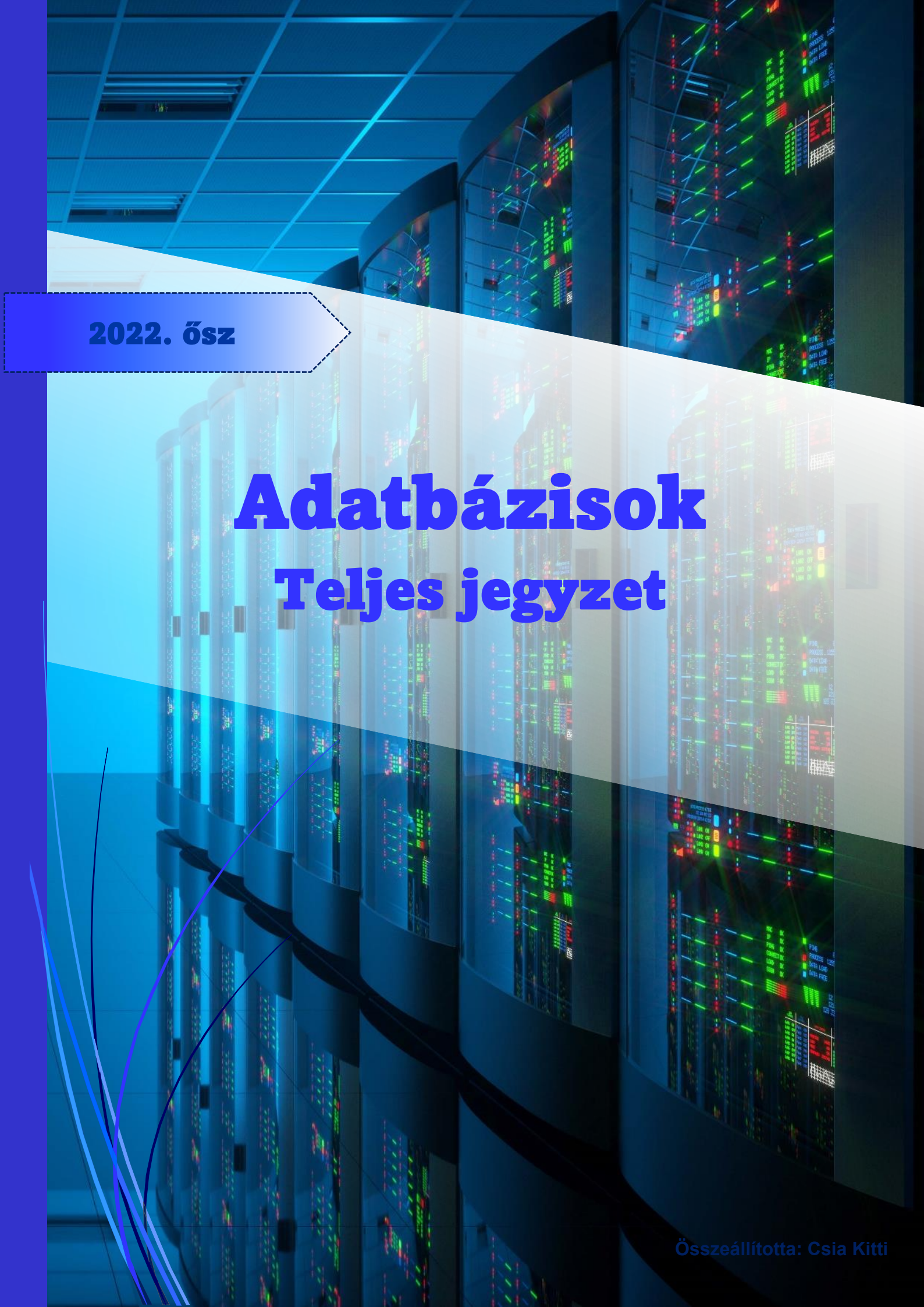


2022. ősz

Adatbázisok

Teljes jegyzet

Összeállította: Csia Kitti

Tartalomjegyzék

Felhasznált irodalom:

Gajdos Sándor – Adatbázisok (2019)

Gajdos Sándor – Relációs lekérdezések optimalizálása laborfelkészítő – előadás/diavetítés (2014)

Engedy Balázs – Relációs lekérdezőnyelvek gyakorlat feladatmegoldások

Saját előadásjegyzet

Talált **HIBA** esetén jelzés: nospatium@gmail.com

BACK

1. előadás: Adatbázis-kezelők felépítése

1. Alapdefiníciók

- Definíció
  - ***adat***
    - nem értelmezett, de értelmezhető darabja a valóságnak (38)
  - ***információ***
    - értelmezett adat (38-as cipőméret)
  - ***tudás***
    - kontextusba helyezett információ (potenciális vásárlónak 38-as a cipőmérete)
  - ***metaadat***
    - adat adata
  - ***szintaxis***
    - adatok ábrázolási módja
  - ***szemantika***
    - valaminek a jelentése
  - ***strukturált adat***
    - ahol a szintaxis megegyezik a szemantikával
  - ***szemistrukturált adat***
    - a szintaktika nem egyezik meg a szemantikával (pl. JSON, XML)
  - ***nemstrukturált adat***
    - nincs szemantika, így nincs szerkezete sem (kép, videó → kezelhetőek bitfolyamként)

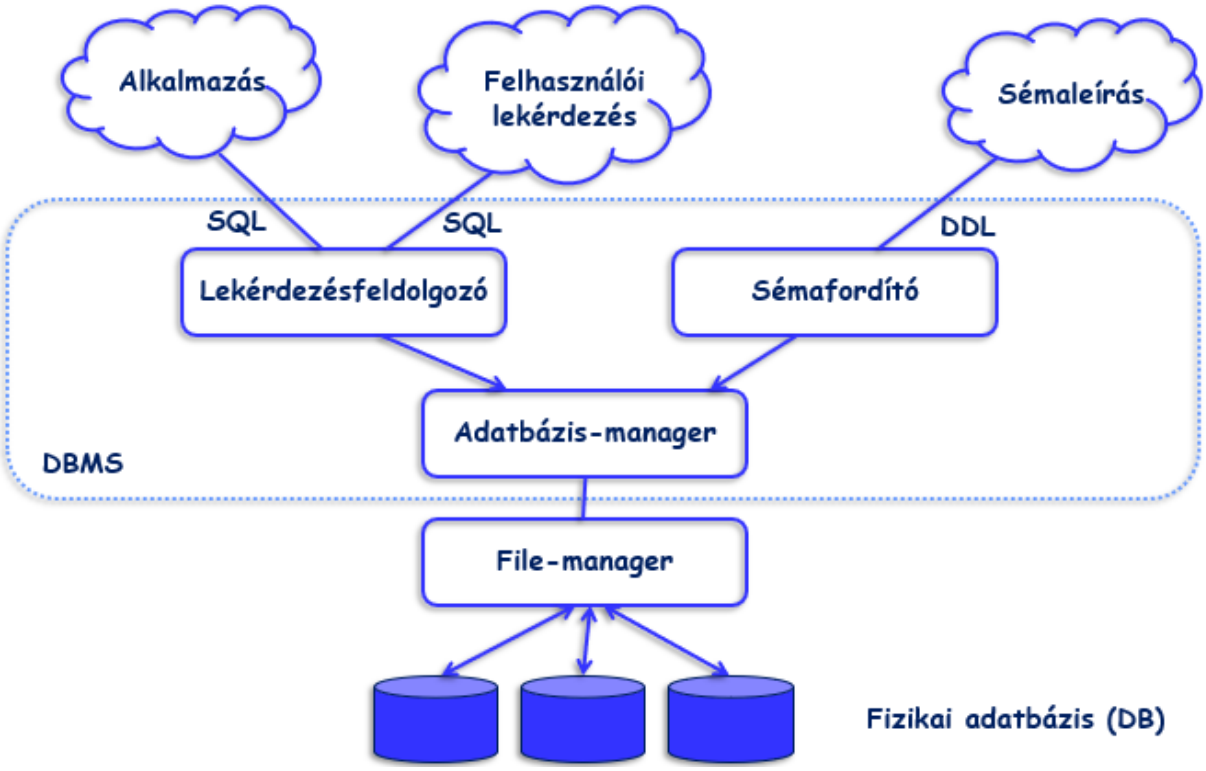
2. Adatbázis-kezelő rendszerek[1]

- Definíció
  - ***adatbázis***
    - valós világ részhalmazának leírásához használt adatok összefüggő, rendezett halmaza
  - ***adatbázis-kezelő rendszer***
    - hardver-szoftver rendszer
    - 1 vagy több személy számára
    - *magas szinten*[2]
    - lehetővé teszi az adatok olvasását, módosítását
- Tulajdonságok
  - ***nagy adatmennyiség***
  - ***gazdag struktúra***
    - rekordok között logikai kapcsolat hozható létre
    - →adatbázisműveletek felgyorsíthatja
  - ***hosszú életciklus***
    - pl. népességi adatbázisok
      - hosszú ideig fent kell maradni, sok technikai váltást túl kell élnie
- **tárolás manapság**:
  - mágnesszalag (kisebb részt)
  - HDD
  - SSD

[1] *Database Management System (DBMS)* – adatbázis alatt általában fizikait értjük
[2] Felhasználó anélkül tudja elvégezni a teendőit, hogy tudatában legyen a háttérben futó folyamatoknak.

3. Programozói/ felhasználói szemlélet

- „klasszikus” adatbázisrendszerek **két fázisa**
  - ***majdani adatok tárolási rendje ~ támogató: DDL***3
    - *séma*[4] megteremtése
    - adatai: technikai *metaadatok* → adatbázis különösen védett részébe kerülnek
      - elvesztésük/sérülésük miatt az adatbázis elérhetetlen lesz



  - ***adatok tárolása, lekérdezése ~ támogató: DML***5
    - *lekérdezés*[6]
      - speciális adatbázis-lekérdező nyelven (pl. SQL) kérdéseket fogalmaz meg (valaki)
      - *interpreter*[7] értelmezi
        - optimalizálás is szükséges: egy esethez több úton keresztül is el tudunk jutni

[3] *Data Definition Language* – megfogalmazhatjuk, hogy milyen adatokat milyen formában tárolunk.
[4] Adatbázis fogalmi váza, más néven struktúrája.
[5] *Data Manipulation Language* – kérések megfogalmazása, értelmezése.
[6] Lekérdezések önálló programként, alkalmazásként is működnek.
[7] Jelen esetben: értelmező, lekérdezés-feldolgozó.

- adatbázis-kezelő válaszol
- lekérdezés-feldolgozó alakítja értelmezhető formába a DB-manager számára → ilyenkor optimalizál is

- DDL és DML gyakran egy egységes nyelvként dolgoznak: pl. SQL
- *állománykezelő* biztosítja a hozzáférést a fizikai adatbázishoz → + szoros kapcsolatban az OS-szel

4. DBM járulékos feladatai

- az adatbázis-manager további feladatai (*PISS*)
  - ***adatvédelem (privacy)***
    - különböző hozzáférési módok vannak
    - jelszóhoz is köthető a hozzáférés vagy célhardver is védheti az adatokat
  - ***integritás[8] (integrity)***
    - olyan szolgáltatás, mely a DB adatainak helyességét ellenőrzi
      - beszúrás, törlés, módosítás kényesek a sikeres végrehajtásra
    - DB logikai felépítése is segíthet ezen
    - típusai:
      - *formai ellenőrzés[9]*
        - adott mezőben valóban az engedélyezett érték áll-e
      - *referenciális integritás*
        - egyik helyről kiolvasott adatelemnek meg kell egyeznie más helyről kiolvasott adatelemmel
      - *strukturális integritás*
        - nem sérült-e a feltételezés, amelyre az DB-t építettük
        - gyakori hiba: *egyértelműség[10]* megszűnése

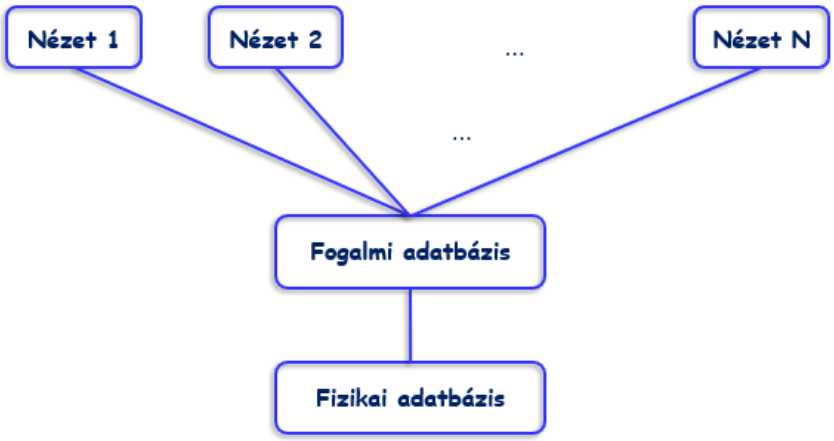
[8] Ellentmondás-mentesség.
[9] Árulkodó jel, ha a testmagasság 3 és fél méter → domain sértés.
[10] Araboknál férfiaknál lehet több feleség is, máshol ez nem egyértelmű. Vagy adatbáziskényszerek, mely miatt az adatok kapcsolatban vannak. Olykor ez egyértelmű, olykor nem. Utóbbiak közé tartoznak a függőségek, amikor egyes adatbázisértékek meghatároznak más értékeket.

- ***adatbiztonság (security)***
  - adatok védelme érdekében, hogy se szoftveres/hardveres hiba esetén se vesszenek el: naplózás, rendszeres mentés, kettőzött adatállomány, elosztott működés
- ***szinkronitás (synchronization)***
  - mai DB-k már többfelhasználósak
  - *tranzakciókezelés*
    - adatokon egyidőben végzett műveletek ne tegyenek keresztbe egymásnak
    - ez biztosítja ezt pl. zárak (*locks)* segítségével

5. Adatbázis-kezelők felépítése

- ***rétegmodell***
  - lényege: probléma több részre bontása, részek épüljenek egymásra, minél kisebb felületen érintkezzenek
  - rétegek egymástól függetlenül megváltoztathatóak legyenek
  - a rétegek közötti interface változatlan marad → ***adatfüggetlenség***
    - *fizikai adatfüggetlenség*[11]
      - fizikai szinten véghez vitt változások nem érintik a logikai DB-t
      - → az adathordozó fizikailag kicserélhető[12], anélkül, hogy bármilyen változás lenne a logikai részben[13]
    - *logikai adatfüggetlenség*[14]
      - logikai DB megváltozása nem jár nézetek megváltozásával
      - → nem mindig teljesül
  - rétegmodellhez példa: *3 rétegű modell*
    - rétegek egymástól függetlenül külön változtathatóak

[11] Eszközfüggetlenség
[12] Pl. fizikai meghibásodás, technikai fejlődés
[13] Tehát nem látunk változást pl. a rendszer teljesítőképességében.
[14] Fogalmi adatfüggetlenség.



- ***fizikai adatbázis***
  - itt valósul meg az adatok fizikai tárolás
  - pl. adatstruktúrák, amelyekben megvalósul az adattárolás
- ***fogalmi adatbázis***
  - való világ egy darabjának leképzése
  - adatok értelmezése
  - ehhez tartozó séma: *fogalmi séma*
- ***nézet***
  - felhasználó az adatbázisból lát
  - több felhasználási lehetőség → több nézet
  - ehhez tartozó séma: *külső séma*

- ***felhasználótípusok***
  - képzetlen felhasználó
  - alkalmazásprogramozó: felhasználói program írása
  - adatbázis admin: jogosultságok, mentés, visszaállítás
  - DBMS tervező/programozó: GOD

BACK

2. előadás:
A fogalmi (logikai) adatbázis

1. Adatmodellek, modellezés

- ***adatbázis létrehozás célja***
  - valós/kitalált világ adatait tároljuk[15]
  - kitalált adatokból információt nyerjünk ki
- adatmodell két részből áll
  - jelölésrendszer, adatkapcsolatok
  - műveletek adatokon
- felhasználó számára legfontosabb: tárolt adatok közötti összefüggések tárolva vannak
  - ábrázolás alapegysége: rekord + típusa
- ***adatmodell meghatározza***
  - adatok milyen struktúrában tárolódnak
  - hogy lehet hozzájuk férni

2. Az entitás-kapcsolat modell

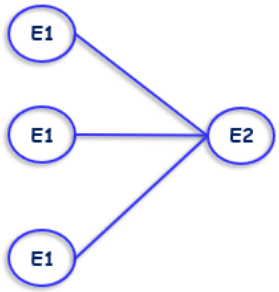
- ER-modell [16]elemei
  - *entitástípus[17]ok*
  - *attribútumtípusok*
  - *kapcsolattípusok*
    - egyes típusokhoz tartoznak *példány[18]*ok is, de a modellezésnél nem kell

[15] Adatok csak egy szűk körét tudjuk csak tárolni.
[16] Nem tekinthető adatmodellnek.
[17] Létező, de hasonló entitások, tulajdonságok, kapcsolatok absztrakciója.
[18] Eset, előfordulás.

- Definíció
  - ***halmaz**/set*
    - azonos tulajdonságú entitások együtt véve
    - halmaz neve: entitás típusa szerint
    - elemei: példányok
  - ***egyed/entitás**/entity*
    - valós világban létező
    - saját léttel rendelkező dolog
    - adatot tárolunk róla
    - FONTOS: megkülönböztethetőnek kell lennie!
  - ***tulajdonság**/property*
    - entitásokat jellemzi
    - ezeken keresztül entitások megkülönböztethetőek
  - ***entitáshalmaz**/entity set*
    - azonos attribútumtípusokkal jellemzett entitások összessége
    - Példa
      - `EMBER(név, szül, szig)`
  - ***kapcsolat**/relationship*
    - entitások névvel ellátott viszonya
    - *kapcsolattípus*: entitás típusok névvel ellátott sorozata
    - Példa
      - `DOLGOZIK: EMBER, CÉG`
      - `ALÁÍR: EMBER, CÉG, SZERZŐDÉS`
      - `TESTVÉR: EMBER, EMBER`
- ***kapcsolatok funkcionalitása/kardinalitása***
  - egy entitáshalmaz egy eleméhez hány entitáshalmaz hány eleme tartozik
  - *egy az egyhez/one-to-one*
    - bináris kapcsolat
    - entitáshalmazok példányai másik entitáshalmaz max 1 példánya van kapcsolatban

- Példa
  - HÁZASSÁG[19]: EMBER, EMBER

- *több az egyhez/many-to-one*



- Példa
  - TANUL: DIÁK, OSZTÁLY

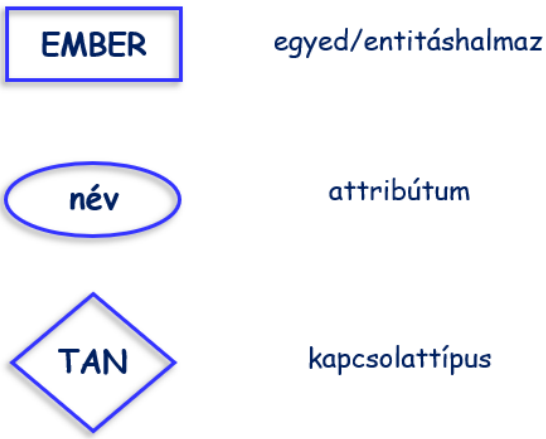
- *több a többhöz/many-to-many*
  - több-több, ha nincs több-egy egyik irányba se
  - Példa
    - TAN[20]: DIÁK, TANÁR

- Definíció
  - ***kulcs***
    - attribútumok azon halmaza, amely az entitást egyértelműen azonosítja
    - Példa
      - EMBER-t egyértelműen tudja azonosítani a személyigazolvány száma
    - ∀ entitáshalmaznak min. egy kulcsa van, hiszen megkülönböztethetőnek kell lenniük
    - tehát az attribútumok teljes halmaza kulcs
    - Jelölés
      - aláhúzással

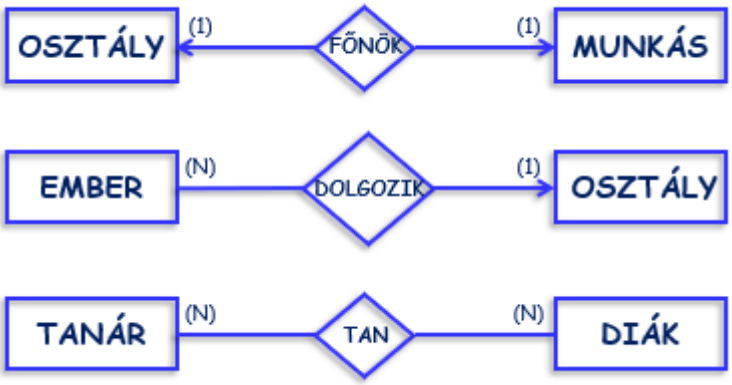
[19] Funkcionalitás meghatározása is modellezési kérdés, mert pl. vannak országok, ahol egy férfinek több felesége is lehet.
[20] Tanul-tanít kapcsolat

2. ER-modell grafikusan: ER-diagram

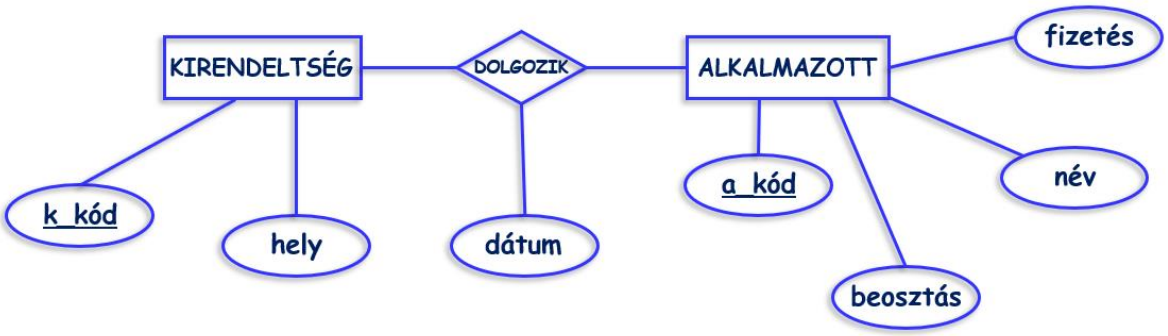
- **jelölésrendszer**



- kapcsolatok funkcionalitása



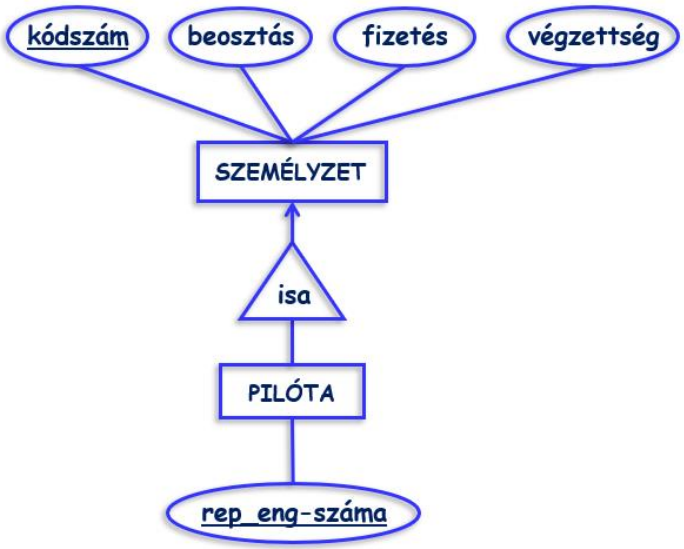
- Példa



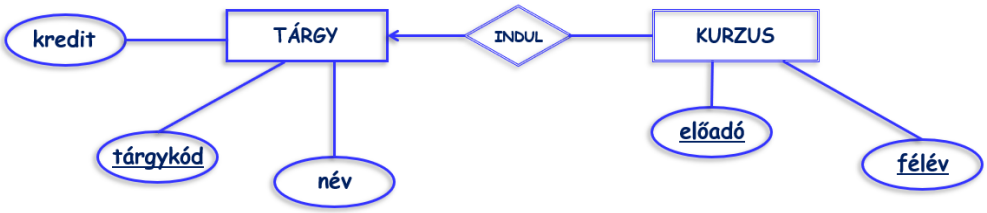
- ebből entitáshalmazok/relációs séma

  KIRENDELTSÉG (<u>k_kód</u>, hely)
  ALKALMAZOTT (<u>a_kód</u>, név, beosztás, fizetés)

- EER[21]-diagram példák
  - ***isa kapcsolat***
    - általánosabb entitáshalmazból konkrétabb entitáshalmaz + speckó egyedi cuccok



  - ***gyenge entitáskapcsolat***
    - entitáshalmaznak nem tudunk kulcsot meghatározni
    - entitást entitással azonosítunk
    - identitását *tulajdonos entitáshalmaz* biztosítja → több-egy kapcsolatban állnak
    - pl. A KURZUS gyenge entitáshalmaz, az INDUL a determináló kapcsolata, ezért a KURZUS példányok egyedisége csak a TÁRGY példányaival együtt biztosítható



[22]

[21] Extended ER
[22] Képen látható KURZUS entitáshalmaznak nincs kulcsa, mert pl. Gipsz Jakab felvett több tárgyat 2012/13-ban is. Kurzusok, mint entitások kurzuskód miatt megkülönböztethetőek.

BACK

3. előadás:
A relációs adatmodell

1. Az adatok strukturálása

- Definíció
  - ***reláció***
    - halmazok Descartes[23]-szorzatának részhalmaza
  - ***domain/tartomány***
    - halmazokban levő értékek domainekből kerülnek ki
    - legyenek ezek: $D_1, D_2, \dots, D_n$
    - Példa

      $D_1 = \{1, 2, 3\} \quad D_2 = \{x, y, z\}$

      $D_1 \times D_2 = \{ (1, x), (2, x), (3, x), (1, y) \dots$
    - reláció lehet az így keletkezettek tetszőleges részhalmaza

      $r_1 = \{(1, z), (2, x)^{24}\} \quad r_2 = \{(3, z), (1, y)\}$
  - ***kardinalitás***
    - oszlopok tartományokhoz rendelése névvel ellátva
    - attribútumok értékeinek száma az attribútum kardinalitása
  - ***relációs séma***
    - milyen relációban milyen attirbútumok vannak
    - Példa

      $R(A_1, A_2, \dots, A_k)$

      $SZEMELY(NEV, KOR, LAKCIM)$
  - ***adatbázis séma***
    - ha az adatbázis több relációs sémát is tartalmaz

[23] Ejtsd: *Dékárt*
[24] Sorrendnek nincs érdemi jelentősége. Hasznos információt az hordozza, hogy a reláció elemeiben kiket tárolunk együtt.

- ***reláció foka***
  - oszlopok száma relációban
- ***reláció számossága***
  - sorok száma relációban
- Szabályok
  - reláció nem tartalmazhat két azonos sort
  - sorok sorrendje nem számít
  - oszlopoknak egyértelmű[25] neve van

2. Műveletek relációkon

- metszetképzés
  - ilyen nem lesz köztük, mert kifejezhető a különbségképzéssel

$$A \cap B = A \setminus (A \setminus B)$$

- ***unió*** *(union)*
  - **feltétele**
    - egyesítendő relációk sémáinak ugyanannyi attribútumból kell állniuk
    - nem szükséges, hogy ténylegesen azonos attribútumok legyenek, max nem tudjuk a sorszámukon kívül névvel is azonosítani
  - Példa

r_1:

A	B	C
a	b	c
c	b	a
a	d	c

r_2:

D	E	F
a	c	d
a	d	c
b	b	c

$r_1 \cup r_2$:

1	2	3
a	b	c
c	b	a
a	d	c
a	c	d
b	b	c

[25] Másiktól egyértelműen megkülönböztethető.

- ***különbségképzés*** *(set difference)*
  - **feltétele**
    - ugyanaz, mint az uniónál
  - Példa

r_1:

A	B	C
a	b	c
c	b	a
a	d	c

r_2:

D	E	F
a	c	d
a	d	c
b	b	c

$r_1 \setminus r_2$:

1	2	3
a	b	c
c	b	a

- ***Descartes-szorzat*** *(Cartesian/cross product)*
  - eredménye $(n_1 + n_2)$-esekből áll, melyeknek első n_1 attribútuma az első operandusból, második n_2 attribútuma a második operandusból áll
  - operandusok szerkezetére nincs kikötés
    - Példa

r_1:

A	B
a	b
b	a

r_2:

A	D
c	d
a	c

$r_1 \times r_2$:

$R_1.A$	B	$R_2.A$	D
a	b	c	d
a	b	a	c
b	a	c	d
b	a	a	c

- ***vetítés*** *(projection[26])*
  - reláció összes rekordjának egyes attribútumait megtartjuk, a többit pedig töröljük
  - Jelölés
    - $\pi_{OSZLOPNEVEK}$
  - **feltétele**
    - kijelöljük, mely attribútumokat akarjuk felhasználni
    - ismétlődéseket megszüntetjük

[26] Tábla oszlopainak kiválasztása.

- Példa

  $GEPKOCSI(AR, RENDSZAM, TIPUS, FOGYASZTAS)$

  - ebből a vetítés:

  $\pi_{AR,RENDSZAM,TIPUS}(gepkocsi)$

  - a vetítés után ez a három oszlopnév fog nekünk megmaradni, és a hozzájuk tartozó sorok

- ***kiválasztás*** *(selection)*[27]
  - részhalmaz képzése az r reláción
  - vezérlésére logikai formula szolgál
  - Jelölés
    - σ_F[28](r)
  - **feltétele**
    - a jelölésben használt F logikai formula [29]ezeket tartalmazhatja
      - konstansok
      - R attribútum azonosítói
      - aritmetikai összehasonlító operátorok $(< = > \leq \geq)$
      - logikai operátorok $(\wedge \vee \neg)$[30]
  - Példa

    $NEVSOR(NEV, KOR, SZIG)$

    - ebből a kiválasztás:

    $\sigma_{KOR < 23 \wedge NEV='KOVACS'}(nevsor)$

    - a szelekció után csak azoknak a halmazát jeleníti meg, akikre igaz, hogy a koruk kevesebb, mint 23 és a név attribútuma pedig Kovács

- ***természetes illesztés*** *(natural join)*
  - **feltétele**
    - adott két reláció, melyeknek min. van egy vagy több megegyező attribútuma

[27] Tábla sorainak kiválasztása.
[28] Ejtsd: *szigma*
[29] Kvantormentes
[30] ÉS VAGY NEGÁLT

- válasszuk ki a megegyező nevű attribútumokat, melyek érték szerint is megegyeznek
- ezeket fűzzük rekordokká, ∀ érték 1x szerepelhet
- ez lesz a természetes illesztés eredménye

▪ Jelölés
- $r \bowtie s$

▪ Példa (1)

r:

A	B
a	b
a	c
b	b

s:

A	C
a	c
b	c

$r \times s$:

A	B	A′	C
a	b	a	c
a	b	b	c
a	c	a	c
a	c	b	c
b	b	a	c
b	b	b	c

$\sigma_{R.A=S.A}(r \times s)$:

A	B	A′	C
a	b	a	c
a	c	a	c
b	b	b	c

$\pi_{ABC}\sigma_{R.A=S.A}(r \times s) = r \bowtie s$:

A	B	C
a	b	c
a	c	c
b	b	c

[31]

▪ Példa (2)

$$OSZTALY(NEV, EVFOLYAM, MELYIK\_OSZT)$$

$$DIAK(NEV, LAKCIM, SZULETES)$$

$$\pi_{NEV,LAKCIM}\sigma_{EVFOLYAM=7}(osztaly \bowtie diak)$$

○ ***θ-illesztés*** *(theta join)*

▪ θ egy kvantormentes feltétel

▪ táblák Descartes-szorzatából a rekordpáron értelmezett θ feltétel szerint választunk ki sorokat:
- $t = \sigma_\theta(r \times s)$

▪ t reláció állítja elő a θ-illesztést

[31] Közös nevű attribútumok hiányában ez csak egy Descartes szorzat lenne.

- Jelölés
  - $r \overset{\bowtie}{_\theta} s$
- Példa

r:

A	B	C
a	b	c
a	a	b
a	d	e
b	c	e
c	d	a

s:

D	E	F
b	c	d
b	c	e
a	e	f
b	e	a
d	a	f

$r \overset{\bowtie}{_{B=D}} s$:

A	B	C	D	E	F
a	b	c	b	c	d
a	b	c	b	c	e
a	b	c	b	e	a
a	a	d	a	e	f
a	d	e	d	a	f
c	d	a	d	a	f

$r \overset{\bowtie}{_{B<D}} s$:

A	B	C	D	E	F
a	a	d	b	c	d
a	a	d	b	c	e
a	a	d	b	e	a
a	b	c	d	a	f
a	a	d	d	a	f
b	c	e	d	a	f

- ***hányados*** *(division)*
  - **feltétele**
    - jelölje $r \div s$ azt a relációt, ahol igaz, hogy s-sel alkotott Descartes-szorzat a legbővebb részhalmaza r-nek
  - Jelölés
    - $r \div s$
  - Példa

r:

A	B	C	D
a	b	a	c
a	b	c	d
a	b	d	c
e	f	a	c
e	f	c	d
e	f	a	d

s:

A	B
a	b
e	f

$r \div s$:

C	D
a	c
c	d

BACK

4. előadás: Relációs lekérdező nyelvek

1. Bevezetés

- Definíció
  - ***sorkalkulus***
    - lekérdező nyelvek részben algebrai, részben logikai
  - ***kvantor***
    - sorvektor változókat kvantifikálhatnak
    - kvantorok:
      - $\forall$ - ezzel a szimbólummal jelölt „minden” szó
      - $\exists, \nexists$ - ezekkel a szimbólumokkal jelölt „*létezik*” „*nem létezik*” szó
  - ***szimbólum***
    - legalacsonyabb szintű építőelem a nyelvben
    - sor-/oszlopkalkulus milyen „karakterek” sorozata lehet
    - Példa
      - ( ) - zárójelek
      - $<, >, =, \neq, \leq, \geq$ - aritmetikai műveletek
      - $\wedge \vee \neg$ - logikai műveleti jelek
      - $s^{(n)}[i]$ – sorváltozók komponensei
      - $R^{(m)}, m$ – (konstans) relációk
      - c - konstansok
      - $\forall, \exists$ - kvantorok
  - ***atom***
    - legkisebb elemek, amelyekhez igazságérték rendelhető
    - Példa
      - $R^{(4)}, (v^{(4)})$ atomhoz igaz értéket rendelünk, ha $v^{(4)} \in R^{(4)}$

- ***(összetett) formula***
  - legmagasabb szint, amelyhez igazságértéket rendelünk
  - atom egyben formula is
  - formulák + ∧ ∨ = formula
  - vannak olyan formulák, akik kvantor hatáskörben állnak
    - ♦ szabad változók: értéküket külön meg kell adni, hogy bool értékké ki lehessen értékelni
- ***kifejezés***
  - legfelső szint
  - nem igazságértéket, hanem halmazt rendelünk hozzá
  - jobb oldalon (a | után) szereplő formula szabad változóinak olyan behelyettesítési-érték kombinációt tartalmazza, amelyre igaz a formula

2. Sorkalkulus[32]

Munkahelyek reláció

1: név	2: foglalkozás	3: munkahely	4: kereset
Aladár	Asztalos	OBI	1001
Béla	Rendőr	BRFK	1002
Cecil	Fogorvos	Rendelő	1003
Dezső	Tanár	BME	1004
Elemér	Tanár	ELTE	1005
Judit	Fodrász	BjutiSzalon	1006
Kata	Tanár	ELTE	1007
Lilla	Igazgató	BjutiSzalon	1008
Mariann	Rendőr	BRFK	1009
Nóra	Műkörmös	BjutiSzalon	1010

Házasságok reláció

1: férj	2: feleség
Aladár	Judit
Béla	Mariann
Elemér	Nóra

Nők reláció

1: név
Judit
Kata
Lilla
Mariann
Nóra

- **Tétel**
  - ∀ az $R_k^{(f_k)} \subseteq A^{f_k}$, $(k = 1, 2, \ldots r)$ relációkból felépített E relációs algebra kifejezéshez ∃ olyan $\Psi^{33}(s^{(m)})$ sorkalkulus formula, hogy Ψ csak az $R_k^{(f_k)}$-k közül tartalmaz relációkat és az E kifejezés megyegyezik az $\{s^{(m)} | \Psi(s^{(m)})\}$ kifejezéssel

[32] Szép, teljes leánykori nevével: relációs sorkalkulus, de mostantól csak sorkalkulus
[33] Ejtsd: *pszí*

- *feladatokon keresztüli értelmezés következik*
- **Kik a relációkban szereplő férfiak**
  - névlista kell nekünk → eredményhalmaz egydimenziós n-esek lesznek
  - ∀ férfi szerepel az M relációban, így:

$$\{s^{(1)} | \exists\, u^{(4)}\ M^{(4)}(u^{(4)}) \wedge u[1] = s[1] \wedge \neg\, (\exists\, v^{(1)}\ N^{(1)}\ (v^{(1)}) \wedge v[1] = s[1])\}$$

  - *no, ezt akkor boncoljuk fel:*
  - az eredményhalmazban azok az $s^{(1)}$ értékek szerepelnek, melyek igazak a következő állítások (*részformulák*)
    - $\exists\, u^{(4)}\ M^{(4)}(u^{(4)}) \wedge u[1] = s[1]$: [34]
      - szerepel az $M^{(4)}$ relációban egy olyan négyes, aminek az első komponense $s^{(1)}$, azaz →
      - az $M^{(4)}$reláció első oszlopában szerepel $s^{(1)}$, tehát $s^{(1)}$ egy név
    - $\neg\, (\exists\, v^{(1)}\ N^{(1)}\ (v^{(1)}) \wedge v[1] = s[1])$[35]
      - nem szerepel az $N^{(1)}$ relációban egy olyan érték, aminek az első (és itt egyetlen) komponense $s^{(1)}$első komponensével egyezik meg, azaz →
      - az $N^{(1)}$ reláció első (és egyetlen) oszlopában nem szerepel $s^{(1)}$, tehát $s^{(1)}$ nem egy női név
      - tehát, ha egy név női név, akkor nem férfi
- **Kik az egyedülálló nők**
  - ugyanaz, mint az előző, csak pont fordítva

$$\{s^{(2)} | (\, H^{(2)}(s^{(2)})) \wedge (\exists v^{(4)} M^{(4)}(u^{(4)}) \wedge M^{(4)}(v^{(4)}) \wedge u[1] = s[1] \wedge v[1] = s[2] \wedge u[4] < v[4])\}$$

[34] Csábító lenne leírni, hogy $M^{(4)}(s^{(1)})$, ez szintaktikai hibás, mivel a reláció és a sorváltozó dimenziója nem egyezik meg, ilyen atomot a sorkalkulus nyelve nem engedélyez.

[35] Itt már lehet, hiszen azonos dimenziójúak: $\neg\, N^{(1)}(s^{(1)})$

- **Mely házaspároknál keres jobban a nő**

$$\{s^{(1)} \mid \neg(\exists\, u^{(2)}\, H^{(2)}(u^{2}) \wedge u[2] = s[1] \wedge\ N^{(1)}\,(s^{(1)})\}$$

  - az eredményhalmazban azok az $s^{(2)}$ értékek szerepelnek, melyek igazak a következő állítások
    - $H^{(2)}(s^{(2)})$
      - vizsgált $s^{(2)}$ a $H^{(2)}$ reláció eleme, tehát házaspárról van szó
    - maradék része...
      - ∃ az M(4) relációnak két olyan sora, hogy az egyik („név” alapján) az adott házaspár férfi tagjához, a másik pedig a női tagjához tartozik
      - a feleséget leíró négyesben a kereset értéke több, mint a férjet leíróban
      - tehát az s(2) házaspár nő tagja többet keres a férfinél.

- **Melyek azok a foglalkozások, amelyeket ketten űznek**

$$\{s^{(1)} \mid (\exists\, u^{(4)}, v^{(4)} M^{(4)}(u^{(4)}) \wedge M^{(4)}(v^{(4)}) \wedge u[1] \neq v[1] \wedge u[2] = v[2] \wedge s[1] = u[2]\}$$

  - az eredményhalmazban azoknak az $s^{(1)}$ foglalkozásnak kell szerepelni, melyekhez találunk az M relációban két olyan $u^{(4)}$ és $v^{(4)}$ sort, hogy bennük azonos foglalkozás, de különböző ember neve legyen

- **Melyek azok a foglalkozások, amelyeket csak egy-egy ember űz**

$$\{s^{(1)} \mid (\exists\, u^{(4)} M^{(4)}(u^{(4)}) \wedge s[1] = u[2]) \wedge \neg \Phi^{36}(s^{(1)})\}$$

  - előző halmaz komplementere az összes foglalkozásra nézve
    - ez a rész azért kell, hogy valóban csak azokat a foglalkozásokat kapjuk meg
  - a formulát negáljuk ($\Phi(s^{(1)})$: előző feladatban szereplő formula)

[36] Ejtsd: *fí*

3. Oszlopkalkulus[37]

- ***változások* az oszlopkalkulusnál:**
  - itt vektorváltozók ($s^{(n)}$) helyett skalárváltozók (u, v, w …) lesznek
    - pl. $s^{(n)}$-t n darab oszlopváltozóval ($u_1, u_2, \dots, u_n$) helyettesítjük[38]
    - $R^{(n)}(u_1, u_2, \dots, u_n)$ jelölhetjük, hogy az u_i-kból képzett n-nes eleme az R relációnak
  - egyetlen sorváltozó helyett oszlopváltozókkal adjuk meg az eredményhalmazt
    - $\{u_1, \dots, u_n \mid \Psi(u_1, \dots, u_n)\}$
    - eredményben pontosan azok a $(u_1, \dots, u_n)$ n-nesek lesznek, melyeknek u_i komponenseit (egyszerre) Ψ-be helyettesítve igazat ad
- **Tétel**
  - rögzített A interpretációs halmaz és $R_k^{(n_k)} \subseteq A^{n_k}$ relációk esetén:
    - sorkalkulus bármely kifejezéséhez ∃ az oszlopkalkulusnak olyan kifejezése, amely az előzővel azonos relációt határoz meg
- *feladatokon keresztüli értelmezés következik*
- **Kik a relációkban szereplő férfiak**

$$\{t \mid \left(\exists\, u, v, w : M^{(4)}(t, u, v, w)\right) \wedge \neg N^{(1)}(t)\}$$

  - az eredményhalmazban azok az t értékek szerepelnek, melyek igazak a következő állítások (*részformulák*)
    - $\exists\, u, v, w : M^{(4)}(t, u, v, w)$
    - szerepel az $M^{(4)}$ relációban olyan négyes, melynek első komponense t, tehát szerepel benne, így t egy név
    - $\neg N^{(1)}(t)$

[37] Az, hogy sor – vagy oszlopkalkulust használunk, az teljesen ránk van bízva, hacsak a feladat nem köti ki, hogy melyiket kell használni. Általában érdemes azt, amelyikkel az eredmény rövidebb lesz.
[38] Skalárok rendezhetősége, kombinálhatósága miatt sokkal rugalmasabb kifejezéseket is kreálhatunk.

- reláció első és egyetlen oszlopában nem szerepel t, tehát t nem női név

- **Kik a házasságban élő tanárok**

$$\{t \mid (\exists\, u,v,w:\ M^{(4)}(t,u,v,w) \wedge u = 'tanar'^{39}) \wedge (\exists\, s: H^{(2)}(s,t) \vee H^{(2)}(t,s)\}$$

  - az eredményhalmazban azok az t értékek szerepelnek, melyekre tudunk találni olyan u,v,w-t, hogy az ezekből alkotott négyes az $M^{(4)}$ reláció sorát kézpik, ahol a második attribútum 'tanár'
  - illetve tudunk t-hez egy olyan s-et találni, hogy valamilyen sorrendben benne legyen a H relációban
  - amelyik t-kre mindkét állítás igaz, azok lesznek a házas tanárok

- **Bjútiszalonban dolgozó nők férjei**

$$\{t \mid (\exists\, s,u,v,w:\ M^{(4)}(s,u,v,w) \wedge v = 'Bjutiszalon') \wedge H^{(2)}(t,s)\}$$

  - az eredményhalmazban azok az t férfinevek szerepelnek, melyekre tudunk találni olyan s,u,v,w-t
  - az ezekből alkotott négyes olyan dolgozó adatait adja meg, akinek a munkahelye a 'Bjútiszalon'
  - továbbá, olyan házastársi kapcsolat „feleség" oldalát adja, ahol a férj a t

3. Biztonságos sorkalkulus

- Definíció

1. Egy $\{t|\Psi(t)\}$ kifejezés biztonságos, ha:

  - $\forall t$ esetén, ahol $\Psi(t)$ igaz, t $\forall$ komponense $DOM(\Psi)$-beli
  - a részformulák biztonságosak, azaz Ψ $\forall$ $\exists$ $u\omega(u)$ részformulájára fennáll, hogy $\forall$ u igaz az ω-beli szabad változók valamely értéke mellett, akkor u $\forall$ komponense $DOM(\omega)$-beli

[39] Ez amúgy 'tanár' akar lenni, csak a betűtípus nem tartalmazza az ékezetes betűket.

2. ***Egy $\Psi(t)$ formula $DOM(\Psi)$ doménje egy olyan halmaz, amely tartalmazza:***
    - a $\Psi(t)$-ben található alaprelációk valamennyi attribútumának értékeit
    - a $\Psi(t)$-ben előforduló konstansokat
  - fontos:
    - ha egy kifejezés biztonságos, akkor véges sok helyettesítés teszi igazzá, és végtelen sok nem
    - → ha egy biztonságos kifejezés formuláját negáljuk, kifejezés már nem biztonságos
    - fordítva nem igaz!
      - ha egy nem biztonságos formulát negálunk, akkor az nem feltétlen lesz biztonságos

- **Tétel**
  - a relációs algebra és a biztonságos sorkalkulus kifejezőereje ekvivalens
- **kvantor nélküli kifejezések**
  - $\{s^{(m)} | s^{(m)}[1] < 3\}$
    - nem biztonságos
      - végtelen olyan sok ennes van, melynek első komponense 3-nál kisebb
      - → eredményhalmaz végtelen
      - sérül: 1. feltétel
      - domain: $\{3\}$ halmaz
  - $\{s^{(m)} | R^{(m)}(s^{(m)}) \wedge s^{(m)}[1] < {}^{40}3\}$
    - biztonságos
      - formulában van egy ÉS kapcsolat
      - → csak az $R^{(m)}$-beli $s^{(m)}$-ek elégítik ki
      - ezek elemei definíció szerint benne vannak a formula domainjében[41]
  - $\{s^{(m)} | \neg R^{(m)}(s^{(m)}) \wedge s^{(m)}[1] = 3\}$

[40] Ugyanez a helyzet, ha itt egyenlőség lenne vagy ha a vége így nézne ki: $\neg(s^{(m)}[1] = 3)$
[41] Hiszen tartalmazza a benne szereplő alaprelációk minden elemét.

- függően biztonságos
  - ha $m = 1$, akkor biztonságos, mert s, akkor csak a 3-at veheti fel
  - ha $m > 1$, akkor már nem biztonságos, mert akkor csak az s első komponensét köti le, többi lehet bármilyen $R^{(m)}$-en kívüli, így végtelen nagy lehet az eredményhalmaz

- **kvantort tartalmazó kifejezések**
  - $\{x \mid R(x) \wedge (\exists\, z : x = z)\}$
    - nem biztonságos
      - mert, az első feltétel teljesíti, viszont van egy $\exists\, u\omega(u)$ részformula, melynek doménje üres halmaz[42]
  - $\{x \mid (\exists\, z : R(x) \wedge x = z\}$
    - biztonságos
      - mert csak olyan z-kre igaz, melyek R-beliek, így mindkét feltétel teljesül
- **összetett kifejezések**
  - $\Phi(x, y) = R_1(x, y) \wedge y > 0$
    - biztonságos
      - mert $R_1(x, y)$-t tartalmaz ÉS-kapcsolatban
      - R_1 y-okra teljesülhet
      - $\rightarrow$ negáltja nem biztonságos
  - $\Psi(\mathrm{x}, \mathrm{y}) = \neg R_1(x, y) \vee y > 0$
    - nem biztonságos
      - ez az előző negáltja, emiatt nem biztonságos
  - $\Omega(x, y) = R_1(x, y) \vee y > 0$
    - nem biztonságos
      - mert a VAGY-kapcsolat miatt végtelen sok y igazzá teszi
      - ennek a negáltja se lesz biztonságos

[42] Tehát nincs benne se konstans, se reláció.

BACK

5. előadás:
Fizikai adatszervezés I.

1. A fizikai adatbázis

- célja: négy művelet minél gyorsabb elvégzése
  - ennek az ideje függ attól, hogy egy blokk tartalmáért hányszor kell a háttértárhoz fordulni
  - adatokat úgy kell tárolni, hogy a legkevesebb blokkművelettel[43] legyen elérhető
- rekordok blokkhatárt nem lépnek át
  - blokkok nincsenek teljesen kihasználva
  - feltételezzük, hogy igen...
- Definíció
  - **kötött rekord**
    - ha a blokkra mutat pointer
    - rekord csak pointerrel mozdítható
  - **szabad rekord**
    - nem mutat rá pointer
    - segítenek a hatékonyabb kihasználásban
  - **fizikai címzés[44]**
    - megadjuk a blokk fizikai címét (fizikai címe memóriában)
  - **logikai címzés**
    - kulcs-érték megadása
    - → egyedi azonosításra
  - **rögzített formátumú[45] blokkok**
    - fix formátumú mezőt adunk meg

[43] Ez lehet írás vagy olvasás.
[44] Rekord + offset (blokkokon belüli kezdőcím)
[45] Vagy hosszúságú

- **változó formátumú blokkok**
  - változó formátumú mezőt vagy ismétlődő csoportot tartalmaz
  - pl. hálós adatbázis
- *feltételezzük, hogy mezők ugyanabban a sorrendben fordulnak elő, hosszuk is azonos*
  - **header/fejrész**
    - blokk fejléce
    - utána tudjuk, hogy a rekordok egymás után jönnek
    - megkülönbözteti a már „élő”[46] részt a még üresektől
      - élőket egy számlálóval jelző a header részben

$$f_r = \left\lfloor \frac{b}{s_r} \right\rfloor$$

- b: blokkméret
- r: állomány
- s_r: rekordok mérete
- f_r: blokkban elhelyezhető rekordok száma

$$b_r = \left\lfloor \frac{n_r}{f_r} \right\rfloor$$

- b_r: állomány által elfoglalt blokkok száma
- n_r: rekordok száma

2. Heap szervezés (*halom/kupac*)

- tárolási probléma megközelítése
  - adatokat annyi blokkban tároljuk, amennyit a rekordok megkövetelnek
  - nem rendelünk hozzá segédstruktúrát
- Definíció
  - **keresés**
    - tárolás: nem hozunk létre semmi újat
    - feladat: egy rekord azonosítása

[46] Tehát már használatban levő.

- blokkművelet hossza (**lineáris keresés**): $\frac{n_r+1}{2}$

- **törlés**
  - tárolás: gyakori törlés miatt kell *szemétgyűjtést* [47]végrehajtani
    - lemezterületek összegyűjtése, egyesítése
  - feladat:
    - egy rekord azonosítása
    - terület felülírása
    - ha nem kötött, akkor headerben jelezzük, hogy a terület [48]felszabadult
    - megváltoztatott blokk visszaírása a lemezre
  - blokkművelet hossza: $\frac{n_r+1}{2}$ [49]
- **beszúrás**
  - tárolás:
    - 1) a törlés által felszabadított helyeken próbáljuk
    - 2) állomány végén
    - 3) adminnak bővíteni kell az állományt
  - feladat:
    - új adat beillesztése az állományba
    - **rekordok egyediségét biztosító mezők értékei egyediek maradjanak beszúrás után**
  - blokkművelet hossza:
    - változó (tárolás pontjai miatt)
    - célszerű egybefüggő diszkterülettel bővíteni
- **módosítás**
  - tárolás:
    - előzőek majdhogynem együtt

[47] Garbage collector
[48] Subblock
[49] Technikailag ez egy keresés megint.

- feladat:
  - keresés + törlés + a törölt rész ismét felülírása[50]
  - **két teljesen azonos rekord ne legyen az adatbázisban! (*egyediség továbbra is biztosítva*)**
- blokkművelet hossza:
  - előzőek majdhogy nem együtt

3. Hash[51]-állományok

- keresés kulcsának bitmintájából csonkolással kinyerhető a cím, amelyből a keresett rekord megtalálható
- ∀ rekordhoz egyedi cím a hash-függvény segítségével
- Definíció
  - **bucket hashing** (*vödrös hashelés)*
    - felosztjuk az adatállományt B részre
    - ∀ rész min. 1 blokkból áll
    - létrehozunk egy B számból álló *vödörkatalógust*[52]
    - ∀ állomány egy vödör címét tartalmazza
    - legyen egy hash-függvény, ami leképezi a kulcsok értékkészletét
    - lényege: egy rekord, aminek kulcsa K, a $h(K)$ vödörben kell tárolni
      - gyakran használt hash-függvény:
      - $h(K) = (c \cdot K) \bmod B$
- **keresés**
  - folyamat:
    - 1) meghatározzuk a rekord kulcsát[53]: K
    - 2) kiszámítjuk: $h(K)$
    - 3) kiolvassuk a vödör katalógus $h(K)$-adik bejegyzését

[50] Fix beszúrás, mert tudjuk, hogy oda fogunk írni, ahol épp töröltünk. Technikailag ez egyesíti az előző három műveltet.
[51] Hash-címzés vagy csonkolásos címzés.
[52] Hash table, bucket directory.
[53] T.F.H: ez a record számára egyediséget biztosít

        - ezen a címen kezdődő vödörben kell lennie[54]
        - ezzel a technikával az állomány csak $\frac{1}{2B}$-ed részét kellett csak átnézni
    - **törlés**
        - folyamat:
            - 1) megkeressük a rekordot
            - 2) törölt bitet bebillentjük
            - 3) módosított blokkot visszaírjuk a diszkre
    - **beszúrás**
        - folyamat:
            - 1) keresési folyamat végigfuttatása
            - 2) *ha megtaláltuk*, hibaüzenet küldése, hogy ne sérüljön az egyediséget biztosító mező értéke
            - 3) *ha nem*, akkor első szabad/törölt helyre beírjuk, vagy vödörhöz új blokkot fűzünk, és oda helyezzük el
            - 4) beírt hely bitjét átállítjuk
    - **módosítás**
        - folyamat:
            - 1) keresési folyamat végigfuttatása
            - 2) *ha megtaláltuk*, akkor módosítjuk, majd visszaírjuk a diszkre
            - 3) *ha nem találtuk*, akkor hibaüzenet
            - 4) *ha megtaláltuk, de kulcsmező*t érint, akkor a művelet egy törlés és egy beszúrás egymás utáni végrehajtására vezet
                - a módosított rekord egy másik vödörbe fog kerülni

[54] Ha egyáltalán van ilyen rekordunk.

BACK

6. előadás: Fizikai adatszervezés II.

1. Indexelt állományok

- Definíció
  - keresés kulcsát indexállományban megismételjük
  - a kulcshoz egy mutatót rendelünk, ez a tárolt adatrekord helyére mutat

$$f_i = \left\lfloor \frac{b}{k+p} \right\rfloor$$

    - b: blokkméret
    - k: kulcshossz
    - p: pointerhossz
    - i: indexállomány neve
    - f_i: indexállomány blocking faktora
  - **indexállomány**
    - mindig rendezve van, kulcs típusától függ
    - numerikus kulcs
      - növekvő/csökkenő sorrend
    - szöveges kulcs
      - lexikografikus rendezés
    - összetett kulcs[55]
      - kulcs ilyenkor több mezőből áll
      - nekünk kell definiálni, hogy milyen sorrendet szeretnénk → megválasztása befolyásolja index használhatóság hatékonyságát
      - *(keresési)* kulcsokra építjük fel az indexeket
      - indexállomány azonos hosszúságú rekordjai szabadok, így könnyen mozgathatóak
  - indexrekordok megfeleltetése indexállományokhoz

[55] Composite key

- **sűrű index**
  - indexrekordot rendelünk ∀ egyes adatrekordhoz
- **ritka index**
  - indexrekordot rendelünk adatrekordok egy csoportjához, egy blokkban lévőkhöz

2. Ritka indexek[56]

- meghatározza, hogy az adatállomány rekordjai melyik blokkban vannak
- adatállományt is rendezetten kell tárolni
  - egy blokkban kell lennie ∀ adatrekordnak, melynek kulcsa meghatározott intervallumba esik
  - ez az indexrekord tartalmazza: blokk címét, legkisebb/nagyobb értékű kulcs

$$b_{ri} = \left\lfloor \frac{n_{ri}}{f_{ri}} \right\rfloor$$

  - i: indexállomány neve
  - ri: ritka indexállomány blokkjainak száma
  - f_{ri}: ritka indexállomány blokkjainak blocking faktora
  - n_{ri}: ritka indexállomány blokkjainak mérete
  - b_{ri}: ritka indexállomány blocking faktora
- **keresés**
  - folyamat:
    - 1) keresni kívánt rekord kulcsa kulcsunk: k_1
    - 2) megkeressük az őt tároló blokkot: B_i
    - 3) azt a rekordot keressük ki, amire igaz, hogy: $k_1 \leq i \leq k_2$
      - keresés lehet bináris, mert az indexállomány kulcs szerint rendezett
    - 4) k_2 pointere megcímzi azt a blokkot, amelyben rejtőzködik a k_1

[56] Sparse indices

- **törlés**
  - folyamat:
    - 1) törölni kívánt rekord kulcsa: k_1
    - 2) keresési folyamat végigfuttatása B_i-ben
    - 3) ha k_1 kulcs nem a legkisebb, akkor a rekordot töröljük
      - lyukat, meg 'befoltozzuk' többi rekord eltolásával
    - 4) ha a legkisebb, akkor az indexállományt is korrigálni kell B_i-t legkisebb kulcsának megfelelően
      - ha csak k_1 volt az egyetlen rekord a blokkban, akkor az egész rekordot törölni kell, az üres adatblokkot meg felszabadítani
- **beszúrás**
  - folyamat:
    - 1) beszúrni kívánt rekord kulcsa: k_1
    - 2) megkeressük a blokkot, amelyben a rekordnak kell lennie: B_i
    - 3) *ha van hely* a kulcs számára, akkor beszúrjuk a blokkba
    - 4) *ha nincs*, akkor kérünk egy új blokkot: B_n, és megfelezzük a kulcsokat[57] a két blokk között
      - mindkét blokkban meghatározzuk a legkisebb kulcsot
      - új indexrekordot képzünk, esetleg új állományt is
- **módosítás**
  - folyamat:
    - 1) megkeressük a módosítani kívánt rekordot
    - 2) *ha megtaláltuk és nem érint kulcsot*, akkor módosítjuk, majd visszaírjuk a diszkre
    - 3) *ha nem találtuk*, akkor hibaüzenet
    - 4) *ha megtaláltuk, de kulcsmező*t érint, akkor a művelet egy törlés és egy beszúrás egymás utáni végrehajtására vezet

[57] Beleértve a beszúrni kívánt kulcsot is.

3. B*-fák[58], mint többszintes ritka indexek

- heap szervezés $<$ indexelt szervezés $(\log_2 b_i) <$ hash-szervezés[59]
- indexek gyorsítása érdekében: B*-fák
  - indexeket k-ágú fában tároljuk
  - karbantartási, működtetési igénye magas ↑
  - indexeket addig rendezzük, ameddig az utolsó index egyetlen blokkba befér
    - $i-1$: egyidejűleg ritka indexe az i-nek, és adatállománya $i-2$
    - legalsó szint pointerei az adatállomány egy blokkjára mutatnak
  - felsőbb szintek az indexállomány egy részfájára

$$f_i = \left\lfloor \frac{b+k}{k+p} \right\rfloor$$

    - b: blokkméret
    - k: kulcshossz
    - p: pointerhossz
    - i: indexállomány neve
    - f_i: indexállomány blocking faktora
  - az r állományhoz tartozó B*-fa szintjeinek számát HT_i-vel szoktuk jelölni

$$HT_i = \lceil \log_{fi} br \rceil$$

- **keresés**
  - folyamat:
    - 1) keresett rekordunk: v_1
    - 2) az indexállomány csúcsán álló blokkban keresgélünk

[58] Ejtsd: Bé-csillag-fa. Mi kibalanszolt (balanced) fákról fogunk beszélni, melyeknek minden levele és csomópontja pontosan blokkméretű és a gyökértől a levelekig vezető út mindig ugyanolyan hosszú.
[59] Ennek a keresésnek a művelete akár konstans 1 is lehet

- 3) azt a rekordot keressük ki, amire igaz, hogy: $v_1 \leq i \leq v_2$
  - ennek a rekordnak a pointere az eggyel alacsonyabb szintű indexben rámutat arra a blokkra, amelyben a keresést kell folytatni
  - olyan indexrekord után, melyeknek legnagyobb kulcsa v_3, azok közül, melyek $\leq v_1$-nél
- 4) addig folytatjuk, ameddig v_1-ig rekordig nem jutunk

- **törlés**
  - folyamat:
    - 1) megkeressük a kívánt adatot és töröljük
    - 2) adatblokkokat összevonjuk, utolsó rekordját töröljük
    - 3) megszűnt blokkhoz tartozó kulcsot ki kell venni
      - lehet itt a fa ∀ részét kell változtatni
- **beszúrás**
  - folyamat:
    - hasonlít a sima indexeshez, csak a bonyolultságát a fának magával rángatja
- **módosítás**
  - folyamat:
    - előzőhöz hasonlóan

4. Sűrű indexek[60] – több kulcs szerinti keresés

- önmagában nem állományszervezés ☹
  - ráépül egy másik módszerre (ritka/hash)
  - főállományt segíti jobban kezelni

$$b_{si} = \left\lfloor \frac{n_{si}}{f_{si}} \right\rfloor$$

- i: indexállomány neve
- si: sűrű indexállomány blokkjainak száma
- f_{si}: sűrű indexállomány blokkjainak blocking faktora
- n_{si}: sűrű indexállomány blokkjainak mérete

[60] Dense indices

- b_{si}: sűrű indexállomány blocking faktora

- előnyei
  - adatállományt nem kell rendezett tárolni
  - meggyorsítja a rekordkezelést
  - támogatja a több kulcsos keresést
  - adatállomány rekordjai szabaddá tehetőek, ha sűrű indexeken keresztül történik a rekordhivatkozás
- hátrányai
  - nagyobb igény: hely, adminisztráció, karbantartás
  - eggyel több indirekció rekord kiolvasásakor
- **keresés**
  - folyamat:
    - 1) keresett rekordunk: v_1
    - 2) az indexállomány csúcsán álló blokkban keresgélünk
    - 3) azt a rekordot keressük ki, amire igaz, hogy: $v_1 \leq i \leq v_2$
      - ennek a rekordnak a pointere az eggyel alacsonyabb szintű indexben rámutat arra a blokkra, amelyben a keresést kell folytatni
      - olyan indexrekord után, melyeknek legnagyobb kulcsa v_3, azok közül, melyek $\leq v_1$-nél
    - 4) addig folytatjuk, ameddig v_1-ig rekordig nem jutunk
- **törlés**
  - folyamat:
    - 1) megkeressük a kívánt adatot és töröljük
    - 2) adatblokkokat összevonjuk, utolsó rekordját töröljük
    - 3) megszűnt blokkhoz tartozó kulcsot ki kell venni
      - lehet itt a fa ∀ részét kell változtatni
- **beszúrás**
  - folyamat:
    - hasonlít a sima indexeshez, csak a bonyolultságát a fának magával rángatja

- **módosítás**
  - folyamat:
    - előzőhöz hasonlóan

4. Invertálás

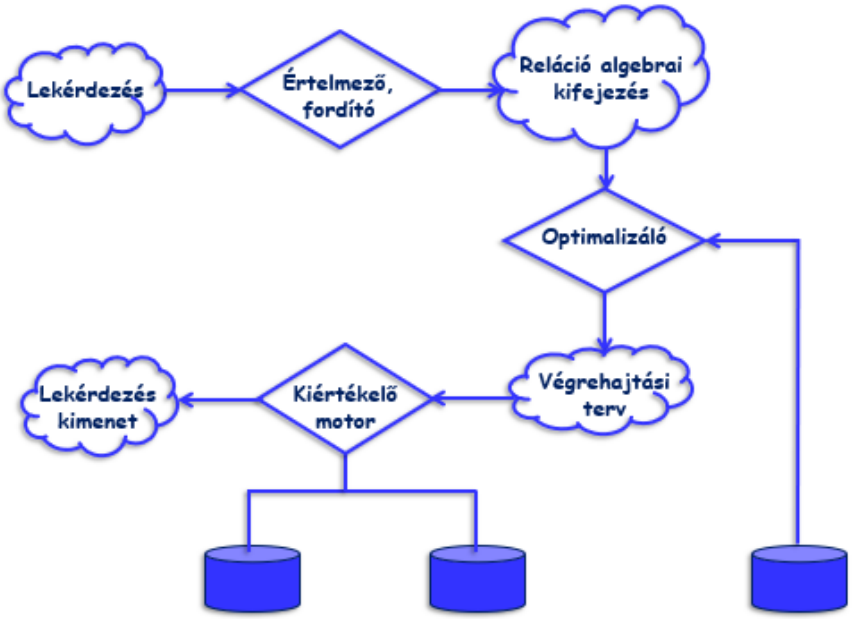
- Definíció
  - **invertált állomány**
    - az az indexállomány, amely nem kulcsmezőre tartalmaz indexeket
- **állomány pointerei**
  - fizikai pointerek
    - közvetlenül az adatállomány megfelelő blokkjára mutat
      - adatállomány blokkjai kötöttek
      - csak egy invertált adatállomány esetén működik
    - vagy az adatállomány elsődleges kulcsa szerinti sűrű indexállomány megfelelő rekordjára
      - eggyel több indirekción keresztül érjük el a keresett rekordot
      - az adatrekordok változtatásakor csak az érintett mezőt tartalmazó invertált index - és sima állományokat kell módosítani
  - logikai pointerek
    - adatállomány valamely kulcsának értékét tartalmazzák
      - adatállomány rekordjai lehetnek szabadok
      - keresett rekord címét még nem ismerjük
      - ennek megtalálását hash vagy indexeléses módszerrel támogatjuk

BACK

7. előadás:
Relációs lekérdezések optimalizálása I.

1. Áttekintés

- célja: adatok adatbázisból való kinyerése
- adatbázis feladatai
  - 1) lekérdezés elemzése
    - az adatbázis motorja elemezni fogja a beérkező lekérdezést, hogy milyen adatokra van szükség, milyen táblákból és milyen kapcsolatokkal kell dolgozni
  - 2) tervek generálása
    - az optimalizálási motor több lehetséges lekérdezési tervet generál, amelyek közül választhat
    - ezek a tervszerűségek különböző módokon kombinálják és végrehajtják a táblák közötti kapcsolatokat
  - 3) költségbecslés/optimalizálás
    - az optimalizálási motor minden lekérdezési tervhez becsüli a végrehajtási költséget, például a szükséges memóriát, a lemezműveletek számát vagy a hálózati forgalmat
  - 4) tervválasztás
    - az optimalizálási motor kiválasztja a legköltséghatékonyabb lekérdezési tervet a becsült költségek alapján
    - ez az a terv, amely a leghatékonyabban hajtja végre a lekérdezést.
  - 5) lekérdezési végrehajtás
    - az adatbázis motorja végrehajtja a kiválasztott lekérdezési tervet, és eredményt ad a lekérdezés alapján



- Definíció
  - **relációs algebrai fa**
    - ha a primitív műveletek a relációs algebra műveletei, akkor a primitív műveletek szekvenciája a relációs algebrai fa

2. Katalógus költségbecslés

- ez csak egy becslés, nem érdemes egzakt számokat elvárni tőle
- a katalógus eltárolja
  - egyes relációkra vonatkozó információkat
  - indexekről információ
  - lekérdezés költsége
  - statisztika (nem mindig friss[61] → rendszer akkor frissíti, ha van rá ideje)
- **katalógusban tárolt relációk**
  - n_r: az r relációban rekordok száma
  - b_r: az r relációban blokkok száma
  - s_r: rekord mérete
  - f_r: blokkban rekordok száma

[61] Statisztika nem mindig konzisztens a rendszer állapotával, de általában egész jól leírja a rendszerben zajló folyamatokat.

- $V(A,r)$: kardinaltiás, hány különböző értéke fordul elő az A attribútumnak az r relációban
  - $V(A,r) = |\pi_A(r)|$
  - ha A kulcs, akkor $V(A,r) = n_r$
- SC[62](A,r): kiválasztásos kardinaltiás, azon rekordok átlagos száma, amelyek egy kiválasztási feltételt kielégítenek
  - $SC(A,r) = \frac{n_r}{V(A,r)}$
  - ha A kulcs, akkor $SC(A,r) = 1$
- ha a relációk rekordjai fizikailag együtt vannak tárolva, akkor:

$$b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$$

- **katalógus információk az indexekről**
  - f_i: pointer kimenetek átlagos száma a fa struktúrájú indexeknél, pl. B*-fáknál
  - HT_i[63]: az indexek szintjeinek száma
    - $HT_i = \lceil \log_{fi} V(A,r) \rceil$ (B*-fa)
    - $HT_i = 1$ $(hash)$
  - LB_i[64]: legalacsonyabb szintű/levélszintű indexblokkok száma
- Definíció
  - **lekérdezés költsége**
    - háttértár blokkolvasások és írások száma a válasz kiírásának költsége nélkül
      - írásba csak a köztes blokkok írását számítjuk bele
- Jelölés
  - E[65]$_{alg}$ – az algoritmus becsült költsége

[62] Selection Cardinality
[63] Height of Tree
[64] Lowest level index Block
[65] Estimate

BACK

8. előadás:
Relációs lekérdezések optimalizálása II.

1. Műveletek költsége

- **kiválasztás/szelekció** *(selection)*
  - alap szelekciós algoritmus
    - $A1$: lineáris keresés (*„full table scan”)*
      - ∀ rekordot beolvasunk, megvizsgáljuk, hogy kielégíti-e a kiválasztást
      - **költsége:**
        - $E_{A1} = b_r$
    - $A2$: bináris keresés
      - blokkoknak folyton a diszken kell lenniük
      - A attribútum szerint rendezettek
      - feltétele az egyenlőség az A attribútumon
      - **költsége:**
        - $E_{A2} = \lceil \log_2(b_r + 1) \rceil + \left\lceil \frac{SC(A,r)}{f_r} \right\rceil$
  - indexelt szelekciós algoritmus
    - elsődleges és másodlagos index elnevezés közt az a különbség, hogy az elsődlegesnél az index lehetővé teszi a rekordokat fizikai tárolási sorrendjükben való beolvasását, ∀ más másodlagos
    - $A3$: elsődleges index, egyenlőségi feltétel a kulcson vizsgálva
      - **költsége:**
        - $E_{A3} = HT_i + 1$
    - $A4$: elsődleges index, egyenlőségi feltétel **nem** a kulcson [66]vizsgálva

[66] A nem kulcs attribútumon van az elsődleges index.

- **költsége:**
  - $E_{A4} = HT_i + \left\lceil \frac{SC(A,r)}{f_r} \right\rceil$ [67]

- $\boldsymbol{A5}$: másodlagos index
  - **költsége:**
    - $E_{A5} = HT_i + SC(A,r)$
    - ha A kulcs, akkor $E_{A5} = HT_i + 1$

- összehasonlítás alapú szelekciós algoritmus
  - eredményrekordok számának becslése
    - ha v-t ismerjük, egyenletes eloszlás esetén

$$n_{avg} = n_r \cdot \frac{v - \min(A,r)}{\max(A,r) - \min(A,r)}$$

    - ha v-t nem ismerjük

$$\frac{n_r}{2}$$

  - $\boldsymbol{A6}$: elsődleges index
    - **költsége:**
      - ha v-t ismerjük $E_{A6} = HT_i + \left\lceil \frac{c^{68}}{f_r} \right\rceil$
      - ha v-t nem ismerjük $E_{A6} = HT_i + \frac{b_r}{2}$
  - $\boldsymbol{A7}$: másodlagos index
    - **költsége:**
      - $E_{A7} = HT_i + \frac{LB_i}{2} + \frac{n_r}{2}$

- **join operáció**
  - Definíció - theta illesztés
    - $r_1 \bowtie_\theta r_2 = \sigma_\theta(r_1 \times r_2)$
  - típusai
    - természetes illesztés:
      $r_1 \bowtie r_2 = \pi_{A \cup B}(\sigma_{R1.X=R2.X}(r_1 \times r_2))$

[67] Teljes blokkművelet.

[68] Azon rekordok számát jelöli, ahol $A \leq v$

- külső illesztés[69]
  - baloldali külső illesztés: $r_1 * (+) r_2$
  - jobboldali külső illesztés: $r_1 (+) * r_2$
  - teljes külső illesztés: $r_1 (+) * (+) r_2$

- nested-loop join[70]

adott két reláció, r és s:

```
FOR ∀ t_r ∈ r rekordra DO BEGIN
    FOR ∀ t_s ∈ s rekordra DO BEGIN
        teszteljük (t_r, t_s) párt, hogy kielégíti-e a θ-joint
        IF igen, THEN adjuk a t_r.t_s rekordot az eredményhez
    END
END
```

  - **költsége:**
    - „worst case” $n_r \cdot b_s + b_r$
    - ha min. egyik belefér a memóriába: $b_s + b_r$

- block nested-loop join[71]
  - hatékonyabb a sima nested-loop joinnál

```
FOR ∀ b_r ∈ r rekordra DO BEGIN
    FOR ∀ b_s ∈ s rekordra DO BEGIN
        FOR ∀ t_r ∈ b_r rekordra DO BEGIN
            FOR ∀ t_s ∈ b_s rekordra DO BEGIN
            teszteljük (t_r, t_s) párt
            END
        END
    END
END
```

[69] Outer join
[70] Egymásba ágyazott ciklikus illesztés.
[71] Blokkalapú egymásba ágyazott ciklikus illesztés

- **költsége:**
  - „worst case” $b_r \cdot b_s$[72] $+ b_r$
  - sok memóriával: $b_s + b_r$
- indexed nested-loop join[73]
  - az egyik relációhoz (s) van indexünk
  - tegyük az első algoritmus belső ciklusába az indexelt relációt
  - →a keresés index alapján kisebb költséggel is elvégezhető
  - **költsége:**
    - c[74] $\cdot n_r + b_r$
- sorted-merge join
  - a relációkat a join feltételben meghatározott attribútumok mentén rendezzük, majd összefésüljük
  - **költsége:**
    - ha nincsenek sorrendben: $b_r \cdot b_s + b_r * \log_2 b_r + b_s * \log_2 b_s$
    - „worst case”: $b_s + b_r$
- hash join
  - az egyik relációt hash-táblán keresztül érjük el, miközben a másik reláció egy adott rekordjához illeszkedő rekordokat keressük
- egyéb
  - pl. bitmap indexekkel (bitmap join)

- **egyéb operációk**
  - ismétlődés kiszűrése
    - rendezés, majd szűrés
  - projekció
    - projekció, majd ismétlődés kiszűrése

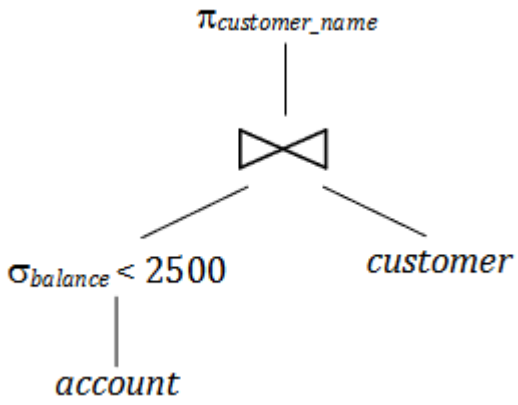
[72] b_r az egyik adatállomány blokkszáma, a b_s pedig a másiké.
[73] Indexalapú egymásba ágyazott ciklikus illesztés.
[74] Szelekció költsége s-en.

- unió
  - mindkét relációt rendezzük
  - összefésülésnél kiszűrjük a duplikációkat
- metszet
  - mindkét relációt rendezzük
  - összefésülésnél csak a másodpéldányokat hagyjuk meg
- különbség
  - mindkét relációt rendezzük
  - összefésülésnél csak az első relációbeli rekordokat hagyjuk meg
- aggregáció
  - pl. rendezés márkanévre
  - összegzés on-the-fly

2. Kifejezés kiértékelése

- Definíció
  - **materalizáció**
    - összetett kifejezésének egyszerre egy műveletét értékeljük ki, valamilyen rögzített sorrend szerint
    - kanonikus alak

      $\pi_{costumer\_name}(\sigma_{balance<2500}(account) \bowtie costumer)$
    - műveleti fa



    - folyamata
      - csomópontokban számítás

- *heavy táblák elmentése későbbre, ne kelljen újraszámolni → csökken a számítási idő később*

- költsége
  - operációk + részeredmények tárolása + visszaolvasás költsége
- előnye
  - egyszerű implementálhatóság
- hátránya
  - sok háttértárművelet

- **pipelining**
  - folyamata
    - nem számítjuk ki az egészet, darabonként csordogál tovább
    - eredmények közvetlenül átad
  - költsége
    - adott reláció
  - előnye
    - kiküszöböli az ideiglenes tárolás szükségességét
    - kis memóriaigény
  - hátránya
    - szűkíti a felhasználható algoritmusok körét

3. Relációs kifejezések transzformációi

- **ekvivalenciai szabályok**
- jelölések
  - $\theta, \theta_1, \ldots$: predikátum
  - L, L_1 ...: attribútumhalmaz
  - $E, E_1, \ldots$: relációs algebrai kifejezések
- szelekció kaszkádosítása
  - $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$
- szelekció kommutativitása
  - $\sigma_{\theta_1}\left(\sigma_{\theta_2}(E)\right) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$

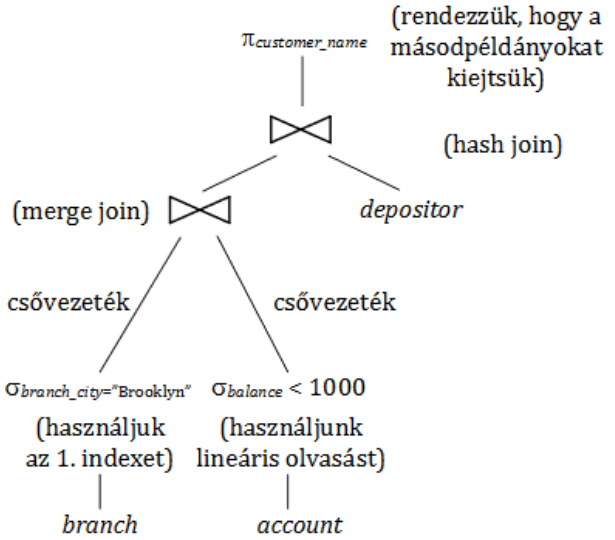
- projekció kaszkádosítása
  - $\pi_{L_1}\left(\pi_{L_2}\left(\ldots\left(\pi_{L_n}(E)\right)\ldots\right)\right) = \pi_{L_1}(E)$
- θ-illesztés és Descartes-szorzat
  - $\sigma_\theta(E_1 \times E_2) = E_1 \overset{\bowtie}{_\theta} E_2$
  - $\sigma_{\theta_1}\left(E_1 \overset{\bowtie}{_{\theta_2}} E_2\right) = E_1 \overset{\bowtie}{_{\theta_1 \wedge \theta_2}} E_2$
- θ-illesztés kommutativitása
  - $E_1 \overset{\bowtie}{_\theta} E_2 = E_2 \overset{\bowtie}{_\theta} E_1$
- természetes illesztés asszociativitása
  - $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$
- a szelekció disztributivitása θ felett, ha a θ_0 csak E_1-beli attribútumokat tartalmaz
  - $\sigma_{\theta_0}(E_1 \overset{\bowtie}{_\theta} E_2) = \sigma_{\theta_0}(E_1) \overset{\bowtie}{_\theta} \sigma_{\theta_0}(E_2)$
- a projekció disztributivitása θ felett, ha a L_1 csak E_1-beli attribútumokat tartalmaz és az illesztés feltételében csak $L_1 \cup L_2$-beli attribútumok vannak
  - $\pi_{L_1 \cup L_2}(E_1 \overset{\bowtie}{_\theta} E_2) = (\pi_{L_1}(E_1)) \overset{\bowtie}{_\theta} (\pi_{L_2}(E_2))$
- metszet és unió kommutativitása
  - $E_1 \cup E_2 = E_2 \cup E_1$
  - $E_1 \cap E_2 = E_2 \cap E_1$
- metszet és unió asszociativitása
  - $(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$
  - $(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$
- a szelekció disztributivitása az unió, metszet és különbség felett
  - $\sigma_\theta(E_1 \cup E_2) = \sigma_\theta(E_1) \cup \sigma_\theta(E_2)$
  - $\sigma_\theta(E_1 \cap E_2) = \sigma_\theta(E_1) \cap \sigma_\theta(E_2) = \sigma_\theta(E_1) \cap E_2$
  - $\sigma_\theta(E_1 \setminus E_2) = \sigma_\theta(E_1) \setminus \sigma_\theta(E_2) = \sigma_\theta(E_1) \setminus E_2$
- a projekció disztributivitása az unió felett
  - $\pi_L(E_1 \cup E_2) = \pi_L(E_1) \cup \pi_L(E_2)$
- **összefoglaló szabályok**
  - konjunktív szelekciós feltételeket szelekciós feltételek

- szelekciós műveleteket felcseréljük a többi művelettel
- átrendezzük a lekérdezési fa leveleit
- a Descartes-szorzatokat és fölöttük lévő szelekciós kapcsolási feltételt egy join műveletté vonjuk össze
- a projekciós műveleteket felcseréljük a többi művelettel

4. Kiértékelési/végrehajtási terv kiválasztása

- Definíció
  - **kiértékelési/végrehajtási terv**
    - fa + algoritmusok + fizikai segédstruktúrák + egymásra épülő műveletek párhuzamosságának ismerete
- **a kiválasztás**
  - milyen műveletek
  - milyen sorrendben
  - milyen algoritmus szerint
  - milyen work flowban

egy konkrét kiértékelési terv



- **költség alapú optimalizálás**
  - ∀ ekvivalens kifejezés felsorolása
  - ∀ forma kiértékelése
  - optimális kiválasztás
  - Példa
    - $r_1 \bowtie r_2 \bowtie r_3 \rightarrow 12$

- általános esetben: n reláció illesztésére $\frac{(2(n-1))!}{(n-1)!}$ ekvivalens lehetősége
- hogy ne legyen nagy terhelés → heurisztikus optimalizálás

- **heurisztikus alapú optimalizálás**
  - lekérdezési fa

```
SELECT LAST_NAME FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE EMPLOYEE.BIRTH_DATE > '1957.12.31'
    AND WORKS_ON.PROJECT_ID = PROJECT.PROJECT_ID
    AND WORKS_ON.EMPLOYEE_ID = EMPLOYEE.
    EMPLOYEE_ID
    AND PROJECT.PNAME = 'AQUARIUS';
```

  - cél: a leggyorsabb alak kiválasztása, lépések
    - 1) kanonikus alak felírása
    - 2) szelekció süllyesztése
    - 3) levelek átrendezése
    - 4) join
    - 5) projekció süllyesztése

BACK

9. előadás: Tranzakciók adatbázis-kezelő rendszerekben I.

2. Tranzakció

- Definíció
  - **tranzakció** *(transaction)*
    - egy program egyszeri futása, melynek ∀ művelete hatásos, vagy egyik sem
    - fontos feladata: ACID tulajdonságok biztosítása
    - tulajdonságai: **ACID**
      - atomicitás (***a****tomicity)*
        - » egy tranzakció oszthatatlan: vagy végrehajtódik vagy nem, nincs köztes
      - konzisztencia (***c****onsistency)*
        - » csak sikeres tranzakciónak van hatása a DB-re
        - » az adatbázis tartalmát egyik konzisztens állapotból a másikba viszi
      - izoláció (***i****solation)*
        - » ∀ tranzakció úgy fut le, mintha közben más tranzakció nem futna
      - tartósság (***d****urability)*
        - » ha egy tranzakció sikeresen lefutott, annak hatása nem veszhet el

3. Ütemezés

- Definíció
  - **ütemezés** *(schedule)*
    - tranzakciók elemi műveleteinek összessége
    - műveletek időbeli sorrendje egyértelműen meghatározható

- típusai
  - soros ütemezés *(serial schedule)*
    - » tranzakciók szigorúan egymás után futnak le
    - » egyidejűleg 1 tranzakció fut
    - » → időben nem lapolódnak át
    - » mindig megvalósítható, ha természetes módon elkülönülnek
  - nem soros ütemezés *(non-serial schedule)*
    - » 1 vagy több CPU-n
    - » *sorosíthatóak* vagy *nem sorosíthatóak*

- **sorosíthatóság**[75] *(serializablitiy)*
  - akkor sorosítható, ha ∃ olyan soros ütemezés, amelynek ∀ hatása a módosított adatokra azonos az adott ütemezésével
- **nem soros ütemezés problémái:**
  - piszkos olvasás *(dirty read)*
    - egy T_2 tranzakció piszkos adatot olvas be
    - egy másik T_1 tranzakció azelőtt írt az adatbázisba, hogy sikeresen befejeződött volna
    - ha T_1 tranzakció sikertelennek bizonyul → piszkos adat mihamarabb eltávolítandó
  - elveszett módosítás *(lost update)*
    - több tranzakció ugyanazon az adategységen végez módosításokat
    - T_1 tranzakció felülírja T_2 eredményét
  - nem megismételhető olvasás *(non-repeatable read)*
    - T_1 tranzakció különböző eredményeket kap azonos adategység többszöri olvasásakor, mert T_2 módosította
    - módosítás nem vész el

[75] Kisebb hibát vétünk, ha sorosítható ütemezést nem sorosíthatónak minősítünk, mint fordítva.

- fantom olvasás *(phantom read)*
  - T_1 tranzakció többször is végrehajtja ugyanazt a lekérdezést
  - T_2 olyan rekordokat szúr be/töröl ki, ami T_1 szelekciójához kellett volna → ez jól elront ∀t

- **izolációs elv** [76]*(serializablitiy)*
  - futás közben más ne fusson
- **korrekt** *(correct)*
  - ütemezés korrekt, ha sorosítható
- **adathozzáférés**
  - mód szabályozása időbélyegekkel
  - egységeinek megválasztása
- **ütemező** *(scheduler)*
  - DBMS azon része, amely az adatelérési igények megítélése felett dönt (műveletek engedélyezése, abortálása stb.)

4. Tranzakciókezelés zárakkal

- Definíció
  - **zár** *(lock)*
    - hozzáférési privilégium az adategységen
- problémák a zárakkal – **pattról holtpontra** *(deadlock)*
  - T_1 és T_2 (ahol $T_1 \neq T_2$) egyidőben tart fenn zárat egy adaton, amire a másiknak szüksége lenne
    - több zár részvételénél is előfordulhat
  - **megoldási lehetőségek**
    - tranzakciók lockoljanak ∀t zárat egyszerre, ami futáshoz kell, ha valamelyik nem sikerül, akció lefújva
    - ha valami túl sokáig várakozik → abortálás
    - adategységeknek sorrend
    - zárak monitorozása, pattot okozó tranzakció lelövése

[76] Ha nem lehet egy nem soros ütemezésnél biztosítani, hogy ugyanazt adja, mintha sorosan futna → ütemezés nem helyes

- Definíció
  - **várakozási gráf** *(wait-for graph)*
    - *zárak monitorozásához*
    - irányított gráf
    - gráf csomópontjai a tranzakciók
    - élek két csomópont (tranzakció között): ha egyik várakoztatja a másikat
- **Tétel**
  - adott időpillanatban nincs patt → nincs várakozási gráfban kör
- Bizonyítás (indirekt)
  - t. f. h. S van kör → résztvevők várakoztatják egymást
  - nem tudnak tovább lépni, patthelyzet van, ami ellentmondás azzal, hogy nincs patt, akkor nem lehetne kör a gráfban
  - illetve: ha a gráf DAG, akkor ∃ topologikus rendezés → tranzakcióknak olyan sorrendje, amelyben sorban egymás után indulnak el várakoztatás nélkül → nincs patt
- Definíció
  - **éhezés** *(starving, livelock)*
    - egy tranzakció egy adategység lockolására vár
    - más tranzakciók mindig lockolják előtte az adategységet
    - elkerülése
      - sikertelen zárkérések feljegyzése

5. Tranzakció modellek

- Definíció
  - **egyszerű tranzakció modell** *(simple transaction modell)*
    - csak egyfajta zár létezik
    - egy adatelemen egyidőben csak egy zár lehet
- Definíció
  - **sorosítási/precedenciagráf** *(precedence graph)*
    - irányított gráf
    - gráf csomópontjai a tranzakciók

- élek két csomópont (tranzakció között): ha egy adategységen, egy ütemezésben T_i tranzakció zárat helyez el, majd ezen zár felszabadítása után T_j teheti rá a zárját

- **Tétel**
  - egy S ütemezés sorosítható
  - a sorosítási gráf DAG
- Bizonyítás (indirekt)
  - t. f. h. S sorosítható, de tartalmaz kört a sorosítási gráfja
  - a kört alkotó tranzakciók közül egyik sem előzi meg a másikat
  - egy soros ekvivalensben egyik sincs legelöl
    - *az ütemezés nem sorosítható ellentmondásban a feltétellel*
- Definíció
  - **kétfázisú zárolás, 2PL** *(two-phase locking, 2PL)*
    - ha az első zárfelszabadítást megelőzi mindegyik zárkérés
      - *első fázisban zárakat kér, másodikban felszabadít*
  - **legális/nem legális ütemezés**
    - lockolt adategység felszabadítása unlockkal
    - ha egy adategység foglalt, akkor a tranzakció a zár felszabadításáig vár
    - nem legálisnál szimplán nem vár, hanem beleszalad, lol
- **Tétel**
  - ha egy legális ütemezés ∀ tranzakciója a 2PL protokollt követi, akkor az ütemezés sorosítható
- Bizonyítás
  - mivel sorosíthatóságnak szükséges és elégséges feltétele, hogy a sorosítási gráfban ne legyen kör, elegendő ezt belátni
- Definíció
  - **zárpont** *(synchronization point)*
    - ha az első zárfelszabadítást megelőzi mindegyik zárkérés
- **Tétel**
  - a fenti tétel a zárpont segítségével is bizonyítható

- Bizonyítás
  - rendezzük a tranzakciókat növekvő zárpontjuk szerinti sorrendbe
  - ez egy soros ekvivalens ütemezés lesz
- ha valamely ütemezésben van egyetlen nem kétfázisú tranzakciót, akkor ∃ olyan tranzakció is, amellyel együtt már nem sorosítható
- Definíció
  - **RLOCK modell (shared lock model)**
    - ha T: RLOCK
    - A: érvényes
    - → más tranzakció is olvashatja A-t, de nem írhatja felül
  - **WLOCK modell (shared lock model)**
    - ha T: WLOCK
    - A: érvényes
    - → semmilyen más tranzakció se nem írhatja se nem olvashatja A-t
  - **UNLOCK**
    - RLOCK, WLOCK felszabadítása
    - ez automatikus legális ütemezések végén
- **Tétel**
  - egy RLOCK-WLOCK modellbeli S ütemezés sorosítható
  - → a sorosítási gráf DAG
- Bizonyítás
  - analóg az egyszerű tranzakció modell hasonló tételével
- Definíció
  - **kétfázisú tranzakció**
    - Egy RLOCK-WLOCK modell szerinti tranzakció kétfázisú, ha ∀ RLOCK és WLOCK megelőzi az első UNLOCK-ot.
- **Tétel**
  - ha egy ütemezésben csak kétfázisú, RLOCK-WLOCK modell szerinti tranzakciók vannak
  - → ütemezés sorosítható

- Bizonyítás
  - analóg az egyszerű tranzakció modell hasonló tételével
- Definíció
  - **zárkompatibilitási mátrix** *(lock compatibility matrix)*
    - → *további hatékonyság, ha zármódok között van több olyan, hogy egy adategységen tartható fenn, kompatibilisek*
    - zármódok közötti összeférhetőség ábrázolása
    - „igen”: ha egy adatelemen lévő adott típusú zár mellett az új zárkérés is elfogadható
    - „nem”: új zárkérés nem teljesíthető
    - ha ezt ismerjük, fel tudjuk rajzolni valamely ütemezés sorosítási gráfját
    - Példa

		meglévő zár az adategységen	
		RLOCK	WLOCK
zárkérés	RLOCK	igen	nem
	WLOCK	nem	nem

- **fa protokoll**
  - ha az adategységek hierarchiában vannak
  - ekkor lehetőség van arra, hogy adategység zárolásakor a hierarchia alacsonyabb részeit is zároljuk egyidejűleg
  - egyszerű tranzakciómodellt követjük, ekkor a csomópont zárolása nem jelenti a gyerek zárolását is
  - szabályok
    - tranzakció első lockot bárhova teheti
    - további lock akkor tehető belülre, ha a szülőn is ugyanaz a tranzakció lockolt már
    - egyazon tranzakció 2x ugyanazt az adategységet nem zárolhatja

- Tétel
  - a fa protokollnak eleget tevő legális ütemezés sorosítható
- Bizonyítás
  - rendeljünk hozzá ∀ tranzakcióhoz egy irányított gráf 1 csúcsát
  - ebben a gráfban adott szabályok szerint rajzoljunk éleket
    - zárpont bizonyítása ütemezés sorosíthatóságára

6. A figyelmeztető protokoll

- Definíció
  - **implicit zár**
    - egyszerű tranzakciós modellt követjük, d egy csomópont zárolása az összes leszármazott csomópont zárolását jelenti
- **WARNING protokoll zárműveletek**
  - LOCK A
    - zárolja A-t és az összes leszármazott csomópontot
  - WARN A
    - A-ra figyelmeztetés kerül → A-t más nem zárolhatja
  - UNLOCK A
    - A-ról zár vagy figyelmeztetés eltávolítása
  - szabályai
    - egy tranzakció 1. művelete kötelezően LOCK vagy WARN a gyökérben
    - LOCK A vagy WARN A akkor helyezhető el, ha A szülőjén ugyanaz a tranzakció már helyezett el WARN-t
    - UNLOCK A akkor lehet, ha A gyerekein már ugyanaz a tranzakció nem tart fenn LOCK-ot vagy WARN-t
    - kétfázisú: első UNLOCK után már nem jöhet LOCK vagy WARN
- Tétel
  - a figyelmeztető protokollt követő legális ütemezések
    - 1) zárkonfliktus-mentesek
    - 2) sorosíthatóak

- Bizonyítás
  - 1) a protokoll 1-3 szabályai biztosítják, hogy bármely T tranzakció csak akkor tehessen zárat A-ra, ha figyelmeztetés van akkor ∀ ősén
  - 2) adott R ütemezés átalakítható olyan ekvivalens S ütemezésbe, amely az egyszerű tranzakció modellnek felel meg, ∀ adategységet explicit módon zárolunk
  - WARN-LOCK kompatibilitási mátrix

	LOCK	WARN
LOCK	x	x
WARN	x	✓

  - RLOCK-WLOCK modellnek megfelelően értelmezhetünk hierarchikus adategységek esetén RWARN-WWARN-t is, amely a tranzakciós teljesítmény további növekedését eredményezheti

BACK

10. előadás: Tranzakciók adatbázis-kezelő rendszerekben II.

1. Tranzakcióhibák kezelése

- **tranzakció hibák okai**
  - tranzakció félbeszakad
    - felhasználó félbeszakítja
    - 0-val való osztás
    - valamilyen adategységhez nem fér hozzá
  - patt miatt kiesés
  - sorosíthatóság miatti kiütés
  - rendszerhiba (*operatív tár, memória sérül*)
    - szoftveres
    - hardveres
  - megsérül a háttértár tartalma
    - pl. médiahiba
- Definíció
  - **konzisztens állapot** *(consistent state)*
    - adatbázis olyan állapota, amely csak teljesen lefutott tranzakciók hatását tükrözi
    - *automatikusan nem működik*
  - **kész pont** *(commit point)*
    - időpillanat, amikor egy tranzakció futása során ∀ befejeződött → akár tranzakció megszakítása
    - *tranzakció ∀ számítást elvégzett, ∀ zárat megkapott → COMMIT utasítás != eredmény véglegesítése*

- **piszkos adat** *(dirty data)*
  - adat, ami COMMIT előtt kerül be az adatbázisba
  - mihamarabb eltávolítandó
    - *kivéve, ha másik tranzakció olvassa a piszkos adatot → sikeres tranzakció, de eredmény helytelen*
- **lavina** *(cascading aborts)*
  - iterált piszkos adat írás, olvasás

- **tranzakció költségek kezelése**
  - nem kommitált tranzakciót nem olvasunk
    - ha mégis, azok hatása törlés adatbázisból (*undo*)
  - piszkos adat olvasásának ellehetetlenítése

2. **Szigorú kétfázisú protokoll** *(strict- two-phase locking protocol)*

- egy protokoll ezt a tranzakciót követi, ha… (ebben a sorrendben!)
  - kétfázisú
  - nem ír az adatbázisba, amíg a készpontját (*commit point*) el nem érte
    - *így nem kell az adatbázist helyreállítani, ha egy tranzakció abortál, max egy redo*
  - zárait csak az adatbázisba írás után engedi el
    - *piszkos adatot ne olvasson be → commit után nincs abort → nincs lavina se*

- **Tétel**
  - a szigorú kétfázisú protokollt követő tranzakciókból álló legális ütemezések sorosíthatók és lavinamentesek

- Bizonyítás
  - az ütemezés sorosítható, mert
    - kétfázisú
    - lavinamentes
    - nincs lehetőség piszkos adat olvasására

3. Agresszív és konzervatív protokollok

- több alternatíva is van, választás „hatékonyságuk” szerint
- **hatékonysági mérce:**
  - mennyi idő alatt fut le a tranzakció
  - tranzakció teljesítmény
    - adott idő alatt mennyi tranzakció fut le sikeresen
    - *erre szoktunk nagyobb hangsúlyt fektetni jó ütemező és protokoll megválasztásával*
- Definíció
  - **agresszív protokoll** *(aggressive protocol)*/***optimista konkurenciakezelés*** *(optimistic concurrency control)*
    - megpróbál olyan gyorsan lefutni, ahogy csak lehet
    - nem törődik, hogy az esetleges aborttal, zárakkal
  - **konzervatív protokoll** *(conservative protocol)*/***pesszimista konkurenciakezelés*** *(pessimistic concurrency control)*
    - megpróbálja elkerülni olyan tranzakciók futását, melyek nem biztosan eredményesek
  - **választási szempont**
    - ha kevés a tranzakciók által közösen használt adat, akkor kicsia a nem sorosítható ütemezés, patt/éhezés veszélye → agresszív protokollok hatékonyabbak

4. Időbélyeges tranzakciókezelés *(timestamp-based concurrency control)*

- sorosíthatóság biztosításának egy olyan módja, ahol az adategységekhez egy vagy több járulékos adatot/időbélyeget rendelünk
- tehát csak olyan műveleteket engedélyez, melyek hatása a tranzakciók növekvő időbélyegei által meghatározott soros ütemezés hatásaival egyezik meg
  - *eldöntheti, hogy egy tranzakció adott adategységre vonatkozó kérése sérti-e a sorosíthatóságot, ha igen, abortálja*→ ez egy agresszív protokoll
- Definíció
  - **időbélyeg** *(timestamp)*
    - olyan érték, amelyet ∀ tranzakcióhoz szigorú egyediséget biztosítva rendelünk hozzá

- arányos (*lehet azonos is*) a tranzakció kezdőidejével
- Jele: t
  - *tranzakciók egy egyértelmű sorrendjét határozzák meg*
  - *olyanok kezdetben, mintha zérus idő alatt futnának le*

- Definíció
  - **soros ekvivalens ütemezés**
    - tranzakciók kezdő idejének növekvő sorrendbe állítása
    - kétfázisú zárkezeléshez hasonlóan, csak ott zárpontok szerinti növekedő sorrend soros ekvivalens
    - működése
      - megvizsgálja ∀ írás-olvasás előtt a hivatkozott adategység időbélyegét
      - ha ez a sorosítási szabályokkal összhangban van, akkor az adategység időbélyegét felülírja a műveletet végrehajtó tranzakció időbélyegével → nincs összhangban → abortálja
      - felételek megsértése többféle modell alapján is detektálható (egyszerű vagy read/write modell)
    - egyértelműsége
      - az időegység meghatározásához jó alap egyprocesszoros környezetben a tranzakció indulásakor a rendszeróra által mutatott érték
        - több vagy elosztott processzorosnál processzor azonosítója is kell
    - kezelése
      - zár alapúnál hasznos

- Definíció
  - **olvasási és írási idő adott t pillanatban**
    - $R(A)$: az adategység olvasási ideje az A adategységet t-nél nem később olvasó tranzakciók időbélyegei közül a legnagyobb
    - $W(A)$: az adategység írási ideje az A adategységet t-nél nem később író tranzakciók időbélyegei közül a legnagyobb

- az időbélyeg tesztelés és maga a művelet oszthatatlan kell, hogy legyen

	T olvasni akar	T írni akar
$t(T) < R(A)$ $t(T) < W(A)$	abort T *(1)*	abort T *(2)*
$t(T) < R(A)$ $t(T) > W(A)$	T olvassa A-t, de $R(A)$ nem változik *(3)*	abort T *(4)*
$t(T) > R(A)$ $t(T) < W(A)$	abort T *(5)*	abort T *(6)*
$t(T) > R(A)$ $t(T) > W(A)$	T olvassa A-t és $R(A) := t(T)$	T írja A-t és $W(A) := t(T)$

(1) egy később indult tranzakció már írta A-t, tehát nem olvashatjuk

(2) egy később indult tranzakció már olvasta A-t, tehát nem írhatjuk

(3) egy később indult tranzakció már olvasta A-t, ettől még T is kiolvashatja, de az időbélyeget a későbbi értéken kell hagyni

(4) ugyanaz, mint (2)

(5) ugyanaz, mint (1)

(6) egy később indult tranzakció már írta A-t, így T nem írhatja felül, azaz T -nek abortálnia kell.

- A fenti táblázatban a $t(T) = W(A)$ $t(T) = R(A)$vagy teljesülése nyilván akkor következhet be, ha maga T írta vagy olvasta legutóbb A-t.
  - abort T, ha T olvasni akar és $t(T) < W(A)$ vagy T írni akar és $t(T) < R(A)$ vagy
  - a READ művelet elvégzendő, ha $t(T) \geq W(A)$ és
  - WRITE elvégzendő, ha $t(T) \geq R(A)$ és $t(T) \geq W(A)$
- **R/W modell és 2PL összehasonlítása**
  - Elképzelhető, hogy egy ütemezés sorosítható…
    - időbélyegesen, de kétfázisú zárakkal nem
    - időbélyegesen is és zárakkal is (pl. ∀ olyan ütemezés, amelyben a tranzakciók nem használnak közös adatokat)
    - kétfázisú zárakkal, de időbélyegesen nem
    - időbélyegesen sem és zárakkal sem

- **időbélyegek kezelése**
  - zár alapú tranzakciókezelésnél praktikus megközelítés, ha
    - külön tároljuk a zárinformációt az adategységektől (*kicsi abortos/lavinás helyeken*)
    - megakadályozzuk a piszkos adat olvasását pl. azzal, hogy nem írunk az adatbázisba, amíg a tranzakció el nem érte a készpontját (*időbélyeges szigorú protokoll*)
  - lépések:
    - módosítás (*csak munkaterületen*)
    - tranzakció commit
    - írás véglegesítése adatbázison
  - fontos, hogy az időbélyeg ellenőrzése a készpont előtt kell, hogy történjen, utána már a tranzakció nem abortálhat (*időbélyegek miatt sem!*)
  - időbélyeg ellenőrzés és adatkiírás között jelentős idő telhet el → okozhat gondot
    - T. f. h. $\mathrm{t(T)} = 100$ és T írni akarja A -t. Legyen $R(A) = W(A) = 60$ az írás előtt, tehát T írhat (a munkaterületen).
    - Ha jönne egy T_1 tranzakció a készpont előtt, ahol $t(T_1) = 60$ és olvasni akar, akkor – olvashat (helytelenül), amennyiben nem állítjuk be.
    - A időbélyegét az írással egyidőben, – nem olvashat (helyesen), amennyiben beállítjuk A időbélyegét az írással egy- időben $W(A) = 100$-ra.
  - megoldási lehetőségek
    - így fontos: tényleges írásig A-ra zárat kell tenni
      - ha T közben abortál → zár elenged → $W(A)$ helyreállít
    - közvetlenül commit előtt ellenőrizzük az időbélyegeket (*optimista*)
    - akkor ellenőrizzük az időbélyegeket, amikor írás vagy olvasás van (*pesszimista*)

- **verziókezelés (MVCC[77])**
  - csökkentheti az abortok számát, ha tároljuk a verziókeletkezés idejét is
  - alapvetően pesszimista módszer (pesszimista)
  - hatékonyan támogatja a sok/hosszú olvasási tranzakciókat
  - több háttértárat igényel
  - bonyolultabb adat-visszakeresési algoritmus
  - a DBMS-nek kell felderítenie, ha egy verzió a futó tranzakciók számára már közömbös, így törölhető
    - feltételezzük ezzel, hogy a régi értékeket is megőrizzük
    - kézenfekvő, ha idősor jellegű adatokat kívánunk tárolni (pl. betegek adatai)
    - fizikai megoldás: egyszer írható diszkek
    - olvasásokat abortmentessé tudja tenni
- **Összefoglalva:**

	zárak	abort	helyreállítás	hátrány
általános	nincs	lehet	lavina lehet	helyreállítás
„pesszimista”	hosszabb időre	lehet	redo	zárak
„optimista”	rövid időre	gyakoribb	redo	abortok
verziók	nincs	ritkább	redo	tárigény, felesleges értékek eltávolítása

[77] Multi-version concurrency control

BACK

11. előadás: Tranzakciók adatbázis-kezelő rendszerekben III.

1. Naplózás

- a rendszerhibák elleni védekezés általános módszere a tranzakciós naplózás
- Definíció
  - **napló** *(journal, log)*
    - adatbázison végrehajtott változások története
    - naplót azelőtt írunk, mielőtt a naplózott műveletre sor kerülne (*kivéve abort*)
    - alábbi rekordokat tartalmazhatja:

$$(< tranzakció\ azonosító >, begin)$$

$$(< tranzakció\ azonosító >, commit)$$

$$(< tranzakció\ azonosító >, abort)$$

$$(< tranzakció\ azonosító >, < adategység >, < új\ érték >, < régi\ érték^{78} >)$$

- **hatékonysága**
  - helyreállítás igénye miatt a napló háttértáron vagy diszken tárolandó
    - ennek költsége a háttértárra írt naplóblokkok számával arányos, ezeket valamilyen lapozási stratégiával kezelik
  - hatékonyabb, ha írjuk a naplót → kevesebb blokkművelet
  - szabály, hogy tranzakció vége után napló akkor is kiírandó háttértárra, ha a lapozási stratégia ezt nem követeli meg → tranzakció tartósan megismételhetően végrehajtottá válik (ACID tartósság)

[78] Ha tudható, hogy undo műveleteket nem kell a napló alapján végezni, akkor elég az adategységek új értékének naplózása. Ha így is túl nagy a méret, elég akkor az is, hogy hogyan kell az új adategységet előállítani.

2. Redo protokoll

- olyan tranzakciókezelést valósít meg, amely rendszer/tranzakcióhiba után szükségtelenné teszi az undo műveletet
- **redo naplózás**
  1. $(T, begin)$ naplóba
  2. $(T, A, < A\ új\ értéke >)$ naplóba, ha T megváltoztatja valamely A értékét
  3. $(T, commit)$ naplóba
  4. új naplóoldalak tárba írása
     - *ekkor tranzakció véglegesítve lett*
  5. A értékek adatbázisba írása
  6. piszkos DB blokkok diszkre írása
  7. zárak elengedése
     - *ezután férnek hozzá más tranzakciók a megváltoztatott adatokhoz →× lavina*
- **redo helyreállítás**
  1. összes zár felszabadítása
  2. napló vizsgálata visszafele; véglegesített tranzakciók feljegyzése
  3. konzisztens állapot keresése a naplóban
  4. bejegyzések alapján felülírjuk az adatbázisban található értékeket az újakkal
  5. ha elszáll commit előtt a helyreállítás, újrakezdés[79]

3. Ellenőrzési pont (checkpoint)

- konzisztens állapot keresésében segít → kikényszerít az adatbázisból egyet
  1. ideiglenesen megtiltjuk új tranzakciók indítását és megvárjuk, amíg ∀ tranzakció befejeződik vagy abortál
  2. módosult, de háttértárra nem írt memóriablokkok keresése
  3. ezen blokkok háttértárra írása
  4. itt checkpoint naplóba

[79] Hiszen a 4-es pontban lévő végzett műveletek hatása az adatbázisra mindig ugyanazt az eredményt adják, tehát idempotensek

5. napló háttértárra írása

- előny
  - csak a leutóbbi checkpointig kell visszamenni naplóban
  - régebbi checkpointok eldobhatóak akár
  - lavinák számának csökkentése
- hátrány
  - tranzakciós teljesítmény csökken
    - *többletírások háttértárra*
- ütemezése
  - adott idő eltelte után
  - egy tranzakció lefutása után

4. Médiahibák elleni védekezés

- **megoldások**
  - rendszeres mentés (backupok)
  - fájlok duplikálása más eszközön
  - adatbázis, napló más-más eszközön

BACK

12. előadás:
Relációs adatbázisok logikai tervezése

1. Tervezési lépések

- legelterjedtebb adatbázis-kezelő rendszereke ma is relációs adatmodellben alapulnak
- **tervezési lépések**
  1. adatbázis logikai tervezése
     - relációs sémák, adattípusok definiálása
     - adatbázis alkalmazás megírása (vagy épp létező kiválasztása :D)
       - ez alapulhat magas szintű procedurális nyelvbe alapozott SQL-en, vagy API-n[80]
  2. adatbázis fizikai tervezése
     - segédstruktúrák kialakítása
     - tárolási paraméterek meghatározása

2. Tervezési ER-diagramból

- ER-diagramm elemeinek átalakítása relációs adatmodell adatstruktúráiba
  - entitáshalmazok
    - ∀ sora entitás
    - ∀ oszlopa attribútum
    - ISA: entitás attribútumai is kell
  - kapcsolattípusok
    - oszlopa a kapcsolat végén lévő entitások kulcsai/attribútumai a kapcsolatnak
    - sorai a kapcsolatban levő entitások / típusaik
      - *így lényegre törőbb a függvényjelleg bevitele is*

[80] Alkalmazásprogramozási interfész

3. Anomáliák

- Definíció
  - **anomália**
    - redundáns relációk esetén előforduló kellemetlenségek
  - **redundáns reláció**
    - ha egy relációban valamely attribútum értékét a relációban található más attribútumok értékéből is ki tudjuk nyerni valamely ismert következtetési szabály segítségével
  - **vertikális dekompozíció**
    - relációk függőleges felbontása
      - *lenti problémákra ez megoldás lehet*
- **szabályok**
  - módosítási anomália
    - attribútumot ∀ hol megváltoztatjuk, de valahol kihagyjunk egy változtatást
  - beszúrási anomália
    - nem tudunk a relációba olyan elemet felvenni, melynek egy kötelező mezője[81] kitöltetlen
  - törlési anomália
    - ha kulcs elemet kell töröljünk, akkor az egész sort törölnünk kell vele → ezzel elveszíthetünk más fontos adatot

4. Adatbázis kényszerek

- **kényszerek**
  - szabályok az adatbázis jellemzéséhez, hogy az megfelelő legyen
- **kényszertípusok**
  - értékfüggő kényszerek
    - *pl. 0 < testmagasság < 300*

[81] Értsd a kötelezőt úgy, hogy pl. egy másik adattal kapcsolatban kéne lennie. Pl. nem vehetünk fel úgy egy új szállítót, ha még nem szállított semmit.

- értékfüggetlen kényszerek
  - tartalmazási függőségek
    - a tartalmazási kényszerek meghatározzák, hogy egy adott érték vagy halmaz tartalmaznia kell-e egy másik értéket vagy halmazt
  - funkcionális függőségek
    - a funkcionális kényszerek meghatározzák, hogy egy adott attribútum értéke függ-e más attribútum(ok) értékétől
  - többértékű függőségek
    - a többértékű kényszerek meghatározzák, hogy az egyik attribútum értékei hogyan függhetnek össze egy másik attribútum értékeivel

- **redundancia kezelése**
  - ha X (szuper)kulcs, akkor nincs redundancia a függés miatt
    - Funkcionális függőséghalmaz jele: F_r
- **függés verziók**
  - determináns
    - X és Y valódi részhalmaza R-nek, X ⇒ [82] Y
    - $\nexists X' \subset X$
    - így X' ⇒ Y, akkor X Y determinánsa
  - részleges függés
    - X és Y valódi részhalmaza R-nek, X ⇒ Y
    - $\exists X' \subset X$[83]
    - így X' ⇒ Y, így Y részlegesen függ X-től
  - teljes függés
    - X és Y valódi részhalmaza R-nek, X ⇒ Y
    - $\nexists X' \subset X$[84]
    - így X' ⇒ Y teljesen (funkcionálisan) függ X-től

[82] X függ X-tól, ez a kis nyilacska a jele
[83] X-nek van olyan részhalmaza, amelytől függ Y.
[84] X-nek nincs olyan részhalmaza, amelytől függ Y.

Relációs sémák kulcsai

- Definíció
  - **kulcs**
    - X kulcs R relációs sémán, ha
      - $X \Rightarrow R$
      - $\nexists X' : X' \subset X, X' \Rightarrow R$ (teljesen függ R X-től)
    - *reláció egy 1 sorát határozza meg*
    - *nem lehet NULL az értéke*
  - **szuperkulcs**
    - *tartalmazhat felesleges attribútumokat*
    - rekordok egyedülálló azonosítására
  - **minimális kulcs**
    - szuperkulcs, mely nem tartalmaz felesleges attribútumokat
    - a legkisebb méretű kulcs, amely képes egyértelműen azonosítani a rekordokat
  - **egyszerű kulcs**
    - egyszerű kulcs egyetlen attribútumból áll, amely egyedileg azonosítja a rekordokat a táblában
    - *else* összetett kulcs
- **Tétel**
  - ∀ relációs sémának van kulcsa
- Bizonyítás
  - válasszuk ki az attribútumok teljes halmazát (*tehát ∀t kiválasztunk*)
  - relációs séma ∀ attribútum értéket meghatároz (*mivel ∀t is kiválasztottunk*)
    - ha második feltétel is teljesül, akkor kulcs *else* szuperkulcs

- Definíció
  - **elsődleges kulcs**
    - X és Z az R kulcsa, X ≠ Z
    - mivel R-nek így több kulcs van, így egyik elsődleges lesz, többi meg kulcsjelölt
    - egyedileg azonosítja a rekordokat a táblában
  - **idegen kulcs**
    - attribútum vagy attribútumok kombinációja
    - egy táblában hivatkoznak egy másik tábla elsődleges kulcsára
    - ennek a segítségével kapcsolódás alakítható ki a táblák között

5. Armstrong axiómái a funkcionális függőségekről

- **triviális függőségek**
  - tetszőleges attribútumhalmazból következik ugyanaz az attribútumhalmaz
  - *példa: ha van egy tábla az "Ember" nevű attribútummal, akkor az ember minden attribútuma egyértelműen meghatározza önmagát, tehát az Ember -> Ember triviális függőség igaz*
- **axiómáktól elvártak**
  - csak igaz függőséget lehessen előállítani
  - elő lehessen ∀ ilyet állítani, amely több függőséggel nincs ellentmondásban
  - igaz
    - X ⇒ Y
    - R séma F_r függéshalmaza mellett is mindenre fennáll

$$F_r \vDash X \Rightarrow Y$$

  - meg lehet kapni
    - W ⇒ Z
    - F_r -ből, ha axiómák ismételt alkalmazásával ismét megkapjuk W ⇒ Z

$$F_r \vdash W \Rightarrow Z$$

- **Armstrong axiómák (RAT)**
  - Y $\subset X$, akkor X ⇒ Y (*triviális függőség*)
    - *ha egy attribútumhalmaz tartalmaz egy attribútumot, akkor az attribútumhalmazból következik az adott attribútum*
  - X ⇒ Y és Y ⇒ Z, akkor X ⇒ Z (*tranzitivitás*)
    - *ha két attribútumhalmazból következik egy harmadik attribútumhalmaz, akkor az első attribútumhalmazból következik a harmadik is*
  - X ⇒ Y, akkor XZ ⇒ YZ (*bővíthetőség*)
    - *ha egy attribútumhalmazból következik egy másik attribútumhalmaz, akkor a kiegészítésével egy harmadik attribútumhalmaz is következik*
- **Tétel: igazság**
  - Armstrong axiómákra építve minden funkcionális függőség következik a kiindulási függőségekből

$$F_r \vdash X \Rightarrow Y \rightarrow F_r \vDash X \Rightarrow Y$$

  - *a X-ből Y következik, akkor a X-ből Y függőséget megállapíthatjuk az Armstrong axiómák ismételt alkalmazásával, ez azt jelenti, hogy az axiómák valóban helyesek és megbízható eszközöket nyújtanak a funkcionális függőségek meghatározására*
- Bizonyítás
  - ha axiómákat egyesével belátjuk, hogy igazak, akkor ∀ olyan függőség is igaz lesz, melyet az axiómák véges számban ismételt alkalmazásával kapunk
- **Tétel: teljesség**
  - az Armstrong axiómák teljesek, azaz belőlük ∀ igaz függőség levezethető

$$F_r \vDash X \Rightarrow Y \rightarrow F_r \vdash X \Rightarrow Y$$

    - *X-ből Y függőség igaz az adatbázisban, akkor ez a függőség kinyerhető az axiómák segítségével, így az axiómák alkalmazása során minden lehetséges funkcionális függőséget megtalálunk, és nincsenek elhagyott vagy kihagyott függőségek*

BACK

13. előadás: Normálformák és jelentésük

1. Relációs sémák normálformái

- Definíció
  - **normálforma**
    - anomáliák elkerülése
    - feltételek teljesítése → ezek a feltételek a normálformák
    - *megszorítások*
  - **0NF**[85]
    - ahol min. 1 attribútum nem atomi[86]
  - **1NF**
    - *kiindulási alap*
    - 1NF, ha csak atomi értékek vannak benne
      - könnyen bővíthető legyen
      - ha 1NF sincs meg → DB poorly designed
      - értékek azonos oszlopba kell legyenek azonos típussal
      - oszlopnevek egyediek
      - adatok tárolási sorrendje nem számít
  - **elsődleges és másodlagos attribútum**
    - R relációs séma A attribútuma elsődleges, ha A eleme a séma valamely K kulcsának
    - *elsődleges attribútum egy olyan attribútum, amely egy adott táblában az entitásokat egyértelműen azonosítja*
    - *else* másodlagos attribútum

[85] NF = Normálforma
[86] Olyan értelemben nem atomi, hogy nem tekinthető egyetlen egységnek, azaz az egyes részeihez külön is hozzá akarunk férni. Pl. relációs séma, amely ismétlődő csoportot tartalmaz

- **2NF**
  - 1NF relációs séma 2NF alakú, ha ∀ másodlagos attribútum a séma bármely kulcsától teljesen függ
  - célja a redundancia csökkentése
    - → ha megsérti ezt a két pontot bármi, akkor másodlagos attribútumban az attribútum értékek redundáns tárolása megvalósulhat
  - Következmény
    - 1) ha ∀ kulcs egyszerű, akkor 1NF ⇒ 2NF
    - 2) ha nincs másodlagos attribútumok, akkor 1NF ⇒ 2NF

- **Tétel**
  - ∀ 1 NF séma felbontható 2MF sémákba úgy, hogy azok felhasználásával az 1NF sémára illesztett „eredeti" relációk helyreállíthatóak

- Definíció
  - **3NF**
    - 1NF 3NF, ha 1 másodlagos attribútuma sem függ tranzitívan egy kulcstól sem
    - 1NF 3NF, ha ∀ X ⇒ A nem triviális függőség esetén
      - X szuperkulcs vagy A elsődleges attribútum
      - *funkcionális függés-alapú redundanciától mentes → 3NF-ben így is lehet redundancia*

- **Tétel**
  - függőségi definíciók egymásból következnek

- **Tétel**
  - ahhoz, hogy (R, F) sémáról eldöntsük tranzitív függés alapján, hogy 3NF-e, elég csak F-beli funkcionális függőségeket vizsgálni

- **Tétel**
  - ∀ min 1NF felbontható 3NF sémákba úgy, hogy azokból eredeti reláció infóvesztés nélkül helyreállítható

- **Tétel**
  - ha egy séma 3NF, akkor 2 is
    - **3NF(R) ⇒ 2NF(R)**

- Definíció
  - **Boyce-Codd normálforma (BCNF)**
    - ha ∀ A attribútum ∀ X kulcsára igaz, hogy nem ∃ olyan Y, melyre X ⇒ Y (Y ⇏ X) és Y ⇒ A (A ⊈ Y), nincs tranzitív függés kulcstól
    - 1NF R séma BCNF, ha ∀ X ⇒ A nem triviális függőség esetén X szuperkulcs
      - *érdemes használni, mert redundanciamentességet lehet vele kialakítani, ami könnyíti az adatbázis használatát*
      - *nincs mindig mód rá kialakítani (pl. ütemezés)*
- **Tétel**
  - a tranzitív és a triviális függés definíció egymásból következik
- **Tétel**
  - ahhoz, hogy (R, F) sémáról 2. definíció alkalmazásával eldönthessük, hogy BCNF-e, elég az F-beli funkcionalitás függőségek vizsgálat
- **Tétel**
  - ha egy séma BCNF, akkor 3NF is
- **Tétel**
  - BCNF sémára illeszkedő relációk nem tartalmaznak redundanciát a funkcionális függőségek következtében
    - Következmény
      - emiatt egyetlen attribútum értékét sem lehet kikövetkeztetni más attribútumok értékeinek ismeretében, ismert funkcionális függőségek alapján
- Definíció
  - egy adatbázis BCNF (3/2/1NF) alakú, ha benne található összes relációs séma rendre min. BCNF (3/2/1NF)