
	Write(A)
	Read(B)
	B := B * 1.06
	Write(B)
Read(A)	
A := A - 100	
Write(A)	
Read(B)	
B := B + 100	
Write(B)	

A=960, B=1160, **A+B=2120**

Mi a baj a soros ütemezésekkel?

Mi a baj a soros ütemezésekkel?

- Problémák:
  - A tranzakcióknak egymásra kell várni.
  - Többmagos CPU-kat nem tudjuk így kihasználni.
- Megoldás:
  - A különböző tranzakciók műveletei legyenek **váltakozva** is végrehajthatók!
  - Így ha egy tranzakció sokáig is tart, a többi tranzakció végrehajtása akkor folytatódhat.

Nem soros ütemezés – „jó” ütemezés

T1	T2
Read(A)	
A := A - 100	
Write(A)	
Read(B)	
B := B + 100	
Write(B)	
	Read(A)
	A := A * 1.06
	Write(A)
	Read(B)
	B := B * 1.06
	Write(B)

A=954, B=1166, **A+B=2120**

T1	T2
Read(A)	
A := A - 100	
Write(A)	
	Read(A)
	A := A * 1.06
	Write(A)
Read(B)	
B := B + 100	
Write(B)	
	Read(B)
	B := B * 1.06
	Write(B)

A=954, B=1166, **A+B=2120**

Nem soros ütemezés – „rossz” ütemezés

T1	T2
Read(A)	
A := A - 100	
Write(A)	
Read(B)	
B := B + 100	
Write(B)	
	Read(A)
	A := A * 1.06
	Write(A)
	Read(B)
	B := B * 1.06
	Write(B)

A=954, B=1166, **A+B=2120**

T1	T2
Read(A)	
A := A - 100	
Write(A)	
	Read(A)
	A := A * 1.06
	Write(A)
	Read(B)
	B := B * 1.06
	Write(B)
Read(B)	
B := B + 100	
Write(B)	

A=954, B=1160, **A+B=2114**

Mi a különbség a két ütemezés között?

- Miért volt **„jó”** az egyik ütemezés és miért volt **„rossz”** a másik?
  - A **jó** ütemezés hatása megegyezett az egyik soros ütemezés hatásával.
  - A **rossz** ütemezés hatása nem egyezett meg egyik soros ütemezés hatásával sem.
- Egy ütemezés **sorba rendezhető** (serializable schedule), ha ugyanolyan hatással van az adatbázis állapotára, mint valamelyik soros ütemezés, függetlenül attól, hogy mi volt az adatbázis kezdeti állapota.

- A nem sorba rendezhető ütemezések azok, amelyeket az ütemezőnek el kell kerülnie!

Jelölések

- Egyszerűsítsük a jelölést.
- Egy T_i tranzakció az „i” indexű műveletekből álló sorozat.
- Mivel számunkra csak az **olvasás** és **írás** műveletek a fontosak, ezért használjunk egyszerűbb jelöléseket a tranzakciók műveleteire:
  - $r_i(A)$ – a T_i tranzakció olvassa az A adatbáziselemet.
  - $w_i(A)$ – a T_i tranzakció írja az A adatbáziselemet.
- Egy S **ütemezés** olyan műveletek sorozata, amelyben minden T_i tranzakcióra teljesül, hogy T_i műveletei ugyanabban a sorrendben fordulnak elő S-ben, mint a T_i -ben.

Konfliktus

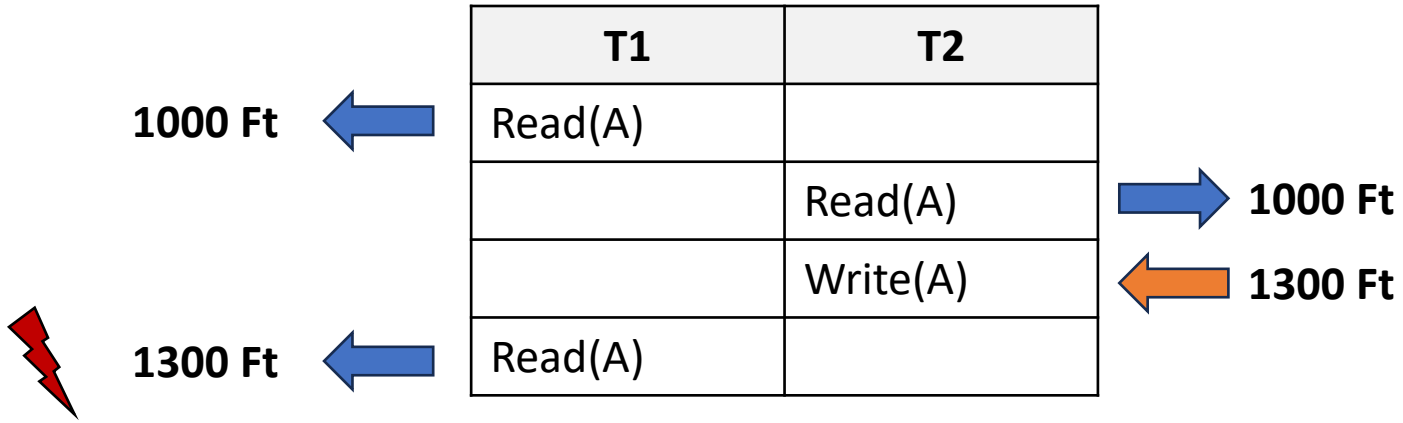
- A **konfliktus** (conflict) vagy **konfliktuspár** olyan egymást követő műveletek (műveletpár) az ütemezésben, amelynek ha a sorrendjét felcseréljük, akkor legalább az egyik tranzakció viselkedése megváltozhat.
- A következő esetekben nem cserélhetjük fel a műveletek sorrendjét:
  - $\boldsymbol{r_i(X); r_i(Y)}$ – mivel egyetlen tranzakción belül a műveletek sorrendje rögzített, ezért ezt a sorrendet nem lehet átrendezni (írás-olvasás, olvasás-írás, írás-írás esetén is ugyanez a helyzet).
  - $\boldsymbol{w_i(X); w_j(X)}$ – két különböző tranzakció ugyanarra az adatbáziselemre vonatkozó írás művelete (írás-írás konfliktus)
  - $\boldsymbol{w_i(X); r_j(X)}$ és $\boldsymbol{r_i(X); w_j(X)}$ – két különböző tranzakció olvasás és írás műveletei, ha ugyanarra az adatbáziselemre vonatkoznak (írás-olvasás és olvasás-írás is konfliktus).

Konfliktus - következtetés

- Különböző tranzakciók bármely két műveletének sorrendje felcserélhető, hacsak nem
  - … **ugyanarra** az **adatbáziselemre** vonatkoznak, és
  - … legalább az **egyik** művelet **írás**.

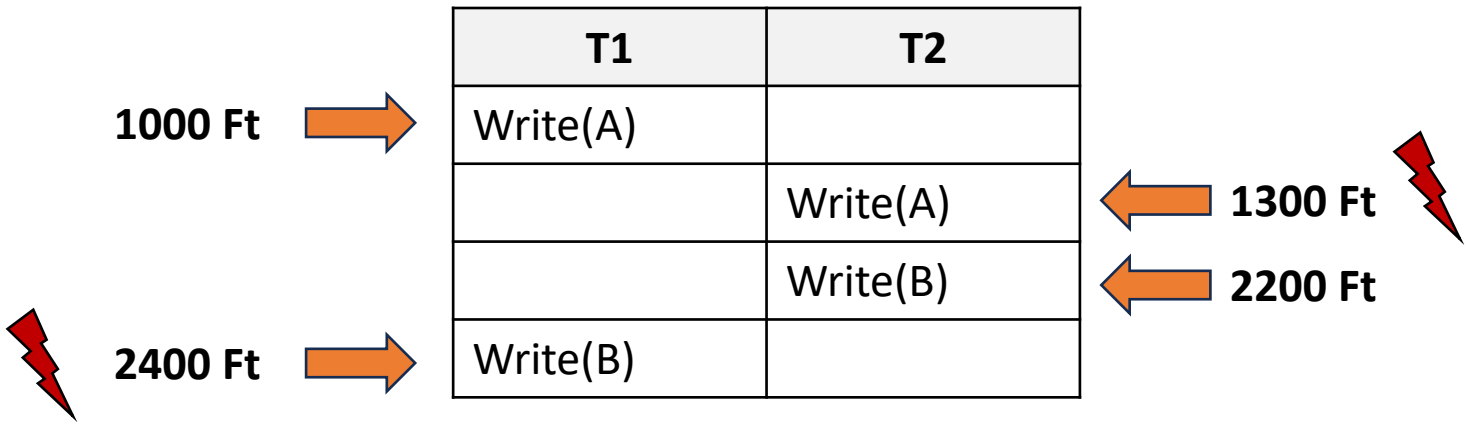
Olvasás-írás konfliktus

- **Unrepeatable read** (nem megismételhető olvasás) – amikor egy tranzakció ugyanazt az adatbáziselemet olvasva különböző értékeket kap.



Írás-írás konfliktus

- **Lost update** (elveszett módosítás) – amikor egy tranzakció felülírja egy másik tranzakció módosításait.



Konfliktusekvivalencia

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

$$r_1(A); w_1(A); r_2(A); \mathbf{w_2(A); r_1(B)}; w_1(B); r_2(B); w_2(B)$$

$$r_1(A); w_1(A); \mathbf{r_2(A); r_1(B)}; w_2(A); w_1(B); r_2(B); w_2(B)$$

$$r_1(A); w_1(A); r_1(B); r_2(A); \mathbf{w_2(A); w_1(B)}; r_2(B); w_2(B)$$

$$r_1(A); w_1(A); r_1(B); \mathbf{r_2(A); w_1(B)}; w_2(A); r_2(B); w_2(B)$$

$$\underbrace{r_1(A); w_1(A); r_1(B); w_1(B);}_{T_1} \underbrace{r_2(A); w_2\,(A); r_2(B); w_2(B)}_{T_2}$$

Konfliktus-sorbarendezhetőség

- Azt mondjuk, hogy egy ütemezés **konfliktus-sorbarendezhető** (conflict-serializable schedule), ha konfliktusekvivalens valamely soros ütemezéssel.
- Az előző példa első és utolsó sora:
- Kiinduló állapot:

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

- Ekvivalens soros ütemezés:

$$\underbrace{r_1(A); w_1(A); r_1(B); w_1(B);}_{T_1}\ \underbrace{r_2(A); w_2\ (A); r_2(B); w_2(B)}_{T_2}$$

Nem konfliktus-sorbarendezhető ütemezés

- A korábbi „rossz” ütemezés átírva egyszerűsített jelölésre:

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

- A T_2T_1 soros ütemezést nem tudjuk előállítani, mert:

$$r_1(A); \boldsymbol{w_1(A); r_2(A)}; w_2(A); r_2(B); w_2(B); r_1(B); w_1(B);$$

- A T_1T_2 soros ütemezést nem tudjuk előállítani, mert:

$$r_1(A); w_1(A); r_2(A); w_2(A); r_2(B); \boldsymbol{w_2(B); r_1(B)}; w_1(B);$$

Következtetések

- A konfliktus-sorbarendezhetőség **elégséges feltétel** a sorbarendezhetőségre, vagyis egy konfliktus-sorbarendezhető ütemezés sorbarendezhető ütemezés is egyben.
- A konfliktus-sorbarendezhetőség **nem szükséges** ahhoz, hogy egy ütemezés sorbarendezhető legyen.
    - A **használatban lévő rendszerek** viszont általában a konfliktus-sorbarendezhetőséget ellenőrzik!

Megjegyzések

- Létezik még ún. **nézet-sorbarendezhetőség** is, amely enyhébb feltétel, mint a konfliktus-sorbarendezhetőség, azonban ezt nem szokták a rendszerek használni (mindkét tankönyvben megtalálható a leírása).
- Léteznek **sorbarendezhető**, de **nem konfliktus-sorbarendezhető** ütemezések, pl:

$$S\colon\ w_1(Y); w_2(Y); w_2(X); w_1(X); w_3(X)$$

- A fenti S ütemezés nem konfliktus-ekvivalens egyetlen soros ütemezéssel sem, viszont sorbarendezhető, mivel hatása megegyezik a $T_1T_2T_3$ soros ütemezéssel.
  - Az ilyen eseket nehéz ellenőrizni, ezért a valós rendszerek általában nem is engedélyezik.

Konfliktusok értelmezése

- **Alapötlet**: ha valahol konfliktusban álló műveletek szerepelnek S-ben, akkor az ezeket a műveleteket végrehajtó tranzakcióknak ugyanabban a sorrendben kell előfordulniuk a konfliktus-ekvivalens soros ütemezésekben, mint ahogyan az S-ben voltak.
- Tehát a konfliktusban álló műveletpárok **megszorítást** adnak a feltételezett konfliktusekvivalens soros ütemezésben a tranzakciók sorrendjére.

Megelőzés

- Adott T_1 és T_2 tranzakcióknak, esetleg további tranzakcióknak egy *S* ütemezése. Azt mondjuk, hogy T_1 **megelőzi** T_2-t, ha van a T_1-ben olyan A_1 művelet és a T_2-ben olyan A_2 művelet, hogy:
  - A_1 megelőzi A_2-őt *S-ben*.
  - A_1 és A_2 ugyanarra az adatbáziselemre vonatkoznak, és
  - A_1 és A_2 közül legalább az egyik írás művelet.
- **Másképpen fogalmazva**: A_1 és A_2 konfliktuspárt alkotna, ha szomszédos műveletek lennének. **Jelölése**: $T_1 <_S T_2$
- Ezeket a megelőzéseket a **megelőzési gráfban** (precedence graph) összegezhetjük.

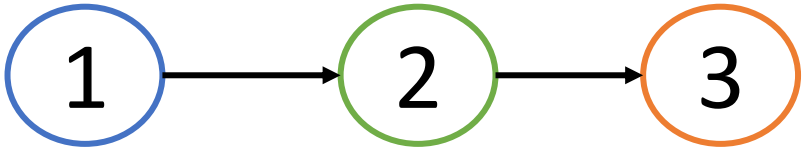
Megelőzési gráf

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

Példa megelőzési gráfra

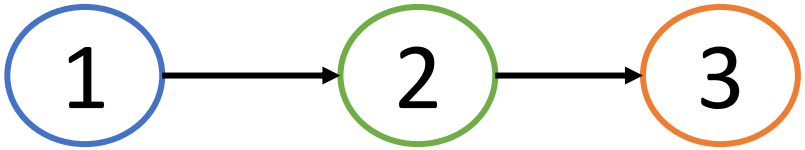
- Adott egy S ütemezés, amely a T_1, T_2 és T_3 tranzakciókat tartalmazza.
- S: $r_2(A); r_1(B); w_2(A); r_3(A); w_1(B); w_3(A); r_2(B); w_2(B)$

- Piros nyilakkal jelöltem néhány megelőzést (a többi megelőzés nem adna új éleket a gráfba).
- Az S ütemezéshez tartozó megelőzési gráf a következő:



Teszt konfliktus-sorbarendezhetőségre

- Ha az S megelőzési gráf tartalmaz **irányított kört**, akkor S **nem konfliktus-sorbarendezhető**, ha nem tartalmaz irányított kört, akkor S konfliktus-sorbarendezhető, és a csúcsok bármelyik **topologikus sorrendje** megad egy konfliktusekvivalens soros sorrendet.
- **Topologikus sorrend**: Egy körmentes gráf csúcsainak topologikus sorrendje a csúcsok bármely olyan rendezése, amelyben minden $a \rightarrow b$ élre az a csúcs megelőzi a b csúcsot a topologikus rendezésben.
- Példa:



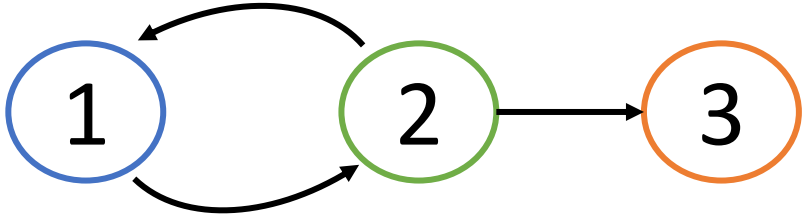
- Csak egy topologikus sorrend létezik: 1, 2, 3
- Azaz az ekvivalens soros ütemezés: $T_1 T_2 T_3$

Példa megelőzési gráfra (irányított körrel)

- Adott egy S ütemezés, amely a T_1, T_2 és T_3 tranzakciókat tartalmazza.
- S: $r_2(A); r_1(B); w_2(A); r_2(B); r_3(A); w_1(B); w_3(A);\ w_2(B)$

- Az $r_2(A) \ldots w_3(A)$; miatt $T_2 <_S T_3$
- Az $r_2(B) \ldots w_1(B)$; miatt $T_2 <_S T_1$
- A $w_1(B) \ldots w_2(B)$; miatt $T_1 <_S T_2$

Az irányított kör miatt nincs topologikus ütemezés!



Az ütemezés megelőzési gráfja

Miért működik a gráfon alapuló tesztelés?

- **Állítás** (egyik irány): Ha van kör a gráfban, akkor cserékkel nem lehet soros ütemezésig eljutni.
- Tegyük fel, hogy létezik $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_n \rightarrow T_1$ kör. Ekkor a feltételezett sorrendben T_1 műveleteinek meg kell előznie T_2 műveleteit és így tovább egészen T_n-ig. Tehát T_n műveletei T_1 műveletei mögött vannak. Ugyanakkor T_n műveleteinek meg kell előznie T_1 műveleteit a $T_n \rightarrow T_1$ él miatt.
- **Ebből következik**, hogy ha a megelőzési gráf tartalmaz kört, akkor az ütemezés nem sorbarendezhető.

Miért működik a gráfon alapuló tesztelés?

- **Állítás** (másik irány): Ha nincs kör a gráfban, akkor cserékkel el lehet jutni egy soros ütemezésig.
- **Alapeset**: Ha n=1, vagyis csak egyetlen tranzakcióból áll az ütemezés, akkor önmagában soros ütemezés, tehát konfliktus-sorbarendezhető.
- **Indukció**: Legyen S a $T_1, T_2, \dots, T_n$ tranzakciók műveleteiből álló ütemezés, és S-nek körmentes megelőzési gráfja van. Ha egy véges gráf körmentes, akkor van olyan csúcsa, amelybe nem vezet él. Legyen T_i egy ilyen csúcs.
- Ennek a tranzakciónak minden műveletét átmozgathatjuk az S ütemezés legelejére (mivel nem vezet hozzá él egyetlen más csúcsból sem). Az így kapott ütemezés:

$$S_1: (T_i\ műveletei)(a\ többi\ n-1\ tranzakció\ műveletei)$$

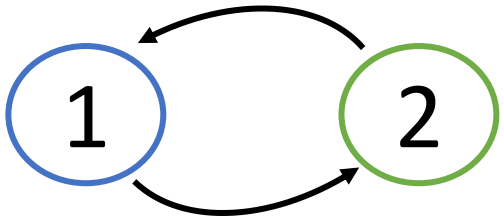
Bizonyítás folytatása

$$S_1: (T_i\ műveletei)(a\ többi\ n-1\ tranzakció\ műveletei)$$

- Most tekintsük S_1 második részét. Mivel ezek a műveletek egymáshoz viszonyítva ugyanabban a sorrendben vannak, mint ahogyan S-ben voltak, ennek a második résznek a megelőzési gráfját megkapjuk S megelőzési gráfjából, ha elhagyjuk belőle az i csúcsot és az ebből kimenő éleket
- Mivel az eredeti gráf körmentes volt, az **elhagyás után is az marad**.
- Ezután alkalmazzunk a maradék $n-1$ tranzakcióra az indukciós felvetést.
- **Végezetül** eljutunk egy soros ütemezéshez, amely az eredeti *S* ütemezés összes tranzakcióját tartalmazza.

Megjegyzések

- Ha S_1 és S_2 ütemezések konfliktusekvivalensek $\Longrightarrow$ S_1 megelőzési gráfja megegyezik S_2 megelőzési gráfjával.
  - Feladat: bizonyítás végig gondolása.
- Ha S_1 megelőzési gráfja megegyezik meg S_2 megelőzési gráfjával $\nRightarrow$ S_1 és S_2 ütemezések konfliktusekvivalensek.
- Példa:
- S_1: $w_1(A); r_2(A); w_2(B); r_1(B)$
- S_2: $r_2(A); w_1(A); r_1(B); w_2(B)$
- S_1 és S_2 megelőzési gráfja megegyezik, mégsem konfliktus-ekvivalensek:



Sorbarendezhetőség elérése

Az ütemező **két megközelítést** használhat a sorbarendezhetőség elérésre:

- **Passzív módszer**:
  - Hagyjuk a rendszert működni.
  - Az ütemezésnek megfelelő megelőzési gráfot tároljuk.
  - Időnként megnézzük, hogy van-e benne kör vagy sem.
  - Ha nincs, akkor örülünk, mert jó az ütemezés.
- **Aktív módszer**:
  - Az ütemező beavatkozik és megakadályozza, hogy kör alakuljon ki.
- A rendszerek **aktív módszert** használnak, azaz olyan ütemezéseket kényszerítenek ki, amelyek konfliktus-sorbarendezhetőek.

Eszközök sorbarendezhetőség elérésére

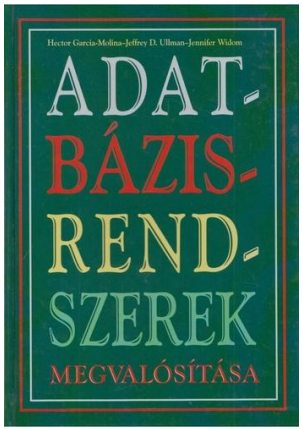
- Az ütemezőnek különböző lehetőségei vannak arra, hogy kikényszerítse a sorbarendezhető ütemezéseket:
  - Zárak
  - Időbélyegek
  - Érvényesítés
- **Fő elv**: inkább legyen szigorúbb és ne hagyjon lefutni egy olyan ütemezést, ami sorbarendezhető, mint hogy fusson egy olyan, ami nem az.

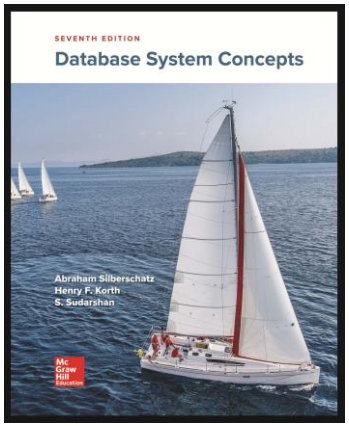
Tankönyv fejezetek

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

  Adatbázisrendszerek megvalósítása

  - 9.1 és 9.2 fejezetek: Konkurenciavezérlés
- Silberschatz, Korth, & Sudarshan: **Database System Concepts**
  - Chapter 17. Transactions





ELTE | IK