

Bevezetés

Adatbázisok használata

Mi is az adatbázis?

- Az adatbázisok ma már az élet számos területén alapvető fontossággal bírnak (Google, Amazon, Flickr, Youtube stb.)
- **Adatbázis:** olyan adatok együttese, amelyet egy adatbázis-kezelő rendszer (DBMS: Database Management System) kezel.
- Megoldandó feladatok:
 - új adatbázisok létrehozása, ezek logikai szerkezetének, **sémájának** definálása, adatdefiníciós nyelv (DDL, Data Definition Language),
 - adatok lekérdezése, módosítása (DML, Data Manipulation Language),
 - nagyméretű adatok hosszú időn keresztül történő tárolása, adatok biztonsága meghibásodásokkal, illetéktelen hozzáférőkkel szemben, hatékony adatbázis-hozzáférés,
 - egyszerre több felhasználó egyidejű hozzáférésének biztosítása.
- **Példa:** banki rendszerek (még a hőskorszakból).

Történelem I.

- Legelőször olyan helyzetekben alkalmaztak ABKR-t (adatbáziskezelő-rendszert), ahol sok kicsi adatalem szerepelt, sokan akartak hozzáférni az adatokhoz egyszerre és gyakoriak voltak a módosítások (a banki rendszerek mellett még pl. repülőgép helyfoglalás).
- Az első modellek: **hierarchikus** és **hálós** adatmodell, az előbbi fa-, utóbbi gráfszerkezetben ábrázolta az adatokat.
- A modelleket végül szabványosították CODASYL jelentésben (Committee on Data Systems and Languages).
- Hátrányuk: nem támogattak magasabb szintű lekérdezőnyelvet (pl. add meg, hogy Szirosi számláin összesen mennyi pénz van), hanem csak a mutatók mentén pontról-pontra lehetett haladni a gráfban, minden lekérdezés külön programkódot igényelt.

Történelem II.

- Ted Codd 1970-ban publikált egy cikket, amelyben azt javasolta, hogy az adatokat táblázatokban, **relációkban** tárolják.
- Az ötlet, habár igen egyszerűnek tűnik, meglepően sikeresnek bizonyult, a 90-es évek elejére a relációs adatbázisok lettek a legelterjedtebbek rendszerek.
- Egyik fő előnyük, hogy lehetővé teszik az adatok magasszintű programnyelvvel, SQL (Structured Query L- pl. Szszi számláin összesen mennyi pénz van. Kell egy művelet, ahol a megfelelő sorokat választjuk ki: név = 'Szszi', kell egy másik, ahol a kívánt oszlopot (összeg), végül az oszlopban lévő számokat összegezni kell. Az SQL-ben ez három utasítás, s az implementációval nem kell törödnünk, sem azzal, hogy az adatokat valójában miként tárolja a rendszer.

Egy példa...

név	számla_azon	összeg
Sziszi	SZ01	45000
Peti	SZ02	543000
Sziszi	SZ03	120000

Mostani irányvonalak

- A rendszerek már nem csupán egyszerű adatok tárolására, hatékony lekérdezésére stb. képesek, hanem igen összetett adatokat is hatékonyan kezelnek (a felsoroltakon kívül (Google stb.) pl. térinformatikai rendszerek).
- Egyre nagyobb mennyiségű adatot kell eltárolni. Ma már egyáltalán nem számít kirívó esetnek, ha egy vállalat terabajtnyi adatot (10^{12}) tárol, de vannak petabajtnyi (10^{15}) adattal dolgozó rendszerek.
- Peer-to-peer fájlmegosztó rendszerek.
- Több adatbázis fölé egy „összefogó” adatbázis felépítése (adattárház, middleware).

Relációs adatmodell

Adatbázisok használata

Mi is az adatmodell?

- Az adatmodell információ vagy adatok leírására szolgáló jelölés. A leírás részei:
 - az adatok struktúrája.
 - Az adatokon végezhető műveletek. Az ABKR esetében általában kevesebb műveletet hajthatunk végre, mint egy általános célú programnyelv esetében. Itt azonban a kevesebb, több. A műveletek egyedi hatékony megvalósításán túl, több művelet - egy lekérdezés - együttes optimalizációja is lehetővé válik.
 - Az adatra vonatkozó megszorítások. Pl. egy személyigazolvány-számhoz nem tartozhat két különböző személy.
 - Legfontosabb adatmodellek: relációs és féligstrukturált (XML).

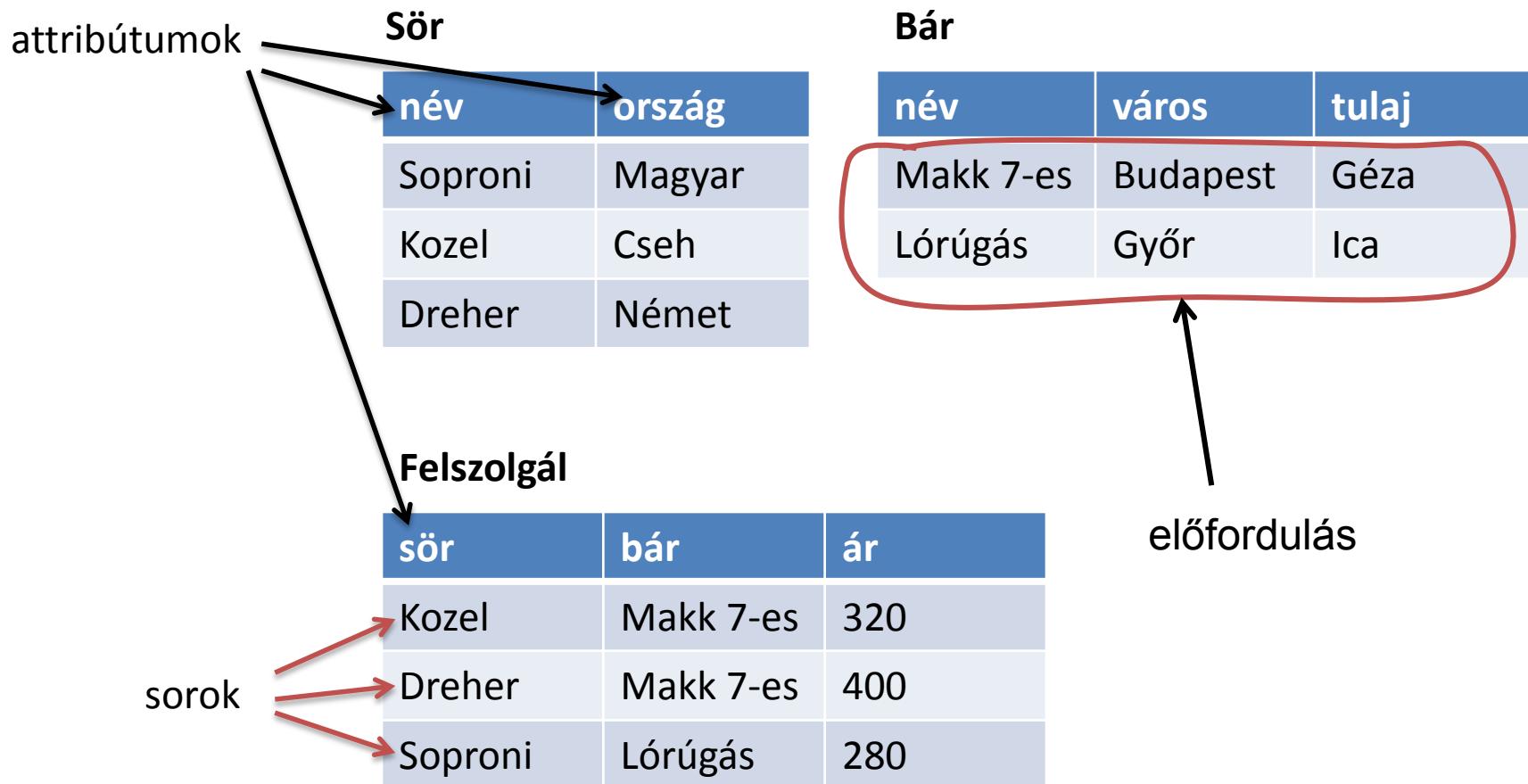
Példa félstrukturált adatra (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<bár típus="étterem">
    <név>Makk 7-es</név>
    <város>Budapest</város>
    <tulaj>Géza</tulaj>
    <telefon>+36-70-123-2345</telefon>
    <telefon>+36-70-123-2346</telefon>
</bár>
<bár típus="kocsma">
    <név>Lórúgás</név>
    <város>Eger</város>
    <telefon>+36-30-451-1894</telefon>
</bár>
</xml>
```

Relációs adatmodell

Egy reláció sémája: Sör (név, ország).

Az adatbázis sémája: Sör (név, ország), Bár (név, város, tulaj),
Felszolgál (sör, bár, ár).



Relációs adatmodell I.

- (Ismétlés: adott X_1, \dots, X_n alaphalmazok esetén a $\rho \subset X_1 \times \dots \times X_n$ részhalmazt **relációnak** nevezzük.)
- A relációs adatmodellben az adatokat **kétdimenziós táblákban** (relációkban) tároljuk.
- A reláció fejrészében találhatók az **attribútumok**.
- minden attribútumhoz hozzátarozik egy **értékkészlet**.
- A reláció neve és a reláció-attribútumok halmaza együtt alkotják a **relációsémát**.
- A **séma** egy adatmodellben általánosságban azt adja meg, hogy egy-egy adatelem milyen „formájú” adatokat tárol.
- Egy-egy reláció **soroknak** egy **halmaza**.
- **Halmaz**: tehát nem számít a sorrend, valamint **egy elem csak egyszer szerepelhet**.

Relációs adatmodell II.

- A reláció sorainak halmazát előfordulásnak nevezzük.
- $p \subset X_1 \times \dots \times X_n$ esetén az attribútumok értékkészlete adja az X_i alaphalmazokat ($1 \leq i \leq n$), egy-egy előfordulás pedig egy-egy relációnak „feleltethető meg”.
- Az attribútumok sorrendje tehát nem rögzített a relációsémában. Egy-egy előfordulás ábrázolása esetén viszont rögzítésre kerül.

Relációs adatmodell II.

- Az adatbázis tulajdonképpen relációk halmaza. A megfelelő relációsémák halmaza adja az **adatbázissémát**, a hozzá tartozó előfordulások pedig az **adatbázis-előfordulást**.
- Egy sor elemeit **mezőnek** (komponens) nevezzük. **Minden mező csak atomi értéket vehet fel.** Léteznek bonyolultabb adatmodellek is, ahol egy mező értéke lehet halmaz, lista, tömb, rekord, referencia stb.
- **Megjegyzés:** a gyakorlatban sokszor megengedik a sorok ismétlődését, hiszen az ismétlődések megszüntetése nagyon időigényes.

Mire kell odafigyelni?

Mivel attribútumok halmazáról van szó, a Példa 1 és Példa 2 relációk nevüktől eltekintve azonosak.

Példa 1

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

Példa 2

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

Mivel sorok halmazáról van szó, a Példa 1 és Példa 3 relációk nevüktől eltekintve azonosak.

Példa 3

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

Példa 4

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

Ebben a modellben Példa 4 nem reláció.

Példa megszorításra

- Az attribútumok egy halmaza egy **kulcsot** alkot egy relációra nézve, ha a reláció **bármely előfordulásában** nincs két olyan sor, amelyek a kulcs összes attribútumának értékein megegyeznének.
- Ilyen egy attribútumú kulcs például a személyiigazolvány-szám vagy a TAJ szám.
- **Megjegyzés:** egy kulcs nem feltétlenül egy attribútumból áll. Például a bár táblában valószínűleg az a jó, ha a név és a város együtt alkotják a kulcsot.
- A kulcsot aláhúzás jelöli:
Bár (név, város, tulaj).

Vigyázat!

Ennél a konkrét előfordulásnál választhatnánk a nevet kulcsnak, sok esetben viszont ez nem megfelelő, hiszen több különböző ember is él ugyanazzal a névvel.

név	telefon
Grasshaus Ignác	20-234-4567
Menyhért Lipót	20-564-2345
Bereg Anna	20-345-1231

Feladat

- Hány különböző módon reprezentálható egy relációelőfordulás (az attribútumok és sorok sorrendjét figyelembe véve), ha az előfordulásnak 4 attribútuma és 5 sora van?

Mit nevezünk algebrának?

- Egy algebra általában műveleteket és atomi operandusokat tartalmaz.
- Az algebra lehetővé teszi kifejezések megfogalmazását az atomi operandusokon és az algebrai kifejezéseken végzett műveletek alkalmazásával kapott relációkon.
- Fontos tehát, hogy minden művelet végeredménye reláció, amelyen további műveletek adhatók meg.
- A relációs algebra atomi operandusai a következők:
 - a relációkhöz reprezentáló változók
 - konstansok, amelyek véges relációt fejeznek ki

Relációs algebra (műveletek) I.

- Projekció (vetítés). Adott relációt vetít le az alsó indexben szereplő attribútumokra. Példa: $\Pi_{A, B}(R)$

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



A	B
a	b
c	d
g	a

Relációs algebra (műveletek) II.

- **Szelekció** (kiválasztás). Kiválasztja az argumentumban szereplő reláció azon sorait, amelyek eleget tesznek az alsó indexben szereplő feltételnek.
- $R(A_1, \dots, A_n)$ reláció esetén a σ_F kiválasztás F feltétele a következőképpen épül fel:
 - atomi feltétel: $A_i \theta A_j$, $A_i \theta c$, ahol c konstans, $\theta \in \{=, <, >\}$,
 - ha B_1, B_2 feltételek, akkor $\neg B_1, B_1 \wedge B_2, B_1 \vee B_2$ is feltételek.
- **Példa:** $\sigma_{A=a \vee C=d}(R)$ ($a \neq, \leq, \geq$ műveleteket ezentúl értelemszerűen használjuk)

The diagram illustrates the selection operation $\sigma_{A=a \vee C=d}(R)$. On the left, there is a 3x3 table with columns labeled A, B, and C, and rows labeled a, b, c in the first row, and c, d, e in the second row, and g, a, d in the third row. A large blue arrow points from this table to a smaller 2x3 table on the right, which has the same column labels and contains the rows (a, b, c) and (g, a, d).

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

A	B	C
a	b	c
g	a	d

Relációs adatmodell (műveletek) III.

- Mivel sorok halmazáról van szó, így értelmezhetők a szokásos halmazműveletek: az **unió**, a **metszet** és a **különbség**. A műveletek alkalmazásának feltétele, hogy az operandusok attribútumai megegyezzenek és azonos sorrendben szerepeljenek. Példa: R – S:

R	S							
A	B	C	A	B	C	A	B	C
a	b	c	a	b	c	g	a	d
c	d	e	c	d	e			
g	a	d	g	d	f			

Relációs algebra (műveletek) IV.

- A **Descartes-szorzat** is értelmezhető. Itt természetesen nem fontos az attribútumok egyenlősége. A két vagy több reláció azonos nevű attribútumait azonban meg kell különböztetni egymástól. **Példa:** $R \times S$.

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

S	B	D
	b	r
	q	s



	A	R.B	C	S.B	D
	a	b	c	b	r
	a	b	c	q	s
	c	d	e	b	r
	c	d	e	q	s
	g	a	d	b	r
	g	a	d	q	s

Relációs algebra (műveletek) V.

- Egyes esetekben szükség lehet egy adott reláció attribútumainak **átnevezésére**. A $\rho_{S(C, D, E)}(R)$ az $R(A, B, C)$ reláció helyett veszi az S relációt, melynek sorai megegyeznek R soraival, az attribútumai pedig rendre C, D, E .
- Ha az attribútumokat nem szeretnénk átnevezni, csak a relációt, ezt $\rho_S(R)$ -rel jelöljük.

Feladatok I.

- A feladatokat a **Szeret (név, gyümölcs)** relációssémájú tábla fölött értelmezzük.
1. Melyek azok a gyümölcsök, amiket Micimackó szeret?
 2. Melyek azok a gyümölcsök, amiket Micimackó nem szeret?
 3. Kik azok, akik szeretik az almát, de nem szeretik a körtét?
 4. Kik azok, akik vagy az almát, vagy a körtét szeretik?
 5. Kik szeretnek legalább két gyümölcsöt?
 6. Kik szeretnek legfeljebb két gyümölcsöt?
 7. Kik szeretnek pontosan két gyümölcsöt?
 8. Kik szeretnek minden gyümölcsöt?

Feladatok II.

9. Kik szeretik legalább azokat a gyümölcsöket, mint Anna?
 10. Kik szeretik legfeljebb azokat a gyümölcsöket, mint Anna?
 11. Kik azok a személy-személy párok, akik pontosan ugyanazokat a gyümölcsöket szeretik?
- Másik séma: **Borivók (név, mennyiség)**
12. Kik fogyasztják a legtöbb bort?

Théta-összekapcsolás I.

- A gyakorlatban szinte kizárolag valamilyen összekapcsolásra visszavezethető műveletet használnak abban az esetben, amikor a lekérdezés megválaszolásához több táblából kell kigyűjteni az adatokat.
- Théta-összekapcsolás: $R(A_1, \dots, A_n)$, $S(B_1, \dots, B_m)$ sémájú táblák esetén:

$R | X |_F S = \sigma_F (R \times S)$ teljesül, itt F

- elemi feltétel $A_i \Theta B_j$, $A_i \Theta c$, ahol $\Theta \in \{=, <, >\}$ és c konstans,
- vagy összetett feltétel, azaz: ha B_1 , B_2 feltétel, akkor
 $\neg B_1$, $B_1 \wedge B_2$, $B_1 \vee B_2$ is feltétel.

Théta-összekapcsolás II.

A	B	C
a	b	c
d	d	g
e	f	r

$|X|_{A=D}$

D	E
b	e
d	r



A	B	C	D	E
d	d	g	d	r

- Egyen-összekapcsolás (equi join): ha a théta-összekapcsolásban a Θ helyén = szerepel.

Természetes összekapcsolás

- Természetes összekapcsolás: $R(A_1, \dots, A_n)$, $S(B_1, \dots, B_m)$ sémájú táblák esetén $R |X| S$ azon sorpárokat tartalmazza R -ből illetve S -ből, amelyek R és S azonos attribútumain megegyeznek.
- A természetes összekapcsolás asszociatív, azaz:
 $(R_1 |X| R_2) |X| R_3 = R_1 |X| (R_2 |X| R_3)$, és kommutatív, azaz :
 $R_1 |X| R_2 = R_2 |X| R_1$.
- A théta-összekapcsolás nem mindenkor asszociatív. Miért?

A	B	C
a	b	c
d	d	g
e	f	r

|X|

B	D
b	e
d	r



A	B	C	D
a	b	c	e
d	d	g	r

Miért olyan gyakori?

Felszolgál

kocsma	sör
Makk 7-es	Dreher
Lórúgás	Kozel
Lórúgás	Gösser

Látogat

X	név	kocsma
	Péter	Makk 7-es
	Feri	Lórúgás



kocsma	sör	név
Makk 7-es	Dreher	Péter
Lórúgás	Kozel	Feri
Lórúgás	Gösser	Feri

A természetes összekapcsolás kifejezhető a többi alapművelettel:

$$R |X| S \equiv \Pi_L(\sigma_C(R \times S)),$$

itt: **C** a közös attribútumok egyenlőségét írja elő, **L** pedig csak egyszer veszi a közös attribútumokat.

Feladatok I.

- A feladatokat a következő táblák fölött kell végrehajtani:

Látogat(név, kocsma)

Felszolgál(kocsma, sör)

Szeret(név, sör)

Fogyasztás(kocsma, dátum, sör, liter).

1. Kik azok, akik szeretik a Drehert és járnak olyan kocsmába, ahol Borsodit is felszolgálnak?
2. Kik járnak legalább egy olyan kocsmába, ahol van legalább egy kedvenc sörtüket felszolgálják?

Feladatok II.

3. Kik járnak olyan kocsmába, ahol nem szolgálnak fel Borsodit?
4. Milyen kocsmákba járnak azok, akik legalább kétféle sört szeretnek?
5. Kik járnak olyan kocsmába, ahol egyetlen kedvenc sörüket sem szolgálják fel?
6. Kik járnak olyan kocsmába, ahol az összes kedvenc italát felszolgálják?
7. Kik azok, akik járnak abba a kocsmába, ahol eddig a legkevesebb Borsodi fogyott egy adott napon?

További feladatok

Termék (gyártó, modell, típus)

PC (modell, sebesség, memória, merevlemez, cd, ár)

Laptop (modell, sebesség, memória, merevlemez, képernyő, ár)

Nyomtató (modell, színes, típus, ár)

1. Melyek azok a PC modellek, amelyek sebessége legalább 150?
2. Mely gyártók készítenek legalább egy gigabájt méretű merevlemezzel rendelkező laptopot?
3. Adjuk meg a B gyártó által gyártott összes termék modellszámát és árát típustól függetlenül!

További feladatok II.

4. Melyek azok a gyártók, amelyek gyártanak legalább két egymástól különböző, legalább 1,2 gigaherzen működő számítógépet (PC-t vagy laptopot)?
5. Melyik gyártó gyárt legalább három különböző sebességű laptopot?
6. Melyik gyártó gyártja a leggyorsabb számítógépet (PC-t vagy laptopot)?

Relációkra vonatkozó megszorítások

- A megszorításokat kétféleképpen fejezhetjük ki (legyenek R és S relációs algebrai **kifejezések**):
 - $R = \emptyset$, azaz R -nek üresnek kell lennie,
 - $R \subseteq S$, azaz R eredményének minden sorának benne kell lennie S eredményében.
- A két megszorítás kifejezőérő szempontjából azonos:
 - $R \subseteq S$ így is kifejezhető: $R - S \subseteq \emptyset$,
 - míg $R = \emptyset$, $R \subseteq \emptyset$ alakban is írható.

Hivatkozási épség megszorítás

- **Hivatkozási épség megszorítás:** ha egy érték megjelenik valahol egy környezetben, akkor ugyanez az érték egy másik, az előzővel összefüggő környezetben is meg kell, hogy jelenjen.
- **Példa:** a **Sör(név, ország), Felszolgál(sör, bár, ár)** táblák esetén megköveteljük, hogy csak olyan sörök szerepeljenek a Felszolgál táblában, amelyek a Sör táblában is szerepelnek.
- A megszorítás: $\Pi_{\text{sör}}(\text{Felszolgál}) \subseteq \Pi_{\text{név}}(\text{Sör})$.
- Általában: $\Pi_A(R) \subseteq \Pi_B(S)$.

Kulcs és egyéb megszorítások

- Példa: a **Bár(név, város, tulaj)** relációban a (név, város) attribútumhalmaz kulcs.

$$\sigma_{B1.\text{név}=B2.\text{név} \wedge B1.\text{város}=B2.\text{város} \wedge B1.\text{tulaj} \neq B2.\text{tulaj}}(B_1 \times B_2) = \emptyset$$

- Tegyük fel, hogy csak a budapesti vagy madridi bárokkal szeretnénk foglalkozni. Ennek kifejezése:

$$\sigma_{(\text{város} \neq 'Budapest') \wedge (\text{város} \neq 'Madrid')}(B) = \emptyset.$$

Feladatok

Termék (gyártó, modell, típus)

PC (modell, sebesség, memória, merevlemez, cd, ár)

Laptop (modell, sebesség, memória, merevlemez, képernyő, ár)

Nyomtató (modell, színes, típus, ár)

1. Az olyan PC-ket, amelyek processzorának sebessége kisebb, mint 2.00, nem árulhatjuk drágábban, mint 12000 Ft.
2. A PC-gyártók nem gyárthatnak laptopokat.
3. Ha egy gyártó készít PC-t, akkor készítenie kell olyan laptopot is, amelynek a sebessége legalább akkora, mint a PC sebessége.

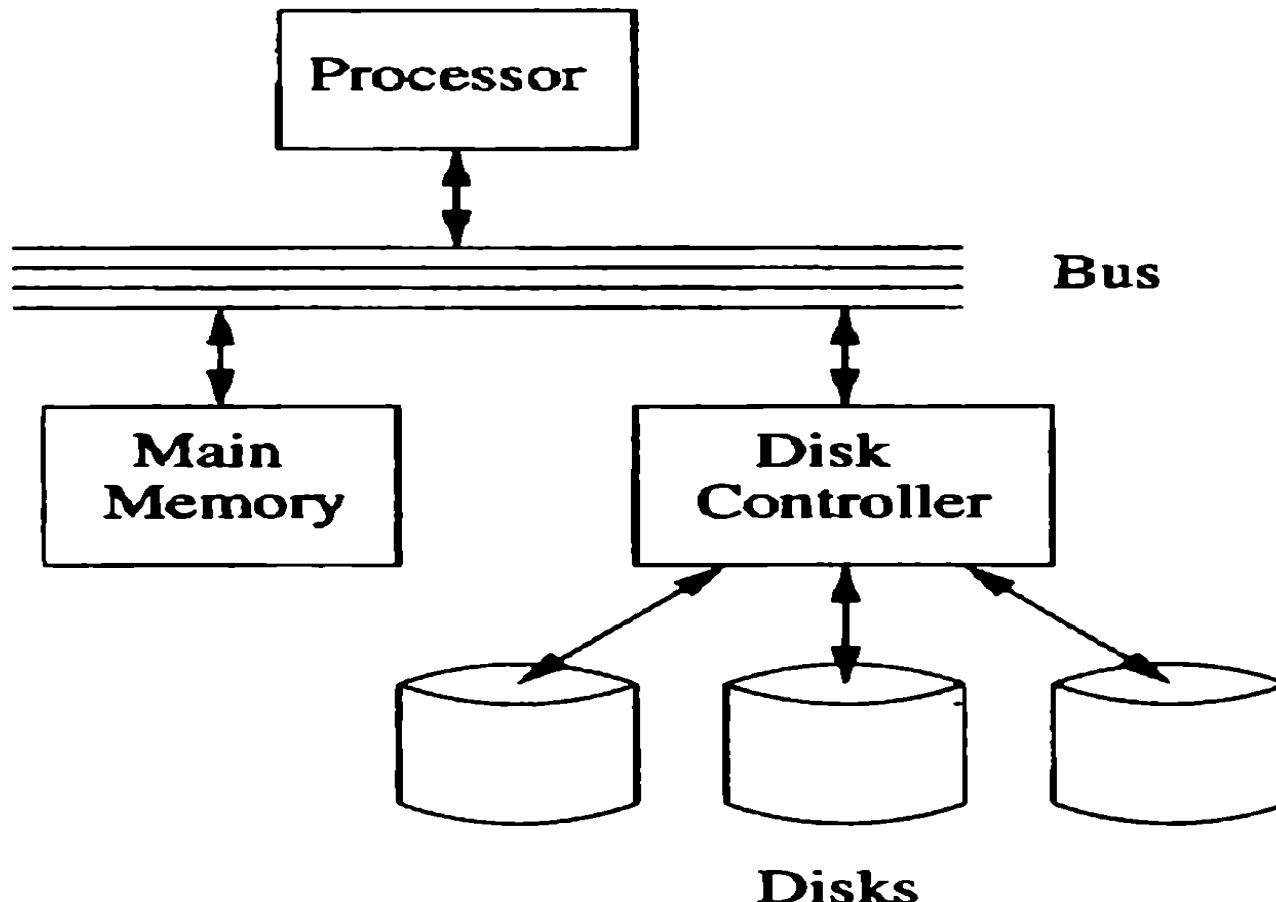
Relációs algebra lekérdezések optimalizációja

Adatbázisok használata

Mi a cél?

- Moore-törvénye: (Gordon Moore) szerint az integrált áramkörök sok jellemzőjének fejlődése exponenciális, ezek az értékek 18 havonta duplázódnak. Ilyenek például:
 - (i) processzorok sebességének és árának aránya,
 - (ii) lemez egy bitre eső ára és a lemezen tárolható bájtok száma.
- Más paraméterek azonban sokkal lassabban fejlődnek. Ilyenek például:
 - (i) központi memóriában milyen gyorsan lehet az adatokat elérni,
 - (ii) az a sebesség, amellyel a lemez mozog.
- Emiatt egy-egy nagy adathalmazzal dolgozó algoritmus optimalizációjánál az a lényeges szempont, hogy a feladatot minél kevesebb adatmozgatással tudjunk megoldani a háttértároló és a központi memória között.

Számítógép rendszer sematikus ábrája



Egy lehetséges megközelítés

- Az adatbázisoknál az előbbiek nyilván úgy értendők, hogy szeretnénk minél kevesebb **lemez olvasási és írási (I/O)** műveletet végrehajtani egy-egy lekérdezés végrehajtása során.
- Az legegyszerűbb megközelítés, ha **igyekszünk minél kisebb méretű relációkkal dolgozni**.
- Az optimalizáció során **relációs algebrai azonosságokat** fogunk alkalmazni. Ezek segítségével egy lekérdezésből az eredetivel ekvivalens lekérdezést készítünk, amelynek kiszámítása az esetek többségében kevesebb I/O műveletet igényel majd.
- A q, q' relációs algebrai lekérdezések (vagy tetszőleges lekérdezések) **ekvivalensek**, ha tetszőleges / előfordulás esetén $q(I) = q'(I)$ fennáll. Jelben: $q \equiv q'$.

Egy példa...

- A táblák legyenek:

Film (cím, év, hossz)

Szerepel (filmcím, év, színésznév)

- Ekkor a következő lekérdezés:

$\Pi_{\text{cím}}(\sigma_{\text{cím}=\text{filmcím} \wedge F.\text{év}=Sz.\text{év} \wedge \text{színésznév}='Edus'} (F \times Sz))$ ekvivalens a

$\Pi_{\text{cím}}(\sigma_{\text{cím}=\text{filmcím} \wedge F.\text{év}=Sz.\text{év}}(F \times (\sigma_{\text{színésznév}='Edus'} (Sz))))$ lekérdezéssel.

- Emellett az utóbbi valószínűleg gyorsabban végrehajtható.

Descartes-szorzat és összekapcsolások

- Asszociativitás:

$(E_1 \Theta E_2) \Theta E_3 \equiv E_1 \Theta (E_2 \Theta E_3)$, ahol $\Theta \in \{\times, | \triangleright \triangleleft | \}$ és

[természetes összekapcsolás]

$(E_1 | \triangleright \triangleleft |_{F1} E_2) | \triangleright \triangleleft |_{F2} E_3 \equiv E_1 | \triangleright \triangleleft |_{F1} (E_2 | \triangleright \triangleleft |_{F2} E_3)$, ha
 $\text{attr}(F1) \subseteq \text{attr}(E1) \cup \text{attr}(E2)$ és $\text{attr}(F2) \subseteq \text{attr}(E2) \cup \text{attr}(E3)$

- [Θ összekapcsolás]

- Kommutativitás:

$E_1 \Theta E_2 \equiv E_2 \Theta E_1$, ahol $\Theta \in \{\times, | \triangleright \triangleleft |, | \triangleright \triangleleft |_F\}$.

Projekció és szelekció

- Projekció sorozat:

$$\Pi_X(\Pi_Y(E)) \equiv \Pi_X(E), \text{ ha } X \subseteq Y.$$

- Kiválasztás és a feltételek konjunkciója:

$$\sigma_{F_1 \wedge F_2}(E) \equiv \sigma_{F_1}(\sigma_{F_2}(E)).$$

- Kiválasztás és a feltételek diszjunkciója:

$$\sigma_{F_1 \vee F_2}(E) \equiv \sigma_{F_1}(E) \cup \sigma_{F_2}(E).$$

- Kiválasztás elő projekció beillesztése:

$$\Pi_X(\sigma_F(E)) \equiv \Pi_X(\sigma_F(\Pi_Y(E))), \text{ ahol } Y = \text{attr}(F) \cup X.$$

Kiválasztás és Descartes-szorzat/összekapcsolás

- Kiválasztás és Descartes-szorzat, összekapcsolás felcserélése:
 $\sigma_F(E_1 \Theta E_2) \equiv \sigma_F(E_1) \Theta E_2$, ahol $\text{attr}(F) \subseteq \text{attr}(E_1)$ és $\Theta \in \{\times, | \triangleright \triangleleft |\}$.
- Általánosabban:
 $\sigma_F(E_1 \Theta E_2) \equiv \sigma_{F1}(E_1) \Theta \sigma_{F2}(E_2)$, ahol $\text{attr}(F_i) \subseteq \text{attr}(E_i)$ ($i = (1, 2)$)
 $F = F_1 \wedge F_2$ és $\Theta \in \{\times, | \triangleright \triangleleft |\}$.
- Ezekből levezethető:
 $\sigma_F(E_1 \Theta E_2) \equiv \sigma_{F2}(\sigma_{F1}(E_1) \Theta E_2)$, ahol $\text{attr}(F_1) \subseteq \text{attr}(E_1)$, $F = F_1 \wedge F_2$,
de $\text{attr}(F_2) \subseteq \text{attr}(E_i)$ nem teljesül ($i = (1, 2)$), $\Theta \in \{\times, | \triangleright \triangleleft |\}$.

Projekció és Descartes-szorzat/összekapcsolás

- Projekció és Descartes-szorzat, összekapcsolás felcserélése:
 $\Pi_X(E_1 \Theta E_2) \equiv \Pi_Y(E_1) \Theta \Pi_Z(E_2),$
ahol $X = Y \cup Z$, $Y \subseteq \text{attr}(E_1)$, $Z \subseteq \text{attr}(E_2)$ és $\Theta \in \{\times, | \triangleright \triangleleft\}$.

Projekció/kiválasztás és halmazműveletek

- Kiválasztás és unió (különbség) felcserélése:

$$\sigma_F(E_1 \Theta E_2) \equiv \sigma_F(E_1) \Theta \sigma_F(E_2), \text{ ahol } \Theta \in \{\cup, -\}.$$

- Projekció unióval való felcserélése:

$$\Pi_X(E_1 \cup E_2) \equiv \Pi_X(E_1) \cup \Pi_X(E_2).$$

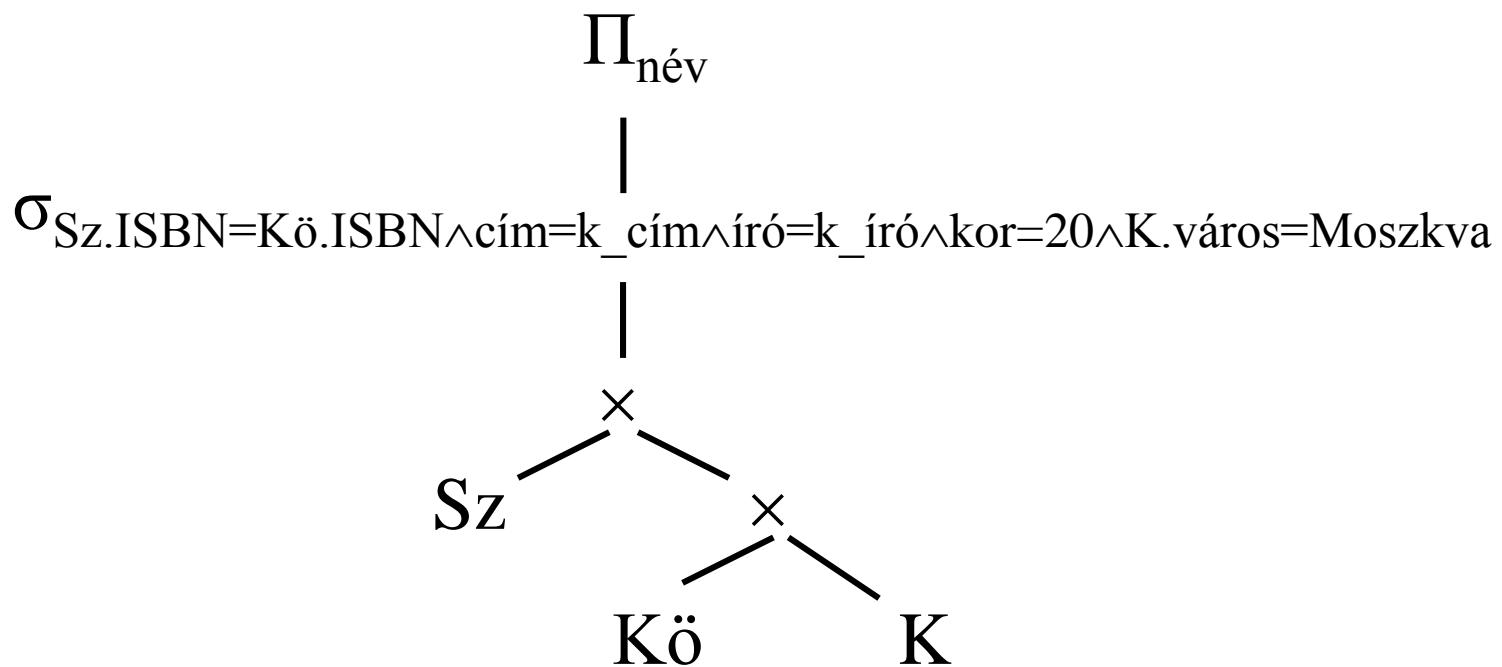
- Megjegyzés: nincs általános szabály a projekció különbséggel való felcserélésére.
- Kérdés: a metszettel mi a helyzet? [reláció séma]

Példa optimalizálásra

- A következő két feladathoz használt táblák:
Személy (név, kor, város, ISBN)
Könyv (cím, író, ISBN, ár)
Kiad (k_cím, k_iró, város, ország)
- Kik azok, akik 20 évesek, és moszkvai kiadású könyvet kölcsönöztek ki?

$$\Pi_N(\sigma_{Sz.ISBN=Kö.ISBN \wedge cím=k_cím \wedge író=k_iró \wedge kor=20 \wedge K.város=Moszkva} (Sz \times Kö \times K))$$

Lekérdezésfa

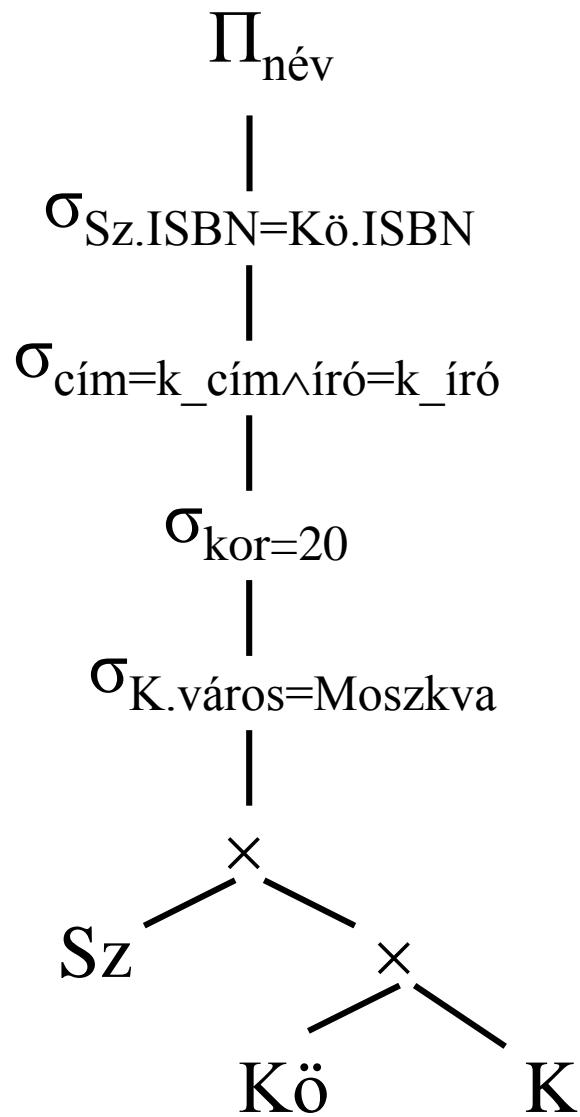


Kiválasztások "lejebb csúsztatása"

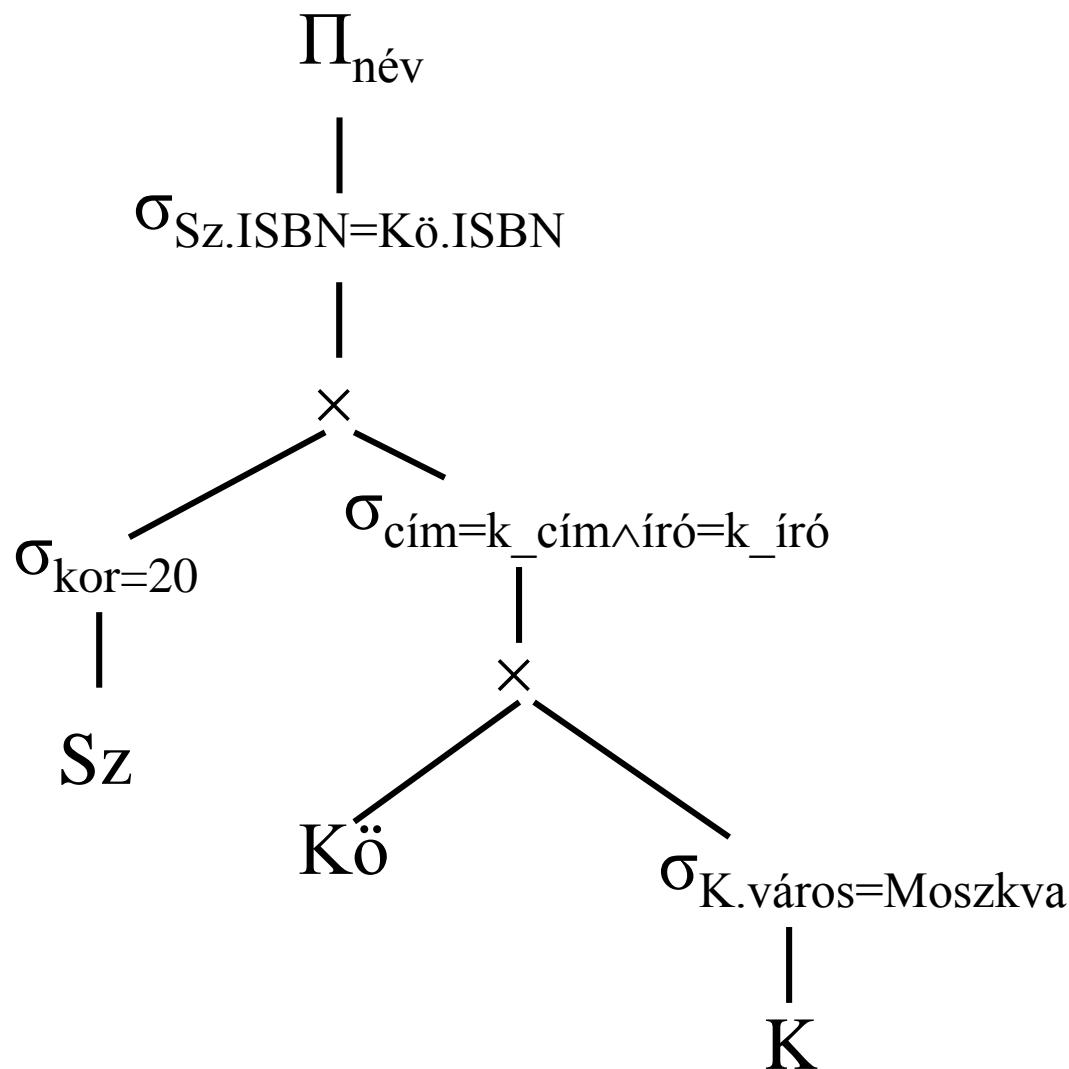
- Első lépésben a kiválasztások konjunkciós feltételeit daraboljuk szét elemi feltételekké a $\sigma_{F_1 \wedge F_2}(E) \equiv \sigma_{F_1}(\sigma_{F_2}(E))$ szabály segítségével.
- Ezek után alkalmazzuk a kiválasztás halmazműveletekkel illetve Descartes-szorzattal és a természetes összekapcsolással való felcserélésének szabályait.
- Azaz: igyekszünk a kiválasztásokat minél hamarabb végrehajtani, hiszen azok jelentősen csökkenthetik a feldolgozandó köztes relációk méretét.
- A Théta-összekapcsolást itt jobb, ha egy Descartes-szorzatra és egy azt követő kiválasztásra bontjuk.

$$R \mid \triangleright \triangleleft \mid_F S \equiv \sigma_F(R \times S).$$

Darabolás

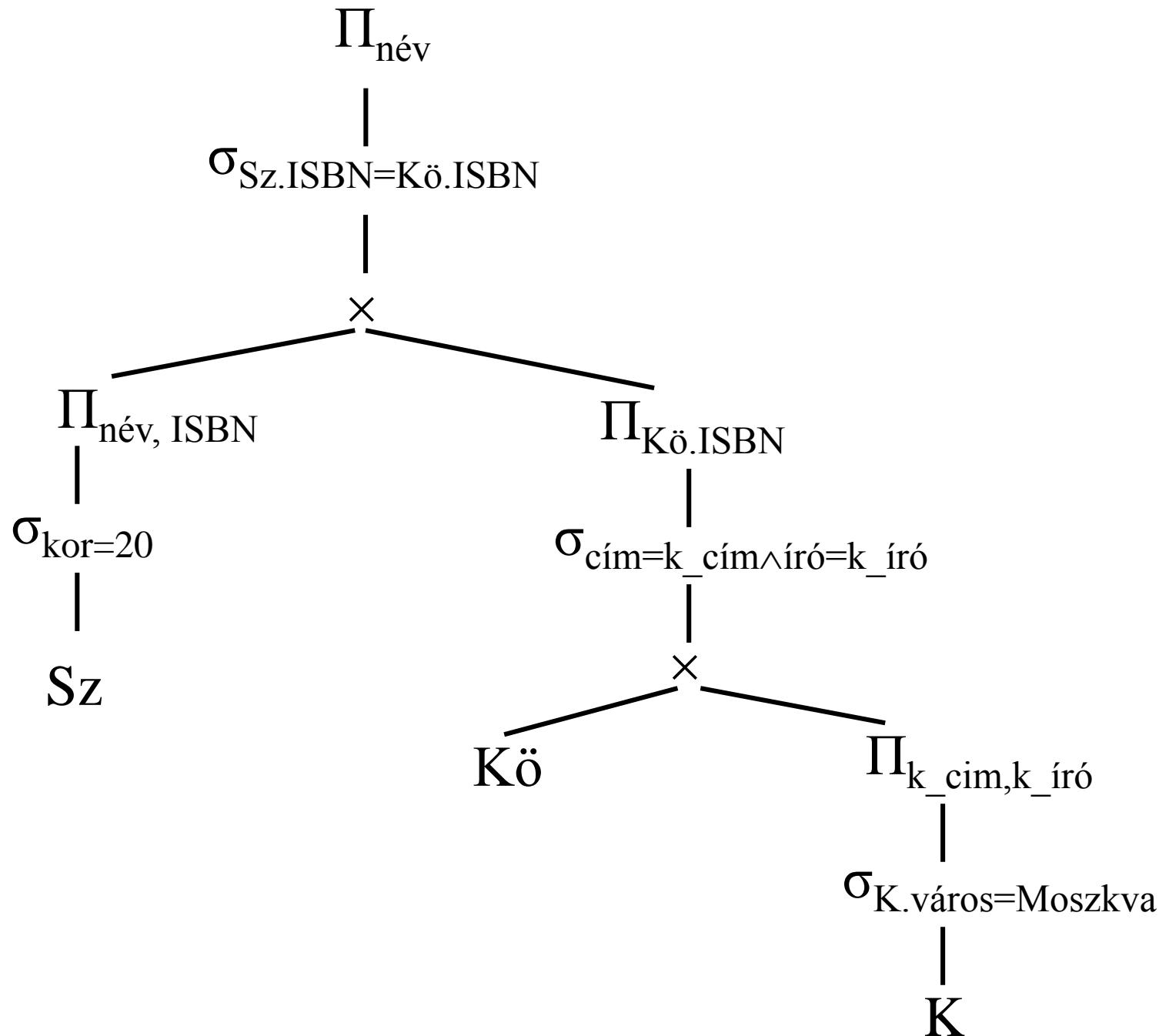


Letolás



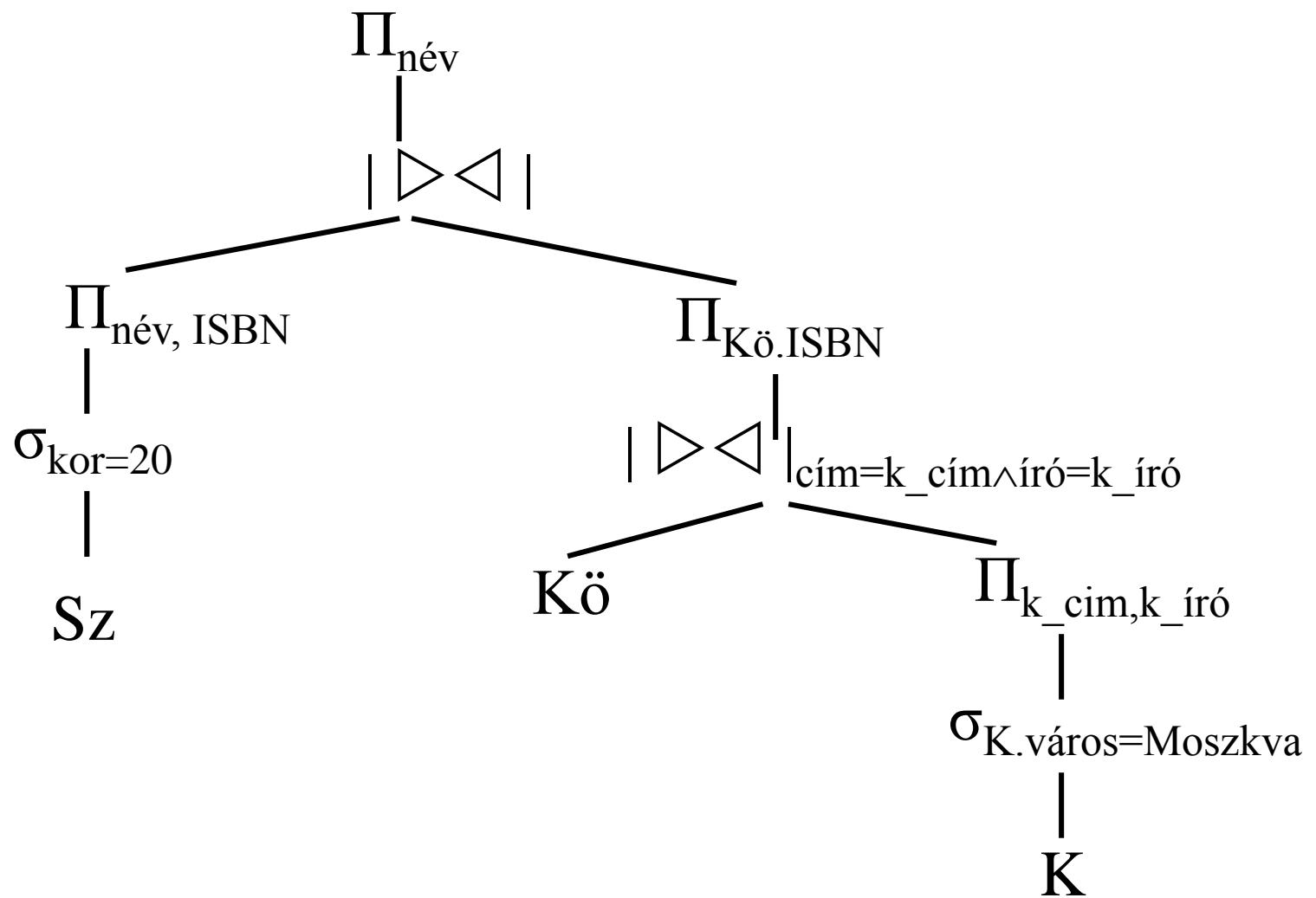
Projekciók "beírása"

- Ennél a lépésnél igyekszünk csak azokat az oszlopokat megtartani a (köztes) relációkban, amelyekre később szükség lesz.
- Általában itt nem olyan nagy a nyereség. A projekciók végrehajtása viszont időt igényel, ezért meg kell gondolni, hogy tényleg végre akarjuk-e hajtani a vetítést.
- Az átalakításoknál értelemszerűen a projekciókra vonatkozó szabályokat használjuk.



Összekapcsolások

- Az utolsó lépésben $\Pi_L(\sigma_C(R \times S))$, $\sigma_C(R \times S)$ kifejezéseket helyettesítjük természetes összekapcsolással, Théta-összekapcsolással úgy, hogy az eddigivel ekvivalens lekérdezést kapjunk.



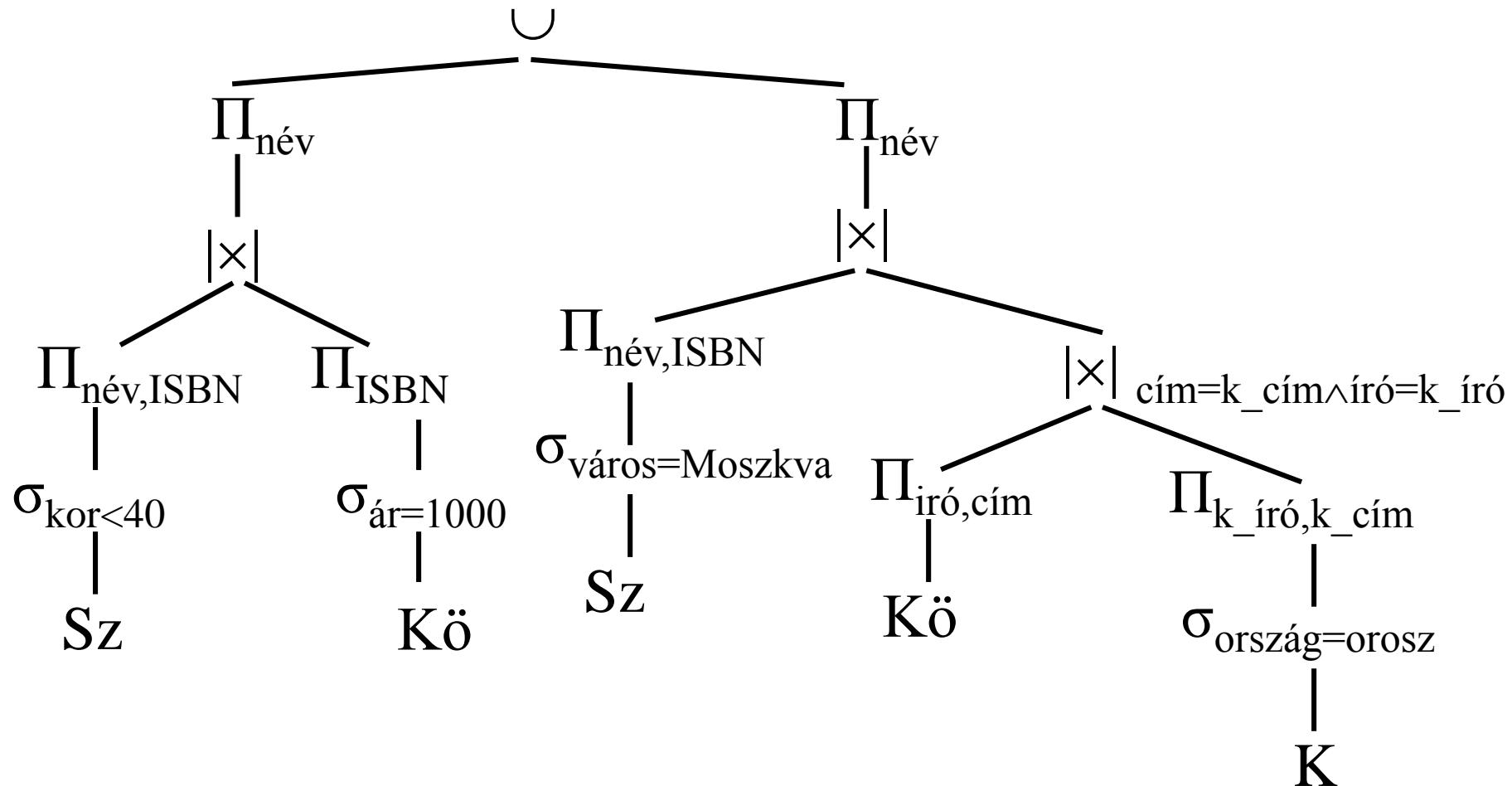
Mi történik, ha a diszjunkció is megjelenik?

- Kik azok, akik 1000 forintos könyvet vásároltak, és még nincsenek 40 évesek, vagy moszkvaiak, és orosz kiadású könyvet vettek?

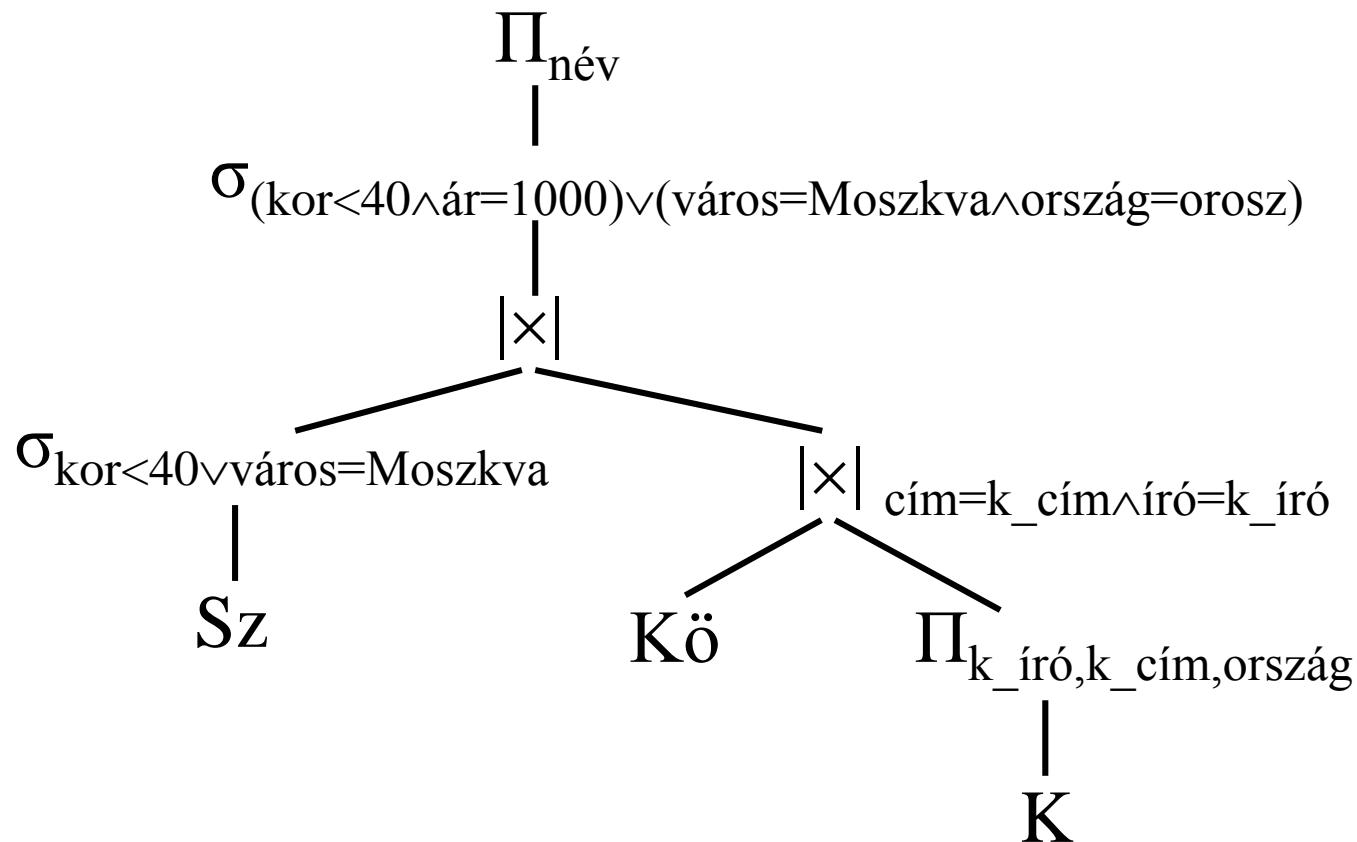
$$\Pi_N(\sigma_{C \wedge ((ár=1000 \wedge kor<40) \vee (Sz.város=Moszkva \wedge ország=orosz))} (Sz \times Kö \times K)).$$

- Itt C az Sz.ISBN = Kö.ISBN \wedge Kö.cím = K.k_cím \wedge Kö.író = K.k_író feltételt jelöli.

Megoldás I.



Megoldás II.



Összegzés

- Ha tehát a kiválasztások feltételei diszjunkciót is tartalmaznak, a helyzet bonyolultabbá válik, és nem adható olyan egyértelmű optimalizációs algoritmus, mint konjunkciók esetén.

Kiválasztások feljebb csúsztatása

- A következő példa azt szemlélteti, amikor egy kiválasztást először felfelé kell csúsztatnunk, hogy aztán letolhassuk.
- A táblák:

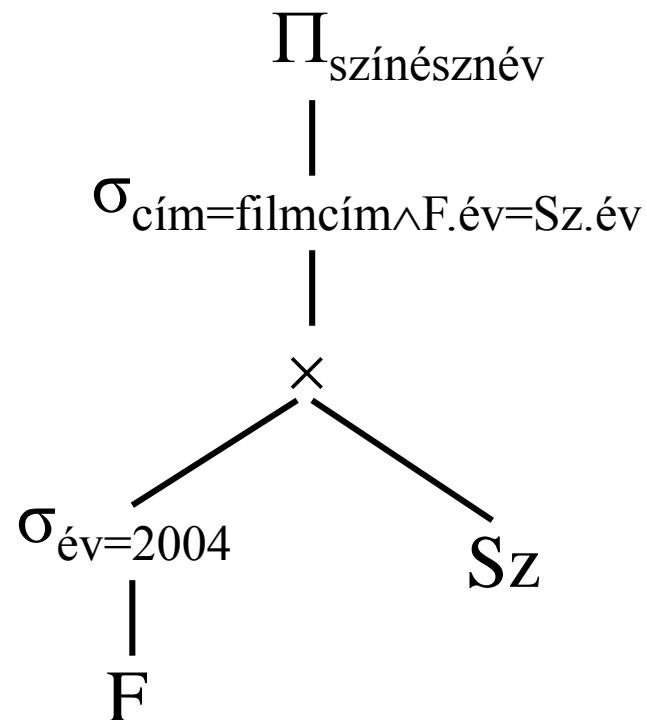
Film (cím, év, hossz)

Szerepel (filmcím, év, színésznév)

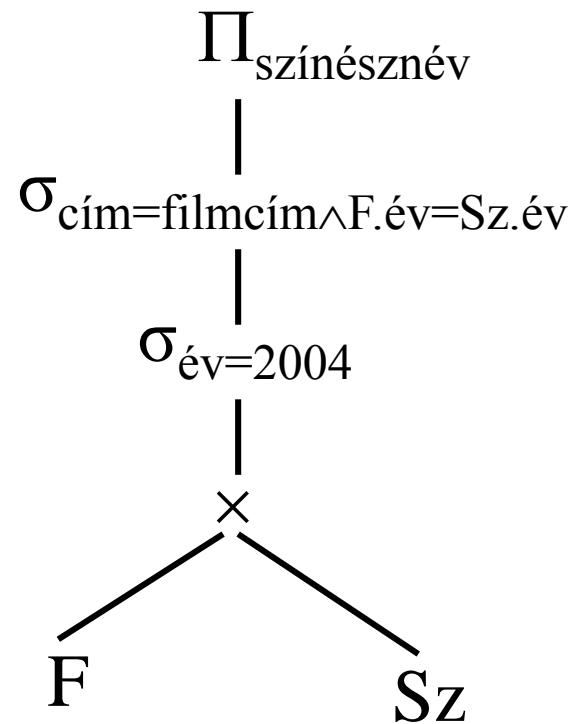
```
CREATE VIEW film04 AS  
(SELECT *  
FROM film  
WHERE év = 2004);
```

```
SELECT színésznév  
FROM film04 f, Szerepel sz  
WHERE cím = filmcím AND  
f.év = sz.év;
```

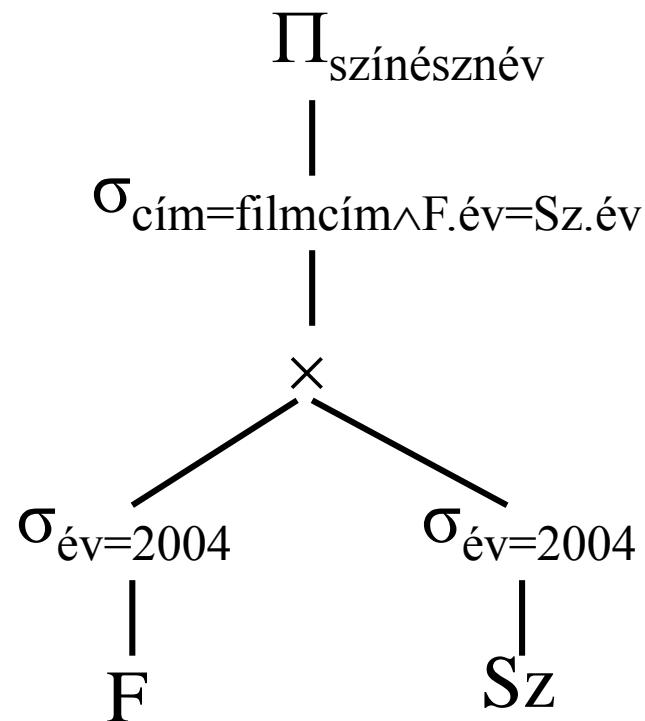
Kezdeti lekérdezésfa



Második lépés



És az eredmény...



Feladat

- A táblák legyenek:
Film (cím, év, hossz)
Szerepel (filmcím, év, színésznév)
Színész (név, kor, város)
- Adjuk meg, hogy a nem budapesti, negyven évesnél idősebb színészek milyen filmekben játszottak 1998-ban. A lekérdezést optimalizáljuk.

SQL bevezetés

Select-From-Where záradékok
Több relációt tartalmazó
lekérdezések
Alkérdések

Miért az SQL?

- ◆ Az SQL magas szintű programozási nyelv.
 - ◆ A "hogyan" helyett azt mondjuk meg, hogy "mit" szeretnénk.
 - ◆ Így elkerülünk egy csomó macerát a procedurális nyelvekhez képest, mint pl C++ vagy Java.
- ◆ Az abkezelő rendszer kitalálja a leggyorsabb végrehajtási módot.
 - ◆ Ezt nevezik "lekérdezés optimalizációnak."

Select-From-Where záradékok

SELECT az érdekes attribútumok

FROM egy vagy több tábla

WHERE a táblák soraira vonatkozó
feltételek

A példa, amit használunk

- ◆ minden SQL lekérdezést a következő adatbázisséma fölött hajtunk végre.
 - ◆ az aláhúzás a kulcsattribútumokat jelöli.

Sörök(név, gyártó)

Kocsmák(név, cím, engedélySzám)

Alkeszek(név, cím, telefon)

Szeret(alkesz, sör)

Felszolgál(kocsma, sör, ár)

Látogat(alkesz, kocsma)

Példa

- ◆ A Sörök(név, gyártó) táblában mely söröket gyártotta az Anheuser-Busch?

```
SELECT név
```

```
FROM Sörök
```

```
WHERE gyártó = 'Anheuser-Busch' ;
```

A lekérdezés eredménye

név
Bud
Bud Lite
Michelob
...

Az eredmény egyetlen attribútumot (**név**) tartalmaz a sorok Anheuser-Busch által gyártott söröket adják.

A lekérdezés jelentése

- ◆ Kezdjük a FROM záradékban megadott relációval.
- ◆ Alkalmazzuk a WHERE záradékban megadott kiválasztási feltételt.
- ◆ Vetítsük le az eredményt a SELECT záradékban megadott oszlopokra.

Szemantika (a példában)

név	gyártó
Bud	Anheuser-Busch

A *t* sor változóval
a sorokat vesszük
egymás után.

t.név bekerül az
eredménybe, ha igen.
Ellenőrizzük, hogy
Anheuser-Busch-e.

* a SELECT záradékban

- ◆ Ha egy reláció szerepel a FROM záradékban, a * SELECT záradékban a reláció összes attribútumát helyettesíti.
- ◆ Példa: Sörök(név, gyártó):

```
SELECT *
FROM Sörök
WHERE gyártó = 'Anheuser-Busch';
```

A válasz:

név	gyártó
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Michelob	Anheuser-Busch
...	...

Azaz a **Sörök** reláció összes attribútuma szerepel.

Attribútumok átnevezése

- ◆ Az attribútumok átnevezéséhez "AS <new name>" utasítást használhatjuk.
- ◆ Példa: Sörök(név, gyártó):

```
SELECT név AS sör, gyártó
```

```
FROM Sörök
```

```
WHERE gyártó = 'Anheuser-Busch';
```

Az eredmény:

sör	gyártó
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Michelob	Anheuser-Busch
...	...

A SELECT záradék kifejezései

- ◆ minden kifejezés, ami „értelmesnek tűnik” megjelenhet a SELECT záradékban.
- ◆ Példa: Felszolgál(kocsma, sör, ár):

```
SELECT kocsma, sör,  
       ár*114 AS árJenben  
FROM Felszolgál;
```

Az eredmény

kocsma	sör	árJenben
Joe's	Bud	285
Sue's	Miller	342
...

Konstansok

◆ Szeret(alkesz, sör):

```
SELECT alkesz,  
       ' szereti a Budot' AS  
      BudIvó  
  
FROM Szeret  
  
WHERE sör = ' Bud' ;
```

Result of Query

alkesz	BudIvó
Sally	szereti a Budot
Fred	szereti a Budot
...	...

Információ integráció

- ◆ Sokszor az adatbázisokat sok forrásból építik fel (adattárházak).
- ◆ Tegyük fel, hogy minden kocsmának van egy saját Menü(sör, ár) táblája.
- ◆ A Felszolgál(kocsma, sör, ár) tábla elkészítéséhez minden ilyen táblát kell dolgoznunk és a kocsma nevét konstansként kell beszúrnunk.

Információ integráció --- (2)

- ◆ Például Joe bárja esetében ezzel a lekérdezéssel dolgozhatunk:

```
SELECT 'Joe bárja', sör, ár  
FROM Menü;
```

Összetett feltételek a WHERE záradékban

- ◆ Logikai műveletek: AND, OR, NOT.
- ◆ Összehasonlítások =, <>, <, >, <=, >=.

Példa összetett feltételre

- ◆ A **Felszolgál(kocsma, sör, ár)** táblában keressük meg Joe bárjában mennyit kérnek a Bud sörért:

```
SELECT ár
```

```
FROM Felszolgál
```

```
WHERE kocsma = 'Joe bárja' AND  
sör = 'Bud';
```

Minták

- ◆ A feltételekben a szavakat mintákra illeszthetjük
 - ◆ <Attribútum> LIKE <minta> vagy <Attribútum> NOT LIKE <minta>
- ◆ *Minta* aposztrófok közötti szöveg az alábbi jelekkel: % = "akármennyi karakter"; _ = "tetszőleges karakter, pontosan egy."

Példa: LIKE

- ◆ Az Alkeszek(név, cím, telefon) keressük a budapestieket.

```
SELECT név  
FROM Alkeszek  
WHERE cím LIKE '%Budapest%';
```

NULL értékek

- ◆ A sorok mezői az SQL relációkban NULL értékeket is tartalmazhatnak.
- ◆ A jelentés a kontextustól függően változhat. Általában:
 - ◆ *hiányzó érték* : pl. nem ismerjük Joe bárja címét.
 - ◆ *értelemetlen* : egy szingli esetében a házastárs neve.

NULL összehasonlítás

- ◆ Az SQL valójában 3-értékű logikát használ: TRUE, FALSE, UNKNOWN.
- ◆ Ha egy értéket (NULL értéket is beleérte) NULL-lal hasonlítunk, az eredmény UNKNOWN.
- ◆ Egy sor akkor és csak akkor kerül be az eredménybe, ha a WHERE záradék TRUE értéket ad.

3-értékű logika

- ◆ Tegyük fel a következőt: TRUE = 1,
FALSE = 0, and UNKNOWN = $\frac{1}{2}$.
- ◆ Ekkor: AND = MIN; OR = MAX, NOT(x)
= $1-x$.

◆ Példa:

TRUE AND (FALSE OR NOT(UNKNOWN))
= MIN(1, MAX(0, (1 - $\frac{1}{2}$))) =
MIN(1, MAX(0, $\frac{1}{2}$)) = MIN(1, $\frac{1}{2}$) = $\frac{1}{2}$.

Meglepetés!

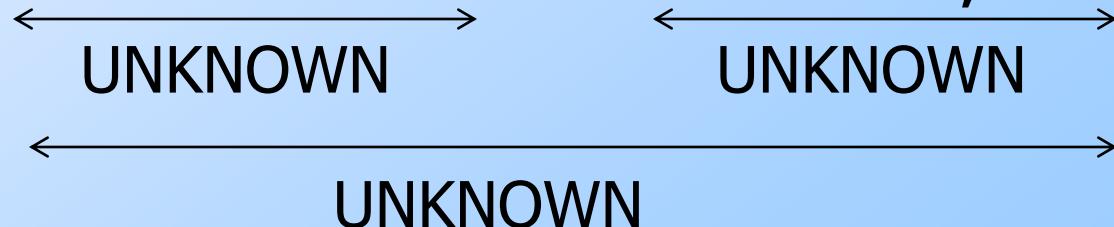
- ◆ Az alábbi Felszolgál tábla esetén:

kocsma	sör	ár
Joe bárja	Bud	NULL

SELECT kocsma

FROM Felszolgál

WHERE ár < 2.00 OR ár >= 2.00;



Többrelációs lekérdezések

- ◆ Általában több táblából kell kinyernünk az adatokat.
- ◆ Ekkor a relációkat a FROM záradékban kell felsorolnunk.
- ◆ Az azonos attribútum neveket az alábbi módon különböztetjük meg egymástól: "<reláció>.<attribútum>".

Példa: két reláció összekapcsolása

```
SELECT sör
FROM Szeret, Látogat
WHERE kocsma = 'Joe bárja' AND
Látogat.alkesz = Szeret.alkesz;
```

Formális szemantika

- ◆ Majdnem ugyanaz, mint korábban:
 1. Vegyük a FROM záradékban szereplő relációs Descartes-szorzatát.
 2. Alkalmazzuk a WHERE záradék feltételét.
 3. Vetítsünk a SELECT záradék oszlopaira.

Működési (operációs) szemantika

- ◆ Képzeljük úgy, mintha minden FROM záradékbeli táblához tartozna egy sorváltozó.
 - ◆ Ezekkel a sorok összes lehetséges kombinációját vesszük.
- ◆ Ha a sorváltozók a WHERE záradékot kielégítő sorra mutatnak, küldjük el ezeket a sorokat a SELECT záradékba.

Példa

alkesz	kocsma
Sally	Joe

sv1

alkesz	sör
Sally	Bud

sv2

Látogat

Joe-e?

Szeret

ellenőrzük az
egyenlőséget

bekerül az
eredménybe

Explicit sorváltozók

- ◆ Esetenként egy tábla több példányára is szükségünk van.
- ◆ A FROM záradékban a relációk neve után adjuk meg a hozzájuk tartozó sorváltozók nevét.
- ◆ Egy relációt minden átnevezhetünk ily módon, akkor is, ha egyébként nincs rá szükség.

Példa: önmagával vett összekapcsolás

```
SELECT b1.név, b2.név  
FROM Sörök b1, Sörök b2  
WHERE b1.gyártó = b2.gyártó AND  
b1.név < b2.név;
```

Alkérdések

- ◆ A FROM és WHERE záradékban zárójelezett SELECT-FROM-WHERE utasításokat (*alkérdés*) is használhatunk.
- ◆ Példa: a FROM záradékban a létező relációk mellett, alkérdéssel létrehozott ideiglenes táblát is megadhatunk.
 - ◆ Ilyenkor a legtöbb esetben explicite meg kell adnunk a sorváltozó nevét.

Példa: alkérdés FROM-ban

- ◆ Keressük meg a Joe bárja vendégei által kedvelt söröket.

```
SELECT sört
```

```
FROM Szeret, (SELECT alkesz
```

```
FROM Látogat
```

```
WHERE kocsma = 'Joe bárja') JD
```

```
WHERE Szeret.alkesz = JD.alkesz;
```

Alkeszek, akik látogatják
Joe bárját.

Egy sort visszaadó alkérdések

- ◆ Ha egy alkérdés biztosan egy sort ad vissza eredményként, akkor úgy használható, mint egy konstans érték.
 - ◆ Általában az eredmény sornak egyetlen oszlopa van.
 - ◆ Futásidéjű hiba keletkezik, ha az eredmény nem tartalmaz sort, vagy több sort tartalmaz.

Példa: egysoros alkérdés

- ◆ A **Felszolgál(kocsma, sör, ár)** táblában keressük meg azokat a kocsmákat, ahol a Miller ugyanannyiba kerül, mint Joe bárjában a Bud.
- ◆ Két lekérdezésre biztos szükségünk lesz:
 1. Mennyit kér Joe a Budért?
 2. Melyik kocsmákban adják ugyanennyiért a Millert?

Kérdés + alkérdés

SELECT kocsma

FROM Felszolgál

WHERE sört = 'Miller' AND

ár = (S

Ennyit kér
Joe a Budáért.

```
SELECT ár
FROM Felszolgál
WHERE kocsma = 'Joe bárja'
AND sört = 'Bud');
```

Az IN művelet

- ◆ <sor> IN (<alkérdés>) igaz, akkor és csak akkor, ha a sor eleme az alkérdés eredményének.
 - ◆ Tagadás: <sor> NOT IN (<alkérdés>).
- ◆ Az IN-kifejezések a WHERE záradékban jelenhetnek meg.

Példa: IN

```
SELECT *  
FROM Sörök  
WHERE név IN (
```

```
SELECT sör  
FROM Szeret  
WHERE alkesz = 'Fred');
```

A sörök,
melyeket Fred
kedvel.

Mi a különbség?

```
SELECT a  
FROM R, S  
WHERE R.b = S.b;
```

```
SELECT a  
FROM R  
WHERE b IN (SELECT b FROM S);
```

IN az R soraira vonatkozó predikátum

```
SELECT a  
FROM R  
WHERE b IN
```

Két 2 érték.

(SELECT b FROM S) ;

Egy ciklus R sorai
fölött.

a	b
1	2
3	4

R

b	c
2	5
2	6

S

(1,2) kielégíti a
feltételt;
1 egyszer jelenik
meg az
eredményben.

Itt R és S sorai párosítjuk

```
SELECT a  
FROM R, S  
WHERE R.b = S.b;
```

Dupla ciklus R és S
sorai fölött

a	b	c
1	2	2
3	4	5

R S

(1,2) és (2,5)
(1,2) és (2,6)
is kielégíti a
feltételt;
1 kétszer kerül
be az eredménybe.

Az EXISTS művelet

- ◆ EXISTS(<alkérdés>) akkor és csak akkor igaz, ha az alkérdés eredménye nem üres.
- ◆ Példa: A Sörök(név, gyártó) táblában keressük meg azokat a söröket, amelyeken kívül a gyártójuk nem gyárt másikat.

Példa: EXISTS

```
SELECT név  
FROM Sörök b1  
WHERE NOT EXISTS (
```

Azon b1 sörtől különböző sörök, melyeknek ugyanaz a gyártója.

```
SELECT *  
FROM Sörök  
WHERE gyártó = b1.gyártó AND  
név <> b1.név);
```

Változók láthatósága: itt a a gyártó a legközelebbi beágyazott FROM-beli táblából való, aminek van ilyen attribútuma.

A „nem egyenlő” művelet SQL-ben.

Az ANY művelet

- ◆ $x = \text{ANY}(<\text{alkérdés}>)$ akkor és csak akkor igaz, ha x egyenlő az alkérdés legalább egy sorával.
 - ◆ $=$ helyett bármilyen aritmetikai összehasonlítás szerepelhet.
- ◆ Példa: $x > \text{ANY}(<\text{alkérdés}>)$ akkor igaz, ha x az alkérdés legkisebb eleménél nagyobb.
 - ◆ Itt az alkérdés sorai egy mezőből állnak.

Az ALL művelet

- ◆ $x <> \text{ALL}(<\text{alkérdés}>)$ akkor és csak akkor igaz, ha x az alkérdés egyetlen sorával sem egyezik meg.
- ◆ $<>$ helyett tetszőleges összehasonlítás szerepelhet.
- ◆ Példa: $x >= \text{ALL}(<\text{subquery}>)$ x az alkérdés eredményének maximum értékével azonos, vagy nagyobb nála.

Példa: ALL

```
SELECT sör  
FROM Felszolgál
```

```
WHERE ár >= ALL(  
    SELECT ár  
    FROM Felszolgál);
```

A külső lekérdezés Felszolgáljának söre egyetlen alkérdezésbeli sörnél sem lehet olcsóbb.

Unió, metszet, különbség

- ◆ A szintaxis:
 - ◆ (<alkérdés>) UNION (<alkérdés>)
 - ◆ (<alkérdés>) INTERSECT (<alkérdés>)
 - ◆ (<alkérdés>) MINUS (<alkérdés>)
- ◆ MINUS helyett EXCEPT is szerepelhet.

Példa: metszet

- ◆ A Szeret(alkesz, sör), Felszolgál(kocsma, sör, ár) és Látogat(alkesz, kocsma) táblák segítségével keressük meg azon alkeszeket és söröket:
 1. ahol az alkesz szereti az adott sört,
 2. az alkesz legalább egy olyan kocsmát látogat, ahol felszolgálják a szóban forgó sört.

Az alkérdés
egy tárolt
táblát ad
vissza.

Megoldás

```
(SELECT * FROM Szeret)
```

INTERSECT

```
(SELECT alkesz, sör
FROM Felszolgál, Látogat
WHERE Felszolgál.kocsma =
Látogat.kocsma);
```

Az alkesz látogatja
azt a kocsmát, ahol
felszolgálják azt a
sört.

Multihalmaz szemantika

- ◆ A SELECT-FROM-WHERE állítások multihalmaz szemantikát használnak, a halmazműveleteknél mégis a halmaz szemantika az érvényes.
 - ◆ Azaz sorok nem ismétlődnek az eredményben.

Motiváció: hatékonyság

- ◆ Ha projektálunk, akkor egyszerűbb, ha nem töröljük az ismétlődéseket.
 - ◆ Csak szépen végigmegyünk a sorokon.
- ◆ A metszet, különbség számításakor általában az első lépésben lerendezik a táblákat.
 - ◆ Ez után az ismétlődések kiküszöbölése már nem jelent extra számításigényt.

Ismétlődések kiküszöbölése

- ◆ mindenéppen törlődjenek az ismétlődések: SELECT DISTINCT . . .
- ◆ Ne törlődjenek az ismétlődések:
pl: SELECT ALL . . . vagy
. . . UNION ALL . . .

Példa: DISTINCT

```
SELECT DISTINCT ár  
FROM Felszolgál;
```

Példa: ALL

- ◆ A Látogat(alkesz, kocsma) and Szeret(alkesz, sör) táblák felhasználásával:

```
(SELECT alkesz FROM Látogat)
```

```
    EXCEPT ALL
```

```
(SELECT alkesz FROM Szeret);
```

- ◆ Kilistázza azokat az alkeszeket, akik több kocsmát látogatnak, mint amennyi sört szeretnek, és a két leszámlálás különbsége azt mutatja, hogy mennyivel több kocsmát látogatnak mint amennyi sört kedvelnek.

Join kifejezések

- ◆ Az SQL-ben számos változata megtalálható az összekapcsolásoknak.
- ◆ Ezek a kifejezések önmagukban is állhatnak lekérdezésként, vagy a FROM záradékban is megjelenhetnek.

Descartes szorzat és természetes összekapcsolás

- ◆ Természetes összekapcsolás:
R NATURAL JOIN S;
- ◆ Szorzat:
R CROSS JOIN S;
- ◆ Példa:
Szeret NATURAL JOIN Felszolgál;
- ◆ A relációk helyén zárójelezett alkérdések is szerepelhetnek.

Théta-összekapcsolás

- ◆ R JOIN S ON <feltétel>
- ◆ Példa: az Alkesz(név, cím) és Látogat(alkesz, kocsma) táblákból:

Alkesz JOIN Látogat ON
név = alkesz;

azokat (d, a, d, b) négyeseket adja vissza,
ahol a d alkesz a címen lakik és a b
kocsmát látogatja.

SQL haladó

Külső összekapcsolások,
Csoportosítás/Összesítés,
Beszúrás/Törlés/Módosítás,
Táblák létrehozása/Kulcs
megszorítások

Külső összekapcsolás

- ◆ Összekapcsoljuk R és S relációkat: $R \bowtie_C S$.
- ◆ R azon sorait, melyeknek nincs S-beli párja *lógó* soroknak nevezzük.
 - ◆ S -nek is lehetnek lógó sorai.
- ◆ A külső összekapcsolás megőrzi a lógó sorokat NULL értékkel helyettesítve a hiányzó értékeket.

Példa: külső összekapcsolás

$$R = (\begin{array}{|c|c|} \hline A & B \\ \hline 1 & 2 \\ \hline 4 & 5 \\ \hline \end{array})$$
$$S = (\begin{array}{|c|c|} \hline B & C \\ \hline 2 & 3 \\ \hline 6 & 7 \\ \hline \end{array})$$

(1,2) és (2,3) összekapcsolható, a másik két sor azonban „lög”.

R OUTERJOIN S =

A	B	C
1	2	3
4	5	NULL
NULL	6	7

Külső összekapcsolás (SQL)

- ◆ R OUTER JOIN S: a külső összekapcsolásoknál mindig ez szerepel.

1. Opcionális NATURAL az OUTER előtt.
2. Opcionális ON <feltétel> JOIN után.
3. Opcionális LEFT, RIGHT, vagy FULL az OUTER előtt.
 - ◆ LEFT = csak R lógó sorait őrzi meg.
 - ◆ RIGHT = csak S lógó sorait őrzi meg.
 - ◆ FULL = az összes lógó sort megőrzi.

Csak az egyik szerepelhet.

Összesítések (aggregáció)

- ◆ SUM, AVG, COUNT, MIN, és MAX összesítő függvényeket a SELECT záradékban alkalmazhatjuk egy-egy oszlopra.
- ◆ COUNT(*) az eredmény sorainak számát adja meg.

Példa: Összesítés

- ◆ A Felszolgál(kocsma, sör, ár) tábla segítségével adjuk meg a Bud átlagos árát:

```
SELECT AVG (ár)  
FROM Felszolgál  
WHERE sör = 'Bud' ;
```

Ismétlődések kiküszöbölése összesítésben

- ◆ Az összesítő függvényen belül DISTINCT.
- ◆ Példa: hány *különféle* áron árulják a Bud sört?

```
SELECT COUNT(DISTINCT ár)  
FROM Felszolgál  
WHERE sör = 'Bud';
```

NULL értékek nem számítanak az összesítésben

- ◆ NULL soha nem számít a SUM, AVG, COUNT, MIN, MAX függvények kiértékelésekor.
- ◆ De ha nincs NULL értéktől különböző érték az oszlopban, akkor az összesítés eredménye NULL.
 - ◆ **Kivétel:** COUNT az üreshalmazon 0-t ad vissza.

Példa: NULL értékek összesítésben

```
SELECT count(*)  
FROM Felszolgál  
WHERE sör = 'Bud';
```

A Bud sört árusító kocsmák száma.

```
SELECT count(ár)  
FROM Felszolgál  
WHERE sör = 'Bud';
```

Azon kocsmák száma, ahol ismerjük a Bud sör árát.

Csoportosítás

- ◆ Egy SELECT-FROM-WHERE kifejezést GROUP BY záradékkal folytathatunk, melyet attribútumok listája követ.
- ◆ A SELECT-FROM-WHERE eredménye a megadott attribútumok értékei szerint csoportosítódik, az összesítéseket ekkor minden csoportra külön alkalmazzuk.

Példa: Csoportosítás

- ◆ A **Felszolgál(kocsma, sör, ár)** tábla segítségével adjuk meg a sörök átlagos árát.

```
SELECT sör, AVG(ár)  
FROM Felszolgál  
GROUP BY sör;
```

beer	AVG(price)
Bud	2.33
Miller	2.45

Példa: Csoportosítás



SELECT alkesz, AVG(ár)

FROM Látogat, Felszolgál

WHERE sör = 'Bud' AND

Látogat.kocsma =

Felszolgál.kocsma

GROUP BY alkesz;

Alkesz-
kocsma-ár
hármasok a
Bud söre.

Alkeszek
szerinti
csoportosítás.

A SELECT lista és az összesítések

- ◆ Ha összesítés is szerepel a lekérdezésben, a SELECT-ben felsorolt attribútumokra a következő érvényes
 1. Összesítések, amelyekben egy összesítési operátort alkalmazunk egy attribútumra vagy egy attribútumot tartalmazó kifejezésre. Ezek a kifejezések csoportonként kerülnek kiértékelésre.
 2. Attribútumok, amelyek a GROUP BY záradékban szerepelnek, mint a példában *alkesz*. Egy összesítéset tartalmazó SELECT záradékban csak a GROUP BY záradékban is megtalálható attribútumok jelenhetnek meg összesítési operátor nélkül.

Helytelen lekérdezés

- ◆ Elsőre sokan gondolhatják azt, hogy az alábbi lekérdezés a Bud sört legolcsóbban áruló kocsmát adja vissza:
 - ◆ *SELECT kocsma, MIN(ár)*
FROM Felszolgál
WHERE sör= 'Bud';
- ◆ Valójában ez egy helytelen SQL lekérdezés.

HAVING záradék

- ◆ A GROUP BY záradékot egy HAVING <feltétel> záradék követheti.
- ◆ Ebben az esetben a feltétel az egyes csoportokra vonatkozik, ha egy csoport nem teljesíti a feltételt, nem lesz benne az eredményben.

Példa: HAVING

- ◆ A Felszolgál(kocsma, sör, ár) és Sörök(név, gyártó) táblák felhasználásával adjuk meg az átlagos árat azon söröknél, melyeket legalább három kocsmában felszolgálnak, vagy Pete a gyártójuk.

Megoldás

```
SELECT sör, AVG(ár)  
FROM Felszolgál  
GROUP BY sör
```

Sör csoportok, melyeket legalább 3 nem-NULL kocsmában árulnak, vagy Pete a gyártójuk.

```
HAVING COUNT(kocsma) >= 3 OR
```

```
sör IN (SELECT név  
        FROM Sörök  
        WHERE gyártó = 'Pete');
```

Sörök,
melyeket
Pete gyárt.

A HAVING feltételére vonatkozó megszorítások

- ◆ Az alkérdesre nincs megszorítás.
- ◆ Az alkérdesen kívül csak olyan attribútumok szerepelhetnek, amelyek:
 1. vagy csoportosító attribútumok,
 2. vagy összesített attribútumok.

(Azaz ugyanazok a szabályok érvényesek, mint a SELECT záradéknál).

- ◆ A HAVING záradékban hivatkozott összesítés csak az éppen feldolgozott csoport soraira vonatkozik.
- ◆ • A FROM záradékban megadott relációk bármely attribútumára képezhetünk a HAVING záradékban összesítést, összesítés nélkül a HAVING záradékban csak azok az attribútumok fordulhatnak elő, amelyek a GROUP BY listában is szerepeltek. (Ugyanaz a szabály, mint ami a SELECT záradékra is vonatkozott.)

Adatbázis módosítások

- ◆ A *módosító* utasítások nem adnak vissza eredményt, mint a lekérdezések, hanem az adatbázis tartalmát változtatják meg.
- ◆ 3-féle módosító utasítás létezik:
 1. *Insert* (beszúrás).
 2. *Delete* (törlés).
 3. *Update* (létező sorok értékeinek módosítása).
- ◆ Data Manipulation Language (DML).

Beszúrás

- ◆ Ha egyetlen sort szúrunk be:

INSERT INTO <reláció>

VALUES (<attribútum lista>);

- ◆ Példa: a *Szeret(alkesz, söر)* táblában rögzítjük, hogy Zsuzska szereti a Bud sört.

INSERT INTO Szeret

VALUES ('Zsuzska', 'Bud');

Az INSERT-nél megadhatjuk az attribútumokat

- ◆ A reláció neve után megadhatjuk az attribútumait.
- ◆ Ennek két oka lehet:
 1. elfelejtettük, hogy a reláció definíciójában, milyen sorrendben szerepeltek az attribútumok.
 2. Nincs minden attribútumnak értéke, és azt szeretnénk, ha a hiányzó értékeket NULL vagy default értékkel helyettesítenék.

Példa: Attribútumok megadása

```
INSERT INTO Szeret(sör, alkesz)  
VALUES ('Bud', 'Zsuzska');
```

Default (alapértelmezett) értékek megadása

- ◆ A CREATE TABLE utasításban az oszlopnevet DEFAULT kulcsszó követheti és egy érték.
- ◆ Ha egy beszúrt sorban hiányzik az adott attribútum értéke, akkor a default értéket kapja.

Példa: Default (alapértelmezett) érték

```
CREATE TABLE Alkeszek (
    név CHAR(30) PRIMARY KEY,
    cím CHAR(50)
        DEFAULT '123 Sesame St.',
    telefon CHAR(16)
) ;
```

Példa: Default (alapértelmezett) értékek

```
INSERT INTO Alkeszek(név)  
VALUES ('Zsuzska');
```

Az eredmény sor:

név	cím	telefon
Zsuzska	123 Sesame St	NULL

Több sor beszúrása

- ◆ Egy lekérdezés eredményét is beszúrhatjuk a következő módon:

```
INSERT INTO <reláció>
( <alkérdés> );
```

Példa: Beszúrás alkérdéssel

- ◆ A Látogat(alkesz, kocsma) tábla felhasználásával adjuk hozzá a Szesztestvérek(név) táblához Zsuzska szesztestvéreit, vagyis azokat, akikkel legalább egy közös kocsmát látogatnak.

A szesztestvér.

Megoldás

```
INSERT INTO Szesztestvérek
```

```
(SELECT I2.alkesz
```

```
FROM Látogat l1, Látogat l2
```

```
WHERE l1.alkesz = 'Zsuzska' AND
```

```
l2.alkesz <> 'Zsuzska' AND
```

```
l1.kocsma = l2.kocsma
```

```
);
```

Alkesz párok: az első Zsuzska, a második nem Zsuzska, de van olyan kocsma, amit mindketten látogatnak.

Törlés

- ◆ A törlendő sorokat egy feltétel segítségével adjuk meg:

```
DELETE FROM <reláció>
WHERE <feltétel>;
```

Példa: Törlés

```
DELETE FROM Szeret  
WHERE alkesz = 'Zsuzska' AND  
sör = 'Bud';
```

Példa: Az összes sor törlése

```
DELETE FROM Likes;
```

Példa: Több sor törlése

- ◆ A **Sörök(név, gyártó)** táblából töröljük azokat a söröket, amelyekhez létezik olyan sör, amit ugyanaz a cég gyártott.

DELETE FROM Sörök s

WHERE EXISTS (

SELECT név FROM Sörök

WHERE gyártó = s.gyártó

AND név <> s.név);

Azok a sörök, amelyeknek ugyanaz a gyártója, mint az s éppen aktuális sorának, a neük viszont különböző.

A törlés szemantikája --- (1)

- ◆ Tegyük fel, hogy az Anheuser-Busch csak Bud és Bud Lite söröket gyárt.
- ◆ Tegyük fel még, hogy *s* sorai közt a Bud fordul elő először.
- ◆ Az alkérdés nem üres, a későbbi Bud Lite sor miatt, így a Bud törlődik.
- ◆ Kérdés, hogy a Bud Lite sor törlődik-e?

A törlés szemantikája --- (2)

- ◆ **Válasz:** igen, a Bud Lite sora is törlődik.
- ◆ A törlés ugyanis két lépésben hajtódik végre.
 1. Kijelöljük azokat a sorokat, amelyekre a WHERE feltétele teljesül.
 2. Majd töröljük a kijelölt sorokat.

Módosítás

- ◆ Bizonyos sorok bizonyos attribútumainak módosítása.

UPDATE <reláció>

SET <attribútum értékkadások lista>

WHERE <sorokra vonatkozó feltétel>;

Példa: Módosítás

- ◆ Fecó telefonszámát 555-1212-re változtatjuk (Fecó itt egy alkesz):

```
UPDATE Alkeszek
```

```
SET telefon = '555-1212'
```

```
WHERE név = 'Fecó' ;
```

Példa: Több sor módosítása

- ◆ Legfeljebb 4 dollárba kerülhessenek a sörök:

```
UPDATE Felszolgál  
SET ár = 4.00  
WHERE ár > 4.00;
```

Adatbázis sémák SQL-ben

- ◆ Data Definition Language (DDL), az SQL nyelv része, ennek segítségével hozhatunk létre adatobjektumokat, deklarálhatunk megszorításokat stb.

Relációk létrehozása

- ◆ A legegyszerűbb forma:

```
CREATE TABLE <név> (  
    <elemek listája>  
);
```

- ◆ Relációk törlése:

```
DROP TABLE <név>;
```

Tábla definíciók elemei

- ◆ A legegyszerűbb elem: az attribútum és annak típusa.
- ◆ A legfontosabb típusok a következők:
 - ◆ INT vagy INTEGER (szinonimák).
 - ◆ REAL vagy FLOAT (szinonimák).
 - ◆ CHAR(n) = rögzített hosszúságú sztring n karakter hosszú.
 - ◆ VARCHAR(n) = változó hosszúságú sztring legfeljebb n karakter hosszú.

Példa : Create Table

```
CREATE TABLE Felszolgál (
    kocsma CHAR(20),
    sör      VARCHAR(20),
    ár       REAL
) ;
```

SQL értékek

- ◆ Az egészek és lebegőpontos típusú konstansokat csak „szimplán le kell írni”. (Pont jelöli a tizedesvesszőt.)
- ◆ A sztringek esetében aposztrófok közé kell tennünk a konstansokat.
 - ◆ Két aposztróf = egyetlen aposztróf,
például: 'Joe''s Bar'.
- ◆ minden érték lehet NULL is.

Dátum és idő

- ◆ A DATE és TIME külön típusok az SQL-ben.
- ◆ A dátum típus formátuma:
DATE 'yyyy-mm-dd'
 - ◆ Példa: DATE '2007-09-30' (2007. szeptember 30.)

Idő típus

- ◆ Az TIME típus formátuma:
TIME 'hh:mm:ss'

Opcionálisan tizedespont is következhet a másodpercek után.

- ◆ Példa: TIME '15:30:02.5' = fél négy
múlt két és fél másodperccel.

Kulcsok megadása (deklarálása)

- ◆ Egy attribútumot vagy attribútum listát kulcsként deklarálhatunk (PRIMARY KEY vagy UNIQUE).
- ◆ Mindkét formája a megszorításnak azt követeli meg, hogy relációnak ne legyen két olyan sora, melyek megegyeznek a kulcs attribútumokon.
- ◆ A különbségekről később lesz szó.

Egy attribútumos kulcs deklarálása

- ◆ PRIMARY KEY vagy UNIQUE kulcsszót írhatjuk közvetlenül az attribútum mögé.
- ◆ Példa:

```
CREATE TABLE Sörök (
    név      CHAR(20) UNIQUE,
    gyártó  CHAR(20)
) ;
```

Több attribútumú kulcsok megadása

- ◆ A kulcs deklaráció a CREATE TABLE utasítás egy eleme is lehet az attribútum-típus elemek után.
- ◆ Több attribútumú kulcsokat csak ebben a formában deklarálhatunk.
 - ◆ Ugyanakkor az egyetlen attribútumból álló kulcsokat is megadhatjuk ily módon.

Példa: Több attribútumú kulcs

```
CREATE TABLE Felszolgál (
    kocsma      CHAR(20),
    sör         VARCHAR(20),
    ár          REAL,
    PRIMARY KEY (kocsma, sör)
) ;
```

PRIMARY KEY vs. UNIQUE

1. Egy relációhoz egyetlen PRIMARY KEY tartozhat és több UNIQUE megszorítás.
2. A PRIMARY KEY egyetlen attribútuma sem kaphat NULL értéket. A UNIQUE megszorításnál szerepelhetnek NULL értékek egy soron belül akár több is.

Megszorítások

Idegen kulcsok
Lokális és globális megszorítások
Triggerek

Megszorítások és triggerek

- ◆ A *megszorítás* adatelemek közötti kapcsolat, amelyet az AB rendszernek fent kell tartania.
 - ◆ Példa: kulcs megszorítások.
- ◆ *Triggerek* olyankor hajtódnak végre, amikor valamilyen megadott esemény történik, mint pl. sorok beszúrása egy táblába.

Megszorítások típusai

- ◆ **Kulcsok.**
- ◆ **Idegen kulcsok, vagy hivatkozási épség megszorítás.**
- ◆ **Érték-alapú megszorítás.**
 - ◆ Egy adott attribútum lehetséges értékeiről mond valamit.
- ◆ **Sor-alapú megszorítás.**
 - ◆ Mezők közötti kapcsolatok leírása.
- ◆ **Globális megszorítás:** bármilyen SQL kifejezés.

Emlékeztető: egy attribútumos kulcsok

- ◆ PRIMARY KEY vagy UNIQUE.

- ◆ Példa:

```
CREATE TABLE Sörök (
    név      CHAR(20) UNIQUE,
    gyártó  CHAR(20)
) ;
```

Emlékeztető: kulcsok több attribútummal

```
CREATE TABLE Felszolgál (
    kocsma      CHAR(20),
    sört        VARCHAR(20),
    ár          REAL,
    PRIMARY KEY (kocsma, sört)
) ;
```

Idegen kulcsok

- ◆ Egy reláció attribútumainak értékei egy másik reláció értékei között is meg kell, hogy jelenjenek együttesen.
- ◆ Példa: a **Felszolgál(kocsma, söör, ár)** táblánál azt várunk, hogy az itteni söörök szerepelnek a **Sörök** tábla **név** oszlopában is.

Idegen kulcsok megadása

- ◆ A REFERENCES kulcssqlt kell használni:
 1. egy attribútum után (egy-attribútumos kulcs)
 2. A séma elemeként:
FOREIGN KEY (<attribútumok listája>)
REFERENCES <reláció> (<attribútumok>)
- ◆ A hivatkozott attribútum(ok)nak kulcsnak kell lennie / lenniük (PRIMARY KEY vagy UNIQUE).

Példa: egy attribútum

```
CREATE TABLE Sörök (
    név      CHAR(20) PRIMARY KEY,
    gyártó  CHAR(20) );
CREATE TABLE Felszolgál (
    kocsma  CHAR(20),
    sör     CHAR(20) REFERENCES Sörök(név),
    ár      REAL );
```

Példa: a séma elemeként

```
CREATE TABLE Sörök (
    név      CHAR(20) PRIMARY KEY,
    gyártó  CHAR(20) );
CREATE TABLE Felszolgál (
    kocsma  CHAR(20),
    sör     CHAR(20),
    ár      REAL,
    FOREIGN KEY(sör) REFERENCES
    Sörök(név));
```

Idegen kulcs megszorítások megőrzése

- ◆ Egy idegen kulcs megszorítás R relációról S relációra kétféleképpen sérülhet:
 1. Egy R -be történő beszúrásnál S -ben nem szereplő értéket adunk meg.
 2. Egy S -beli törlés „lógó” sorokat eredményez R -ben.

Hogyan védekezzünk? --- (1)

- ◆ Példa: R = Felszolgál, S = Sörök.
- ◆ Nem engedjük, hogy Felszolgál táblába a Sörök táblában nem szereplő sört szúrjanak be.
- ◆ A Sörök táblából való törlés, ami a Felszolgál tábla sorait is érintheti (mert sérül az idegen kulcs megszorítás) 3-féle módon kezelhető.

Hogyan védekezzünk? --- (2)

1. *Default*: a rendszer nem hajtja végre a törlést.
2. *Továbbgyűrűzés*: a Felszolgál tábla értékeit igazítjuk a változáshoz.
 - ◆ **Sör törlése**: töröljük a Felszolgál tábla megfelelő sorait.
 - ◆ **Sör módosítása**: a Felszolgál táblában is változik az érték.
3. *Set NULL*: a sör értékét állítsuk NULL-ra az érintett sorokban.

Példa: továbbgyűrűzés

- ◆ Töröljük a Bud sort a **Sörök** táblából:
 - ◆ az összes sort töröljük a **Felszolgál** táblából, ahol sör oszlop értéke 'Bud'.
- ◆ A 'Bud' nevet 'Budweiser'-re változtatjuk:
 - ◆ a **Felszolgál** tábla soraiban is végrehajtjuk ugyanezt a változtatást.

Példa: Set NULL

- ◆ A Bud sort töröljük a **Sörök** táblából:
 - ◆ a **Felszolgál** tábla **sör** = 'Bud' soraiban a Budot cseréljük NULL-ra.
- ◆ 'Bud'-ról 'Budweiser'-re módosítunk:
 - ◆ ugyanazt kell tennünk, mint törléskor.

A stratégia kiválasztása

- ◆ Ha egy idegen kulcsot deklarálunk megadhatjuk a SET NULL és a CASCADE stratégiát is beszúrásra és törlésre is egyaránt.
- ◆ Az idegen kulcs deklarálása után ezt kell írnunk:

ON [UPDATE, DELETE][SET NULL, CASCADE]

- ◆ Ha ezt nem adjuk meg, a default stratégia működik.

Példa: stratégia beállítása

```
CREATE TABLE Felszolgál (
    kocsma      CHAR(20) ,
    sör         CHAR(20) ,
    ár          REAL,
    FOREIGN KEY(sör)
        REFERENCES Sörök(név)
        ON DELETE SET NULL
        ON UPDATE CASCADE
) ;
```

Attribútum alapú ellenőrzések

- ◆ Egy adott oszlop értékeire vonatkozóan adhatunk meg megszorításokat.
- ◆ Adjuk hozzá a CHECK(<condition>) utasítást az attribútum deklarációjához.
- ◆ A feltételben csak az adott attribútum neve szerepelhet, más attribútumok (más relációk attribútumai is) csak alkérdésben szerepelhetnek.

Példa: attribútum alapú ellenőrzés

```
CREATE TABLE Felszolgál (
    kocsma CHAR(20),
    sör      CHAR(20) CHECK ( sör IN
        (SELECT name FROM Sörök)) ,
    ár       REAL CHECK ( ár <= 5.00 )
) ;
```

Mikor ellenőriz?

- ◆ Attribútum-alapú ellenőrzést csak beszúrásnál és módosításnál hajt végre a rendszer.
 - ◆ Példa: CHECK (ár <= 5.00) a beszúrt vagy módosított sor értéke nagyobb 5, a rendszer nem hajtja végre az utasítást.
 - ◆ Példa: CHECK (sör IN (SELECT név FROM Sörök)), ha a Sörök táblából törlünk, ezt a feltételt nem ellenőrzi a rendszer.

Oszlop-alapú megszorítások

- ◆ A CHECK (<feltétel>) megszorítás a séma elemeként is megadható.
- ◆ A feltételben tetszőleges oszlop és reláció szerepelhet.
 - ◆ De más relációk attribútumai csak alkérdésben jelenhetnek meg.
- ◆ Csak beszúrásnál és módosításnál ellenőrzi a rendszer.

Példa: oszlop-alapú megszorítások

- ◆ Csak Joe bárjában lehetnek drágábbak a söröök 5 dollárnál:

```
CREATE TABLE Felszolgál (
    kocsma      CHAR(20) ,
    sör         CHAR(20) ,
    ár          REAL,
    CHECK (kocsma = 'Joe bárja' OR
           ár <= 5.00)
) ;
```

Globális megszorítás

- ◆ Ezek az adatbázissémához tartoznak a relációsémákhoz és nézetekhez hasonlóan.
- ◆ `CREATE ASSERTION <név>`
`CHECK (<feltétel>);`
- ◆ A feltétel tetszőleges táblára és oszlopra hivatkozhat az adatbázissémából.

Példa: globális megszorítás

```
CREATE ASSERTION CsakOlcsó CHECK (  
    NOT EXISTS (
```

```
        SELECT kocsma  
        FROM Felszolgál  
        GROUP BY kocsma  
        HAVING 5.00 < AVG(ár)  
));
```



Kocsmák, ahol
a söörök
átlagosan
drágábbak 5
dollárnál.

Példa: globális megszorítás

- ◆ Az Alkesz(név, cím, telefon) és Kocsma(név, cím, engedélySzám), táblákban nem lehet több kocsma, mint alkesz.

```
CREATE ASSERTION TöbbAlkesz CHECK (
  (SELECT COUNT (*) FROM Kocsma) <=
  (SELECT COUNT (*) FROM Alkesz)
) ;
```

Globális megszorítások ellenőrzése

- ◆ Alapvetően az adatbázis bármely módosítása előtt ellenőrizni kell.
- ◆ Egy okos rendszer felismeri, hogy mely változtatások, mely megszorításokat érinthetnek.
 - ◆ Példa: a **Sörök** tábla változásai nincsenek hatással az iménti TöbbAlkesz megszorításra.

Miért hasznosak a triggerek?

- ◆ A globális megszorításokkal sok minden le tudunk írni, az ellenőrzésük azonban gondot jelenthet.
- ◆ Az attribútum- és oszlop-alapú megszorítások ellenőrzése egyszerűbb (tudjuk mikor történik), ám ezekkel nem tudunk minden kifejezni.
- ◆ A triggerek esetén a felhasználó mondja meg, hogy egy megszorítás mikor kerüljön ellenőrzésre.

Esemény-Feltétel-Akció szabályok

- ◆ A triggereket esetenként *ECA szabályoknak* (*event-condition-action*) is nevezik.
- ◆ *Esemény*: általában valamelyen módosítás a adatbázisban, például egy sor beszúrása.
- ◆ *Feltétel* : bármilyen SQL igaz-hamis-(ismeretlen) feltétel.
- ◆ *Akció* : bármilyen SQL utasítás.

Példa: trigger

- ◆ Ahelyett, hogy visszautasítanánk a **Felszolgál(kocsma, sör, ár)** táblába történő beszúrást az ismeretlen sörökkel esetén, a **Sörök(név, gyártó)** táblába is beszúrjuk a megfelelő sort a gyártónak NULL értéket adva.

Példa: trigger definíció

```
CREATE TRIGGER SörTrig  
  BEFORE INSERT ON Felszolgál  
    REFERENCING NEW ROW AS ÚjSor  
    FOR EACH ROW  
      WHEN (ÚjSor.sör NOT IN  
            (SELECT név FROM Sörök))  
        INSERT INTO Sörök(név)  
          VALUES(ÚjSor.sör);
```

Az esemény

A feltétel

Az akció

Relációs adatbázis tervezés

Funkcionális függősségek

Felbontások

Normálformák

Funkcionális függőségek

- $X \rightarrow Y$ az R relációra vonatkozó megszorítás, miszerint ha két sor megegyezik X összes attribútumán, Y attribútumain is meg kell, hogy egyezzenek.
- **Jelölés:** X, Y, Z, \dots attribútum halmazokat; A, B, C, \dots attribútumokat jelöl.
- **Jelölés:** $\{A, B, C\}$ attribútum halmaz helyett ABC -t írunk.

Jobboldalak szétvágása (ff)

- $X > A_1 A_2 \dots A_n$ akkor és csak akkor teljesül R relációra, ha $X > A_1$, $X > A_2, \dots$, $X > A_n$ is teljesül R -en.
- Példa: $A > BC$ ekvivalens $A > B$ és $A > C$ függőségek kettősével.
- Baloldalak szétvágására nincs általános szabály.
- Általában FF-k jobboldalán egyetlen attribútum szerepel majd.

Példa: FF

Vezetők(név, cím, kedveltSörök, gyártó, kedvencSör)

- FF-k, amelyek vszleg teljesülnek:
 1. név -> cím kedvencSör
 - Ez az FF ugyanaz, mint név -> cím és név -> kedvencSör.
 2. kedveltSörök -> gyártó.

Példa: egy lehetséges előfordulás

név	cím	kedveltSörök	gyártó	kedvencSör
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

Mert név -> cím

Mert név -> kedvencSör

Mert kedveltSörök -> gyártó

Relációk kulcsai

- K *szuperkulcs* R relációra, ha K funkcionálisan meghatározza R attribútumait.
- K *kulcs* R -en, ha K szuperkulcs, de egyetlen valódi részhalmaza sem szuperkulcs.

Példa: szuperkulcs

Főnökök(név, cím, kedveltSörök, gyártó, kedvencSör)

- {név, kedveltSörök} szuperkulcs, hiszen a két attribútum meghatározza funkcionálisan a maradék attribútumokat.
 - név -> cím kedvencSör
 - kedveltSörök -> gyártó

Példa: kulcs

- {név, kedveltSörök} **kulcs**, hiszen sem {név}, sem {kedveltSörök} nem szuperkulcs.
 - név -> gyártó; kedveltSörök -> cím nem teljesülnek.
- Az előbbin kívül nincs több kulcs, de számos szuperkulcs megadható még.
 - minden olyan halmaz, amit tartalmazza {név, kedveltSörök}-t.

Kis kombinatorika

- Feladat: R relációnak legyenek A_1, \dots, A_n az attribútumai. Adjuk meg n függvényeként, hogy R-nek hány szuperkulcsa van, ha
 - (a) csak A_1 kulcs,
 - (b) A_1 és A_2 kulcsok,
 - (c) $\{A_1, A_2\}, \{A_3, A_4\}$ kulcsok,
 - (d) $\{A_1, A_2\}, \{A_1, A_3\}$ kulcsok.

Hogyan kaphatjuk meg a kulcsokat?

1. Szimplán megadunk egy K kulcsot.
 - Az FF-k $K \rightarrow A$ alakúak, ahol A „végigmegy” az összes attribútumon
2. Vagy: megadjuk az FF-keket, és ezekből következtetjük ki a kulcsokat.

Még egy természetesen adódó FF

- Példa: az „ugyanabban az időben nem lehet két előadás ugyanabban a teremben” lefordítva:
idő terem -> előadás.

FF-k kikövetkeztetése

- Legyenek $X_1 \rightarrow A_1, X_2 \rightarrow A_2, \dots, X_n \rightarrow A_n$ adott FF-ek, szeretnénk tudni, hogy $Y \rightarrow B$ teljesül-e olyan relációkra, amire az előző FF-ek teljesülnek.
 - Példa: $A \rightarrow B$ és $B \rightarrow C$ teljesülése esetén $A \rightarrow C$ biztosan teljesül.
 - Ez az adatbázis sémájának megtervezésekor lesz majd fontos.

Armstrong-axiómák I.

- (A1) **Reflexitivitás**: ha $Y \subseteq X \subseteq R$, akkor $X \rightarrow Y$. Az ilyen függőségeket **triviális** függőségeknek nevezzük.
- (A2) **Bővítés**: ha $X \rightarrow Y$ teljesül, akkor tetszőleges $Z \subseteq R$ -ra $XZ \rightarrow YZ$ teljesül.
- (A3) **Tranzitivitás**: ha $X \rightarrow Y$ és $Y \rightarrow Z$, akkor $X \rightarrow Z$.

- Példák a *személy* (*sz_ig_szám*, *TAJ*, *név*, *anyja_neve*, *születés*, *kor*, *fizetés*) tábla esetén:
 - (A1) (*név*, *születés*) \rightarrow *név*
 - (A2) *születés* \rightarrow *kor*, akkor (*születés*, *név*) \rightarrow (*kor*, *név*)
 - (A3) *TAJ* \rightarrow *születés*, *születés* \rightarrow *kor*, akkor *TAJ* \rightarrow *kor*.

Példa levezetésre

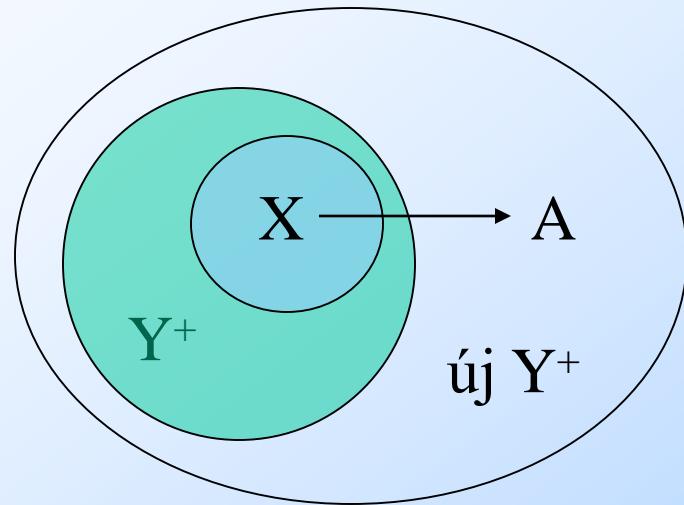
- Legyen $R = ABCD$ és $F = \{ A \rightarrow C, B \rightarrow D \}$:
 1. $A \rightarrow C$ adott.
 2. $AB \rightarrow ABC$ (A2) alapján.
 3. $B \rightarrow D$ adott.
 4. $ABC \rightarrow ABCD$ (A2) alapján.
 5. $AB \rightarrow ABCD$ (A3) alapján 2-ből és 4-ből.
- Példa: bizonyítsuk be levezetéssel, hogy $\{ X \rightarrow Y, XY \rightarrow Z \}$ -ből következik $\{ X \rightarrow Z \}$.

Újabb feladat

- **Feladat:** mutassuk meg, hogy az alábbiak nem érvényes szabályok funkcionális függőségekre:
 - ha $A \rightarrow B$, akkor $B \rightarrow A$,
 - ha $AB \rightarrow C$ és $A \rightarrow C$, akkor $B \rightarrow C$,
 - ha $AB \rightarrow C$, akkor $A \rightarrow C$ vagy $B \rightarrow C$.

Lezárás

- Egy egyszerűbb út, ha kiszámítjuk Y *lezártját*, jelölésben Y^+ .
- **Kiindulás:** $Y^+ = Y$.
- **Indukció:** Olyan FF-ket keresünk, melyeknek a baloldala már benne van Y^+ -ban. Ha $X \rightarrow A$ ilyen, A -t hozzáadjuk Y^+ -hoz.



Mi is az a lezárt?

- Adott R séma és F FF halmaza mellett,
 X^+ az összes olyan A attribútum
halmaza, amire $X \rightarrow A$ következik F -ből.

A lezárást kiszámító algoritmus „helyes” I.

- Az algoritmus „tényleg” X^+ -t számítja ki.
Vagyis:
 1. Ha az A attribútum valamely j -re belekerül X^j -be, akkor A valóban eleme X^+ -nak.
 2. Másfelől, ha $A \in X^+$, akkor létezik olyan j , amire A belekerül X^j -be.
- **Megjegyzés:** az első állítás könnyen bizonyítható indukcióval.

A lezárást kiszámító algoritmus „helyes” II.

- (2) Tegyük fel, hogy $A \notin X^+$, tehát nem létezik olyan j , amire A belekerül X^j -be.

	X^+ elemei	más attribútumok
t	111 ... 111	000 ... 000
s	111 ... 111	111 ... 111

- Ekkor az előbbi előfordulásra minden F -beli függőség teljesül. (Miért?)
- $X \rightarrow A$ viszont nem teljesül.

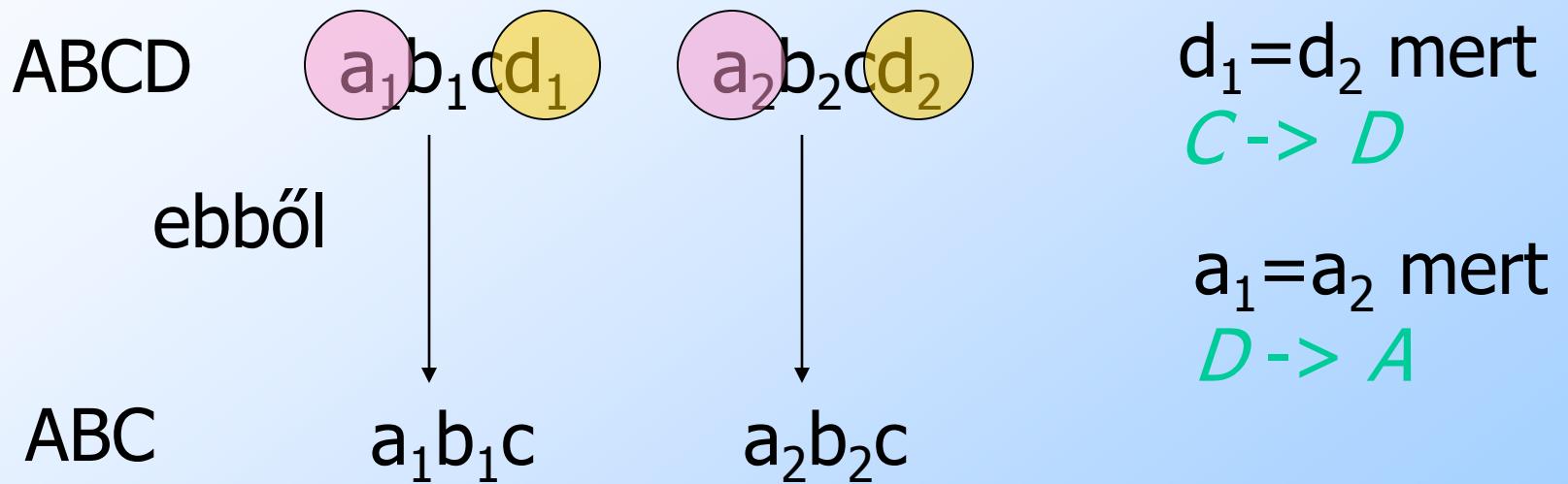
Feladatok

1. $R = SEBADNM$, $F = \{ S \rightarrow EBAD, D \rightarrow NM \}$, mi az S és D attribútumok F-re vonatkozó lezártja?
2. $R = HTUSDK$, $F = \{ H \rightarrow T, U \rightarrow HS, HD \rightarrow KU \}$, adjuk meg az összes F-szerinti kulcsát R-nek és lássuk be, hogy csak a megadottak kulcsok.
3. Bizonyítsuk be, hogy $(X^+)^+ = X^+$.

Az összes következmény FF megtalálása

- Motiváció: „normalizálás”, melynek során egy reláció sémát több sémára bonthatunk szét.
- Példa: $ABCD$ FF-k: $AB \rightarrow C$, $C \rightarrow D$ és $D \rightarrow A$.
 - Bontsuk fel ABC és AD -re. Milyen FF-k teljesülnek ABC -n?
 - Nem csak $AB \rightarrow C$, de $C \rightarrow A$ is!

Miért?



Emiatt, ha két vetített sor C-n
megegyezik, akkor A-n is, azaz:
 $C \rightarrow A$.

Alapötlet

1. Induljunk ki a megadott FF-ekből és keressük meg az összes *nem triviális* FF-t, ami a megadott FF-ekből következik.
 - Nem triviális = a jobboldalt nem tartalmazza a bal.
2. Csak azokkal az FF-kel foglalkozzunk, amelyekben a projektált séma attribútumai szerepelnek.

Exponenciális algoritmus

1. minden X attribútumhalmazra számítsuk ki X^+ -t.
2. Adjuk hozzá a függősségeinkhez $X \rightarrow A$ -t minden A -ra $X^+ - X$ -ből.
3. Dobjuk ki $XY \rightarrow A$ -t, ha $X \rightarrow A$ is teljesül.
 - Mert $XY \rightarrow A$ $X \rightarrow A$ -ból minden esetben következik.
4. Végül csak azokat az FF-ket használjuk, amelyekben csak a projektált attribútumok szerepelnek.

Néhány trükk

- Az üreshalmaznak és az összes attribútum halmazának nem kell kiszámolni a lezártját.
- Ha $X^+ =$ az összes attribútum, akkor egyetlen X^- t tartalmazó halmaznak sem kell kiszámítani a lezártját.

Példa: FF-k projekciója

□ $ABC, A \rightarrow B$ és $B \rightarrow C$ FF-kel.

Projektálunk AC -re.

□ $A^+ = ABC$; ebből $A \rightarrow B, A \rightarrow C$.

- Nem kell kiszámítani AB^+ és AC^+ lezárásokat.

□ $B^+ = BC$; ebből $B \rightarrow C$.

□ $C^+ = C$; semmit nem ad.

□ $BC^+ = BC$; semmit nem ad.

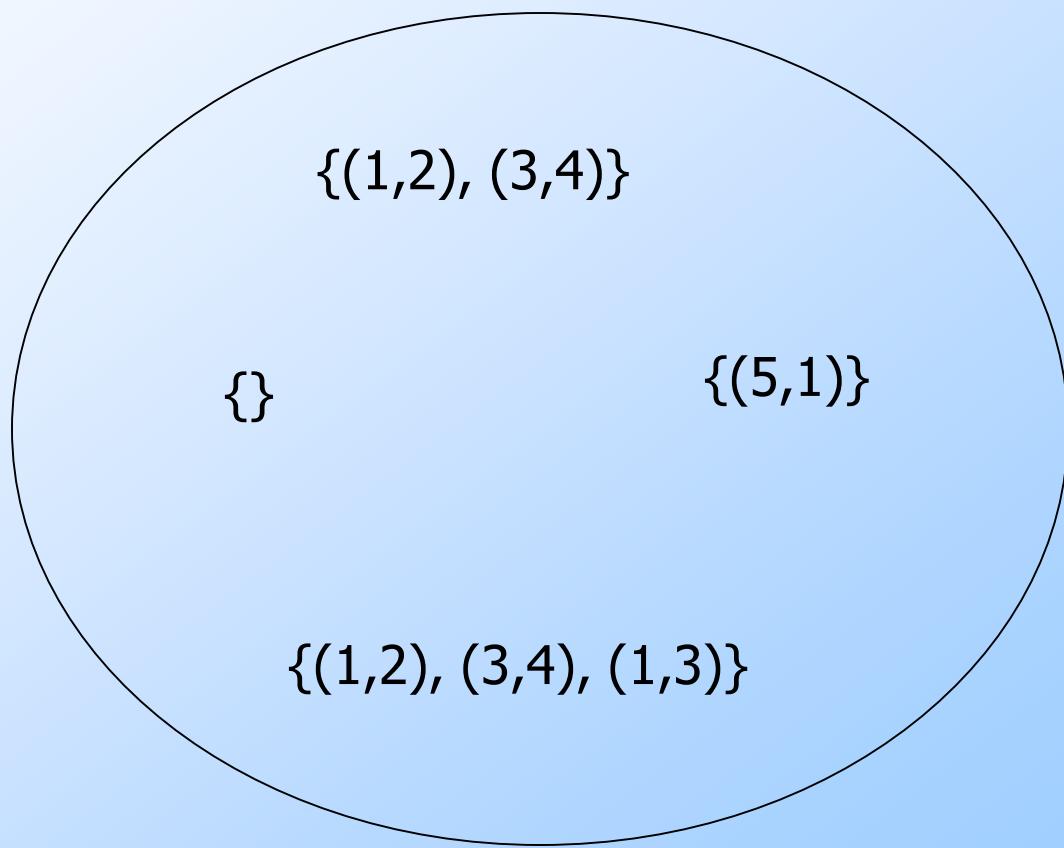
Példa -- folytatása

- A kapott FF-ek: $A \rightarrow B$, $A \rightarrow C$ és $B \rightarrow C$.
- AC -re projekció: $A \rightarrow C$.

Az FF-k geometriai reprezentációja

- Vagyük egy reláció összes lehetséges *előfordulásainak* halmazát.
- Azaz az összes olyan sorhalmazt, mely sorok komponensei a „megfelelők”.
- minden ilyen halmaz egy pont a térben.

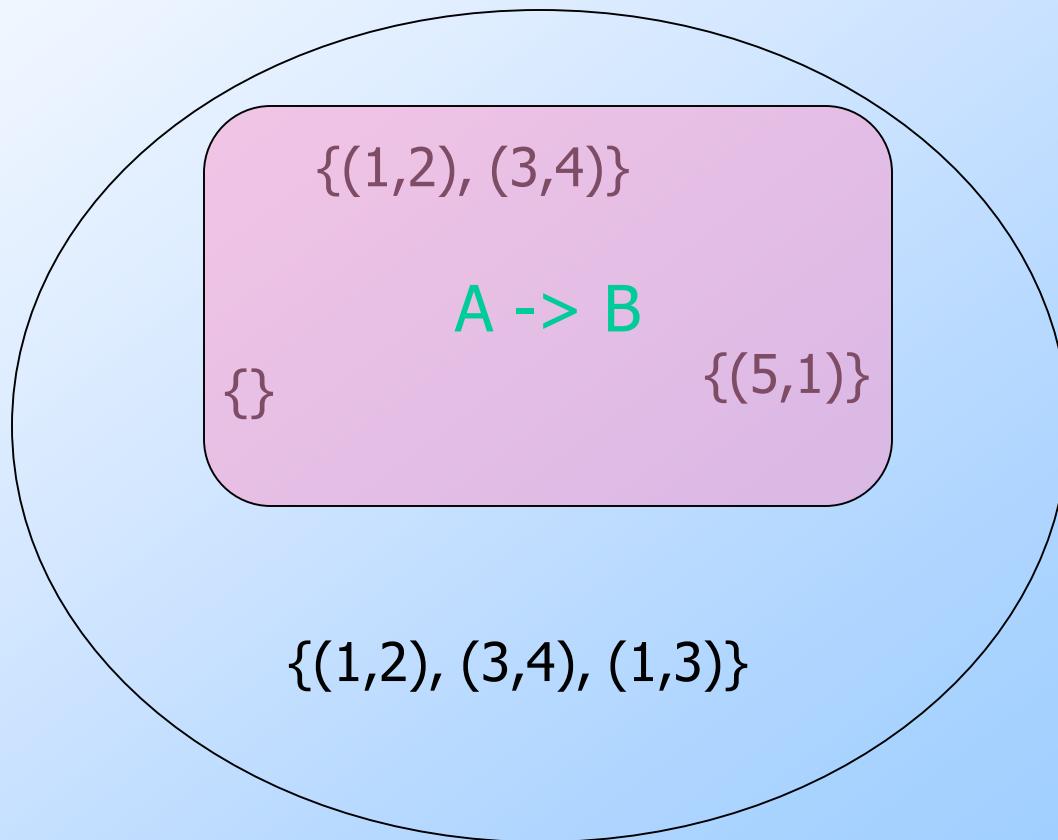
Példa: R(A,B)



Egy FF az előfordulásoknak egy részhalmaza

- minden $X \rightarrow A$ FF megadható azon előfordulások részhalmazaként, mely teljesíti FF-t.
- Így minden FF egy régióval jellemzhető a térben
- A triviális FF-k azok, melyeknél ez a régió a teljes tér.
 - Példa: $A \rightarrow A$.

Példa: $A \rightarrow B$ $R(A,B)$ fölött

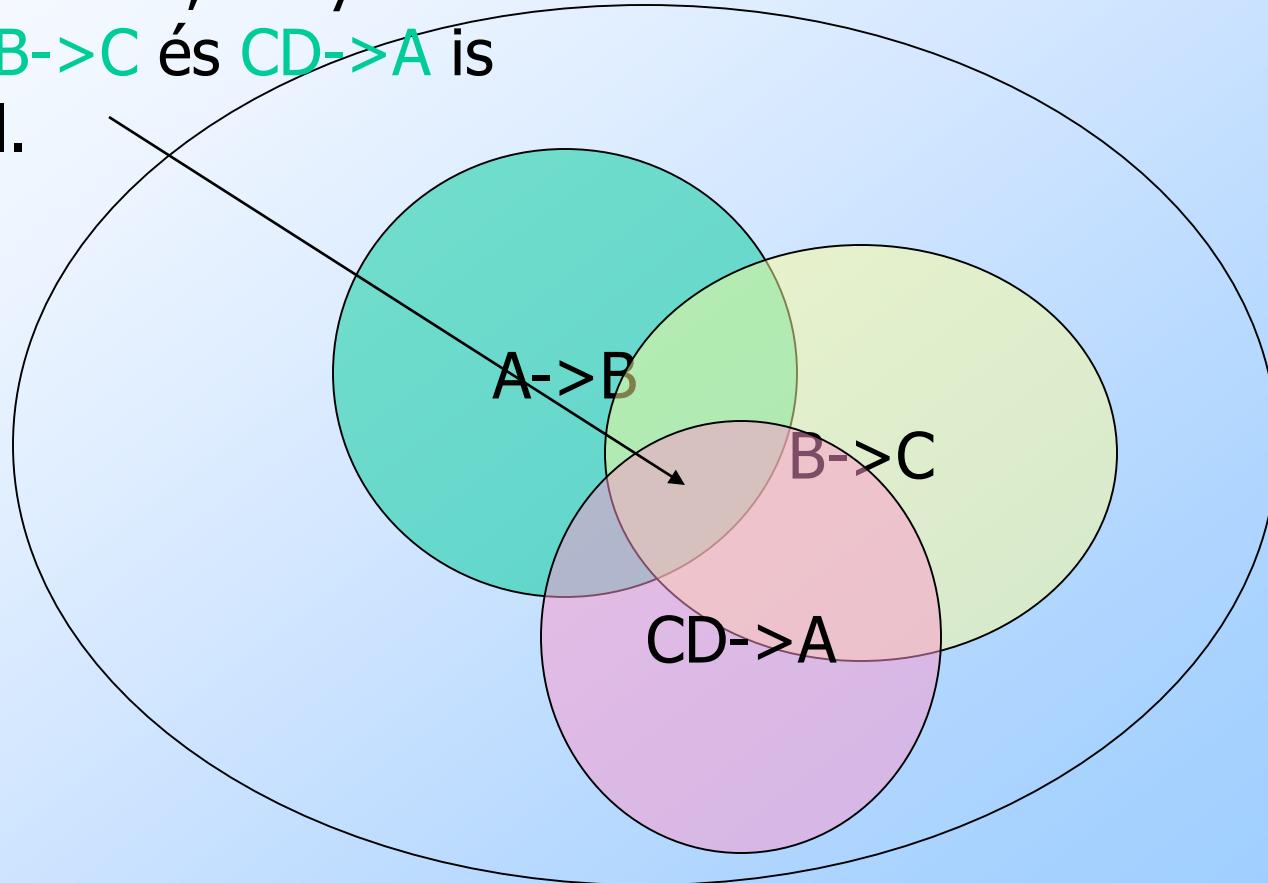


FF-k halmazának reprezentálása

- Ha egy-egy FF előfordulásoknak egy halmzával reprezentálható, akkor az FF-ek halmaza az előbbi halmazok metszetével lesz egyenlő.
- Azaz a metszet = azon előfordulások, amelyekre minden FF teljesül.

Példa

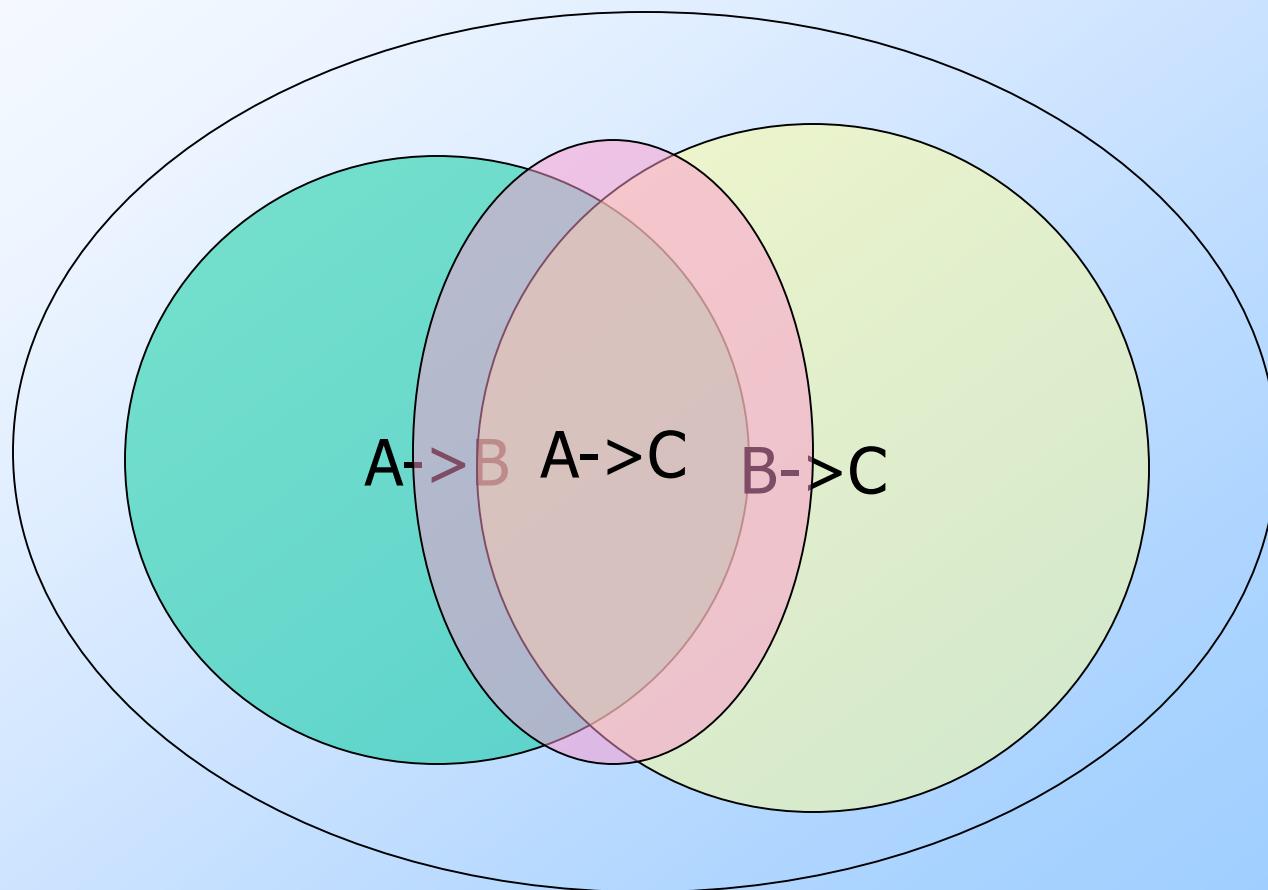
Előfordulások, melyekre
 $A \rightarrow B$, $B \rightarrow C$ és $CD \rightarrow A$ is
teljesül.



FF-k következtetése

- Ha FF $Y \rightarrow B$ következik $X_1 \rightarrow A_1, \dots, X_n \rightarrow A_n$ FF-ekből, akkor az $Y \rightarrow B$ régiójának tartalmaznia kell az $X_i \rightarrow A_i$ FF-ekhez tartozó régiók metszetét.
- Azaz: minden előfordulás, ami teljesíti $X_i \rightarrow A_i$ -t, $Y \rightarrow B$ -t is teljesíti.
- Ugyanakkor ha egy előfordulásra teljesül $Y \rightarrow B$, $X_i \rightarrow A_i$ nem feltétlen teljesül.

Példa



Relációs sémák tervezése

- Cél: az anomáliák és a redundancia megszüntetése.
 - *Módosítási anomália* : egy adat egy előfordulását megváltoztatjuk, más előfordulásait azonban nem.
 - *Törlési anomália* : törléskor olyan adatot is elveszítünk, amit nem szeretnénk.

Példa: rosszul tervezett séma

Főnökök(név, cím, kedveltSörök, gyártó, kedvencSör)

név	cím	kedveltSörök	gyártó	kedvencSör
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	???	WickedAle	Pete's	???
Spock	Enterprise	Bud	???	Bud

Redundáns adat, a ??? helyén a név -> cím kedvencSör és kedveltSörök -> gyártó FF-ek felhasználásával tudjuk, mi szerepel.

A rosszul tervezettség anomáliákat is eredményez

név	cím	kedveltSörök	gyártó	kedvencSör
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

- **Módosítási anomália:** ha Janeway-t *Karcsira* módoítjuk, megtesszük-e ezt minden sornál?
- **Törlési anomális:** Ha senki sem szereti a Bud sört, azt sem tudjuk, ki gyártotta.

Boyce-Codd normálforma

- R reláció ***BCNF*** normálformában van, ha minden $X \rightarrow Y$ nemtriviális FF-re R -ben X szuperkulcs.
 - ***Nemtriviális:*** Y nem része X -nek.
 - ***Szuperkulcs:*** tartalmaz kulcsot (ő maga is lehet kulcs).

Példa

Főnökök(név, cím, kedveltSörök, gyártó, kedvencSör)

FF-ek: név->cím kedvencSör, kedveltSörök->gyártó

- Itt egy kulcs van: {név, kedveltSörök}.
- A baloldalak egyik FF esetén sem szuperkulcsok.
- Emiatt az *Főnökök* reláció nincs BCNF normálformában.

Még egy példa

Sörök(név, gyártó, gyártóCím)

FF-ek: név->gyártó, gyártó->gyártóCím

- Az egyetlen kulcs {név} .
- név->gyártó nem sérti a BCNF feltételét, de a gyártó->gyártóCím függőség igen.

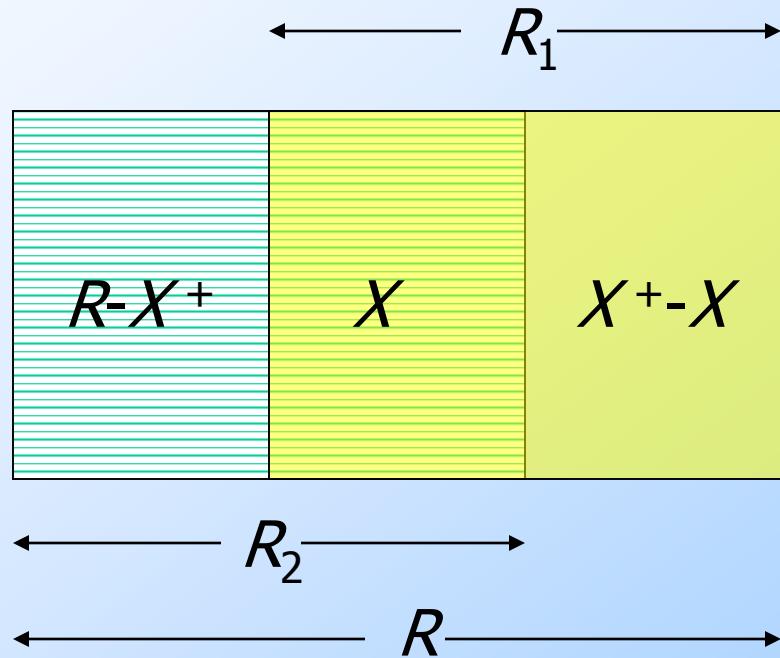
BCNF-re való felbontás

- Adott R reláció és F funkcionális függőségek.
- Van-e olyan $X \rightarrow Y$ FF, ami sérti a BCNF-t?
 - Ha van olyan következmény FF F -ben, ami sérti a BCNF-t, akkor egy F -beli FF is sérti.
- Kiszámítjuk X^+ -t:
 - Ha itt nem szerepel az összes attribútum, X nem szuperkulcs.

R dekomponálása $X \rightarrow Y$ felhasználásával

- R -t helyettesítsük az alábbiakkal:
 1. $R_1 = X^+$.
 2. $R_2 = R - (X^+ - X)$.
- *Projektáljuk* a meglévő F -beli FF-eket a két új relációsémára.

Dekomponálási kép



Példa: BCNF dekompozíció

Alkeszek(név, cím, kedveltSörök, gyártó,
kedvencSör)

$F = \text{név} \rightarrow \text{cím}, \text{név} \rightarrow \text{kedvencSör},$
 $\text{kedveltSörök} \rightarrow \text{gyártó}$

- Végyük $\text{név} \rightarrow \text{cím}$ FF-t:
- $\{\text{név}\}^+ = \{\text{név}, \text{cím}, \text{kedvencSör}\}.$
- A dekomponált relációsémák:
 1. Alkeszek1(név, cím, kedvencSör)
 2. Alkeszek2(név, kedveltSörök, gyártó)

Példa -- folytatás

- Meg kell néznünk, hogy az Alkeszek1 és Alkeszek2 táblák BCNF-ben vannak-e.
- Az FF-ek projektálása könnyű.
- A **Alkeszek1(név, cím, kedvencSör)**, az FF-ek **név->cím** és **név->kedvencSör**.
 - Tehát az egyetlen kulcs: {név}, azaz az Alkeszek1 BCNF-ben van.

Példa -- folytatás

- Az **Alkeszek2**(név, kedveltSörök, gyártó) esetén az egyetlen FF kedveltSörök->gyártó, az egyetlen kulcs: {név, kedveltSörök}.
 - Sérül a BCNF.
- $\text{kedveltSörök}^+ = \{\text{kedveltSörök}, \text{gyártó}\}$, az *Alkeszek2* felbontása:
 1. **Alkeszek3**(kedveltSörök, gyártó)
 2. **Alkeszek4**(név, kedveltSörök)

Példa -- befejezés

- Az *A/keszek* dekompozíciója tehát:
 1. *Alkeszek1*(név, cím, kedvencSör)
 2. *Alkeszek 3*(kedveltSörök, gyártó)
 3. *Alkeszek 4*(név, kedveltSörök)
- Az *A/keszek1* az alkeszékéről, az *A/keszek3* a sörökről, az *A/keszek4* az alkeszek és kedvelt söreikről tartalmaz információt.

Miért működik a BCNF?

- (R, F) esetén ha R_1, \dots, R_k egy veszteségmentes felbontás, S_1, S_2 pedig R_1 veszteségmentes felbontása, akkor $S_1, S_2, R_2, \dots, R_k$ is veszteségmentes felbontás.
- Könnyen ellenőrizhető, hogy a fenti R_1, R_2 veszteségmentes.
Feladat: bizonyítsuk be, hogy ha az $R(A, B, C)$ reláció esetén $B \rightarrow C$ teljesül, akkor az $R_1(A, B), R_2(B, C)$ felbontás minden veszteségmentes.
- minden két attribútumú séma BCNF normálformában van.
- Az algoritmus tehát valóban veszteségmentes felbontást ad, ám sajnos **exponenciális** lépésszámú is lehet a függőségek vetítése miatt.

Feladat

1. Adott $R = ABCD$ reláció és $F = \{ AB \rightarrow C, A \rightarrow D, BD \rightarrow C \}$ függősségi halmaz mellett adjuk meg R egy BCNF dekompozícióját.
2. Legyen most $R = BOISQD$, $F = \{ S \rightarrow D, I \rightarrow B, IS \rightarrow Q, B \rightarrow O \}$. Ugyanez a feladat.

Veszteségmentes szétvágás I.

- Ha $r = \Pi_{R_1}(r) | X | \dots | X | \Pi_{R_k}(r)$ teljesül, akkor az előbbi összekapcsolásra azt mondjuk, hogy **veszteségmentes**. Itt r egy R sémájú relációt jelöl.
- $\Pi_{R_i}(r)$ jelentése: r sorai az R_i attribútumaira projektálva.
- Megjegyzés: könnyen látható, hogy $r \subseteq \Pi_{R_1}(r) | X | \dots | X | \Pi_{R_k}(r)$ mindenkorban teljesül. (Miért?)

R

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

R_1

A	B
a	b
d	e
c	b

R_2

B	C
b	c
e	f

Példa

- A szétvágás után keletkező relációk összekapcsolása nem veszteségmentes:

R	A	B	C
a	b	c	
c	b	e	

R_1	A	B
a	b	
c	b	

R_2	B	C
b	c	
b	e	

Chase-teszt veszteségmentességhoz

I.

- Példa: adott $R(A, B, C, D)$, $F = \{ A \rightarrow B, B \rightarrow C, CD \rightarrow A \}$ és az $R_1(A, D)$, $R_2(A, C)$, $R_3(B, C, D)$ felbontás. Kérdés veszteségmentes-e a felbontás?
- Vegyük $R_1 |X| R_2 |X| R_3$ egy $t = (a, b, c, d)$ sorát. Bizonyítani kell, hogy t R egy sora. A következő tablót készítjük el:

A	B	C	D
a	b_1	c_1	d
a	b_2	c	d_2
a_3	b	c	d

Itt pl. az (a, b_1, c_1, d) sor azt jelzi, hogy R -nek van olyan sora, aminek R_1 -re való levetítése (a, d) , ám ennek a B és C attribútumokhoz tartozó értéke ismeretlen, így egyáltalán nem biztos, hogy a t sorról van szó.

Chase-teszt veszteségmentességhöz

II.

- Az F-beli függőségeket használva egyenlővé tesszük azokat a szimbólumokat, amelyeknek ugyanazoknak kell lennie, hogy valamelyik függőség ne sérüljön.
 - Ha a két egyenlővé teendő szimbólum közül az egyik index nélküli, akkor a másik is ezt az értéket kapja.
 - Két indexes szimbólum esetén a kisebbik indexű értéket kapja meg a másik.
 - A szimbólumok minden előfordulását helyettesíteni kell az új értékkel.
- Az algoritmus véget ér, ha valamelyik sor t-vel lesz egyenlő, vagy több szimbólumot már nem tudunk egyenlővé tenni.

Chase-teszt veszteségmentességhöz III.

A	B	C	D
a	b_1	c_1	d
a	b_2	c	d_2
a_3	b	c	d

$A \rightarrow B$



A	B	C	D
a	b_1	c_1	d
a	b_1	c	d_2
a_3	b	c	d

$B \rightarrow C$



A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a_3	b	c	d

$CD \rightarrow A$



A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a	b	c	d

Chase-teszt veszteségmentességhöz

IV.

- Ha t szerepel a tablóban, akkor valóban R-nek egy sora, s mivel t-t tetszőlegesen választottuk, ezért a felbontás veszteségmentes.
- Ha nem kapjuk meg t-t, akkor viszont a felbontás nem veszteségmentes.
- Példa: $R(A, B, C, D)$, $F = \{ B \rightarrow AD \}$, a felbontás: $R_1(A, B)$, $R_2(B, C)$, $R_3(C, D)$.

A	B	C	D
a	b	c_1	d_1
a_2	b	c	d_2
a_3	b_3	c	d

$B \rightarrow AD$



A	B	C	D
a	b	c_1	d_1
a	b	c	d_1
a_3	b_3	c	d

Itt az eredmény jó ellenpélda, hiszen az összekapcsolásban szerepel $t = (a, b, c, d)$, míg az eredeti relációban nem.

Chase-teszt veszteségmentességhez IV.

- $\{A, B\}, \{(a, b)\}, (a_3, b_3)\}.$
- $\{B, C\}, \{(b, c_1), (b, c), (b_3, c)\},$
- $\{C, D\}, (c_1, d_1), (c, d_1), (c, d)\}$
- $\bowtie, \{(a, b, c_1), (a, b, c), (a_3, b_3, c)\}.$
- $\bowtie, , \{(a, b, c_1, d_1), (a, b, c, d_1), (a, b, c, d), (a_3, b_3, c, d_1), (a_3, b_3, c, d)\}\}$
- 2 extra sor (rekord), és (a, b, c, d) , amit kell is tartalmaznia.

A harmadik normálforma -- motiváció

- Bizonyos FF halmazok esetén a felbontáskor elveszíthetünk függőségeket.
- $AB \rightarrow C$ és $C \rightarrow B$.
 - Példa: $A = \text{f_cím}$, $B = \text{város}$, $C = \text{mozi}$.
- Két kulcs van: $\{A, B\}$ és $\{A, C\}$.
- $\{\text{f_cím}, \text{város}\}$, $\{\text{f_cím}, \text{mozi}\}$
- $C \rightarrow B$ megsérte a BCNF-t, tehát AC , BC -re dekomponálunk. [$\text{mozi} \rightarrow \text{város}$, nem szuperkulcs C]

FF-ek kikényszerítése

- A probléma az, hogy AC és BC sémákkal nem tudjuk kikényszeríteni $AB \rightarrow C$ függőséget.
- Példa $A = f_cím$, $B = \text{város}$, $C = \text{mozi}$, a következő dián.

Egy kikényszeríthetetlen FF

F_cím	mozi
Antz	Guild
Antz	Park

város	mozi
Cambridge	Guild
Cambridge	Park

Kapcsoljuk össze a sorokat (mozi).

F_cím	város	mozi
Antz	Cambridge	Guild
Antz	Cambridge	Park

A szétbontott relációkban egyik FF sem sérül, az eredményben az F_cím város -> mozi nem teljesül.

A probléma megoldása: 3NF

- 3. normálformában (3NF) úgy módosul a BCNF feltétel, hogy az előbbi esetben nem kell dekomponálnunk.
- Egy attribútum *prím*, ha legalább egy kulcsnak eleme.
- $X \rightarrow A$ megsérti 3NF-t akkor és csak akkor, ha X nem szuperkulcs és A nem prím (elsődleges attribútum).
- minden nem triviális függőségre igaz, hogy bal oldala szuperkulcs, vagy jobb oldala csak elsődleges attribútumokat tartalmaz
- 3NF feltétel és a BCNF feltétel közötti különbség a „vagy jobb oldala csak elsődleges attribútumokat tartalmaz”

Példa: 3NF

- A problematikus esetben az $AB \rightarrow C$ és $C \rightarrow B$ FF-ek esetén a kulcsok AB és AC .
- Ezért A , B és C mindegyike prím.
- Habár $C \rightarrow B$ megséríti a BCNF feltételét, 3NF feltételét már nem sérti meg.

Miért hasznos 3NF és BCNF?

- A dekompozícióknak két fontos tulajdonsága lehet:
 1. *Veszteségmentes összekapcsolás* : ha a projektált relációkat összekapcsoljuk az eredetit kapjuk vissza.
 2. *Függőségek megőrzése* : a projektált relációk segítségével is kikényszeríthetőek az előre megadott függőségek.

3NF és BCNF -- folytatás

- Az (1) tulajdonság teljesül a BCNF esetében.
- A 3NF (1) és (2)-t is teljesíti.
- A BCNF esetén (2) sérülhet.
- Az F_cím - város - mozi erre volt egy példa.

Minimális bázis létrehozása

1. Jobboldalak szétvágása.
2. Próbáljuk törölni az FF-eket egymás után. Ha a megmaradó FF-halmaz nem ekvivalens az eredetivel, akkor nem törölhető az épp aktuális FF.
3. Egymás után próbáljuk csökkenteni a baloldalakat, és megnézzük, hogy az eredetivel ekvivalens FF-halmazt kapunk-e.

3NF-re bontás – (2)

- A minimális bázis minden FF-re megad egy sémát a felbontásban.
 - A séma a jobb- és baloldalak uniója lesz ($X \rightarrow A$ FF, XA séma) .
- Ha a minimális bázis FF-jei által meghatározott sémák ($X \rightarrow A$ FF, XA séma) között nincs *szuperkulcs*, akkor hozzáadunk a felbontáshoz egy olyan *sémat*, amely maga egy *kulcs* az R relációra.

Példa: 3NF felbontás

- A reláció: $R = ABCD$.
- FF-ek: $A > B$ és $A > C$.
- Felbontás: AB és AC az FF-ekből és AD-t is hozzá kell venni, mert AB, AC egyike sem kulcs.

Miért működik?

- Megőrzi a függőségeket: minden FF megmarad a minimális bázisból.
- Veszeségmentes összekapcsolás: a CHASE algoritmussal ellenőrizhető (a kulcsból létrehozott séma itt lesz fontos).
- 3NF: a minimális bázis tulajdonságaiból következik.

Minimális bázist kiszámító algoritmus

Jelölje F^+ az F függőségi halmazból következő függőségek halmazát.

1. Kezdetben G az üreshalmaz.
2. minden $X \rightarrow Y \in F$ helyett vegyük az $X \rightarrow A$ függőségeket, ahol $A \in Y - X$.

Megjegyzés: Ekkor minden G -beli függőség $X \rightarrow A$ alakú, ahol A attribútum.

3. minden $X \rightarrow A \in G$ -re, ha $X \rightarrow A \in (G - \{ X \rightarrow A \})^+$, vagyis ha elhagyjuk az $X \rightarrow A$ függőséget G -ből, az még mindig következik a maradékból, akkor $G := G - \{ X \rightarrow A \}$.

Megjegyzés: Végül nem marad több elhagyható függőség.

4. minden $X \rightarrow A \in G$ -re, amíg van olyan $B \in X$ -re, hogy $A \in (X - B)^+$ a G -szerint, vagyis $(X - B) \rightarrow A$ teljesül, akkor $X := X - B$.

Megjegyzés: Ez a lépés után minden baloldal minimális lesz.

Feladat

- Példa: $F = \{ H \rightarrow T, U \rightarrow HS, HD \rightarrow KU \}$. Mi a minimális bázis?
- Feladat: $R = BOISQD$,
 $F = \{ S \rightarrow D, I \rightarrow BS, IS \rightarrow Q, B \rightarrow OQ, SD \rightarrow O \}$. Adjunk meg egy függőségőrző, veszteségmentes 3NF dekompozíciót.

Többértékű függőségek

Negyedik normálforma
Funkcionális és többértékű
függőségek következtetése

A TÉF definíciója

- ◆ A *többértékű függőség* (TÉF): az R reláció fölött $X \rightarrow\!-\!\rightarrow Y$ teljesül: ha bármely két sorra, amelyek megegyeznek az X minden attribútumán, az Y attribútumaihoz tartozó értékek felcserélhetők, azaz a keletkező két új sor R -beli lesz.
- ◆ Más szavakkal: X minden értéke esetén az Y -hoz tartozó értékek függetlenek az R - X - Y értékeitől.

Példa: TÉF

Alkesz(név, cím, tel, kedveltSörök)

- ◆ Az alkeszek telefonszámai függetlenek az általuk kedvelt sörtől.
 - ◆ név->->tel és név ->->kedveltSörök.
- ◆ Így egy-egy alkesz minden telefonszáma minden általa kedvelt sörrrel kombinációban áll.
- ◆ Ez a jelenség független a funkcionális függőségektől.
 - ◆ itt a név->cím az egyetlen FF.

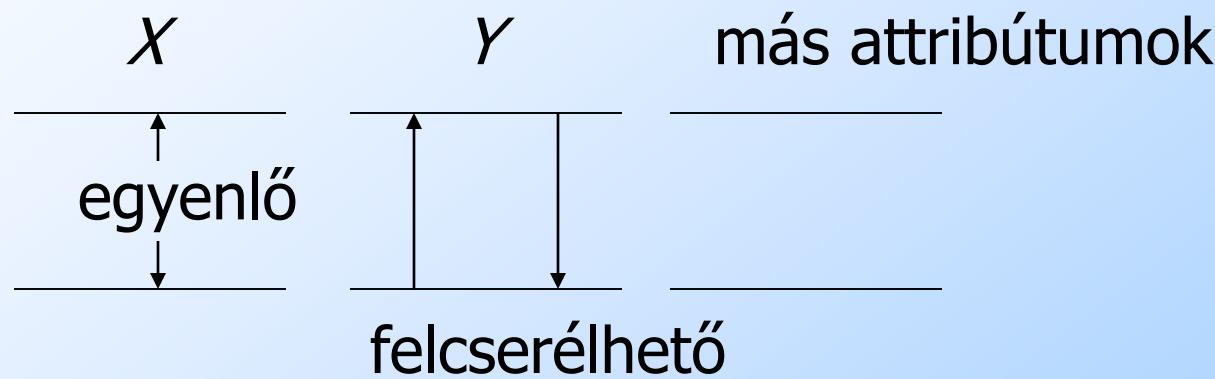
A név->->tel által implikált sorok

Ha ezek a soraink vannak:

név	cím	tel	kedveltSörök
sue	a	p1	b1
sue	a	p2	b2
sue	a	p2	b1
sue	a	p1	b2

Akkor ezeknek a soroknak is szerepelnie kell.

Az $X \rightarrow\rightarrow Y$ TÉF képe



TÉF szabályok

◆ minden FF TÉF.

- ◆ Ha $X \rightarrow Y$ és két sor megegyezik X-en, Y-on is megegyezik, emiatt ha ezeket felcseréljük, az eredeti sorokat kapjuk vissza, azaz: $X \rightarrow \rightarrow Y$.

◆ *Komplementálás* : Ha $X \rightarrow \rightarrow Y$ és Z jelöli az összes többi attribútum halmazát, akkor

$X \rightarrow \rightarrow Z$.

Nem tudunk darabolni

- ◆ Ugyanúgy, mint az FF-ek esetében, a baloldalakat nem „bánthatjuk” általában.
- ◆ Az FF-ek esetében a jobboldalakat felbonthattuk, míg ebben az esetben ez sem tehető meg.

Példa: többattribútumos jobboldal

Alkesz(név, tTársaság, tel, kedveltSörök,
gyártó)

- ◆ Egy alkesznek több telefonja lehet, minden számot két részre osztunk: tTársaság (pl. Vodafone) és a maradék hét számjegy.
- ◆ Egy alkesz több sört is kedvelhet, mindegyikhez egy-egy gyártó tartozik.

Példa folytatás

- ◆ Mivel a tTársaság-tel kombinációk függetlenek a kedveltSörök-gyártó kombinációtól, azt várjuk, hogy a következő TÉF-ek teljesülnek:

név ->-> tTársaság tel

név ->-> kedveltSörök gyártó

Példa adat

Egy lehetséges előfordulás, ami teljesíti az iménti TÉF-et:

név	tTársaság	tel	kedveltS	gyártó
Sue	20	555-1111	Bud	A.B.
Sue	20	555-1111	WickedAle	Pete's
Sue	70	555-9999	Bud	A.B.
Sue	70	555-9999	WickedAle	Pete's

Ugyanakkor sem a név->->tTársaság sem a név->->tel függőségek nem teljesülnek.

Negyedik normálforma

- ◆ A TÉF-ek okozta redundanciát a BCNF nem szünteti meg.
- ◆ A megoldás: a negyedik normálforma!
- ◆ A negyedik normálformában (4NF), amikor dekomponálunk, a TÉF-eket úgy kezeljük, mint az FF-eket, a kulcsok megtalálásánál azonban nem számítanak.

4NF definíció

- ◆ Egy R reláció $4NF$ -ben van ha:
minden $X \rightarrow\!-\!\rightarrow Y$ nemtriviális $TÉF$
(MVD) esetén X szuperkulcs.
 - ◆ *Nemtriviális TÉF*:
 1. Y nem részhalmaza X -nek,
 2. X és Y együtt nem adják ki az összes attribútumot.
 - ◆ A szuperkulcs definíciója ugyanaz marad,
azaz csak az FF-ektől függ.

BCNF kontra 4NF

- ◆ Kiderült, hogy minden $X \rightarrow Y$ FF $X \rightarrow\rightarrow Y$ TÉF is.
- ◆ Így, ha R 4NF-ben van, akkor BCNF-ben is.
 - ◆ Mert minden olyan FF, ami megséríti a BCNF-t, a 4NF-t is megséríti.
- ◆ De R lehet úgy BCNF-ben, hogy közben nincs 4NF-ben.

Dekompozíció és 4NF

- ◆ Ha $X \rightarrow\!-\!\rightarrow Y$ megséríti a 4NF-t, akkor R -t majdnem ugyanúgy dekomponáljuk, mint a BCNF esetén.
 1. XY az egyik dekomponált reláció.
 2. Az $Y - X$ -be nem tartozó attribútumok a másik.

Példa: 4NF dekompozíció

Alkesz(név, cím, tel, kedveltSörök)

FF: név -> cím

TÉF-ek: név ->-> tel

 név ->-> kedveltSörök

- ◆ Kulcs {név, tel, kedveltSörök}.
- ◆ Ezért
- ◆ Az összes függőség megsérte 4NF-et.

Példa folytatás

- ◆ Dekompozíció név -> cím szerint:

1. Alkesz1(név, cím)

- ◆ Ez 4NF-beli; az egyetlen függőség név-> cím.

2. Alkesz2(név, tel, kedveltSörök)

- ◆ Nincs 4NF-ben. A név ->-> tel és név ->-> kedveltSörök függőségek teljesülnek. A három attribútum együtt kulcs (mivel nincs nemtriviális FF).

Példa: Alkesz2 dekompozíciója

- ◆ Bármelyik, név ->-> tel, vagy a név ->-> kedveltSörök TÉF szerinti dekompozíció ugyanazt eredményezi:
 - ◆ Alkesz3(név, tel)
 - ◆ Alkesz4(név, kedveltSörök)

TÉF és FF-ek együttes következtetése

- ◆ **Probléma:** R relációsémához adott a TÉF-ek és FF-ek egy halmaza, kérdés: egy adott FF vagy TÉF következik-e ezekből R fölött?
- ◆ **Megoldás:** használunk egy táblázatot (tablót), hogy a függőségek hatásait feltárjuk. (A chase mögötti ötletet terjesztjük ki.)

Miért foglalkozunk ilyesmivel egyáltalán?

1. 4NF azon múlik, hogy van-e olyan TÉF, ami sérti a feltételt.
 - ◆ Előfordulhat, hogy a megadott FF-ek és TÉF-ek nem sértik a feltételt, de egy belőlük következő függőség igen.
2. Amikor dekomponálunk az FF-eket és TÉF-eket is vetítenünk kell.

Példa: CHASE TÉF-ek és FF-ek esetére

- ◆ Az FF-ek esetén ugyanúgy tegyük egyenlővé a szimbólumokat, mint korábban.
- ◆ Egy TÉF esetén írjuk be azokat a sorokat, melyek szükségesek ahhoz, hogy az előfordulás ne sértse meg a TÉF-et.
- ◆ $X->->Y$: ha van két sor a tablóban, amelyek megegyeznek X -en \rightarrow készíthetünk 2 újabb sort, megcserélve Y -on elhelyezkedő komponenseiket

Példa: CHASE TÉF-ek és FF-ek esetére

- ◆ A 2 új sornak a relációban szerepelnie kell → a tablóban is
- ◆ Ha FF-ekből és TÉF-ekből szeretnénk levezetni egy $X \rightarrow\rightarrow Y$, akkor 2 soros tablóval kezdünk, amelyek X -en megegyeznek a többinél különböznek
- ◆ A fentieket alkalmazzuk; ha észrevesszük, hogy az eredeti sorok egyikében az Y attr.-okat kicseréljük egy másik eredeti sorból ugyanazokkal → beláttuk a függőséget

Példa: CHASE TÉF-ek és FF-ek esetére

- ◆ Kiindulásként: legyen az első sor olyan, hogy nem indexelt betűket tartalmaz X-en és Y-on, a második pedig ugyanilyeneket X-en, és azonkívül a nem Y-belieken.
- ◆ A két sorban fennmaradó helyeken új, egyszer szereplő szimbólumok legyenek
- ◆ Kérdés: előfordul-e az a sor a tablóban, amelynek minden eleme indexeletlen?

A tabló $A \rightarrow C$ bizonyítása

◆ Példa: ha $A \rightarrow \rightarrow BC$ és $D \rightarrow C$, akkor
 $A \rightarrow C$ is teljesül minden esetben.

Cél: bizonyítani, hogy $c_1 = c_2$.

A	B	C	D
a	b1	c1	d1
a	b2	c1	d2
a	b2	c1	d1
a	b1	c1	d2

$A \rightarrow \rightarrow BC$ használata.

$D \rightarrow C$ -t használjuk.

Példa: tranzitivitás TÉF-ek esetén

- ◆ Ha $A \rightarrow\rightarrow B$ és $B \rightarrow\rightarrow C$, akkor $A \rightarrow\rightarrow C$?
 - ◆ Ha a séma ABC , akkor a komplementálási szabályból ez valóban következik.
 - ◆ A példában feltesszük hogy a séma: $ABCD$, és be fogjuk látni, hogy ott is igaz.
 - ◆ (Általában ebben a formában nem igaz, hanem az igaz: ha $A \rightarrow\rightarrow B$ és $B \rightarrow\rightarrow C$, akkor $A \rightarrow\rightarrow C-B$)

A tábló A->->C esetén

Cél: megjelenjen az (a,b,c,d) sor.

A	B	C	D
a	b1	c	d1
a	b	c1	d
a	b	c	d1
a	b1	c1	d
a	b	c1	d1
a	b	c	d
a	b1	c1	d1
a	b1	c	d

A->->B
használata.

B->->C
használata.

Következtetés: FF használata

- ◆ FF $X \rightarrow Y$ alkalmazásánál keressük meg azon sorpárokat, amelyek megegyeznek X attribútumain. Az Y attribútumain is tegyük őket egyenlővé.
 - ◆ Egy változót egy másikra cseréljünk.
 - ◆ Ha a lecserélt változó a célsorban is megjelenik, ott is cseréljünk.

Következtés: TÉF használata

- ◆ Egy $X \rightarrow\rightarrow Y$ TÉF használatánál keressünk két sort, amelyek megegyeznek X attribútumain.
 - ◆ Adjuk hozzá a tablóhoz azokat a sorokat, amelyeket az Y attribútumaihoz tartozó értékek felcserélésével kapunk.

Következtetés: célok

- ◆ Az $U \rightarrow V$ ellenőrzésekor akkor nyertünk, ha a megfelelő változók V -hez tartozó minden oszlopban egyenlőek.
- ◆ $U \rightarrow \rightarrow V$ akkor győztünk, ha sikerül egy olyan sort kigenerálni, ami az eredeti két sorból keletkezik V értékeinek felcserélésével.

Következtetés: Végjáték

- ◆ Használjuk az összes FF-et és TÉF-et, amíg bármiféle változtatás történhet.
- ◆ Ha nyertünk, nyertünk.
- ◆ Ha nem, egy ellenpéldát kaptunk.
 - ◆ A kapott előfordulás az összes előre megadott függőséget teljesíti.
 - ◆ Az eredeti két sor megséríti a kikövetkeztetendő függőséget.

TÉF-ek vetítése

- ◆ Le kell tudnunk vetíteni megadott függőségeket 2 reláció sémára
- ◆ Legrosszabb eset: ki kell próbálnunk minden lehetséges FF-et és TÉF-et a felbontott relációkra
- ◆ Chase teszt alkalmazása; Cél egy TÉF ellenőrzésénél: olyan sor előállítása a tablóban, amely indexeletlen betűket tartalmaz a felbontott reláció oszlopaira³⁰

Példa: vetítés

- ◆ Példa: adott $R(A,B,C,D,E) \rightarrow$ felbontunk
- ◆ Egyik létrejövő reláció: $S(A,B,C)$
- ◆ Tf. $A \rightarrow\!\!> CD$ R-ben fennáll
- ◆ Cél: bizonyítani, hogy $A \rightarrow\!\!> C$ fennáll S-ben

A	B	C	D	E
a	b1	c	d1	e1
a	b	c2	d	e
a	b1	c2	d	e1
a	b	c	d1	e



$A \rightarrow\!\!> CD$ használata.

Egyed-kapcsolat modell

E/K diagramok

Gyenge egyedhalmazok

E/K diagramok átírása relációsémákká

Az E/K modell célja

- ◆ Az E/K modellel az adatbázissémát vázolhatjuk fel.
 - ◆ Bizonyos megszorításokat is megadhatunk, de műveleteket nem.
- ◆ A tervezet *egyed-kapcsolat diagramnak* nevezzük.
- ◆ Később: az E/K diagramot relációs adatbázissémává alakítjuk..

Az E/K modell „kerete”

- ◆ A tervezés komoly része az AB építésnek.
- ◆ A “főnök” általában abban biztos, hogy szeretne egy adatbázist, de azt már nem feltétlen tudja, mi és hogyan legyen benne.
- ◆ A főbb részek felvázolásával hatékonyan lehet megtervezni egy működő adatbázist.

Egyedhalmazok

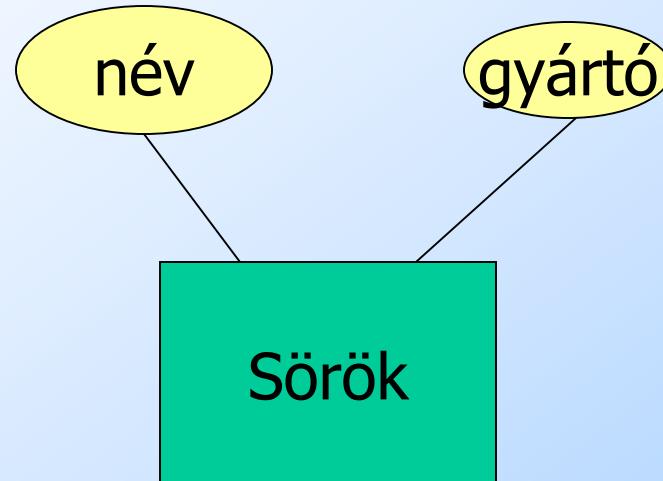
- ◆ *Egyed* = “valóságdarab” vagy objektum.
- ◆ *Egyedhalmaz* = hasonló egyedek kollekciója.
 - ◆ Az objektum-orientált nyelvek ***osztály foglamához*** hasonló.
- ◆ *Attribútum* = az egyedhalmaz egyedeinek egy tulajdonsága.
 - ◆ Az attribútumok atomi értékűek, például egészek (integer) vagy karakter sztringek, de nem rekordok, tömbök é.i.t.

E/K diagramok

◆ Az egyed-kapcsolat diagramban:

- ◆ az egyedhalmaz = téglalap,
- ◆ az attribútum = a megfelelő egyedhalmazt reprezentáló téglalappal összekötött ovális (krumpli).

Példa:

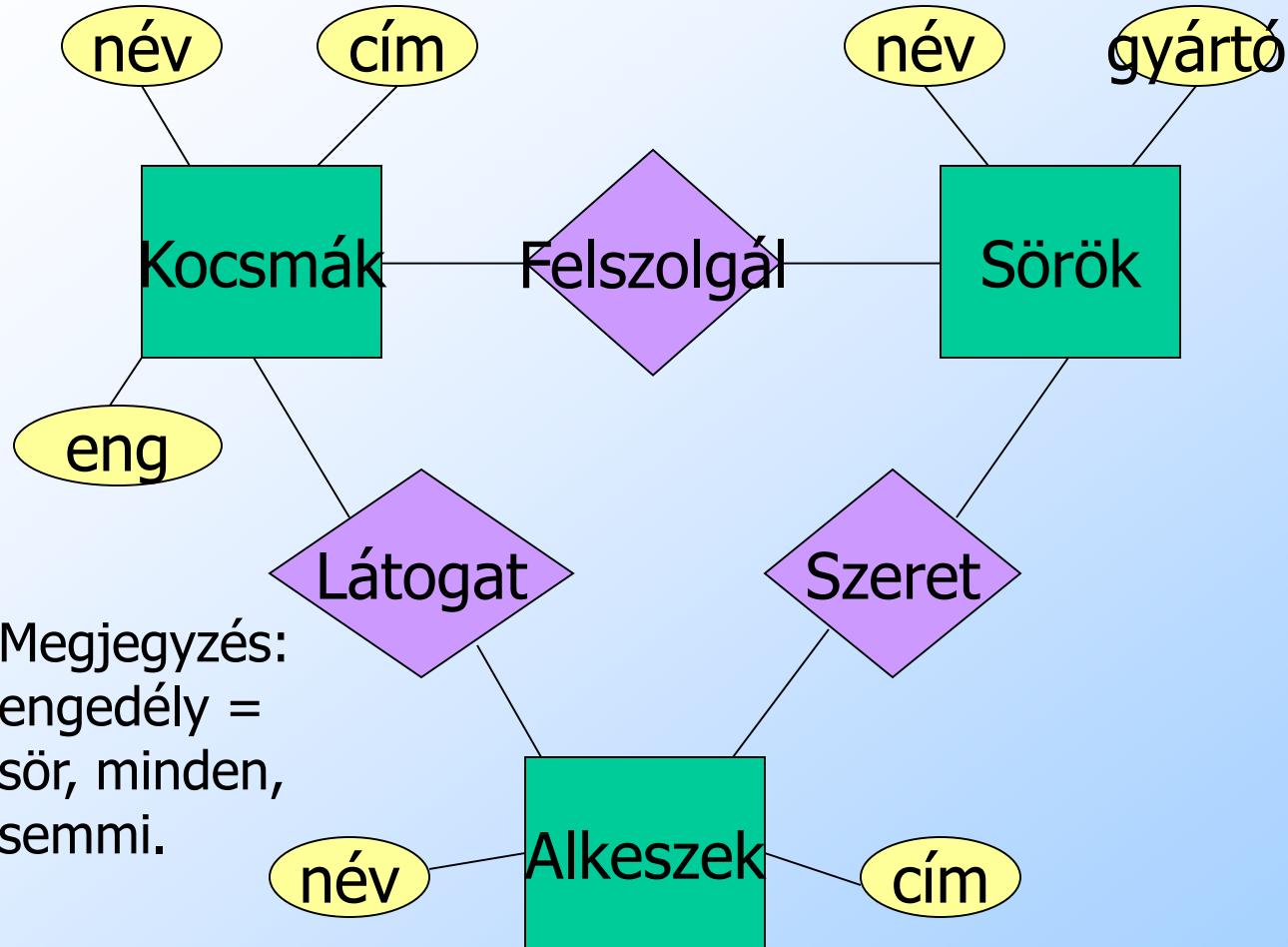


- ◆ A **Sörök** egyedhalmaznak két attribútuma van: **név** és **gyártó**.
- ◆ minden **Sörök** entitás értékeket kap ezeken az attribútumokon, pl. (Bud, Anheuser-Busch).

Kapcsolatok

- ◆ Egy **kapcsolat** kettő vagy több egyedhalmazt köt össze.
- ◆ Rombusszal jelöljük, melyet a kapcsolatban részt vevő egyedhalmazok téglalapjaival kötünk össze.

Példa: kapcsolatok



Kocsmákban
sört árulnak.

Az alkeszek
szeretnek néhány
sört.

Emellett
kocsmákba
járnak.

Kapcsolat halmaz

- ◆ Az egyedhalmaz “értéke” a hozzá tartozó egyedekek halmaza.
 - ◆ Példa: az adatbázisunkban az összes kocsma halmaza.
- ◆ Egy kapcsolat “értéke” a *kapcsolat halmaz*, azaz sorok egy halmaza, melynek minden eleme egy a kapcsolatban résztvevő egyedhalmazból való.

Példa: kapcsolat halmaz

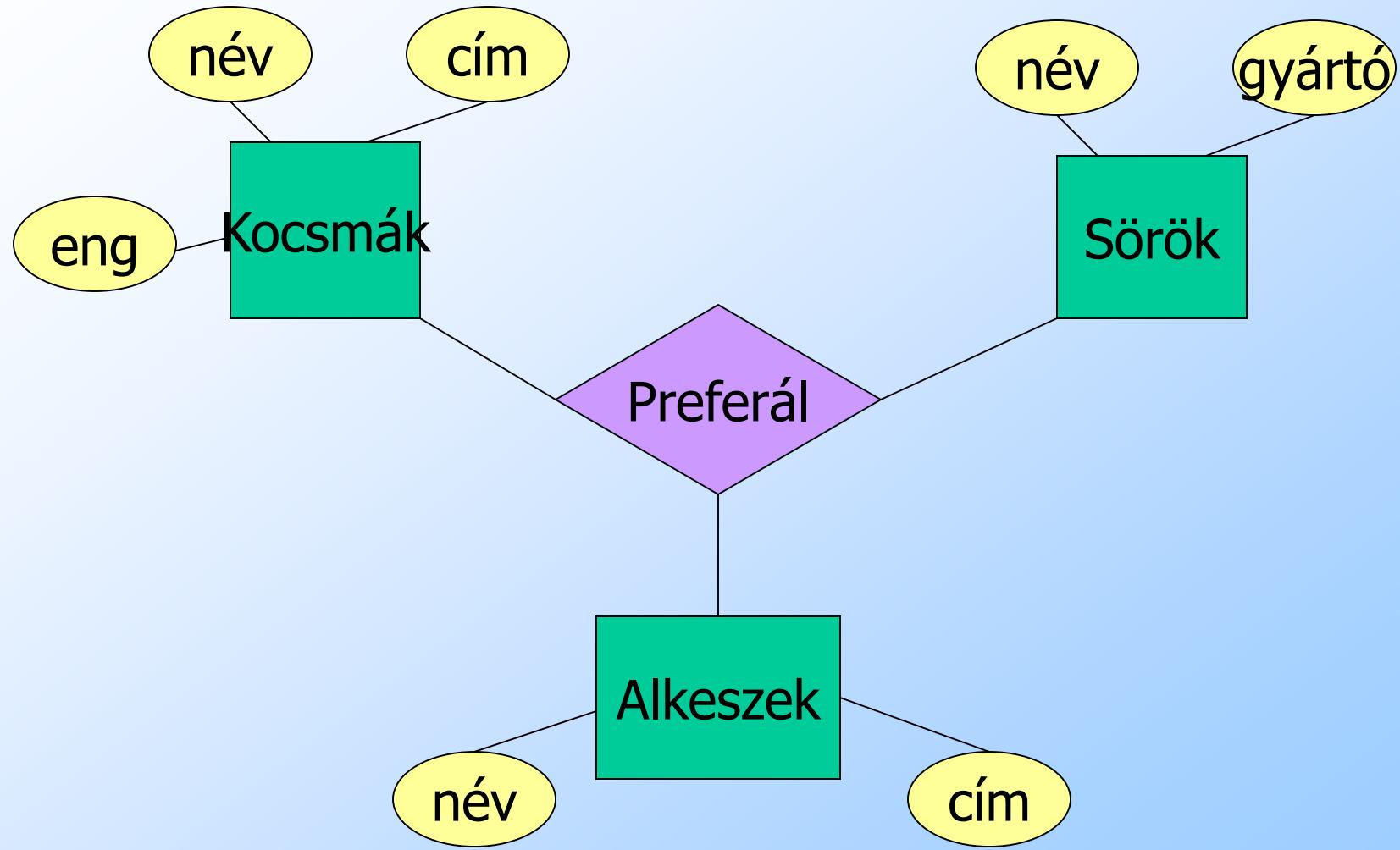
- ◆ A **Felszolgál** kapcsolathoz például az alábbi kapcsolat halmaz tartozhat:

Kocsma	Sör
Joe bárja	Bud
Joe bárja	Miller
Sue bárja	Bud
Sue bárja	Pete's Ale
Sue bárja	Bud Lite

Többirányú kapcsolatok

- ◆ Esetenként több, mint két egyedhalmazt összekötő kapcsolatra van szükség.
- ◆ Tegyük fel, hogy az egyes alkeszek bizonyos söröket csak bizonyos kocsmákban fogyasztanak.
 - ◆ A bináris kapcsolataink **Szeret**, **Felszolgál** és **Látogat** segítségével ezt nem tudjuk leírni.
 - ◆ De egy hármas kapcsolattal igen.

Példa: hármas kapcsolat



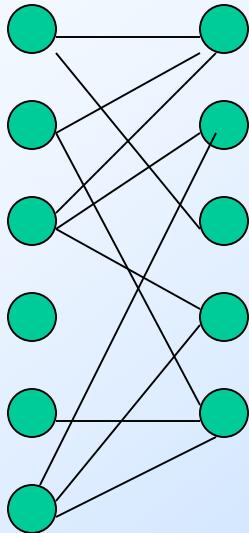
Egy tipikus kapcsolat halmaz

Kocsma	Alkesz	Sör
Joe bárja	Anna	Miller
Sue bárja	Anna	Bud
Sue bárja	Anna	Pete's Ale
Joe bárja	Pál	Bud
Joe bárja	Pál	Miller
Joe bárja	Karcsi	Miller
Sue bárja	Karcsi	Bud Lite

Sok-sok kapcsolat

- ◆ Elsősorban a **bináris** kapcsolatok esetén értelmezzük, mint például a **Kocsmák** és **Sörök** közötti **Felszolgál** kapcsolat.
- ◆ A **sok-sok kapcsolatokban**, minden a két résztvevő egyedhalmaz egyedei több másik egyedhez kapcsolódhatnak.
 - ◆ A példában: egy kocsma több sört árulhat; egy sört is több kocsmában árulhatnak.

Képen

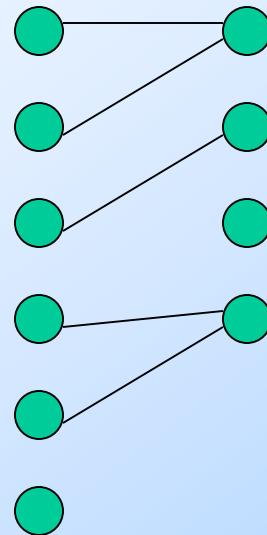


sok-sok

Sok-egy kapcsolat

- ◆ Ezen kívül léteznek még *sok-egy* kapcsolatok.
- ◆ minden egyede az első egyedhalmaznak legfeljebb egy egyedhez kapcsolódhat a másik halmazból.
- ◆ De a második halmaz entitásai nulla, egy vagy több entitáshoz is kapcsolódhatnak az első halmazból.

Képen



sok-egy

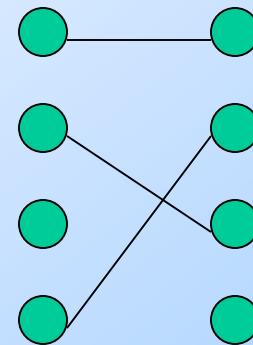
Példa: sok-egy kapcsolat

- ◆ Az Alkeszekből a Kocsmákba menő Kedvenc (Preferál) kapcsolat sok-egy típusú.
- ◆ Egy alkesznek legfeljebb egy kedvenc söre lehet.
- ◆ De egy sör több vagy akár nulla alkesznek is a kedvence lehet.

Egy-egy kapcsolatok

- ◆ Egy *egy-egy kapcsolat* esetén minden egyes entitás legfeljebb egyetlen másik entitáshoz kapcsolódhat.
- ◆ Példa: a Best-seller kapcsolat a Gyártók és Sörök egyedhalmazok között.
 - ◆ Egy sört nem gyárthat több cég és egy gyártónak sem lehet több bestseller söre.

Képen

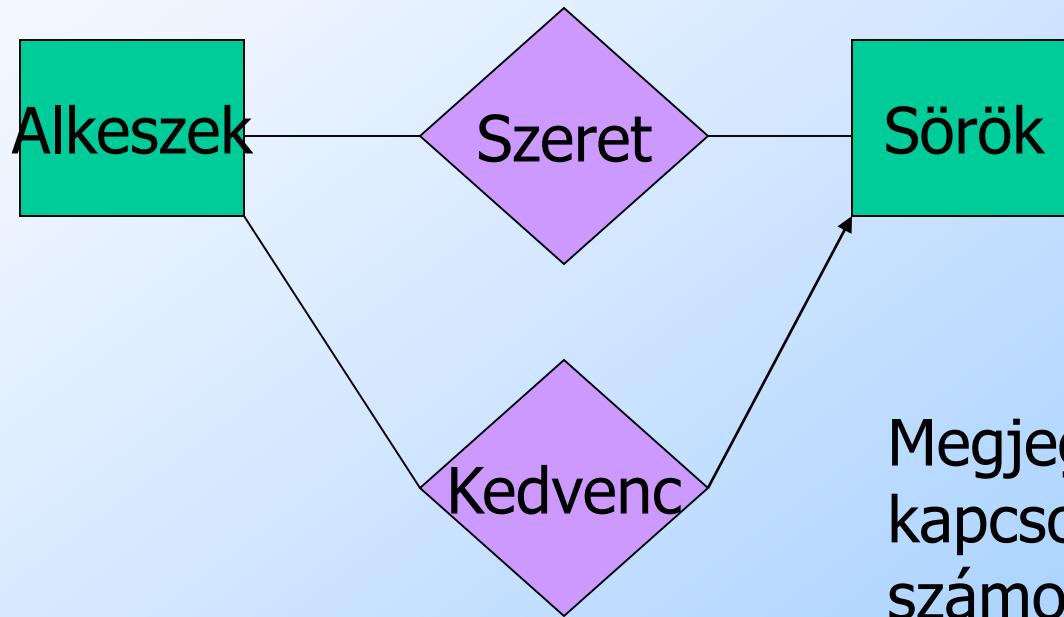


egy-egy

A „számossság” jelzése

- ◆ A sok-egy kapcsolat „egy oldalát” egy nyíl jelzi.
- ◆ Az egy-egy kapcsolatok mindenkét végén nyíl van.
- ◆ A **lekerékített nyíl** jelentése: „pontosan egy”, azaz minden első halmazbeli entitásnak pontosan egy párja van a második egyedhalmazból.

Példa: sok-egy kapcsolat



Megjegyzés: a két kapcsolat különböző számoságú.

Példa: egy-egy kapcsolat

- ◆ Vegyük a Gyártók és Sörök közötti Bestseller kapcsolatot.
- ◆ Néhány sör valószínűleg egyetlen gyártónak sem lesz bestsellere, tehát a Gyártók egyedhalmaz felőli nyíl nem lehet kerek.
- ◆ De egy gyártónak kell lennie bestsellerének.

Az E/K diagramon



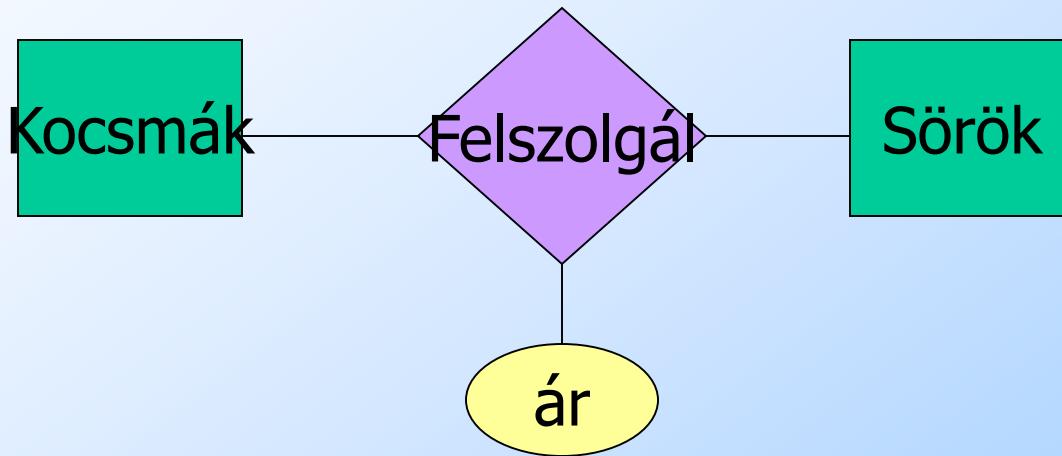
Egy sör 0 vagy 1
gyártó bestsellere.

Egy gyártónak
pontosan egy
bestsellere van.

A kapcsolatok attribútumai

- ◆ Néha hasznos ha a kapcsolathoz is illeszthetünk attribútumot.
- ◆ Erre úgy tekinthetünk, mint a kapcsolat halmazban lévő sorok egy tulajdonságára.

Példa: kapcsolat attribútuma

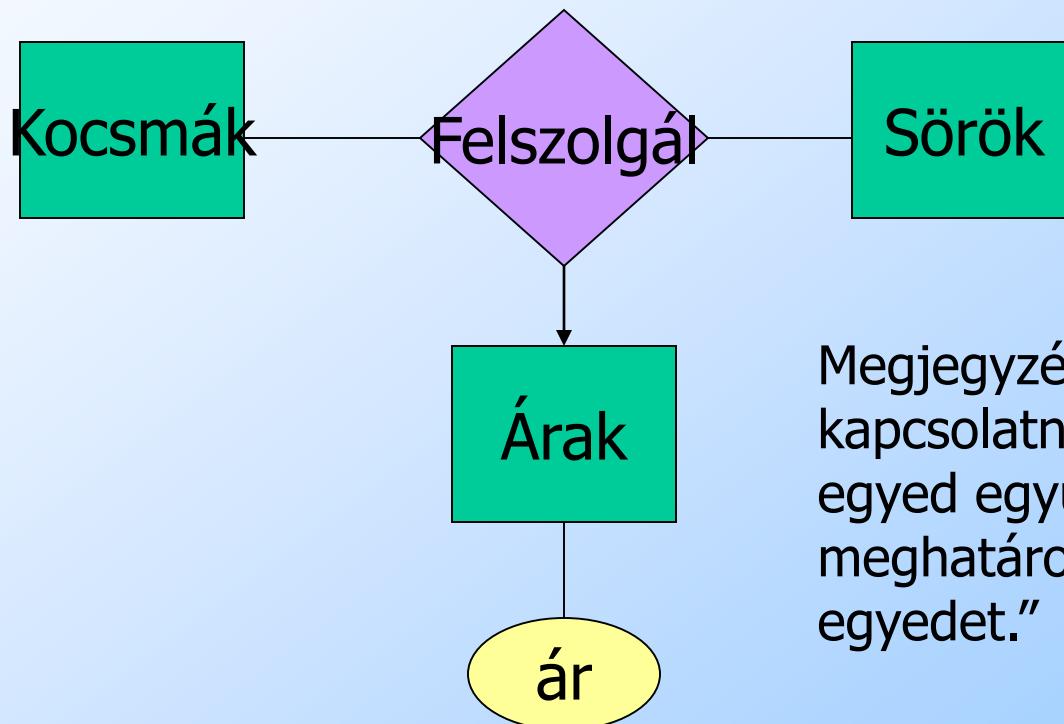


Az ár a kocsmának és a sörnek együttes függvénye.

Egy másik lehetőség kapcsolatok attribútumának ábrázolására

- ◆ „Vegyünk fel” egy egyedhalmazt, ami a kérdéses attribútum értékeit reprezentálja.
- ◆ Ezt az új egyedhalmazt „vonjuk be” a szóban forgó kapcsolatba.

Példa: kapcsolat attribútumának átírása



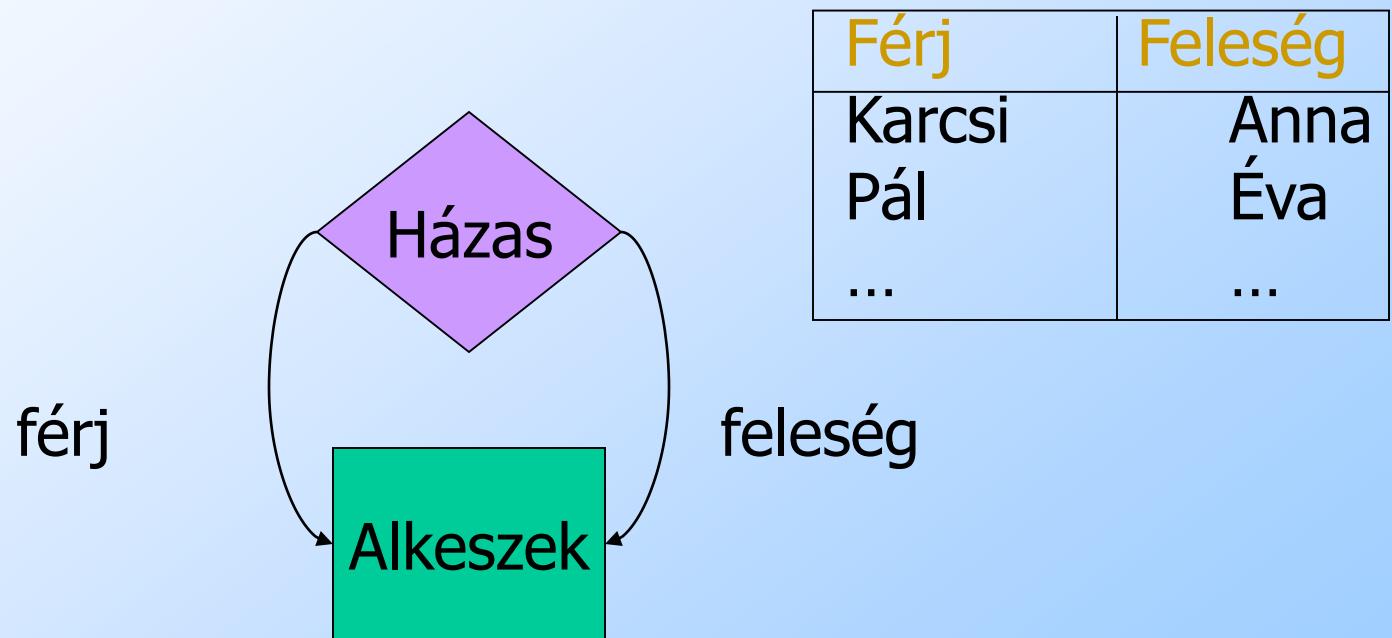
Megjegyzés: nyíl egy többirányú kapcsolatnál = „a többi egyed együttese egyértelműen meghatározza a szóban forgó egyedet.”

Szerepek

- ◆ Néha egy egyedhalmaz többször is megjelenik egy kapcsolatban.
- ◆ Ilyenkor a megfelelő éleket címkézzük és a címkéket *szerepeknek* nevezzük.

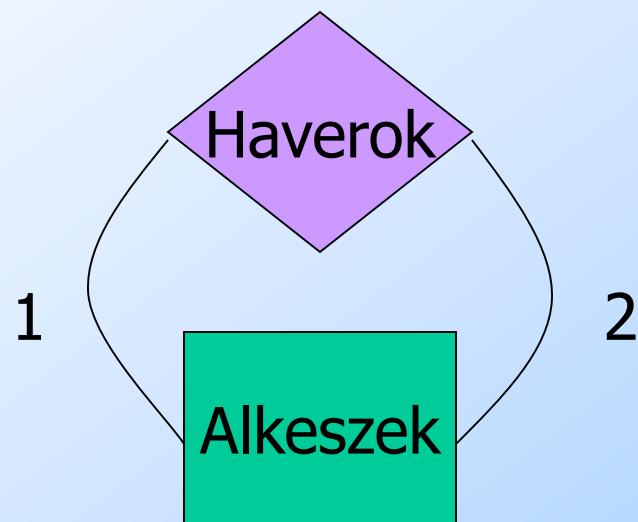
Példa: szerepek

Kapcsolat halmaz



Példa: szerepek

Kapcsolat halmaz



Haver1	Haver2
Pál	Anna
Rezső	Éva
Éva	Karczi
Richárd	Jenő
...	...

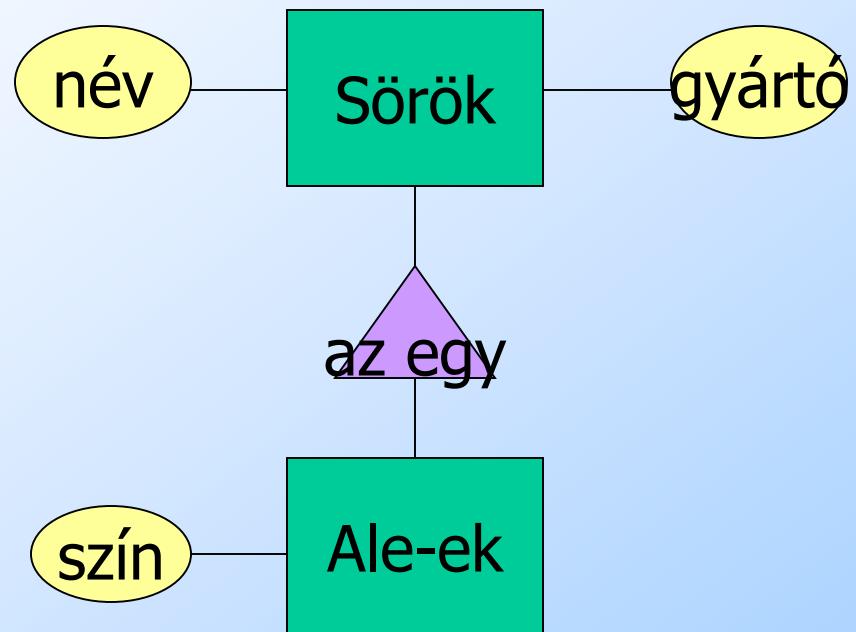
Alosztályok

- ◆ *Alosztály* = speciális eset = kevesebb egyed = több tulajdonság (kapcsolat).
- ◆ **Példa:** Az Ale-ek mind sörök is.
 - ◆ Nem minden sör ale (*angol barna*), de minden ale sör.
 - ◆ Tegyük fel, hogy a sörök már meglévő *tulajdonságai* (attribútumok és kapcsolatok) mellé az ale-ek esetében még egy a **szín** attribútumot is felveszünk.

Alosztályok E/K diagramokon

- ◆ Feltesszük, hogy az alosztályok rendszere fát alkot.
 - ◆ Azaz, nincs többszörös öröklés.
- ◆ Az alosztály kapcsolatot az-egy háromszögek jelölik.
 - ◆ Az ősosztályra mutat a háromszög felső csúcsa.

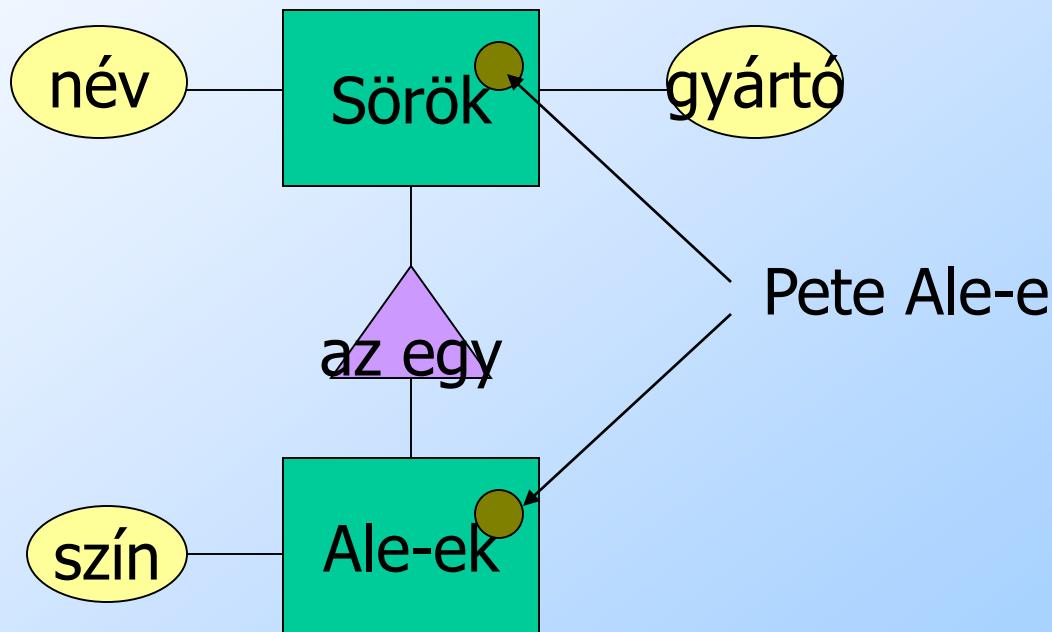
Példa: részosztályok



E/K vs. objektumorientált (OO) részosztályok

- ◆ Az OO paradigmában minden objektum pontosan egy osztálynak lehet eleme.
 - ◆ A részosztályok az ōosztályuktól örökölnek.
- ◆ Ezzel ellentétben az E/K entitásoknak minden részosztályban vannak *reprezentánsai*, amihez hozzá tartoznak.
 - ◆ Szabály: ha az *e* entitás szerepel egy részosztályban, akkor *e* szerepel az ōosztály(ok)ban is.

Példa: entitások előfordulásai



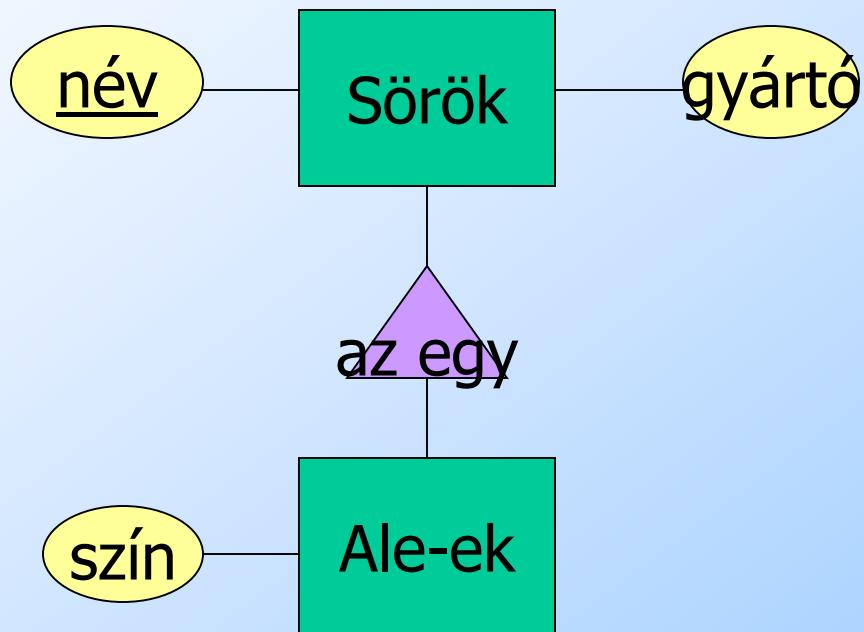
Kulcsok

- ◆ A *kulcs* az attribútumoknak egy olyan halmaza, amelyekre nem létezhet két olyan entitás, amelyek a kulcsattribútumok mindegyikén azonos értéket vennének fel.
 - ◆ Ugyanakkor a kulcs néhány attribútumán megegyezhetnek, de az összesen nem.
- ◆ minden entitáshalmazhoz meg kell adnunk egy kulcsot.

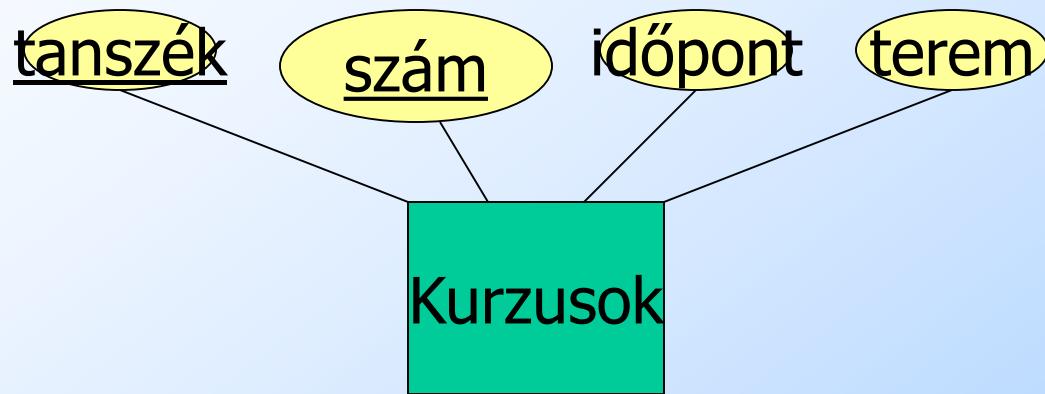
Kulcsok az E/K diagramokon

- ◆ A kulcsattribútumot(ka)t aláhúzással jelöljük.
- ◆ Egy öröklődési hierarchiában csak a gyökér entitáshalmaznak lehet kulcsa, ez lesz a hierarchiában szereplő többi alosztálynak is a kulcsa.

Példa: a név kulcs a Sörökben



Példa: több attribútumú kulcs



- Az **időpont** és **terem** attribútumok együtt szintén kulcsot alkotnak, a modellben azonban csak egy kulcsot adhatunk meg.

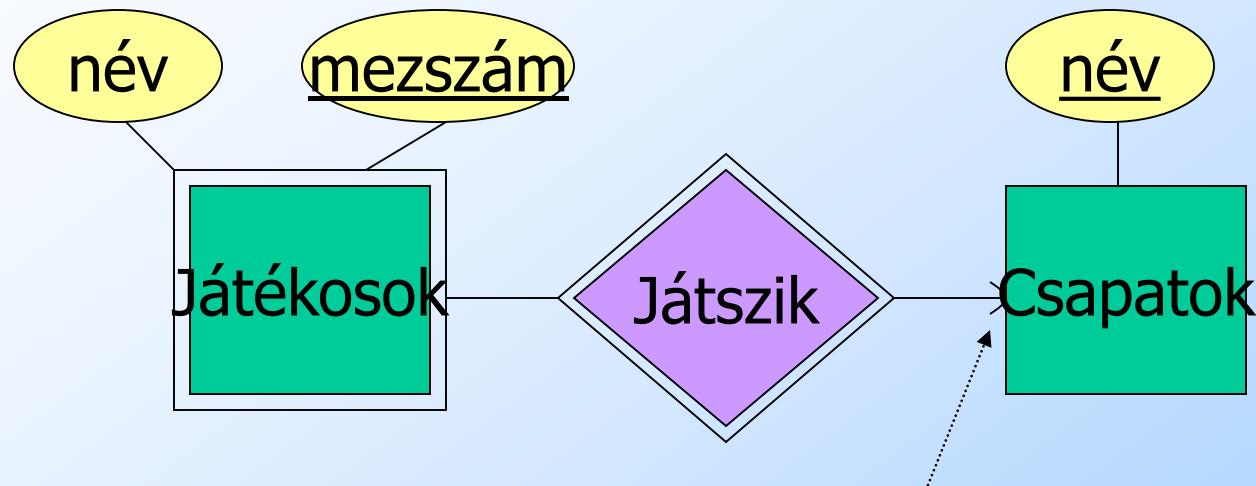
Gyenge egyedhalmazok

- ◆ Esetenként egy-egy egyedhalmaz egyedeit csak "külső segítséggel" lehet egyértelműen azonosítani.
- ◆ Egy E egyedhalmazt *gyengének* nevezünk, ha ahoz, hogy E elemeit azonosítsuk, egy vagy több, E -ből induló sok-egy kapcsolatot követve a kapcsolódó egyedek kulcsértékeire is szükségünk van.

Példa: gyenge egyedhalmazok

- ◆ a **név** majdnem kulcs a focisták esetén, ritkán azonban előfordulhat, hogy két játékosnak ugyanaz a neve.
- ◆ a **mezzám** nyilván nem kulcs.
- ◆ Ám a **mezzám** a csapat **nevével** kombinálva a **Játszik** kapcsolaton keresztül már egyedi minden játékos esetén.

Az E/K diagramon



Megjegyzés: itt minden játékoshoz kell, hogy tartozzon csapat.

- A gyenge egyedhalmazt dupla téglalap jelzi.
- A *támogató* sok-egy kapcsolatot dupla rombusszal jelöljük.

Gyenge egyedhalmaz szabályok

- ◆ Egy gyenge egyedhalmaznak egy vagy több sok-egy kapcsolata lehet más (támogató) egyedhalmazokhoz.
 - ◆ Nem az összes sok-egy kapcsolatnak kell támogatónak lennie.
 - ◆ De a támogató kapcsolatoknak kerek nyílban kell végződniük az egy oldalon (azaz minden entitásnak a gyenge egyedhalmazból pontosan egy egyedhez kell kapcsolódnia a támogató egyedhalmazból).

Gyenge egyedhalmaz szabályok – (2)

- ◆ A gyenge egyedhalmaz kulcsa saját aláhúzott és a támogató egyedhalmaz(ok) aláhúzott attribútumaiból áll.
 - ◆ Például a (játékos) **mezzáma** és a (csapat) **neve** kulcs lesz a **Játékosok** egyedhalmazban.

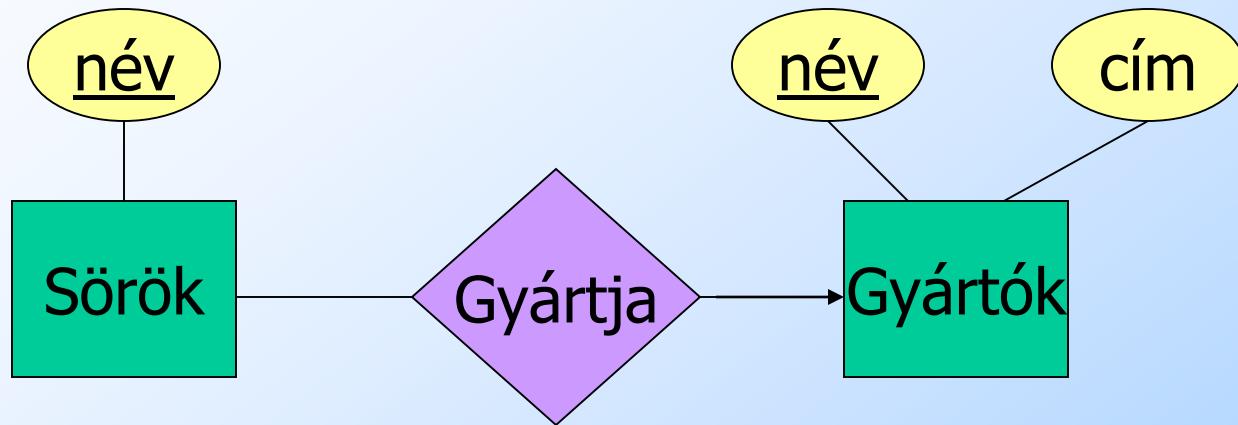
Tervezési technikák

1. Redundancia elkerülése.
2. A gyenge egyedhalmazok óvatos használata.
3. Ne használunk egyedhalmazt, ha egy attribútum éppúgy megfelelne a célnak.

Redundancia elkerülése

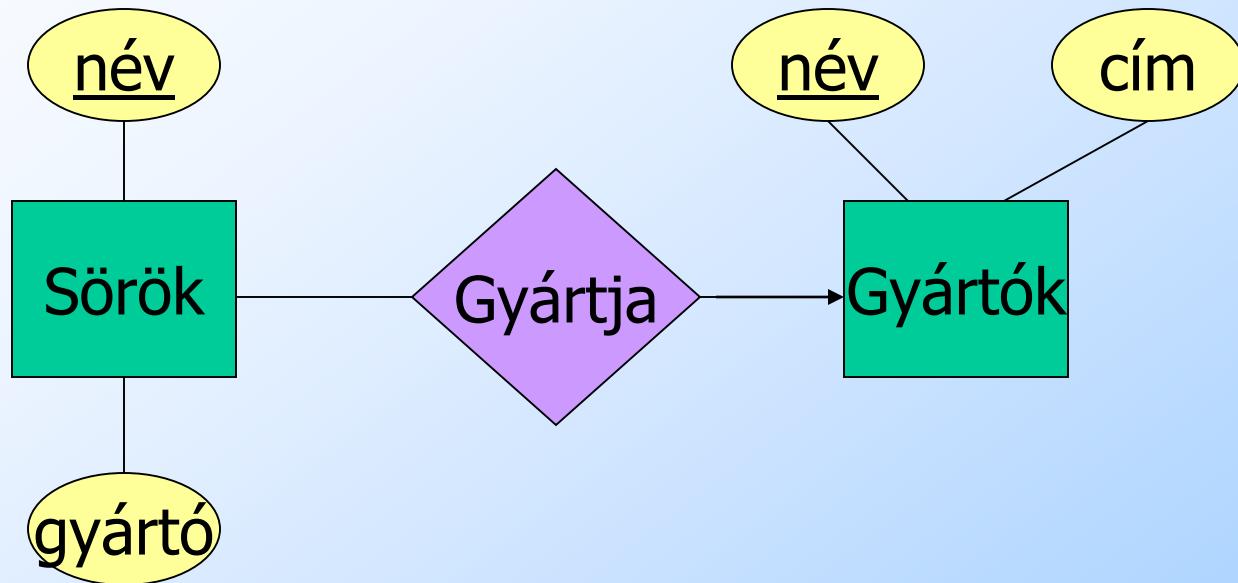
- ◆ *Redundancia* = ugyanazt a dolgot többféle módon is feltüntetjük.
- ◆ Ez egyrészt helypazarlás, másrészt növeli az inkonzisztencia veszélyét.
 - ◆ Ugyanazon tény két leírása akkor válik inkonzisztenssé, ha az egyik értéket megváltoztatjuk, míg a másikról megfeledkezünk.
 - ◆ Emlékezzünk vissza a módosítási anomáliára.

Példa: helyes



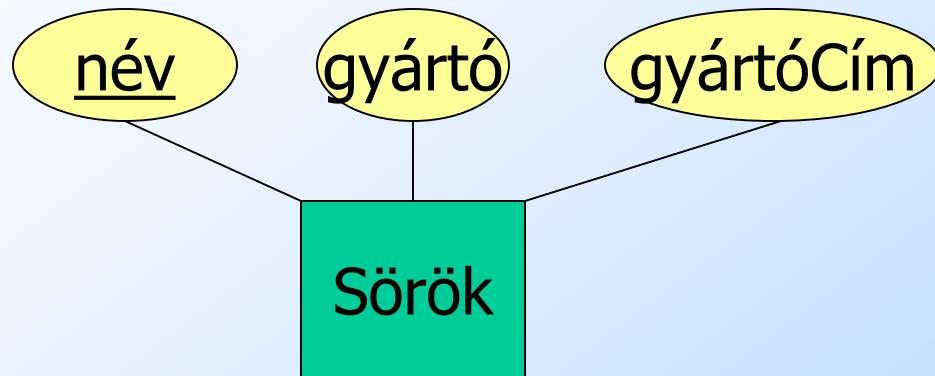
Ennél a diagramnál minden gyártónál pontosan egyszer szerepel a címe.

Példa: helytelen



Ennél a diagramnál viszont egy-egy sör gyártója kétszer is szerepel: attribútumként és a kapcsolódó Gyártók egyedhalmazon keresztül.

Példa: helytelen

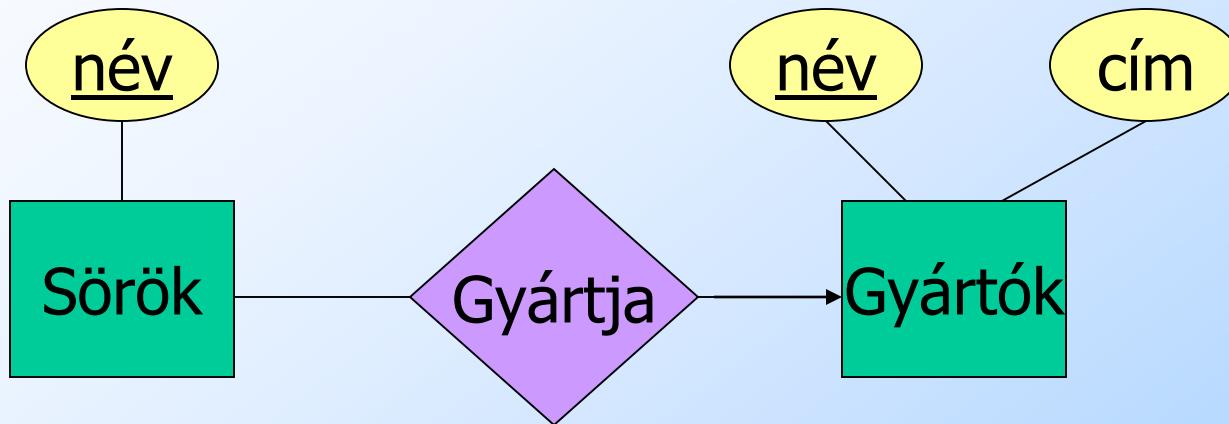


Ez a diagram minden sör esetén felsorolja a gyártó címét. Ha pedig ideiglenesen nincs megadva az adott gyártóhoz sör, akkor elveszítjük ezt a címet.

Egyedhalmazok vs. attribútumok

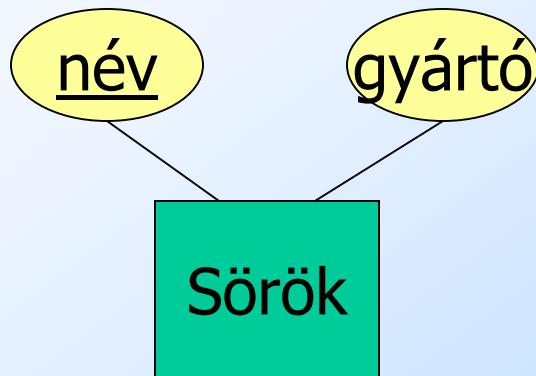
- ◆ Egy egyedhalmaznak legalább egy feltételnek eleget kell tennie az alábbiak közül:
 - ◆ Többnek kell lennie, mint egy egyszerű név, azaz legalább egy nem kulcs attribútumának lennie kell.
 - ◆ Vagy..
 - ◆ a „sok” végén szerepel egy sok-egy kapcsolatnak.

Példa: helyes



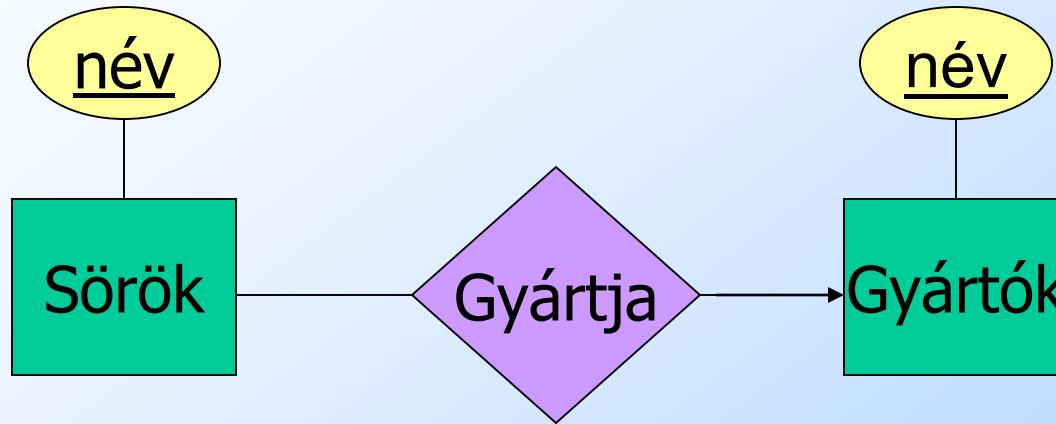
- A **Gyártóknak** valóban egyedhalmaznak kell lennie a nem kulcs **cím** attribútum miatt.
- A **Sörökre** ugyanez igaz, hiszen a **Gyártja** kapcsolat „sok” végén szerepel.

Példa: helyes



Itt nem szükséges külön egyedhalmazt definiálni a gyártók számára, hiszen a nevükön kívül mást nem tárolunk.

Példa: rossz



Mivel a gyártóknál csak a nevet tároljuk, ezen kívül a sok-egy kapcsolat „egy” oldalán szerepel, ezért fölösleges külön egyedhalmazt definiálni a számára.

Ne használunk gyenge egyedhalmazokat fölöslegesen

- ◆ Kezdő adatbázis tervezőknek gyakran kétségeik vannak abban tekintetben, hogy mit is lehet kulcsként definiálni, ami megállna önmaga jogán kulcsként.
 - ◆ minden egyedhalmazt gyengének definiálnak, amit az összes vele összekötött egyedhalmaz támogat.
- ◆ A „nagybetűs életben” gyakran külön azonosítókat készítenek az egyedhalmazokhoz.
 - ◆ Pl. TAJ, rendszámtábla stb.

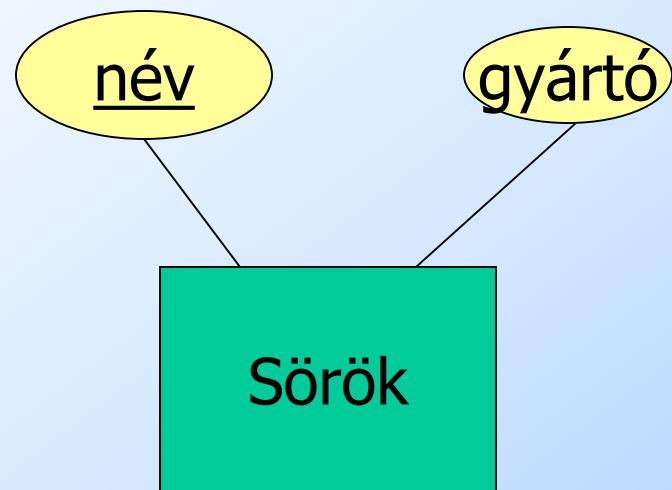
Mikor van szükség gyenge egyedhalmazokra?

- ◆ A legtöbb esetben nem létezik olyan globális fórum, amely kioszthatná az egyedi azonosítókat.
- ◆ Példa: nem valószínű, hogy el lehetne érni, hogy a világ összes focijátékosát külön mezszámmal regisztrálják.

E/K diagramok átírása relációsémáva

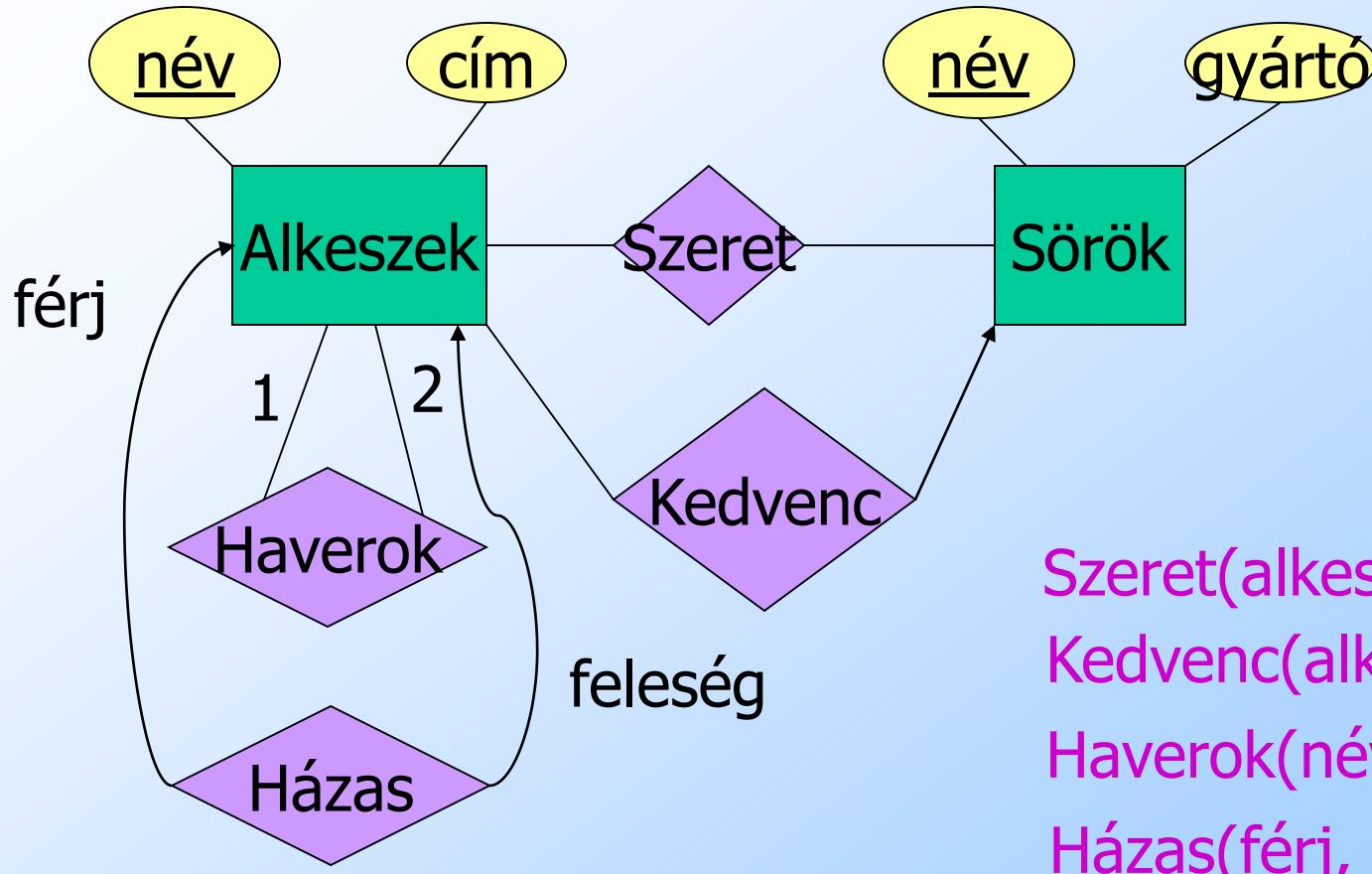
- ◆ Egyedhalmaz -> reláció.
 - ◆ Attribútumok -> attribútumok.
- ◆ Kapcsolat -> relációk, melyeknek az attribútumai csak:
 - ◆ az összekapcsolt egyedhalmazok kulcsattribútumait,
 - ◆ és a kapcsolat attribútumait tartalmazzák.

Egyedhalmaz -> Reláció



Reláció: **Sörök(név, gyártó)**

Kapcsolat -> Reláció



Relációk összevonása

- ◆ Egy relációba összevonható:
 1. Az E egyedhalmazból kapott reláció,
 2. valamint azon sok-egy kapcsolatok relációi, melyeknél az E a „sok” oldalon szerepel.
- ◆ Példa: Alkeszek(név, cím) és Kedvenc(alkesz, sör) összevonható az Alkeszek1(név, cím, kedvencSör) relációvá.

A sok-sok kapcsolat nem összevonható

- ◆ Az Alkeszek és Szeret relációk összevonása hiba lenne, redundanciához vezet.

név	cím	Sör
Zsuzsa	123 Maple	Bud
Zsuzsa	123 Maple	Miller

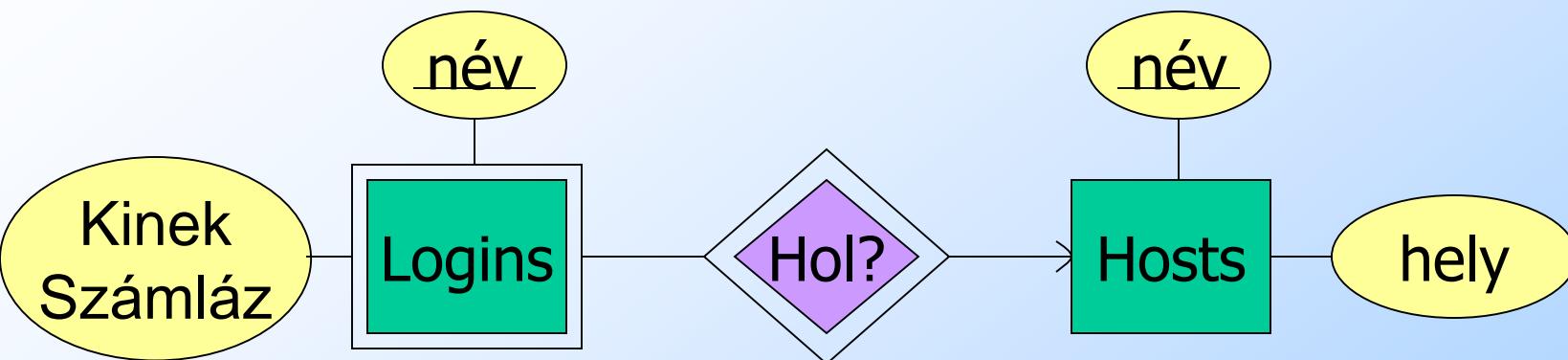
Redundancia



Gyenge egyedhalmazok átírása

- ◆ Egy gyenge egyedhalmazkból kapott relációt a teljes kulcsot tartalmaznia kell (a más egyedhalmazokhoz tartozó kulcs-attribútumokat is), valamint a saját, további attribútumokat.
- ◆ A támogató kapcsolatot nem írjuk át, redundanciához vezetne.

Példa: gyenge egyedhalmaz -> reláció



Hosts(hostNév, hely) (GazdaGép, Kiszolgáló)

Logins(loginNév, hostNév, billTo)

~~Host(loginNév, hostNév, hostNév2)~~

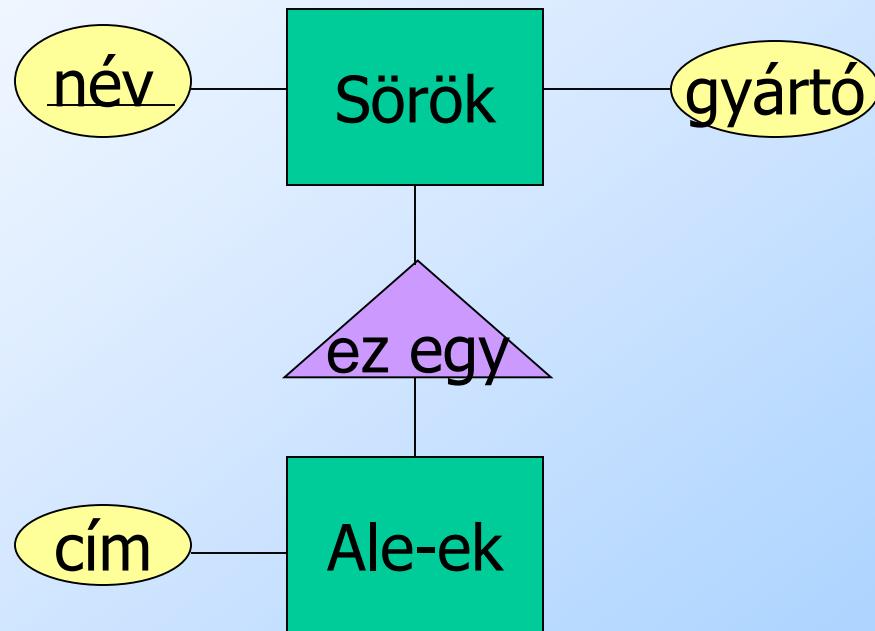
Meg kell egyezniük.

A Hol a Logins részévé válik.

Entitás Alosztályok: három megközelítés

1. *Objektumorientált*: minden alosztályhoz egy reláció tartozik az összes releváns attribútummal.
2. *Null értékek használata*: az öröklődési hierarchiát egyetlen reláció reprezentálja, az entitások a rájuk nem vonatkozó attribútumoknál null értéket kapnak.
3. *E/K style*: minden alosztályhoz külön reláció tartozik:
 - ◆ a kulcs-attribútummal (kulcs-attribútumokkal).
 - ◆ a részosztály további attribútumaival.

Példa: részosztály -> reláció



Objektumorientált

név	gyártó
Bud	Anheuser-Busch
Sörök	

név	gyártó	szín
Summerbrew	Pete's	barna
Ale-ek		

A „Pete által gyártott ale-ek színeit” visszaadó lekérdezések esetén hasznos

E/K Style

név	gyártó
Bud	Anheuser-Busch
Summerbrew	Pete's
Sörök	

név	szín
Summerbrew	barna
Ale-k	

A Pete által gyártott söröket visszaadó lekérdezések esetén hasznos.

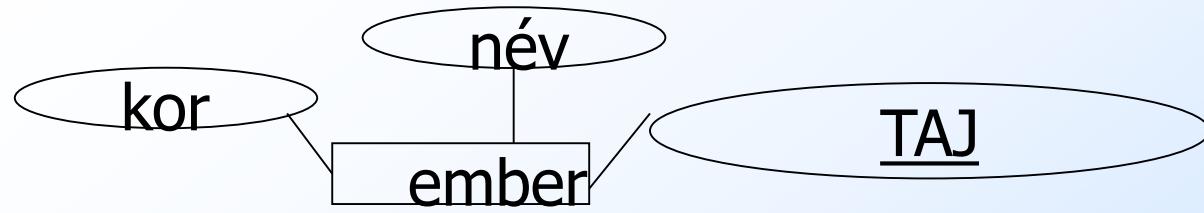
Null értékek használata

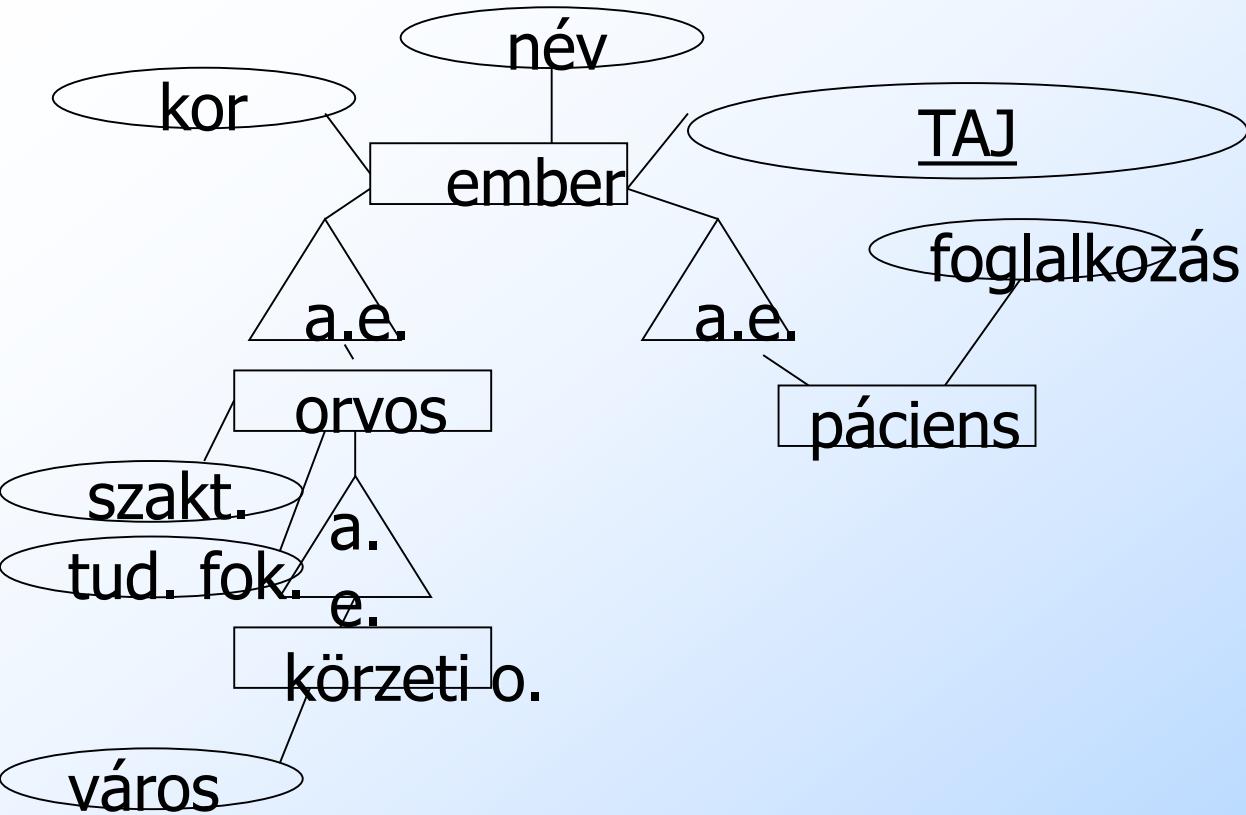
név	gyártó	szín
Bud Summerbrew	Anheuser-Busch Pete's	NONE barna
Sörök		

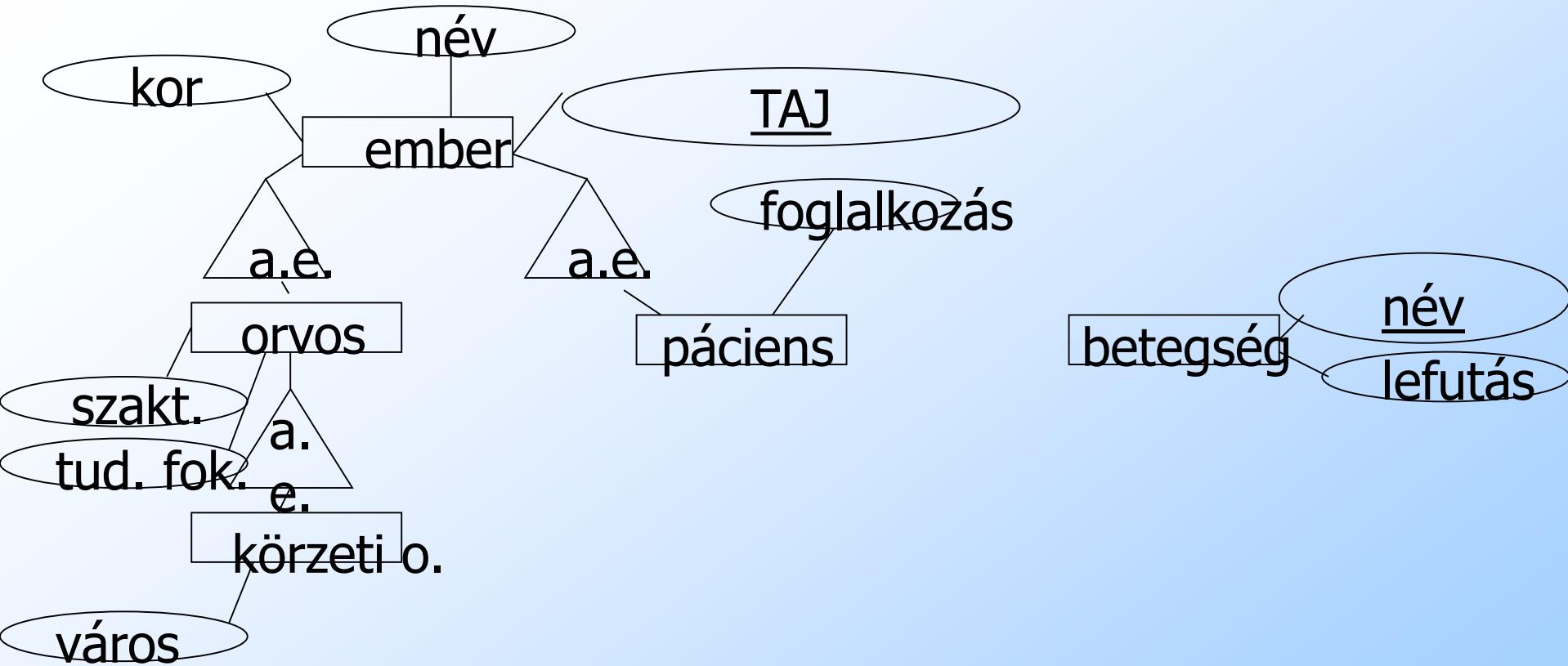
Helytakarékos, ha csak nem túl sok a NULL érték. Összekapcsolásokat is megspórolhatunk.

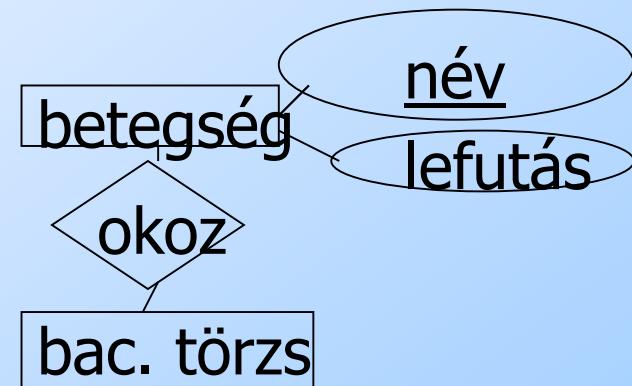
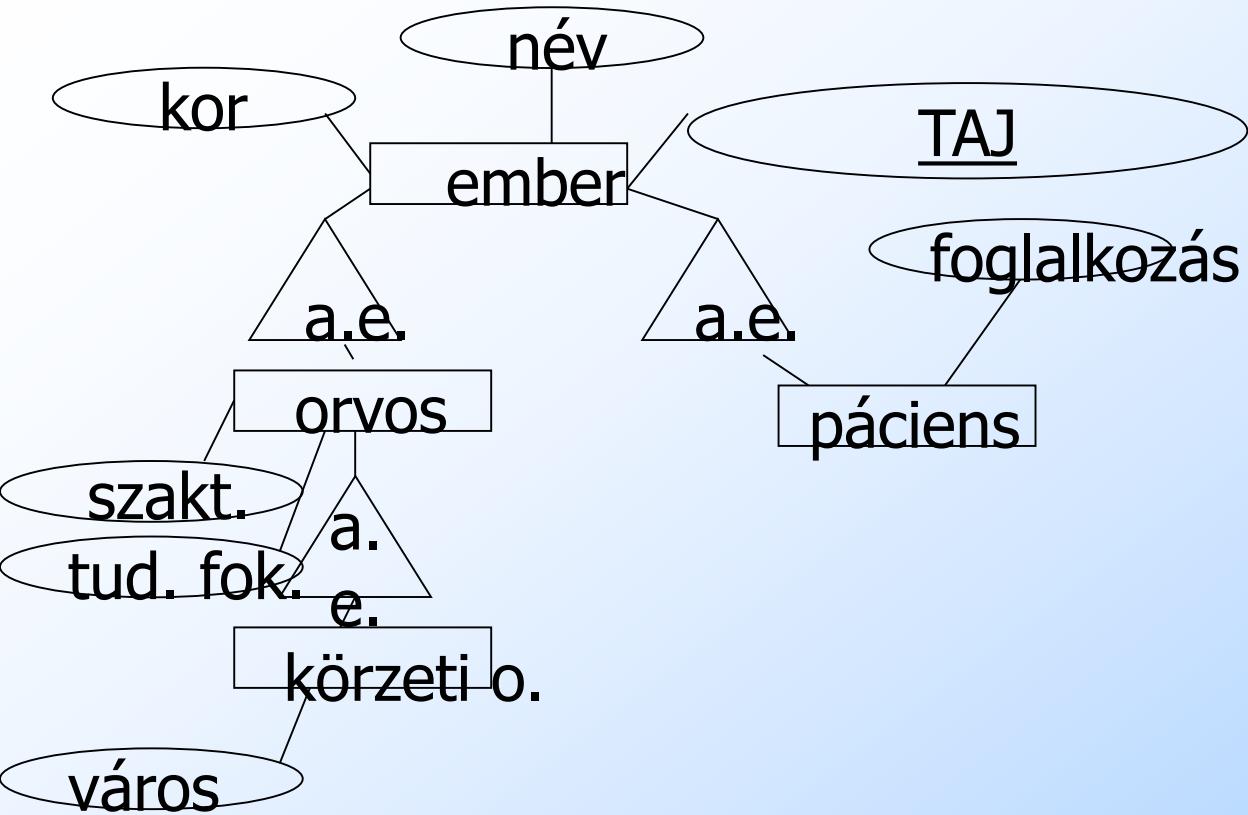
Feladat

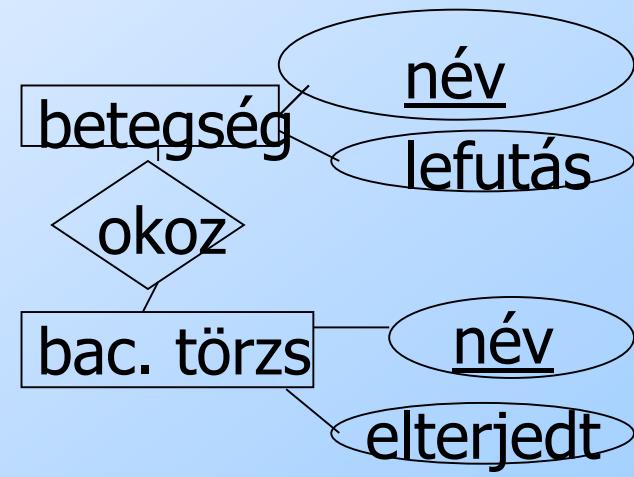
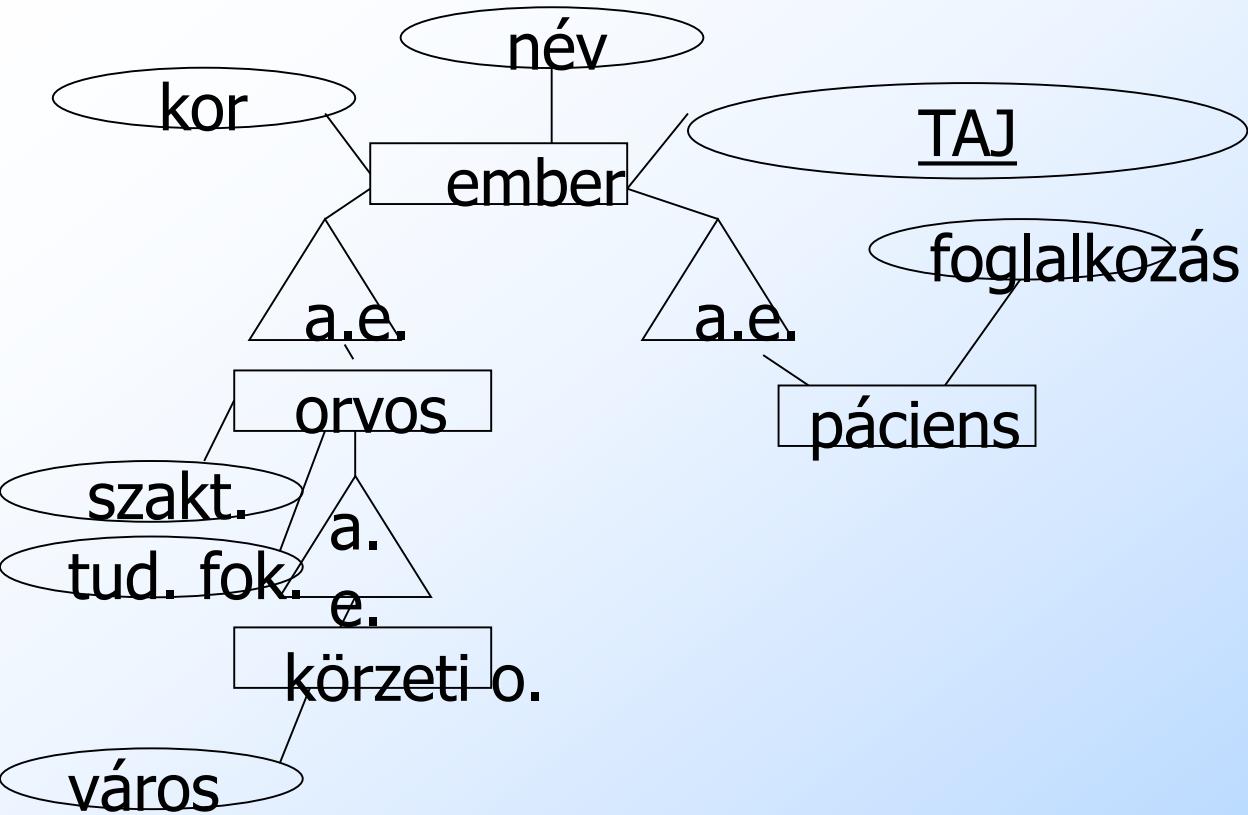
- ◆ Orvosi adatbázist készítünk. minden embernél számon tartjuk a nevét, korát, TAJ számát. Ezen utóbbi alapján egyértelműen azonosítani lehet bárkit. Az orvosoknál tároljuk ezeken kívül még a tudományos fokozatukat és a szakterületüket, a körzeti orvosoknál még annak a városnak a nevét is, ahol rendelnek, a pácienseknél pedig a foglalkozásukat. A betegségeknél számon tartjuk a nevüket és azt, hogy átlagosan mennyi ideig tart a gyógyulási folyamat. minden betegséget valamilyen bacilustörzs okoz. Ezek a nevük alapján egyértelműek, emellett tároljuk az elterjedtségük arányát. A törzsekhez bacillusok tartoznak, ám a név alapján még nem tudhatjuk pontosan melyik bacilusról van szó, mert több törzshöz is tartozhat ugyanolyan nevű bacillus. A különféle betegségekben szenvedő pácienseknek lehet, hogy egyszerre több orvos írja majd fel a különféle árú gyógyszereket. A gyógyszereknél tároljuk a nevüket, és hogy mely bacillusok esetében hatásosak. minden páciensnek tartoznia kell egy körzeti orvoshoz, s valaki csak úgy lehet körzeti orvos, ha legalább húsz páciense van.

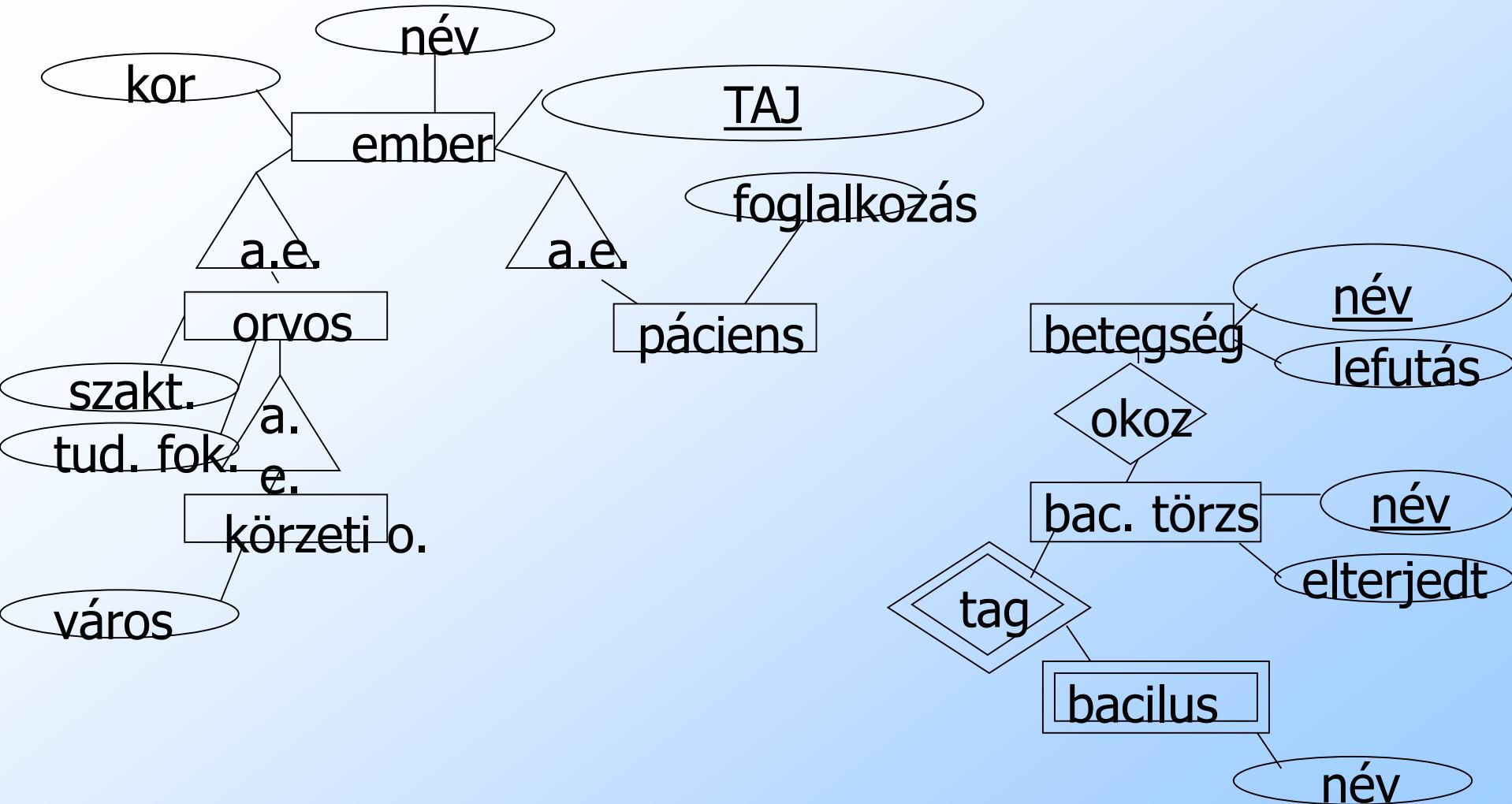


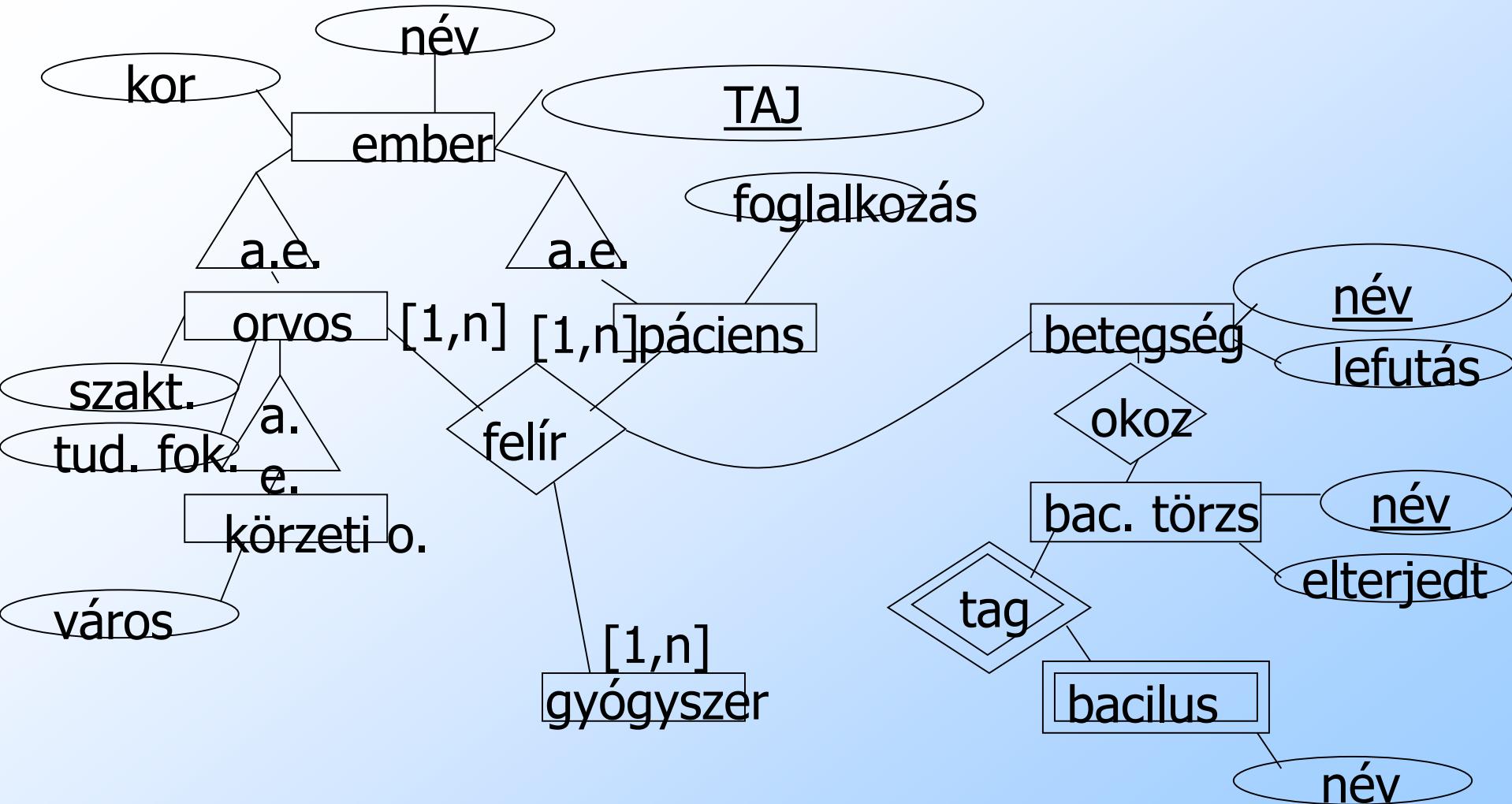


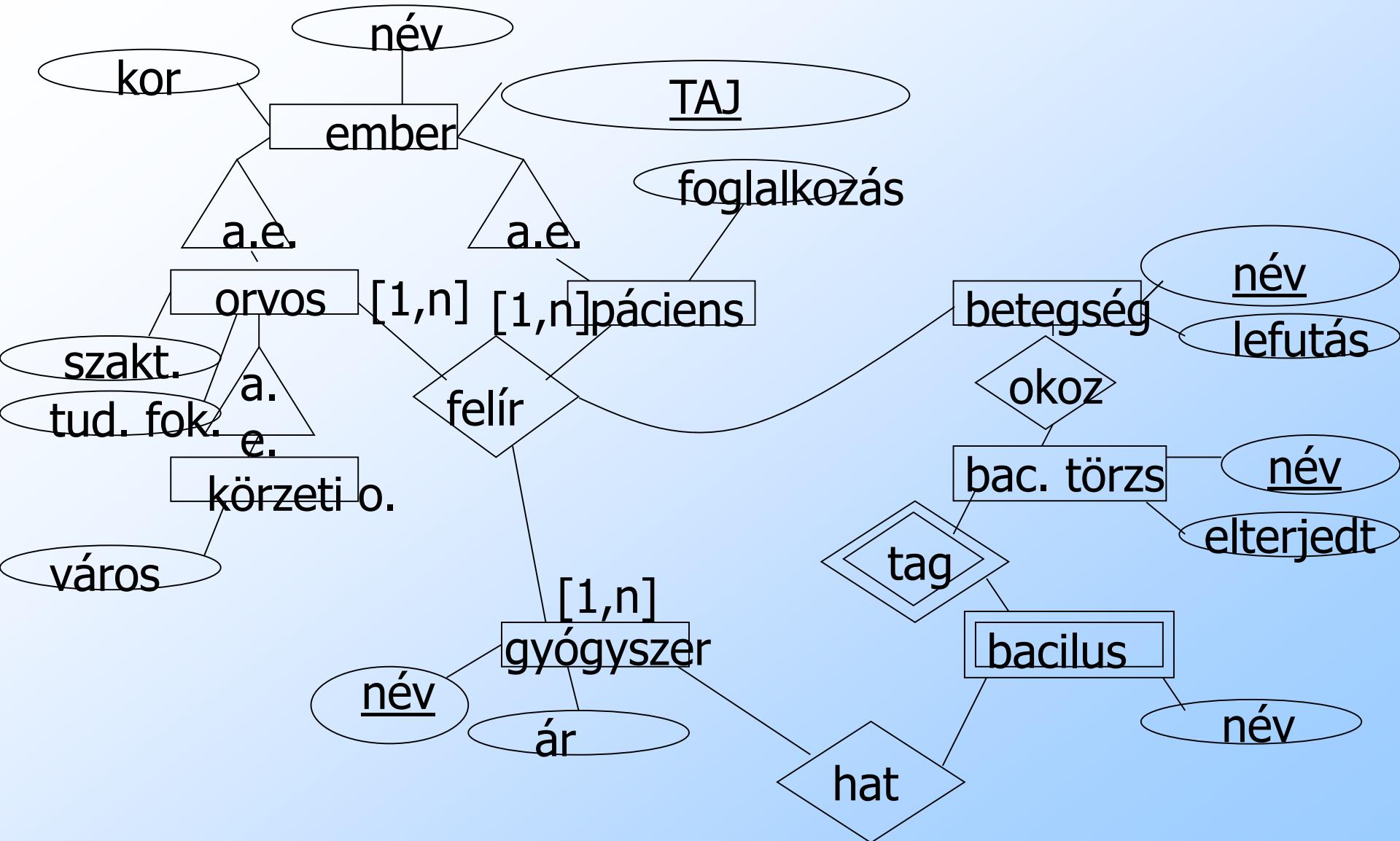


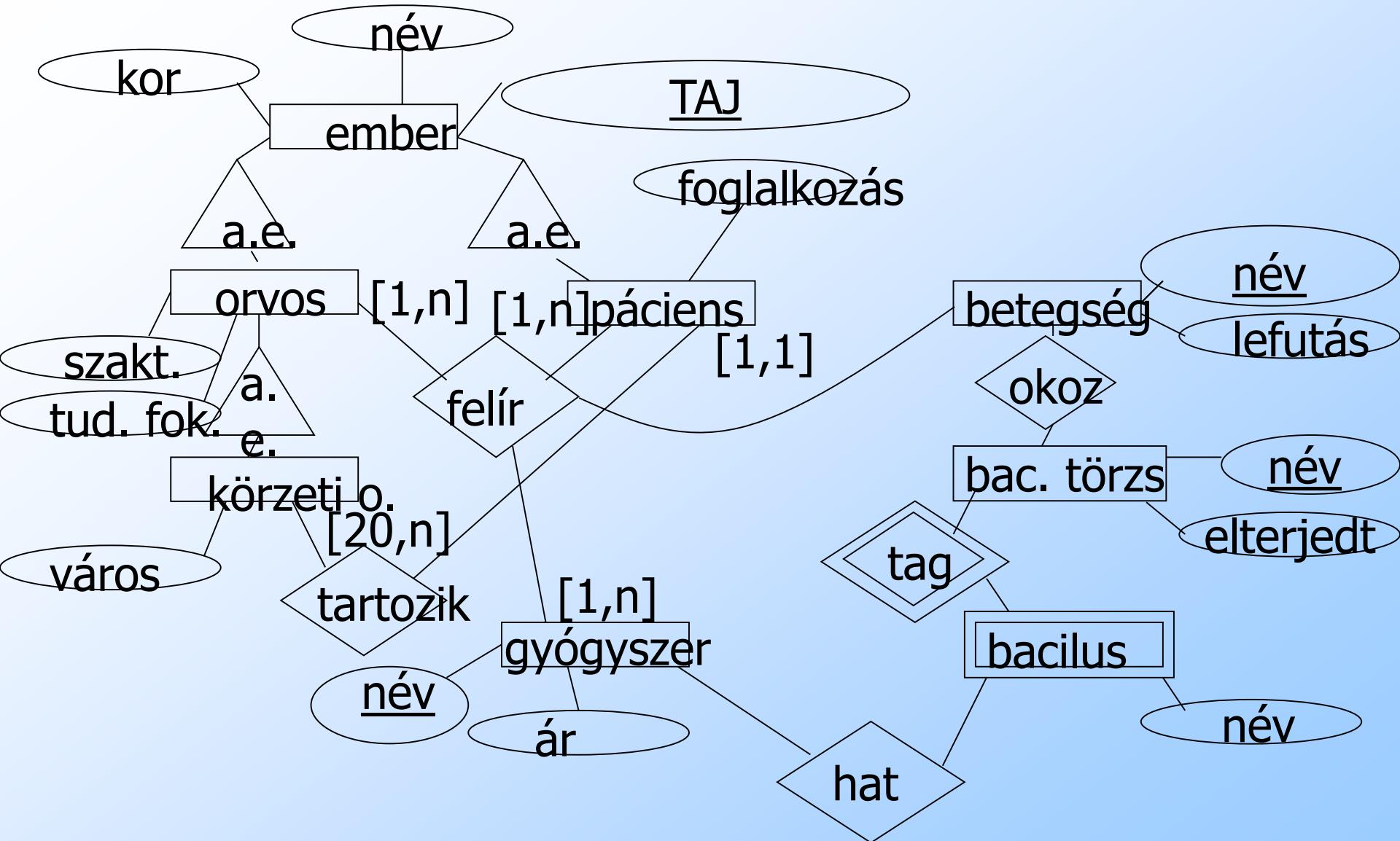












Adatbázis-séma az összevonások előtt

- ◆ Ember (név, kor, TAJ)
- ◆ Orvos (TAJ, szakt., tud_fok)
- ◆ Körzeti orvos (TAJ, város)
- ◆ Páciens (TAJ, foglalkozás)
- ◆ Gyógyszer (név, ár)
- ◆ Betegség (név, lefutás)
- ◆ Bacillus törzs (név, elterjedtség)
- ◆ Bacillus (név, törzs_név)
- ◆ Felír (orvos_TAJ, páciens_TAJ, gy_név, betegség_név)
- ◆ Tartozik (orvos_TAJ, páciens_TAJ)
- ◆ Hatásos (gy_név, bac_név, törzs_név)
- ◆ Okoz (bet_név, törzs_név)

Egyszerűsítés

- ◆ Ember (név, kor, TAJ)
- ◆ Orvos (TAJ, szakt., tud_fok)
- ◆ Körzeti orvos (TAJ, város)
- ◆ Páciens (TAJ, foglalkozás, orvos_TAJ)
- ◆ Gyógyszer (ár, név)
- ◆ Betegség (név, lefutás, törzs_név)
- ◆ Bacillus törzs (név, elterjedtség)
- ◆ Bacillus (név, törzs_név, gy_név) (esetleg)
- ◆ Felír (orvos_TAJ, páciens_TAJ, gy_név, betegség_név)

Tranzakciók, nézettáblák, indexek

Párhuzamos folyamatok irányítása
Virtuális és materializált
nézettáblák
Az adathozzáférés felgyorsítása

Miért van szükség tranzakciókra?

- ◆ Az adatbázis rendszereket általában több felhasználó és folyamat használja egyidőben.
 - ◆ Lekérdezések és módosítások egyaránt történhetnek.
- ◆ Az operációs rendszerektől eltérően, amelyek *támogatják* folyamatok interakcióját, az *ab* rendszereknek kell különíteniük a folyamatokat.

Példa: rossz interakció

- ◆ Te és egy barátod egy időben töltötök fel 100 dollárt ugyanarra a számlára ATM-en keresztül.
 - ◆ Az ab rendszernek biztosítania kell, hogy egyik művelet se vesszen el.
- ◆ **Ezzel szemben** az operációs rendszerek megengedik, hogy egy dokumentumot ketten szerkesszenek egy időben. Ha mind a ketten írnak, akkor az egyik változtatás elvész (elveszhet).

Tranzakciók

- ◆ *Tranzakció* = olyan folyamat, ami adatbázis lekérdezéseket, módosításokat tartalmaz.
- ◆ Az utasítások egy „értelmes egészt” alkotnak.
- ◆ Egyetlen utasítást tartalmaznak, vagy az SQL-ben explicit módon megadhatóak.

ACID tranzakciók

◆ Az ACID tranzakciók:

- ◆ *Atomiság (atomicity)*: vagy az összes vagy egy utasítás sem hajtódik végre.
- ◆ *Konzisztencia (consistency)*: az adatbázis megszorítások megőrződnek.
- ◆ *Elkülönítés (isolation)*: a felhasználók számára úgy tűnik, mintha folyamatok, elkülönítve, egymás után futnának le.
- ◆ *Tartósság (durability)*: egy befejeződött tranzakció módosításai nem vesznek el.

◆ Opcionálisan: gyengébb feltételek is megadhatóak.

COMMIT

- ◆ A COMMIT SQL utasítás végrehajtása után a tranzakció vélegesnek tekinthető.
 - ◆ A tranzakció módosításai vélegesítődnek.

ROLLBACK

- ◆ A ROLLBACK SQL utasítás esetén a tranzakció *abortál*.
 - ◆ Azaz az összes utasítás visszagörgetésre kerül.
- ◆ A 0-val való osztás vagy egyéb hibák, szintén visszagörgetést okozhatnak, akkor is, ha a programozó erre nem adott explicit utasítást.

Példa: egymásra ható folyamatok

- ◆ A **Felszolgál**(kocsma, sör, ár) táblánál tegyük fel, hogy Joe bárjában csak Bud és Miller söröket kaphatók 2.50 és 3.00 dollárért.
- ◆ Sally a **Felszolgál** táblából Joe legolcsóbb és legdrágább sörét kérdezi le.
- ◆ Joe viszont úgy dönt, hogy a Bud és Miller söröket helyett ezentúl Heinekent árul 3.50 dollárért.

Sally utasításai

- (max) SELECT MAX(ár) FROM Felszolgál
 WHERE kocsma = 'Joe bárja';
- (min) SELECT MIN(ár) FROM Felszolgál
 WHERE kocsma = 'Joe bárja';

Joe utasításai

- ◆ Ugyanabban a pillanatban Joe a következő utasításokat adja ki:

(del) DELETE FROM Felszolgál
WHERE kocsma = 'Joe bárja';

(ins) INSERT INTO Felszolgál
VALUES('Joe bárja', 'Heineken', 3.50);

Átfedésben álló utasítások

- ◆ A **(max)** utasításnak a **(min)** előtt kell végrehajtódnia, hasonlóan **(del)** utasításnak az **(ins)** előtt, ettől eltekintve viszont nincsenek megszorítások a sorrendre vonatkozóan, ha Sally és Joe utasításait nem gyűjtjük egy-egy tranzakcióba.

Példa: egy furcsa átfedés

- ◆ Tételezzük fel a következő végrehajtási sorrendet: **(max)(del)(ins)(min)**.

Joe árai: {2.50,3.00}{2.50,3.00} {3.50}

Utasítás: (max) (del) (ins) (min)

Eredmény: 3.00 3.50

- ◆ Mit lát Sally? MAX < MIN!

A probléma megoldása tranzakciókkal

- ◆ Ha Sally utasításait, **(max)(min)**, egy tranzakcióba gyűjtjük, akkor az előbbi inkonzisztencia nem történhet meg.
- ◆ Joe árait ekkor egy adott időpontban látja.
 - ◆ Vagy a változtatások előtt vagy utánuk, vagy közben, de a MAX és a MIN ugyanazokból az árakból számolódik.

Egy másik hibaforrás: a visszagörgetés

- ◆ Tegyük fel, hogy Joe a **(del)(ins)** és utasításokat nem, mint tranzakció hajtja végre, utána viszont úgy dönt, jobb ha visszagörgeti a módosításokat.
- ◆ Ha Sally az **(ins)** után, de visszagörgetés előtt hajtatja végre a tranzakciót, olyan értéket kap, 3.50, ami nincs is benne az adatbázisban végül.

Megoldás

- ◆ A **(del)(ins)** és utasításokat Joe-nak is, mint tranzakciót kell végrehajtatnia, így a változtatások akkor válnak láthatóvá, ha tranzakció egy COMMIT utasítást hajt végre.
 - ◆ Ha a tranzakció ehelyett visszagörgetődik, akkor a hatásai sohasem válnak láthatóvá.

Elkülönítési szintek

- ◆ Az SQL négy *elkülönítési szintet* definiál, amelyek megmondják, hogy milyen interakciók engedélyezettek az egy időben végrehajtódó tranzakciók közt.
- ◆ Ezek közül egy szint ("sorbarendezhető") = ACID tranzakciók.
- ◆ minden *ab* rendszer a saját tetszése szerint implementálhatja a tranzakciókat.

Az elkülönítési szint megválasztása

◆ Az utasítás:

SET TRANSACTION ISOLATION LEVEL X

ahol X =

1. SERIALIZABLE
2. REPEATABLE READ
3. READ COMMITTED
4. READ UNCOMMITTED

Sorbarendezhető (Serializable) tranzakciók

- ◆ Ha Sally a **(max)(min)**, Joe a **(del)(ins)** tranzakciót hajtatja végre, és Sally tranzakciója SERIALIZABLE elkülönítési szinten fut, akkor az adatbázist vagy Joe módosításai előtt vagy után látja, a **(del)** és **(ins)** közötti állapotban sohasem.

Az elkülönítési szint személyes választás

- ◆ A döntésed csak azt mondja meg, hogy *te* hogy látod az adatbázist, és nem azt, hogy mások hogy látják azt.
- ◆ Példa: Ha Joe sorbarendezhető elkülönítési szintet használ, de Sally nem, akkor lehet, hogy Sally nem talál árakat Joe bárja mellett.
 - ◆ azaz, mintha Sally Joe tranzakciójának közepén futtathna a sajátját.

Read-Committed tranzakciók

- ◆ Ha Sally READ COMMITTED elkülönítési szintet választ, akkor csak kommitálás utáni adatot láthat, de nem feltétlenül mindenkor ugyanazt az adatot.
- ◆ Példa: READ COMMITTED mellett megengedett a **(max)(del)(ins)(min)** átfedés amennyiben Joe kommitál.
 - ◆ Sally legnagyobb megdöbbensére: MAX < MIN.

Repeatable-Read tranzakciók

- ◆ Hasonló a read-commited megszorításhoz. Itt, ha az adatot újra beolvassuk, akkor amit először láttunk, másodszor is látni fogjuk.
- ◆ De második és az azt követő beolvasások után akár *több* sort is láthatunk.

Példa: ismételhető olvasás

- ◆ Tegyük fel, hogy Sally REPEATABLE READ elkülönítési szintet választ, a végrehajtás sorrendje:
(max)(del)(ins)(min).
 - ◆ **(max)** a 2.50 és 3.00 dollár árakat látja.
 - ◆ **(min)** látja a 3.50 dollárt, de 2.50 és 3.00 árakat is látja, mert egy korábbi olvasáskor **(max)** már látta azokat.

Read Uncommitted

- ◆ A READ UNCOMMITTED elkülönítési szinten futó tranzakció akkor is láthatja az adatokat, amikor a változtatások még nem lettek véglegesítve („kommitolva”).
- ◆ Példa: Ha Sally a READ UNCOMMITTED szintet választja, akkor is láthatja a 3.50 dollárt, mint árat, ha később Joe abortálja a tranzakcióját.

Nézettáblák

- ◆ A *nézettábla* olyan reláció, amit tárolt táblák (*alaptáblák*) és más nézettáblák felhasználásával definiálunk.
- ◆ Kétféle létezik:
 1. *virtuális* = nem tárolódik az adatbázisban; csak a relációt megadó lekérdezés.
 2. *Materializált* = kiszámítódik, majd tárolásra kerül.

Nézettáblák létrehozása

- ◆ Deklaráció:
CREATE [MATERIALIZED] VIEW
 <név> AS <lekérdezés>;
- ◆ Alapesetben virtualizált nézettábla jön létre.

Példa: nézettábla definíció

- ◆ **Ihatja(alkesz, sör)** nézettáblában az alkeszek mellett azon söröket tároljuk, melyeket legalább egy kocsmában felszolgálnak az általa látogatottak közül:

```
CREATE VIEW Ihatja AS  
    SELECT alkesz, sör  
    FROM Látogat, Felszolgál  
    WHERE Látogat.kocsma =  
          Felszolgál.kocsma;
```

Példa: nézettáblákhoz való hozzáférés

- ◆ A nézettáblák ugyanúgy kérdezhetők le, mint az alaptáblák.
 - ◆ A nézettáblákon keresztül az alaptáblák néhány esetben módosíthatóak is, ha a rendszer a módosításokat át tudja vezetni.

◆ Példa lekérdezés:

```
SELECT söR FROM Ihatja  
WHERE alkesz = 'Sally';
```

Materializált nézettáblák

- ◆ **Probléma:** minden alkalommal, amikor az alaptáblák valamelyike változik, a materializált nézettábla frissítése is szükségessé válhat.
 - ◆ Ez viszont néha túl költséges.
- ◆ **Megoldás:** Periodikus frissítése a materializált nézettábláknak, amelyek egyébként „nem aktuálisak”.

Példa: levelezési lista

- ◆ A következő levelezési lista cs145-aut0708 valójában egy materializált nézettábla, ami a kurzusra beiratkozott hallgatókat tartalmazza.
- ◆ Ezt négyeszer frissítik egy nap.
 - ◆ A feliratkozás után közvetlen még nem feltétlen kapja meg az ember az akkor küldött emaileket.

Példa: adattárház

- ◆ Wal-Mart minden áruházának minden eladását egy adatbázisban tárolja.
- ◆ Éjszaka az új adatokkal frissítik az áruház lánc *adattárházát*, ami itt az eladások materializált nézeteiből áll.
- ◆ Az adattárházat aztán elemzők használják, hogy trendeket figyeljenek meg és odamozgassák az árukat, ahol azok a legjobb áron értékesíthetők.

Indexek

- ◆ *Index* = olyan adatszerkezet, amivel egy-egy reláció sorait gyorsabban érhetjük el adott attribútumának, attribútumainak értéke, értékei alapján.
- ◆ Lehet hash tábla, de az *ab* rendszerekben a legtöbb esetben kiegyensúlyozott keresési fával valósítják meg (*B-fák*).

Indexek deklarálása

- ◆ Nincs standard megoldás!
- ◆ Tipikus szintaxis:

```
CREATE INDEX SörInd ON  
Sörök(gyártó);
```

```
CREATE INDEX EladásInd ON  
Felszolgál(kocsma, sör);
```

Indexek használata

- ◆ Adott v értékre az index azokhoz a sorokhoz irányít, ahol ez a v érték megjelenik a megfelelő attribútum(ok)nál.
- ◆ Példa: a SörInd és az EladásInd indexek segítségével megkeressük azokat a söröket, melyeket Pete gyárt és Joe árul. (következő dia)

Indexek használata --- (2)

```
SELECT ár FROM Sörök, Felszolgál  
WHERE gyártó = 'Pete' AND  
Sör.név = Felszolgál.sör AND  
kocsma = 'Joe bárja';
```

1. A SörInd segítségével megkapjuk azokat a söröket, melyeket Pete gyárt.
2. Aztán a EladásInd használatával a Joe bárjában felszolgált sörök árait kapjuk meg.

Adatbázisok hangolása

- ◆ Az adatbázisok hangolásánál komoly kérdést jelent annak eldöntése, hogy milyen indexeket használjanak.
- ◆ **Mellette:** az index felgyorsíthatja a lekérdezések végrehajtását.
- ◆ **Ellene:** a módosítások lassabbak lesznek, hiszen az indexeket is módosítani kell.

Példa: hangolás

- ◆ Tegyük fel, hogy a sörös adatbázisunkban a következők történhetnek:
 1. Új tények kerülnek egy relációba (10%).
 2. Adott kocsmára és sörre keressük az ottani árat (90%).
- ◆ Ekkor az **EladásInd** a Felszolgál(kocsma, sör) fölött nagyszerű szolgálatot tesz, a **SörInd** a Sörök(gyártó) fölött pedig inkább a kárunkra van.

Hangolási szakértők

- ◆ Fontos kutatási feladat.
 - ◆ A kézi hangolás nagy szakérteletet kíván.
- ◆ A szakértő először egy *lekérdezés terhelési kimutatást (query load)* kap kézhez:
 1. véletlenszerűen lekérdezéseket választanak a korábban végrehajtottak közül.
 2. A tervező átad egy mintát.

Hangolási szakértők --- (2)

- ◆ A szakértő létrehozza a szerinte fontos indexeket, majd megvizsgálja azok hatását.
 - ◆ minden minta lekérdezés esetén a lekérdezés optimalizálónak használnia kell az indexeket.
 - ◆ Így meg tudja mondani, hogy javult-e összességében a lekérdezések végrehajtási ideje.

XML

Document Type Definitions (DTD)
XML séma

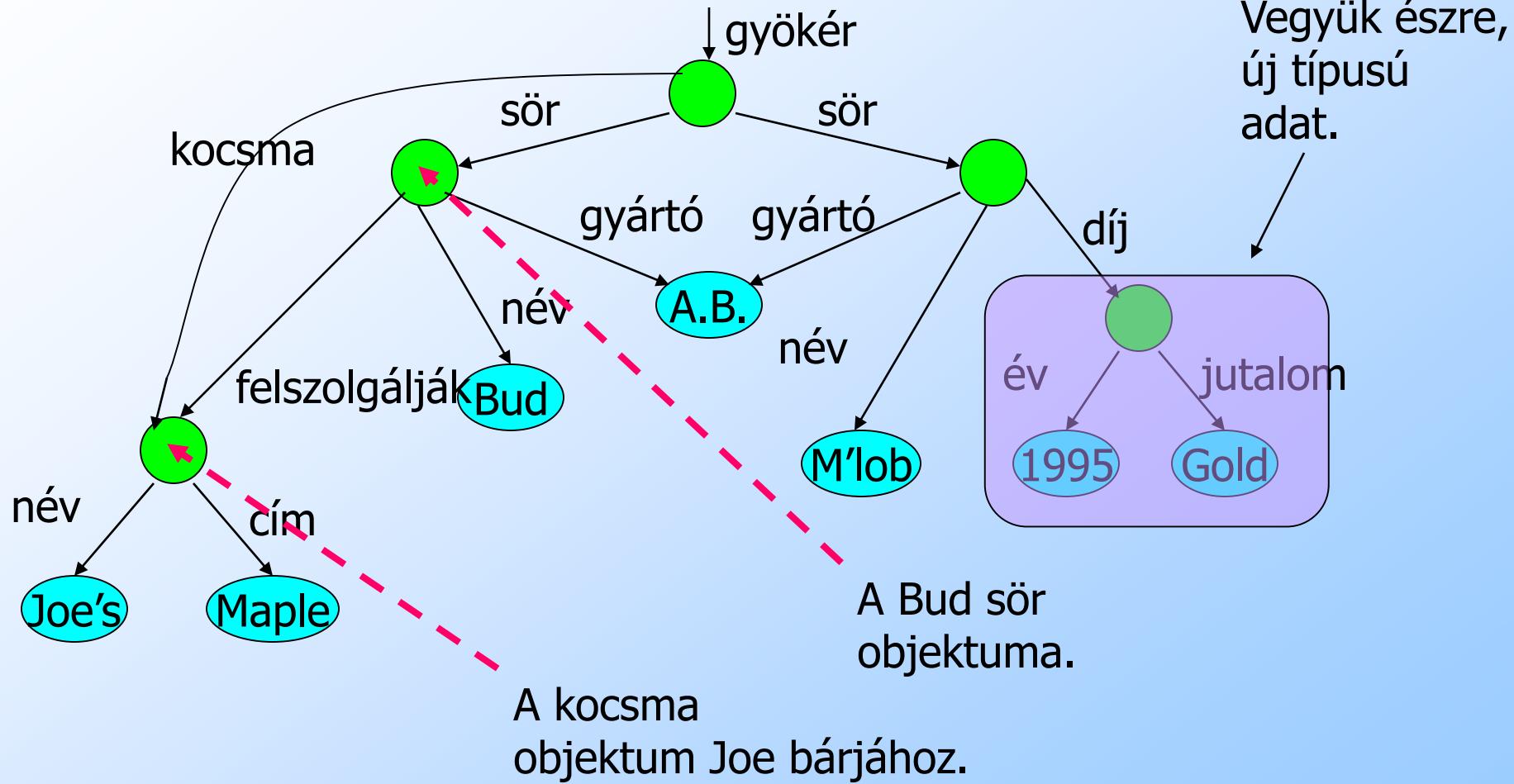
Féligstrukturált adat

- ◆ Egy másik, fákon alapuló adatmodell.
- ◆ Motiváció: az adatok rugalmas megjelenítése.
- ◆ Motiváció: *dokumentumok* megosztása rendszerek és adatbázisok között.

A félígystrukturált adatok gráfja

- ◆ Pontok = objektumok.
- ◆ Az élek címkéi tulajdonképpen az attribútumnevek, kapcsolatok.
- ◆ Az atomi értékek a levelekben tárolódnak.

Példa: adatgráf



XML

- ◆ XML = *Extensible Markup Language*.
- ◆ A HTML a dokumentumok kinézetét írja le (pl. *< i > rózsa </ i >* “italic”), az XML-ben a jelentés, szemantika leírására használják a tageket (pl. “ez egy cím”).

XML dokumentumok

- ◆ A dokumentum *deklarációval* kezdődik, amit egy speciális <?xml ... ?> tag határol.
- ◆ Tipikusan:

```
<?xml version = "1.0" encoding  
= "utf-8" ?>
```

```
<?xml version = "1.0"  
standalone = "yes" ?>
```

- ◆ “standalone” = “DTD-t csak validációra használjuk (yes) vagy nem (no, ez a default)”

Tagek

- ◆ A tagek, mint a HTML esetében is nyitó és záró elemből állnak <FOO> ... </FOO>.
 - ◆ Opcionálisan <FOO/>.
- ◆ Tageket tetszőlegesen egymásba ágyazhatjuk.
- ◆ Az XML tagek érzékenyek a kis- és nagybetű különbségre.

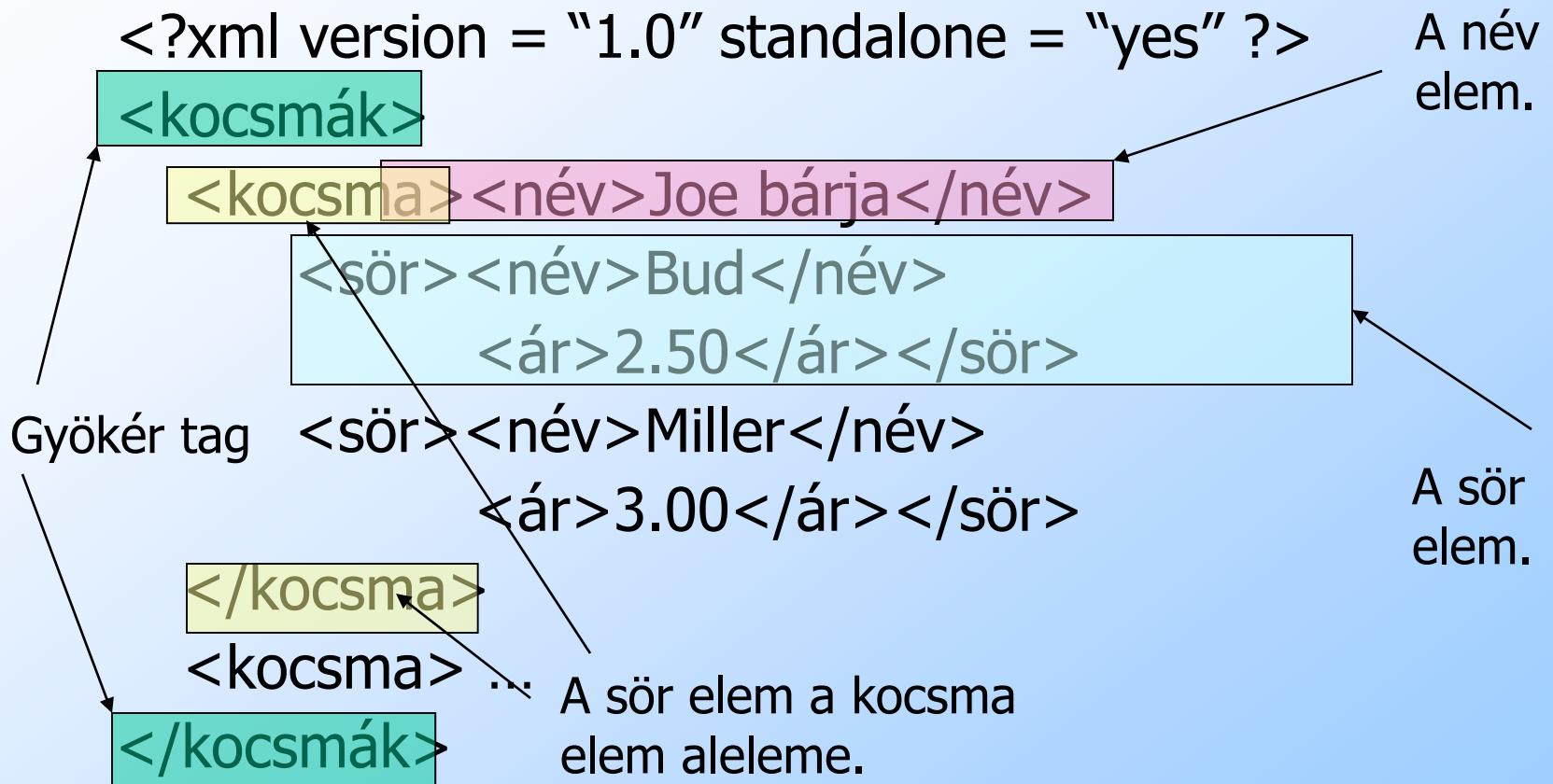
Jól formált és valid XML

- ◆ *Jól formált XML* megengedi, hogy önálló tageket vezessünk be.
- ◆ *Valid XML* illeszkedik egy előre megadott sémára: DTD vagy XML séma.

Jól formált XML

- ◆ Nem hiányzik a *deklaráció*: <?xml ... ?>.
- ◆ minden nyitó tagnek megvan a záró párja.

Példa: jól formázott XML



DTD felépítése

```
<!DOCTYPE gyökér elem [
    <!ELEMENT elem név(összetevők) >
    . . . további elemek . . .
] >
```

DTD ELEMENT

- ◆ Egy-egy elem leírása az elem nevét és zárójelek között az alelemek megadását jelenti.
 - ◆ Ez magában foglalja a alelemek sorrendjét és multiplicitását.
- ◆ A levelek (szöveges elemek) típusa #PCDATA (*Parsed Character DATA*).

Példa: DTD

```
<!DOCTYPE kocsmák [
```

```
    <!ELEMENT kocsmák (kocsma*)>
```

```
    <!ELEMENT kocsma (név, sör+)>
```

```
    <!ELEMENT név (#PCDATA)>
```

```
    <!ELEMENT sör (név, ár)>
```

```
    <!ELEMENT ár (#PCDATA)>
```

```
]>
```

A kocsmák elem 0 vagy több kocsma elemet tartalmazhat.

A kocsma elemnek van egy név eleme és egy vagy több sör eleme.

A sör elemnek neve és ára van.

A név és ár elemeknek nincsenek alelemek, szövegesek.

Elem leírások

- ◆ Az alelemek a felsorolás sorrendjében kell, hogy kövessék egymást.
- ◆ Egy elemet a felsorolásban egy további szimbólum követhet megadva a multiplicitását.
 - ◆ * = 0 vagy több.
 - ◆ + = 1 vagy több.
 - ◆ ? = 0 vagy 1.
- ◆ A | szimbólum jelentése: vagy.

Példa: elem leírás

- ◆ A névhez opcionálisan hozzátarozik egy titulus, egy kereszt- és vezetéknév (ebben a sorrendben), vagy egy IP cím:

```
<!ELEMENT név (  
    (titulus?, kereszt, vezeték) |  
    IP cím  
)>
```

Példa: DTD-k használata (a)

```
<?xml version = "1.0" standalone = "no" ?>
```

```
<!DOCTYPE kocsmák [  
    <!ELEMENT kocsmák (kocsma*)>  
    <!ELEMENT kocsma (név, sör+)>  
    <!ELEMENT név (#PCDATA)>  
    <!ELEMENT sör (név, ár)>  
    <!ELEMENT ár (#PCDATA)>  
>]
```

a DTD

```
<kocsmák>  
    <kocsma><név>Joe bárja</kocsma>  
        <sör><név>Bud</név> <ár>2.50</ár></sör>  
        <sör><név>Miller</név> <ár>3.00</ár></sör>  
    </kocsma>  
    <kocsma> ...  
</kocsmák>
```

a dokumentum

Példa: DTD-k használata (b)

- ◆ Feltesszük, hogy a kocsmák DTD a bar.dtd fájlban van.

```
<?xml version = "1.0" standalone = "no" ?>
<!DOCTYPE kocsmák SYSTEM "bar.dtd">
<kocsmák>
    <kocsma><név>Joe bárja</név>
        <sör><név>Bud</név>
            <ár>2.50</ár></sör>
        <sör><név>Miller</név>
            <ár>3.00</ár></sör>
    </kocsma>
    <kocsma> ...
</kocsmák>
```

A DTD-t
bar.dtd fájlból
nyeri.

Attribútumok

- ◆ Az XML-ben a nyitó tag-ek mellett szerepelhetnek *attribútumok*.

- ◆ Egy DTD-ben,

```
<!ATTLIST E . . . >
```

Az *E* taghez tartozó attribútumokat adja meg.

Példa: attribútumok

- ◆ A kocsma elemhez hozzátarozik egy típus **attribútum**.

```
<!ELEMENT kocsma (név sört*)>  
<!ATTLIST kocsma típus CDATA  
      #IMPLIED>
```

Az attribútum opcionális,
ennek ellentéte: #REQUIRED

Sztring.

Példa: attribútum használatára

```
<kocsma típus = "skót">
  <név>McGuiver sörozője</név>
  <sör><név>Scottish ale</név>
    <ár>5.00</ár></sör>
  ...
</ kocsma>
```

ID és IDREF attribútumok

- ◆ Az attribútumok segítségével egyik elemről a másikra mutathatunk.
- ◆ Emiatt az XML dokumentum reprezentációja inkább gráf, mintsem egyszerű fa (fenntartásokkal kell fogadni ezt az információt).

ID-k létrehozatala

- ◆ Adjuk meg egy E elemet és egy A attribútumát, aminek típusa: ID.
- ◆ Amikor az E elemet ($<E>$) egy XML dokumentumba használjuk, az A attribútumnak egy máshol nem szereplő értéket kell adnunk.
- ◆ Példa:

```
<E A = "xyz">
```

IDREF-ek létrehozatala

- ◆ Egy F elem az IDREF attribútum segítségével hivatkozhat egy másik elemre annak ID attribútumán keresztül.
- ◆ Vagy: az IDREFS típusú attribútummal több másik elemre is hivatkozhat.

A DTD

```
<!DOCTYPE kocsmák [
    <!ELEMENT kocsmák (kocsma*, sör*)>
    <!ELEMENT kocsma (felszolgál+)>
        <!ATTLIST kocsma név ID #REQUIRED>
    <!ELEMENT felszolgál (#PCDATA)>
        <!ATTLIST felszolgál melyikSör IDREF #REQUIRED>
    <!ELEMENT sör EMPTY>
        <!ATTLIST sör név ID #REQUIRED>
        <!ATTLIST sör kapható IDREFS #IMPLIED>
]>
```

lásd
később

A kocsma elemek neve ID attribútum, emellett felszolgál alelemeik lehetnek.

A felszolgál elemhez sörökre vonatkozó hivatkozások tartoznak.

A sör elemeknek is van egy ID típusú attribútuma (név), a kapható attribútum pedig kocsmák nevét tartalmazza.

Példa: a DTD-re illeszkedő dokumentum

```
<kocsmák>
  <kocsma név = "Joe bárja">
    <felszolgál melyikSör = "Bud">2.50</felszolgál>
    <felszolgál melyikSör = "Miller">3.00
    </felszolgál>
  </kocsma> ...
  <sör név = "Bud" kapható = "Joe bárja
    Suzy bárja ..." /> ...
</kocsmák>
```

Példa: üres elem

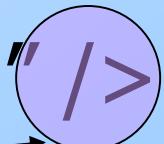
◆ A DTD:

```
<!ELEMENT felszolgál EMPTY>
<!ATTLIST felszolgál melyikSör IDREF
#REQUIRED>
<!ATTLIST felszolgál ár CDATA #REQUIRED>
```

◆ Példa a használatra:

```
<felszolgál melyikSör = "Bud" ár = "2.50" />
```

a záró tag itt lerövidül



XML séma

- ◆ Az XML sémák segítségével szintén XML dokumentumok szerkezetét adhatjuk meg. Itt több megszorítást lehet előírni, mint a DTD-k esetén.
- ◆ Az XML séma maga is egy XML dokumentum.

Az XML séma dokumentum szerkezete

```
<? xml version = ... ?>  
<xs:schema xmlns:xs =  
    "http://www.w3.org/2001/XMLSchema">  
    . . .  
</xs:schema>
```

A schema tehát az xs névtérhez tartozó elem.

Az "xs" *névteret* adja meg, amit a megadott URL azonosít. Itt a névtérhez tartozó elemek leírása is megtalálható.

Az xs:element elem

- ◆ Az attribútumai:
 1. **name** = a definiált elem neve (tagben miként szerepel).
 2. **type** = az elem típusa.
 - ◆ Lehet XML séma típus, pl. xs:string.
 - ◆ Vagy egy olyan típus, amit az adott XML sémában deklarálunk.

Példa: xs:element

```
<xs:element name = "név"  
           type = "xs:string" />
```

◆ A következő elemet írja le:

```
<név>Joe bárja</név>
```

EMPTY

Séma definíció szinten üres
Dokumentumban <tag> között

Összetett típusok

- ◆ alelemeket tartalmazó elemek leírásához **xs:complexType** használjuk.
 - ◆ A **name** attribútummal nevet adhatunk ennek a típusnak.
- ◆ Az **xs:complexType** egy tipikus aleleme az **xs:sequence**, amihez **xs:element** elemek egy sorozata tartozik.
 - ◆ A **minOccurs** és **maxOccurs** attribútumok használatával az adott **xs:element** előfordulásainak számát korlátozhatjuk.

Példa: típus a sörökhöz

```
<xs:complexType name = "sörTípus">
  <xs:sequence>
    <xs:element name = "név"
      type = "xs:string"
      minOccurs = "1" maxOccurs = "1" />
    <xs:element name = "ár"
      type = "xs:float"
      minOccurs = "0" maxOccurs = "1" />
  </xs:sequence>
</xs:complexType>
```

Pontosan egy előfordulás.

Mint a ? a DTD-ben.

Egy sör típusú elem

<xxx>

<név>Bud</név>

<ár>2.50</ár>

</xxx>

Nem ismerjük az ilyen típusú
elem nevét.

Példa: típus a kocsmákhoz

```
<xs:complexType name = "kocsmaTípus">
  <xs:sequence>
    <xs:element name = "név"
      type = "xs:string"
      minOccurs = "1" maxOccurs = "1" />
    <xs:element name = "sör"
      type = "sörTípus"
      minOccurs = "0" maxOccurs =
      "unbounded" />
  </xs:sequence>
</xs:complexType>
```

Mint a * a
DTD-ben.

xs:attribute

- ◆ xs:attribute elemek használatával az összetett típuson belül a típushoz tartozó elemek attribútumait adhatjuk meg.
- ◆ Az xs:attribute elem attribútumai:
 - ◆ name és type mint az xs:element esetén.
 - ◆ use = "required" vagy "optional".

Példa: xs:attribute

```
<xs:complexType név = "sörTípus">
  <xs:attribute name = "név"
    type = "xs:string"
    use = "required" />
  <xs:attribute name = "price"
    type = "xs:float"
    use = "optional" />
</xs:complexType>
```

Az új sörTípusnak megfelelő elem

```
<xxx név = "Bud"  
     ár = "2.50" />
```

Továbbra sem tudjuk az elem nevét.

Üres elemmel van dolgunk, mert nincsenek alelemei.

Egyszerű típus megszorítások

- ◆ Az **xs:simpleType** segítségével felsorolásokat adhatunk meg és az alaptípusra vonatkozó megszorításokat.
- ◆ **name** attribútuma van és
- ◆ **xs:restriction** alelemme.

Megszorítások: xs:restriction

- ◆ A **base** attribútum adja meg, hogy melyik egyszerű típusra (simple type) vonatkozik a megszorítás: pl. **xs:integer**.
- ◆ **xs:{min, max}{Inclusive, Exclusive}** a négy attribútum használatával alsó és felső korlátokat adhatunk meg.
- ◆ **xs:enumeration** alelem, a **value** attribútuma után megadhatjuk a felsorolás elemeit.

Példa: engedély attribútum a kocsmákhoz

```
<xs:simpleType name = "engedély">
  <xs:restriction base = "xs:string">
    <xs:enumeration value = " minden" />
    <xs:enumeration value = " csak sör" />
    <xs:enumeration value = " csak bor" />
  </xs:restriction>
</xs:simpleType>
```

Példa: az árak [1,5) intervallumba eshetnek

```
<xs:simpleType name = "ár">  
  <xs:restriction  
    base = "xs:float"  
    minInclusive = "1.00"  
    maxExclusive = "5.00" />  
</xs:simpleType>
```

Kulcsok az XML sémában

- ◆ Az **xs:element** elemhez tartozhat **xs:key** alelem.
- ◆ **Jelentése:** ezen az elemen belül, minden elem, ami egy adott *szelektor* ösvényen keresztül elérhető egyedi értékekkel kell, hogy rendelkezzen a megadott mezőinek (*field*) kombinációin (alelem, attribútum).
- ◆ **Példa:** egy kocsmák elemen belül a kocsma elemek **név** attribútumának egyedinek kell lennie.

Példa: kulcs

```
<xss:element name = "kocsmák" ...>
```

A @
attribútumot
jelöl.

• • •

```
<xss:key name = "kocsmaKulcs">
```

```
    <xss:selector xpath = "kocsma"/>
```

```
    <xss:field xpath = "@név" />
```

```
</xss:key>
```

• • •

```
</xss:element>
```

Az XPath segítségével
az XML fákat járhatjuk be
(következő óra).

Idegen kulcsok

- ◆ A **xs:keyref** alelem az **xs:element** elemen belül előírja, hogy ezen az elemen belül bizonyos értékek, melyeket ugyanúgy a szelektor és mezők használatával adhatunk meg, egy kulcs értékei között kell, hogy szerepeljenek.

Példa: idegen kulcs

- ◆ Tegyük fel, hogy a név alelem kulcs a kocsma elemekre vonatkozóan.
 - ◆ A kulcs neve legyen *kocsmaKulcs*.
- ◆ Az alkeszek elemhez *látogat* alelemeket szeretnénk hozzáadni. Ezen elemek *kocsma* attribútuma *idegen kulcs* lesz, a *kocsma* elem *név* alelemére hivatkozik.

Példa: idegen kulcs az XML sémában

```
<xs:element name = "alkeszek"  
    . . .  
<xs:keyref name = "kocsmaRef"  
    refers = "kocsmaKulcs">  
    <xs:selector xpath =  
        "alkeszek/látogat" />  
    <xs:field xpath = "@kocsma" />  
</xs:keyref>  
</xs:element>
```

Framework - Keretrendszer

1. *Information Integration* : Making databases from various places work as one.
 2. *Semistructured Data* : A new data model designed to cope with problems of information integration.
 3. *XML* : A standard language for describing semistructured data schemas and representing data.
-
1. *Információ integráció*: A különböző helyekről származó adatbázisokat úgy üzemeltetni, mintha egységes egészet alkotnának.
 2. *Félig-strukturált adat*: Viszonylag új adatmodell, amely segít megbirkózni az adatintegráció problémájával.
 3. *XML*: Szabványos nyelv a félig-strukturált adatok leírására.

The Information-Integration Problem

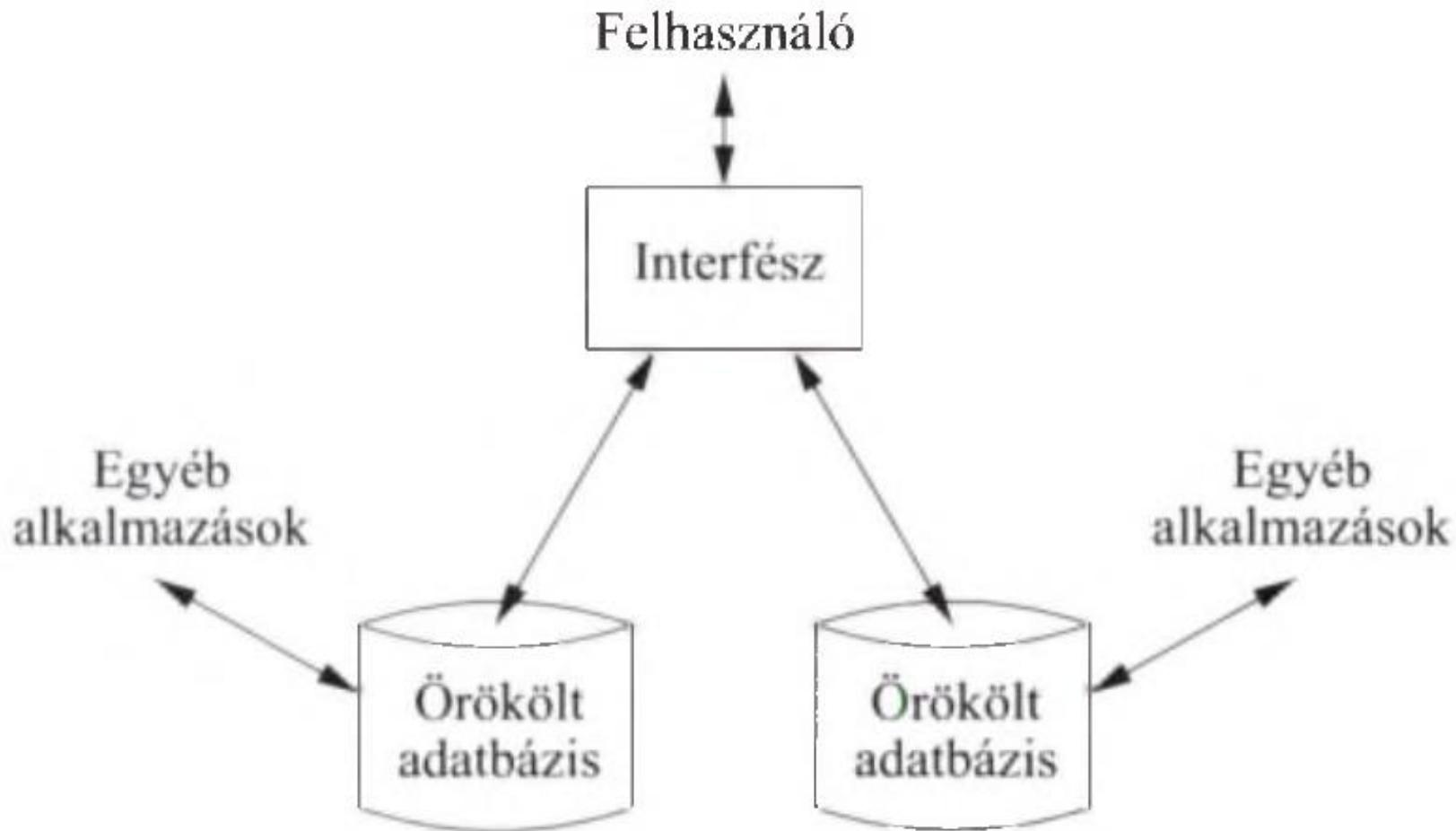
- ◆ Related data exists in many places and could, in principle, work together.
- ◆ But different databases differ in:
 1. Model (relational, object-oriented?).
 2. Schema (normalized/unnormalized?).
 3. Terminology: are consultants employees? Retirees? Subcontractors?
 4. Conventions (meters versus feet?).
- ◆ Egymáshoz kapcsolható adat elemek sok helyen léteznek és elvileg együtt működésre alkalmasak volnának:
- ◆ De a különböző adatbázisok több tekintetben különböznek:
 1. Adatbázis modellek (relációs, objektum-orientált, NoSQL, dokumentum stb.)
 2. Séma (normalizált, nem normalizált)
 3. Szakkifejezések: tanácsadó alkalmazott-e? Visszavonult nyugdíjas-e? Alvállalkozó?
 4. Konvenciók (méter kontra láb [metrikus (SI, CGI), birodalmi]).

Example



- ◆ Every bar has a database.
 - ◆ One may use a relational DBMS; another keeps the menu in an MS-Word document.
 - ◆ One stores the phones of distributors, another does not.
 - ◆ One distinguishes ales from other beers, another doesn't.
 - ◆ One counts beer inventory by bottles, another by cases.

- ◆ Mindegyik kocsmának van adatbázisa:
 - ◆ Az egyik relációs adatbáziskezelőt használ; másik MS-Word-ben tartja nyilván a menüt.
 - ◆ Az egyik nyilvántartja a mobil telefonok forgalmazóit, a másik nem.
 - ◆ Az egyik megkülönbözteti az angol barna sört a többi söröttl, a másik nem.
 - ◆ Az egyik a leltárban a söröket palackokként tartja nyilván, másik a göngyöleg egységei alapján.



Two Approaches to Integration



1. *Warehousing* : Make copies of the data sources at a central site and transform it to a common schema.

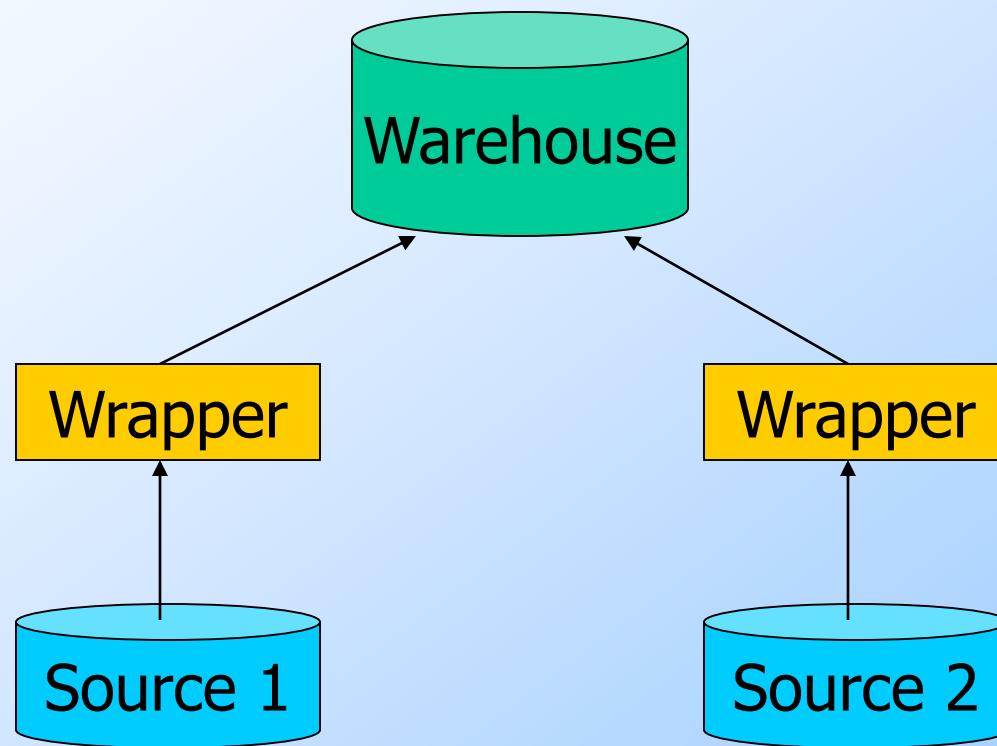
 - ◆ Reconstruct data daily/weekly, but do not try to keep it more up-to-date than that.
 2. *Mediation* : Create a view of all sources, as if they were integrated.

 - ◆ Answer a view query by translating it to terminology of the sources and querying them.
1. Adattárház: Az adatforrásokról egy központi másolatot készít, és egy közös adat sémává transzformálja.

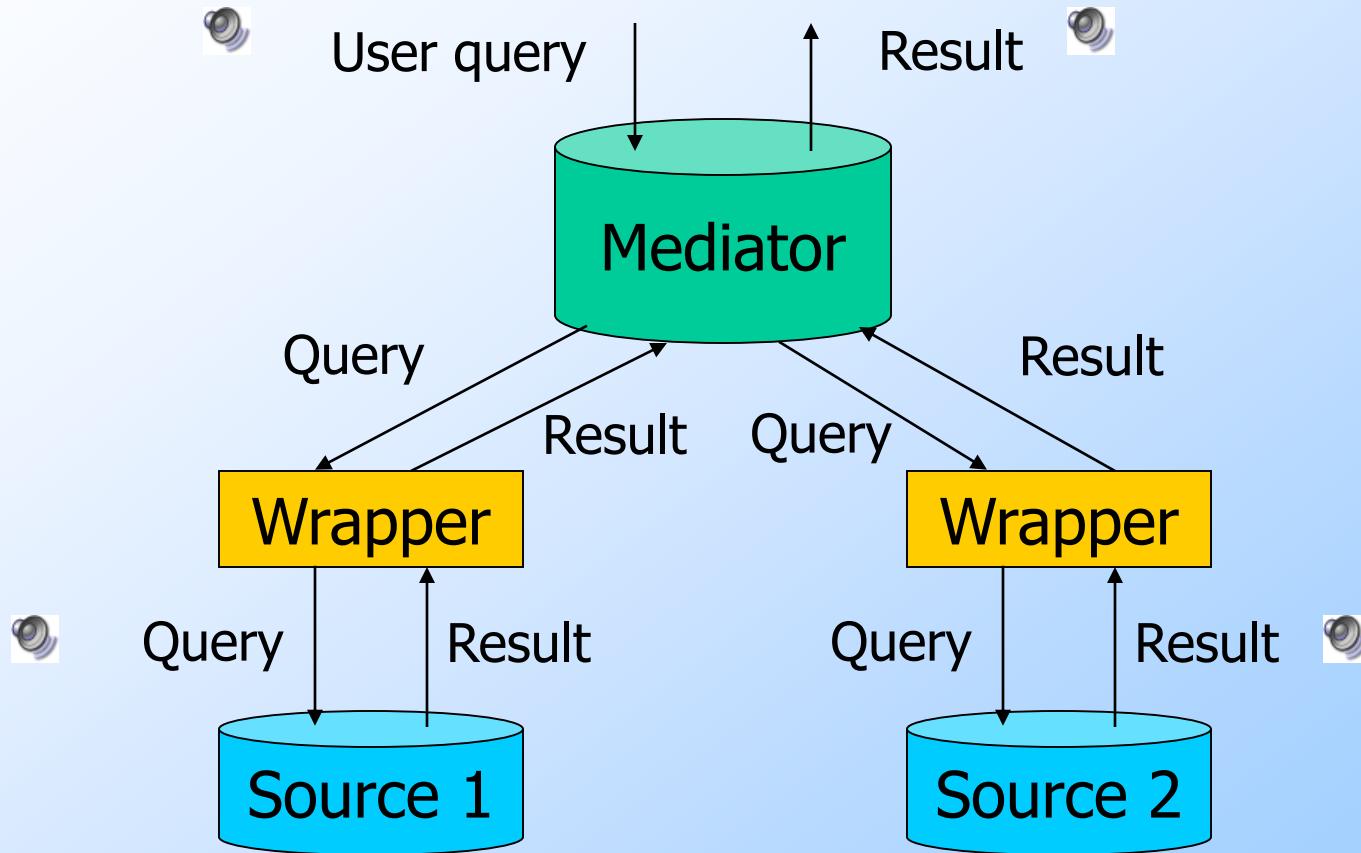
 - Az adatokat naponta, hetente frissítik, de ennél nem szabad nagyobb pontosságot megcélozni.
 2. Mediáció, közvetítés: Az összes adatforrásra egy nézetet kell létrehozni, mintha egy integrált rendszer részei volnának:

 - A nézetre vonatkozó lekérdezést úgy lehet megválaszolni, hogy a lekérdezést az egyes adatforrások szakkifejezéseire fordítják le és azután kérdezik le az eredeti adatforrásokat

Warehouse Diagram



A Mediator



XML lekérdezőnyelvek

XPath
XQuery

Az XPath/XQuery adatmodell

- ◆ A relációk megfelelője ebben a környezetben a *tételek (item) listája (sequence)*.
- ◆ Ez azt jelenti, hogy a bemenetet, a köztes lépések eredményeit és a végeredményt is tételek listáként kezeljük.
- ◆ Egy *tétel* lehet:
 1. egyszerű érték, pl.: egész vagy sztring.
 2. Csomópont.

A csomópontok fő típusai

1. A *dokumentum csomópontok* a teljes dokumentumot reprezentálják.
2. *Elem csomópontok*: a tagek (jelölők) és a közöttük lévő dokumentumrészlet.
3. *Attribútumok*: a nyitó tagekben szerepelnek és ott kapnak értéket.

Öt téTEL szekvenciája

10

"tíz"

10.0

<Számrendszer bázis = "8">

 <Digit>1</Digit>

 <Digit>2</Digit>

 </Számrendszer>

 @val="10"

- ◆ Az első három téTEL egyszerű típusú (egész, szöveg, valós sz.)
- ◆ A negyedik elem csomópont típus
- ◆ Az utolsó egy attribútum csomópont típus

Dokumentum csomópont

- ◆ A doc(URL) vagy document(URL) parancs hatására jön létre.
- ◆ Példa: doc(<http://abc.com/sales.xml>)
- ◆ minden XPath (és XQuery) lekérdezés hivatkozik egy dokumentum csomópontra.
 - ◆ Példa: az XML Sémában szereplő XPath kifejezések arra a dokumentumra hivatkoznak, amelyre a séma éppen vonatkozik.

A használt példa DTD-je

```
<!DOCTYPE kocsmák [  
    <!ELEMENT kocsmák (kocsma*, sör*)>  
    <!ELEMENT kocsma (ár+)>  
        <!ATTLIST kocsma név ID #REQUIRED>  
    <!ELEMENT ár (#PCDATA)>  
        <!ATTLIST ár melyikSör IDREF #REQUIRED>  
    <!ELEMENT sör EMPTY>  
        <!ATTLIST sör név ID #REQUIRED>  
        <!ATTLIST sör árulja IDREFS #IMPLIED>  
]>
```

Példa dokumentum

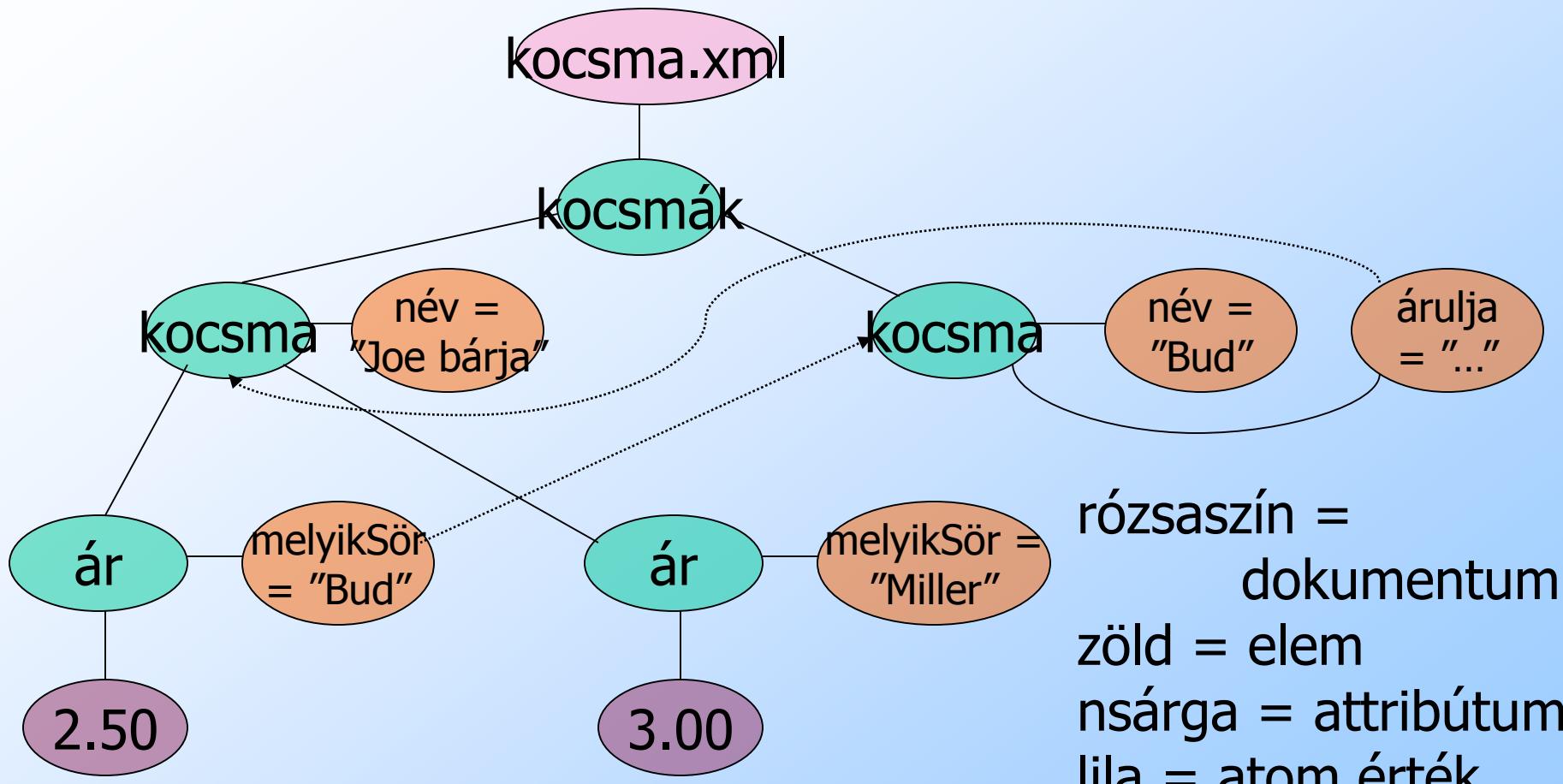
```
<kocsmák>
  <kocsma név = "Joe bárja">
    <ár melyikSör = "Bud">2.50</ár>
    <ár melyikSör = "Miller">3.00</ár>
  </kocsma> ...
<sör név = "Bud" árulja = "Joe bárja
  Sue bárja ... "/> ...
</kocsmák>
```

Elem csomópont.

Attribútum csomópont

A dokumentum csomópont minden, plusz az <? xml version... rész.

A csomópontok típusa



Utak az XML dokumentumban

- ◆ Az XPath segítségével az XML dokumentumokat járhatjuk be. Más szóval utakat adhatunk meg.
- ◆ Az utak mindenkorban tételek egy listáját választják ki.

Út kifejezések

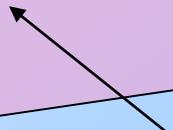
- ◆ Egyszerű formájában az utak perjel és jelölők (tagek) sorozatából állnak. A kifejezés perjellel kezdődik.
 - ◆ Példa: /kocsmák/kocsma/ár
- ◆ **Informális jelentés:** a dokumentum csomópontból kiindulva balról-jobbra haladva kövessük a jelölőket sorra.

Útkifejezések kiértékelése

- ◆ Tegyük fel, hogy az első jelölő a gyökér.
 - ◆ Ennek a jelölőnek a feldolgozása a gyökér pontot tartalmazó, azaz egyelemű, sorozatot eredményezi.
- ◆ Tegyük fel, hogy rendelkezésünkre áll tételek egy sorozata és a következő jelölő X .
 - ◆ minden elempontot helyettesítsünk az X jelölőjű alelemeivel a listában.

Példa: /kocsmák

```
<kocsmák>
  <kocsma név = "Joe bárja">
    <ár melyikSör = "Bud">2.50</ár>
    <ár melyikSör = "Miller">3.00</ár>
  </kocsma> ...
  <sör név = "Bud" árulja = "Joe bárja
    Sue bárja ... "/> ...
</kocsmák>
```



A kocsmák elem

Példa: /kocsmák/kocsma

```
<kocsmák>
```

```
  <kocsma név = "Joe bárja">
```

```
    <ár melyikSör ="Bud">2.50</ ár>
```

```
    <ár melyikSör = "Miller">3.00</ ár>
```

```
  </kocsma>
```

```
  ...<sör név = "Bud" árulja = "Joe bárja
```

```
    Sue bárja ..."/> ...
```

```
</kocsmák>
```

A kocsma elem, amit más kocsma elemek is követhetnek.

Példa: /kocsmák/kocsma/ár

```
<kocsmák>
  <kocsma név = "Joe bárja">
    <ár melyikSör ="Bud">2.50</ár>
    <ár melyikSör = "Miller">3.00</ár>
  </kocsma> ...
  <sör név = "Bud" árulja = "Joe bárja
    Sue bárja ..."/> ...
</kocsmák>
```

ár elemek, melyeket más kocsmák ár elemei követhetnek.

Attribútumok az utakban

- ◆ Az attribútumokat a @ jel jelöli, ez után következik az attribútum neve.

Példa:

/kocsmák/kocsma/ár/@melyikSör

```
<kocsmák>
  <kocsma név = "Joe bárja">
    <ár melyikSör ="Bud">2.50</ár>
    <ár melyikSör = "Miller">3.00</ár>
  </kocsma> ...
  <sör név = "Bud" árulja = "Joe bárja
    Sue bárja ..."/> ... A "Bud" "Miller" értékek
  </kocsmák>                                belekerülnek az eredménybe.
```

Ne felejtsük: tételek listája

- ◆ Eddig a tételek mindenkor elemek voltak.
- ◆ Ha egy útkifejezés attribútummal végződik, akkor a lista atomi típusú elemekből áll (az előző példában sztringekből).

Utak, melyek akárhol kezdődhetnek

- ◆ Ha az út a dokumentum pontból indul, akkor a $//X$ útkifejezés megtalálja az összes X gyerekelementet a beágyazottság bármely szintjén, és visszaadja abban a sorrendben, ahogy az a dokumentumban megjelenik,
- ◆ illetve ha ez a útkifejezés folytatódik: $//X/...,$ akkor vagy a gyökérből fog indulni vagy bármely olyan részelemből, aminek a jelölője $X.$

Példa: //ár

```
<kocsmák>
  <kocsma név = "Joe bárja">
    <ár melyikSör ="Bud">2.50</ár>
    <ár melyikSör = "Miller">3.00</ár>
  </kocsma> ...
  <sör név = "Bud" árulja = "Joe bárja
    Sue bárja ..."/> ...
</kocsmák>
```

Ezek az ár elemek és bármely másik ár eleme a dokumentumnak.

Jolly-joker: *

- ◆ A csillag (*) tetszőleges jelölő nevet helyettesít.
- ◆ Példa: /*/*/ár az összes „dé dunoka” (harmadik szinten lévő) ár elemet adja vissza.

Példa: /kocsmák/*

<kocsmák>

Ez a kocsma elem és esetleg más kocsma elemek, plusz ez a sör elem és esetleg más sör elemek.

<kocsma név = "Joe bárja">

<ár melyikSör ="Bud">2.50</ ár>

<ár melyikSör = "Miller">3.00</ár>

</kocsma> ...

<sör név = "Bud" árulja = "Joe bárja

Sue bárja ..."/> ...

</kocsmák>

Szűrési feltétel

- ◆ A jelölőket feltételek [...] követhetik.
- ◆ Ilyenkor csak azok az útkifejezésre illeszkedő utak kerülnek be az eredménybe, melyek a feltételt is kielégítik.

Példa: szűrési feltétel

◆ /kocsmák/kocsma/ár[ < 2.75]
<kocsmák>
<kocsma név = "Joe bárja">

<ár melyikSör ="Bud">2.50</ár>
<ár melyikSör ="Miller">3.00</ár>

Az aktuális elem.

Csak ez az ár elem kerül be az eredménybe a láthatók közül.

Példa: szűrés attribútummal

◆ /kocsmák/kocsma/ár[@melyikSör = "Miller"]

<kocsmák>

<kocsma név = "Joe bárja">

<ár melyikSör ="Bud">2.50</ ár>

<ár melyikSör = "Miller">3.00</ár>

Tengelyek

- ◆ Általánosan: az útkifejezésekben minden egyes lépésnél *tengelyekkel (axes)* adhatjuk meg a következő lépésnél feldolgozandó pontok listáját.
- ◆ A default tengely a `child:: ---` ami az összes gyermekét veszi az aktuális pontoknak.

Példa: tengelyek

- ◆ /kocsmák/sör valójában a /child::kocsmák/child::sör rövidítése.
- ◆ @ pedig az attribute:: tengely rövidítése.
 - ◆ Így a, /kocsmák/sör[@név = "Bud"] jelentése
/kocsmák/sör[attribute::név = "Bud"]

További tengelyek

- ◆ Néhány további hasznos tengely:
 1. parent:: = az aktuális pont(ok) szülője (szülei).
 2. descendant-or-self:: = az aktuális pont(ok) és az összes leszármazott.
 - ◆ // valójában ennek a rövidítése (ez majdnem igaz).
 3. ancestor::, ancestor-or-self éít.
 4. self (ennek rövidítése: .)

XQuery

- ◆ Az XQuery egy SQL-hez hasonló lekérdezőnyelv, ami XPath kifejezéseket használ.
- ◆ Ugyanúgy a tételek lista adatmodellt használja.
- ◆ Az XQuery egy funkcionális nyelv.
 - ◆ Ez azt jelenti, hogy ahol kifejezés szerepelhet, ott tetszőleges XQuery kifejezés szerepelhet. Ez eltérés az SQL-től. Az SQL-ben a SELECT-nél nem szerepelhetett SQL alkérdés például.

Tételek listája (részletesebben)

- ◆ Az XQuery-ben előfordulhat, hogy listák listája generálódik.
- ◆ Az ilyen listákat a rendszer „sima” listává alakítja át.
- ◆ Példa: $(1\ 2\ \boxed{0}\ (3\ 4)) = (1\ 2\ 3\ 4).$
Üres lista.

FLWR kifejezések (flower)

1. Egy vagy több **for** és/vagy **let** záradék.
2. Ezek után opcionálisan egy **where** záradék.
3. Végül egy **return** záradék.

Az FLWR kifejezések szemantikája

- ◆ Mindegyik **for** egy ciklust generál.
 - ◆ a **let** a cikluson belüli hozzárendeléseket adja meg.
- ◆ minden iterációjánál a beágazott ciklusnak, ha van ilyen, ki kell értékelni a **where** záradékot.
- ◆ Ha a **where** záradék IGAZ, a **return** záradéknak megfelelő értéket a végeredményhez kapcsoljuk.

FOR záradék

for <változó> in <kifejezés>, . . .

- ◆ A változók \$ jellet kezdődnek.
- ◆ A **for** változója egy ciklusban sorra bejárja a kifejezés eredményének összes tételeit.
- ◆ A **for** után megadott részek tehát minden egyes tételere végrehajtódnak egyszer.

A korábbi példákban is használt dokumentum.

Példa: FOR

```
for $sor in  
  document("kocsmak.xml")/kocsmák/sör/@név
```

```
return  
  <sörNév> {$sor} </sörNév>
```

- ◆ \$sor a példa dokumentum sör elemeinek név attribútumán fut végig.
- ◆ Az eredmény sörNév elemek listája:

```
<sörNév>Bud</sörNév>  
<sörNév>Miller</sörNév> . . .
```

A {} jelek közötti kifejezést mindenki kiértékeli a rendszer, ha a return záradék végrehajtására kerül a sor a ciklusban.

Kapcsos zárójelek

- ◆ Ha azt szeretnénk, hogy egy változó nevet, pl. \$x, ne sima szövegként kezeljen a rendszer kapcsos zárójelek közé kell tennünk.
 - ◆ Példa: <A>\$x egy A elem lesz "\$x" értékkel ugyanúgy, mint a <A>foo is egy A elem "foo" értékkel.

Kapcsos zárójelek --- (2)

- ◆ De `return $x` értéke egyértelmű.
- ◆ Jelölők vagy idézőjelek nélküli sima sztringet nem adhat vissza a lekérdezés, azaz, ha `$x`-t sima sztringként szeretnénk visszaadni, nem pedig a `$x` változó értékére vagyunk kíváncsiak, akkor az eredménynek `return <a>$x` vagy `return "$x"` alakúnak kell lennie.

LET záradék

let <változó> := <kifejezés>, . . .

- ◆ A változó értéke *tételek listája* lesz, ez a lista a kifejezés eredménye.
- ◆ A **let** záradék hatására nem indul el egy ciklus; a **for** záradék hatására igen.

Példa: LET

```
let $d := document("kocsmák.xml")
```

```
let $sor := $d/kocsmák/sör/@név
```

```
return
```

```
<sörNév> {$sor} </sörNév>
```

◆ Az eredmény egyetlen elemből áll az összes sörnévvel:

```
<sörNév> Bud Miller ... </sörNév>
```

Order-By záradék

- ◆ Az FLWR valójában FLWOR, ahol egy **order-by** záradék előzheti meg a **return** záradékot.
- ◆ Alakja: order by <kifejezés>
 - ◆ Opcionális **ascending** vagy **descending**.
- ◆ A kifejezés a változók minden hozzárendelésére kiértékelődik.
- ◆ A végeredmény listájának sorrendjén változtat.

Példa: Order-By

- ◆ A Bud összes árát kilistázzuk, a legalacsonyabb legelőször.

```
let $d := document("kocsmak.xml")
```

```
for $p in  
$d/kocsmák/kocsma/ár[@melyikSör="Bud"]
```

```
order by $p
```

Rendezi a
hozzárendelés
értékeit.

\$p-hez a megfelelő
ár elemeket rendeli.

```
return $p
```

Az eredmény ár
elemeknek egy listája.

Összehasonlítás: SQL ORDER BY

- ◆ Az SQL ugyanezen az elven működik; a FROM és WHERE záradékok eredménye rendeződik, nem a végeredmény.
- ◆ Példa: R(a,b) relációra:

```
SELECT b FROM R  
WHERE b > 10
```

Aztán, a már rendezett sorokból vesszük a b értékeket.

```
ORDER BY a;
```

R sorai, ahol $b > 10$
az a értékek szerint
rendeződnek.

Feltételek (predicates)

- ◆ A feltételekben a „létezést” követeljük meg.
- ◆ Példa: /kocsmák/kocsma[@név] jelentése “az összes olyan kocsma, aminek létezik neve.”
- ◆ Példa: /kocsmák/sör[@árulja = “Joe bárja”] azon sörök listáját adja vissza, melyeket Joe bárjában megkaphatunk.

Példa: összehasonlítások

- ◆ Az összes söre, amit Joe bárjában árulnak, adjuk vissza az ár elemeket az összes kocsmát figyelembe véve.
- ◆ Az eredmény BBP elemekből áll majd, melynek attribútuma a kocsma és a sör nevét adják majd meg, egy aleleme pedig az árat.

Stratégia

1. Készítsünk egy tripla for ciklust, ahol vesszük az összes **sör** elemet, aztán az összes **kocsma** elemet, majd minden egyes kocsmára a hozzá tartozó **ár** elemeket.
2. Ellenőrizzük, hogy a sört kapható-e Joe bárjában, és hogy a sört neve és az aktuális **ár** elemben a **melyikSör** attribútum értéke egyezik-e.
3. A kívánt alakú eredmény megadása.

A lekérdezés

```
let $bars = doc("kocsmák.xml") / kocsmák  
for $beer in $bars/sör  
for $bar in $bars/kocsma  
for $price in $bar/ár  
where $beer/@árulja = "Joe bárja" and  
      $price/@melyikSör = $beer/@név  
return <BBP kocsma = "{$bar/@név}" sör =  
      "{$beer/@név}>{$price}</BBP>
```

Ez nem
igazán
hatékony.



„Szigorú” összehasonlítások

- ◆ Ha meg szeretnénk követelni, hogy az összehasonlított listák egyetlen elemet tartalmazzanak az alábbi összehasonlításokat kell alkalmaznunk:
 - ◆ eq, ne, lt, le, gt, ge.
- ◆ Példa: \$beer/@árulja eq "Joe bárja" igaz, ha a beer változóhoz egyetlen elem lett hozzárendelve, melynek árulja attribútuma Joe bárja.

Elemek és értékek összehasonlítása

- ◆ Ha egy elemet hasonlítunk össze egy értékkel, akkor az elemnek a hozzá tartozó értékét vesszük, ha az az érték atomi.
- ◆ **Példa:** /kocsmák/kocsma[@név="Joe bárja"]/ár[@melyikSör="Bud"] eq "2.50"

Elemek összehasonlítása

- ◆ Nem elegendő, ha két elem „ugyanolyannak tűnik”.
- ◆ Példa:

/kocsmák/kocsma[@név="Joe bárja"] /
ár[@melyikSör="Bud"] eq

/kocsmák/kocsma[@név="Sue bárja"] /
ár[@melyikSör="Bud"]

hamis, még akkor is ha Joe és Sue
ugyanannyit kérnek a Bud sörért.

Elemek összehasonlítása – (2)

- ◆ Egy elem csak önmagával tud egyenlő lenni.
- ◆ **Tudniillik**: az elemek valójában mutatók, amelyek a dokumentum megfelelő helyére mutatnak, azaz nem azonosak a szöveges tartalmukkal.

Az elemek adatainak kinyerése

- ◆ Tegyük fel, hogy elemek értékeit szeretnénk összehasonlítani nem pedig az elhelyezkedésüket az adott dokumentumban.
- ◆ Az E elem értékét a *data* függvény használatával kaphatjuk meg: $\text{data}(E)$.

Példa: data()

- ◆ Tegyük fel, hogy az előbbi lekérdezésünkben módosítjuk a return záradékot:

```
return <BBP kocsma ={$bar/@name}  
        sör = {$beer/@name}  
        ár = {data($price)} />
```

Ismétlődések kiszűrése

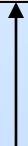
- ◆ Használjuk a distinct-values függvényt, aminek a paramétere: elemek listája.
- ◆ Finomság: a függvény a jelölők nélkül veszi az értékeket és ezek szöveges értékét hasonlítja össze.
 - ◆ Az eredményben nem írja vissza a jelölőket.

Példa: az összes különböző ár

```
return distinct-values(
```

```
let $bars = doc("kocsmák.xml")  
return $bars/kocsmák/kocsma/ár
```

```
)
```



Emlékezzünk vissza: az XQuery funkcionális nyelv, azaz ahol érték jelenhet meg, oda tetszőleges XQuery kifejezést is írhatunk.

Logikai érték (Boolean)

- ◆ Egy-egy kifejezés *logikai értéke* a következő:
 1. ha a kifejezés logikai típusú, akkor az aktuális érték.
 2. FALSE ha a kifejezés kiértékelésének eredménye: 0, "" [az üres sztring] vagy () [az üres lista].
 3. TRUE különben.

Példa: logikai értékek

1. @név="Joe bárja" TRUE, ha név attribútum értéke "Joe bárja".
2. /kocsmák/kocsma[@név="Lórúgás"] TRUE, ha létezik olyan kocsma, amely név attribútumának Lórúgás az értéke.

Logikai műveletek

- ◆ E_1 and E_2 , E_1 or E_2 , not(E) tehát minden kifejezésre alkalmazható.
- ◆ Példa: not(3 eq 5 or 0) az értéke TRUE.
- ◆ A true() és false() függvények (paraméterek nélkül) TRUE és FALSE értéket adnak vissza. A konstansok kiírása helyett ezt használják.

Elágazó kifejezések

◆ if (E_1) then E_2 else E_3 is értelmezhető.

◆ Példa:

if (\$kocsma/@név eq "Joe bárja")

then \$kocsma/ár else 

Az üres lista.

Kvantifikálás

some \$x in E_1 satisfies E_2

1. E_1 -t ki kell értékelni.
 2. \$x vegye fel sor E_1 eredményének értékeit, és értékeljük ki ezzel az értékkel E_2 -t.
 3. Az eredmény IGAZ, ha \$x legalább egy értékére E_2 igaz.
- ◆ Hasonlóan:

every \$x in E_1 satisfies E_2

Példa: Some

```
for $bar in
  doc("kocsmak.xml")/kocsmák/kocsma
where some $p in $bar/ár
  satisfies $p < 2.00
return $bar/@név
```



Vegyük észre: a where \$bar/ár < 2.00
ugyanezt a hatást éri el.

Példa: Every

```
for $bar in
  doc("kocsmak.xml")/kocsmák/kocsma
where every $p in $bar/ár
  satisfies $p <= 5.00
return $bar/@név
```

Dokumentum sorrend

- ◆ A dokumentum sorrend szerinti összehasonlítás műveletei: << és >>.
- ◆ Példa: \$d/kocsmák/sör[@név="Bud"] << \$d/kocsmák/sör[@név="Miller"] igaz, ha a bal oldali sör elem megelőzi a jobb oldalit.

Halmazműveletek

- ◆ A `union`, `intersect`, `except` itt is alkalmazhatóak pontok listájára.
 - ◆ A jelentés hasonló SQL-beli jelentéshez.
 - ◆ Az ismétlődések itt is törlődnek.
 - ◆ Az eredmény elemei dokumentum sorrendben jelennek meg.

Analitikus adatfeldolgozás

Adattárház

Adatkocka

Adatbányászat

Áttekintés

- ◆ A hagyományos adatbázisokat sok, apró, egyszerű lekérdezésre hangolták
- ◆ A jelenlegi alkalmazások kevesebb, de idő igényesebb, bonyolultabb lekérdezéseket használnak
- ◆ Ezért modernebb adatarchitektúrákat fejlesztettek ki azért, hogy a bonyolultabb „analitikus”, elemző jellegű lekérdezéseket kezelní tudják

Adattárház

- ◆ Jelenleg ez az egyik legelterjedtebb formája az adatintegrációnak.
 - ◆ Egyetlen egy közös adatbázisba másolják (adattárház) az adatokat és napra készen tartják.
 - ◆ Módszere: periodikus aktualizálás, gyakran éjszaka.
 - ◆ Gyakran az analitikus lekérdezések végett hozzák létre.

OLTP

- ◆ Adatbázisok tipikusan on-line tranzakció feldolgozást végeznek (OLTP).
 - ◆ Rövid, egyszerű, gyakran feltett kérdések, minden egyik viszonylag kevés sort ad vissza válaszként.
 - ◆ Pl.: Web felületen keresztüli lekérdezések és válaszok, pénztárgépnél vásárlás, rep. jegy értékesítés

OLAP

- ◆ Növekvő jelentőségű az OLAP jellegű lekérdezések.
 - ◆ Kisebb számú, de összetett , amelyek órákig is futhatnak.
 - ◆ A lekérdezések nem igénylik az abszolút időben pontos adatbázist.

OLAP

- ◆ Általában az adatbázis nagyobb részét érintik az idetartozó tranzakciók.
 - ◆ Soros végrehajtás esetén leállhatnának az OLAP-műveletek → nem igazán tolerálható
 - ◆ Nem jó megengedni valamilyen típusú új adatok felvitelét, amikor éppen fut egy konkurens OLAP lekérdezés az ilyen típusú adatokon (ami pl. átlagolja ezeket)
- ◆ Korábban ezt nevezték adatbányászatnak.

OLAP alkalmazások

- ◆ Általában az adattárházat a fogyasztásra vonatkozó adatokból állítják össze.
 - ◆ Akár terabajtnyi adatmennyiségről, hogy melyik cikkből mennyi eladás történt
 - ◆ Előrejelzésre használható: fogyasztási adatok összesítése alapján valamilyen érdekes csoportok azonosítására

OLAP Példák

1. Amazon elemzi vásárlói viselkedését azért, hogy olyan képernyő tartalmat jelenítsen meg, amely valószínűleg érdekli a vásárlót.
2. Wal-Mart-ot az érdekli, hogy melyik régióban melyik termék értékesítése növekszik

Tipikus architektúra megoldások

- ◆ Az áruházláncok egyes áruházai OLTP szinten dolgoznak.
- ◆ A helyi adatbázisokat éjszakánként feltöltik a központi adattárházba.
- ◆ Az adatelemzők az adattárházat OLAP elemzésekre használják fel.

Csillag séma

◆ Csillag séma az adattárházak megszokott adatszerkezete. A következőkből áll:

1. Tény tábla: olyan adatok kumulált tömege mint pl. az értékesítési adatok.
 - Általában csak „beillesztés”-re állított tábla.
2. Dimenzió táblák: kisebb, általában statikus információkat tartalmaznak azokról az entitásokról, amelyekről tényeket tárolunk.

Példa csillag sémára

- ◆ Tegyük fel, hogy az adattárházban fel akarunk jegyezni minden sört értékesítési adatot: a kocsmát, a sörfajtáját, az alkeszt, a napot, időpontot és fizetett árat.
- ◆ A ténytábla egy ilyan reláció lesz:
Értékesítések(kocsma, sör, alkesz, nap, idő, ár)

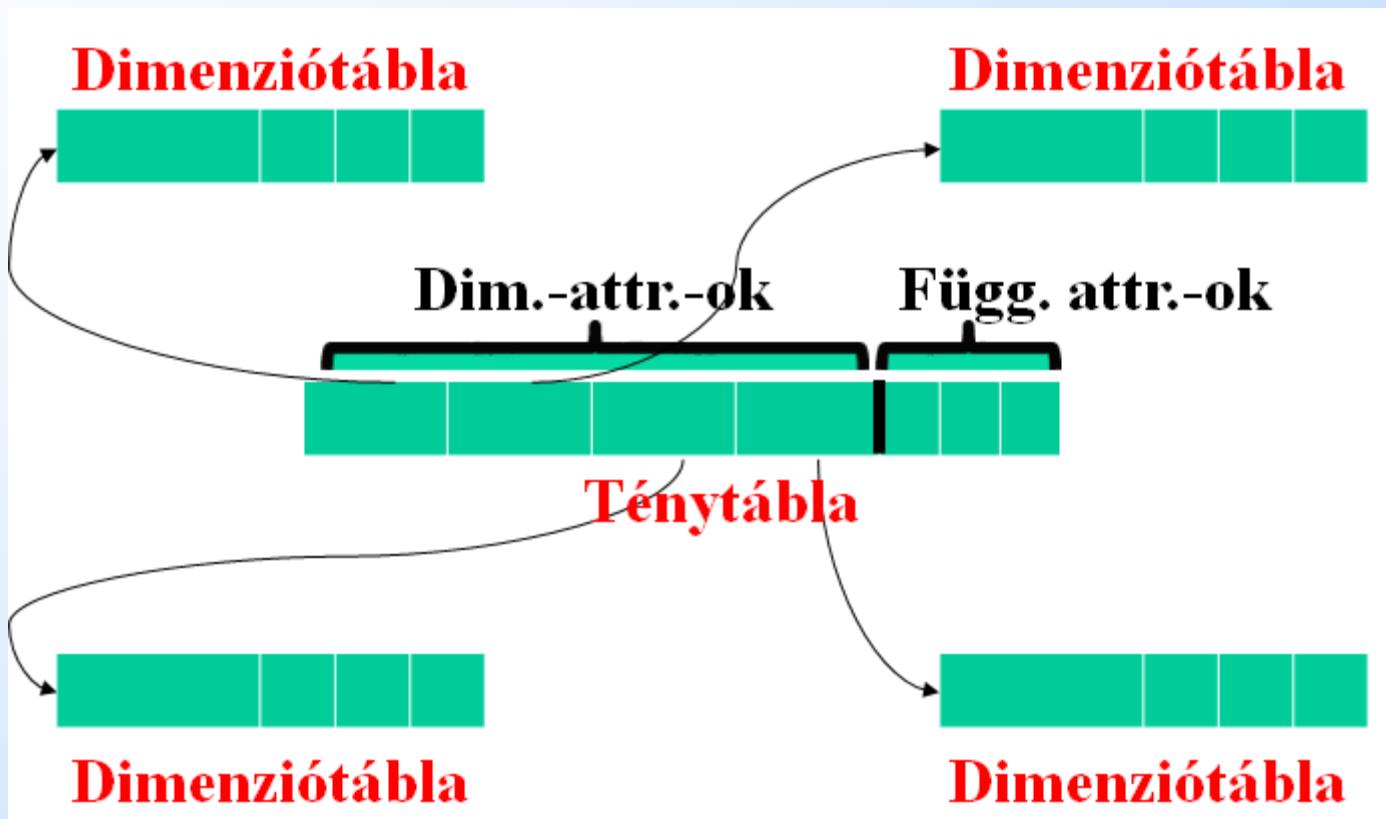
Példa folytatás

- ◆ A dimenzió táblák a kocsma, a sör és a alkesz adatait tartalmazzák

Dimenziók és a függő attribútumok

- ◆ A ténytáblák attribútumainak két osztálya:
 1. Dimenzió attribútumok: A dimenzió táblák kulcsai. (Idegen kulcsok)
 2. A függő attribútum: A sorban a dimenzió attribútumok által meghatározott értékek

Dimenziók és a függő attribútumok



Példa függő attribútumok

- ◆ Az *ár* függő attribútum az *Értékesítés* relációban
- ◆ Amelyet a dimenzió attribútumok kombinációja határoz meg: a *kocsma*, a *sör* és az *alkesz* valamint az időpont (a *nap* és az *idő* attribútumok kombinációja)

OLAP adatok többdimenziós nézete

- ◆ Gondolhatunk úgy is a ténytáblában tárolt adatokra, mintha többdim. térben vagy „kockában” lennének elrendezve
 - ◆ A kocka belsőpontjai ábrázolják az adatobjektumokat: pl. egy-egy gépkocsi eladás.
 - ◆ A dimenziók ennek az eladásnak a jellemzőit írják le.

OLAP adatok többdimenziós nézete

- ◆ Az előbbi általában *nyers adatkockának* hívjuk, hogy megkülönböztessük az összetettebb *formális adatkockától* (ld. később bővebben)

OLAP adatok többdimenziós nézete

◆ Ez utóbbi két dologban tér el:

1. A dimenziók részhalmazaira vett összesítéseket is tartalmazza az adatokon túl.
2. Az itteni belső pontok az adatobjektumoknak egy kezdeti összesítését is ábrázolhatják (pl. nem az összes sörnek az eladásait ábrázoljuk, hanem gyártónként összesítve)

Adattárház készítési módozatok

1. *ROLAP* = relációs OLAP. Relációs adatbáziskezelő rendszer olyan hangolása, amely a csillag sémát támogatja.
2. *MOLAP* = többdimenziós OLAP: specializált adatbáziskezelő használata, amely pl. az adatkocka adatszerkezetet támogatja.

ROLAP technikák

1. *Bitmap indexek*: a dimenzió táblák minden egyik kulcsértékére (pl. minden gyártóra a *Sör* relációban) egy bit vektor létrehozása, amely megmondja, hogy mely sorok tartalmazzák ezt az értéket.

Sör	Gyártó
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Michelob	Anheuser-Busch
Miller Lite	Miller Brewing Co.
Miller Genuine Draft	Miller Brewing Co.
Miller High Life	Miller Brewing Co.

Gyártó='A.-B.'	Gyártó='M.'
1	0
1	0
1	0
0	1
0	1
0	1

ROLAP technikák

2. *Materializált nézetek:* az olyan nézeteket, amelyek több lekérdezés megválaszolásához hasznosak magában az adattárházban eltárolják.

Tipikus OLAP lekérdezések

- ◆ Az OLAP lekérdezések gyakran egy csillag összekapcsolással kezdődnek: a ténytábla és a dimenzió táblák természetes összekapcsolásával.

```
SELECT *
FROM Értékesítések, Kocsmák, Sörök,
Alkeszek
WHERE Értékesítések.kocsma =
      Kocsmák.kocsma AND Értékesítések.sör
      = Sörök.sör AND Értékesítések.alkesz
      = Alkeszek.alkesz;
```

Tipikus OLAP lekérdezések 2

- ◆ Tipikus OLAP lekérdezések:
 1. Egy csillag séma összekapcsolással kezdődik.
 2. Leválogatják a fontos sorokat, a dimenzió táblák adatai alapján
 3. Egy vagy több dimenzió alapján csoportosítjuk.
 4. Az eredmény egyes attribútumait összegezzük

Példa: OLAP lekérdezés

- ◆ Palo Alto mindenki kocsmájára keressük meg az Anheuser-Busch által gyártott mindenki sörre az összes eladás értékét
1. Szűrő: *cím* = “Palo Alto” és *gyártó* = “Anheuser-Busch”.
 2. Csoportosítás: kocsma és sör.
 3. Összesítés: az *ár összege*

Példa: SQL-ben

```
SELECT kocsma, sör, SUM(ár)
FROM Értékesítések NATURAL JOIN
    Kocsmák NATURAL JOIN Sörök
WHERE cím = 'Palo Alto' AND
    gyártó = 'Anheuser-Busch'
GROUP BY kocsma, sör;
```

Materializált nézetek használata

- ◆ Az előbbi példa lekérdezés közvetlen végrehajtása az *Értékesítések* és a dimenzió táblák segítségével túl hosszú ideig tartana.
- ◆ Ha egy materializált nézetet hozunk létre, amely elegendő információt tartalmaz sokkal gyorsabban lehetne megválaszolni

Példa: Materializált nézetek használata

- ◆ Mely nézetek segítenék a lekérdezést:
- ◆ Kulcs kérdések:

1. Össze kell kapcsolni minimum az *Értékesítések*, *Kocsmák*, és *Sörök* táblákat.
2. Csoportosítani kell legalább *kocsma* és *sör* attr. szerint.
3. Sem a Palo-Alto-i kocsmákat (*sör*) sem Anheuser-Busch söreit (*sör*) nem szabad kihagyni.
4. Nem szabad lehagyni az eredményből, vetítéssel, sem a *cím* sem a *gyártó* attribútumokat.

Példa --- folytatás

Itt a materializált nézet:

A funkcionális függések miatt nincs igazi csoportosítás

```
CREATE MATERIALIZED VIEW KCSGE (kocsma,  
cím, sör, gyártó, értékesítés) AS  
SELECT kocsma, cím, sör, gyártó,  
       SUM(ár) értékesítés  
FROM Értékesítések NATURAL JOIN  
Kocsmák NATURAL JOIN Sörök  
GROUP BY kocsma, cím, sör, gyártó;
```

Mivel kocsma -> cím és sör -> gyártó, ezért nincs igazi csoportosítás
Szükségünk van címre és gyártónak a SELECT-ben.

Példa --- Lezárás

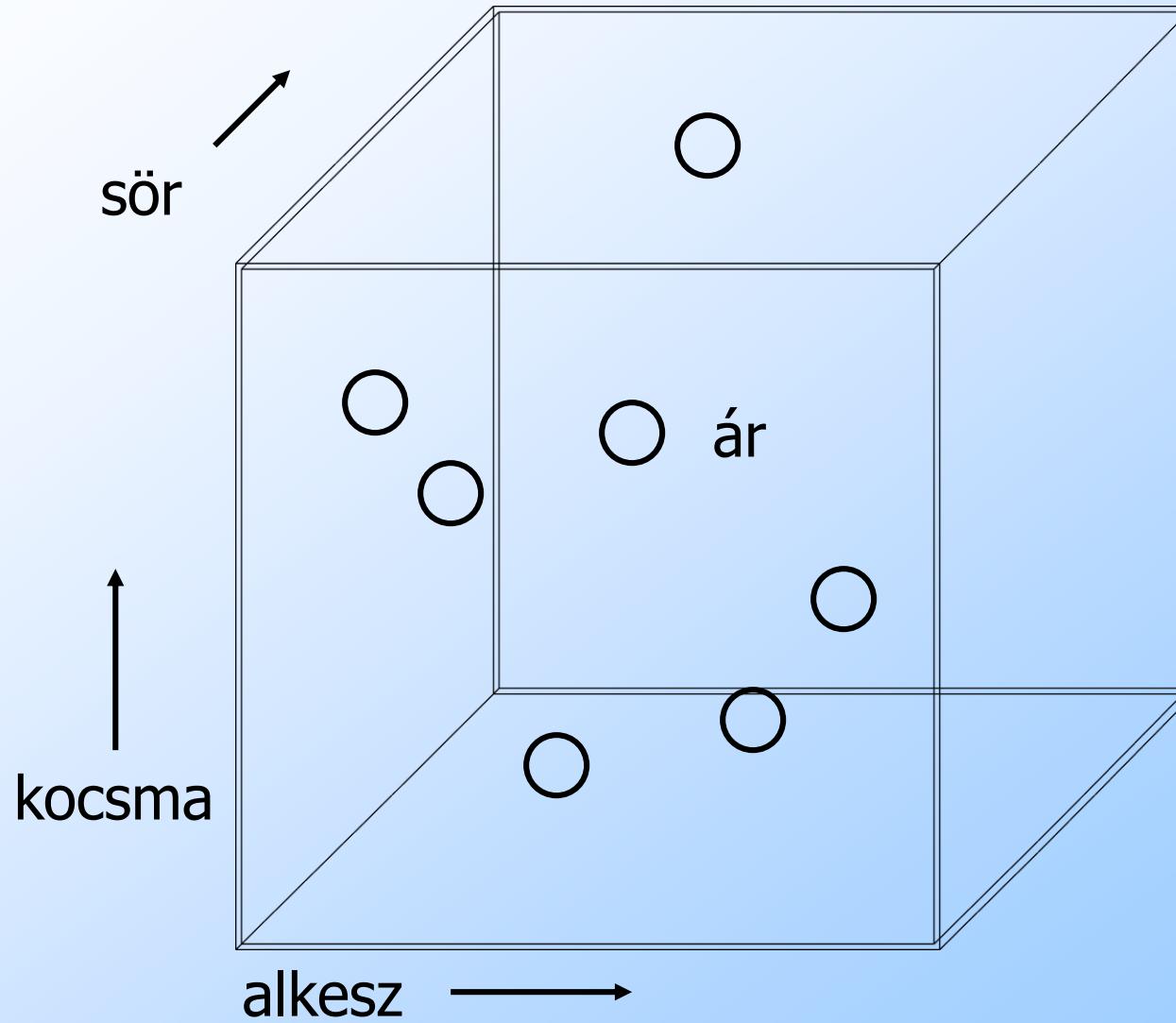
BABMS materializált nézet lekérdezése:

```
SELECT kocsma, sör, értékesítés  
FROM KCSGE  
  
WHERE cím = 'Palo Alto' AND  
gyártó = 'Anheuser-Busch';
```

MOLAP és adatkockák

- ◆ A dimenzió táblák kulcsai a hiper-kocka dimenziói:
 - ◆ Példa: *Értékesítések* adataira, négy dimenzió: *kocsmák, sörök, alkeszek, és idő*.
- ◆ A függő attribútumok (pl. *ár*) a kocka „belső” pontjaiban jelennek meg

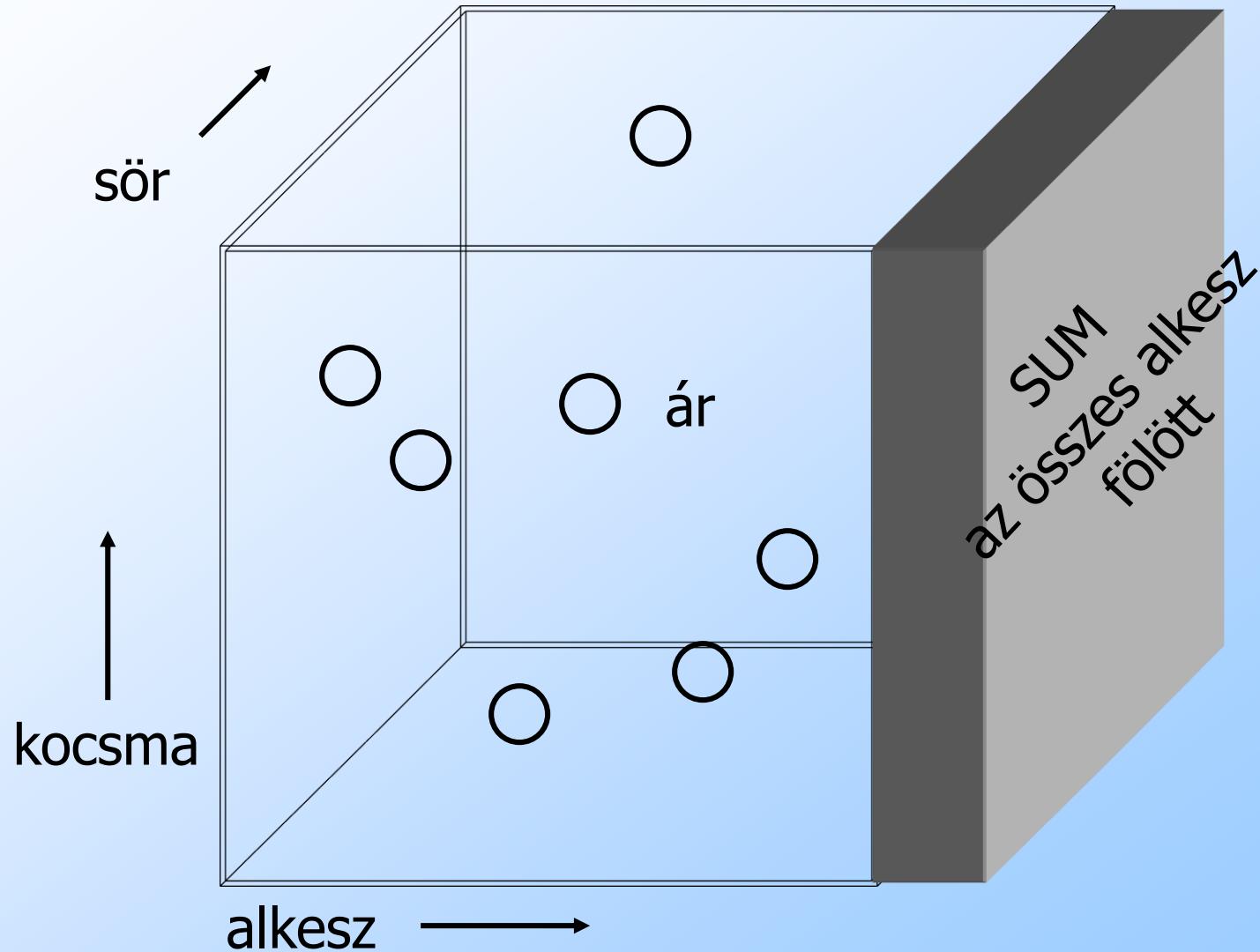
Vizualizáció -- Adatkockák



Kocka oldalai, szélei

- ◆ Az adatkocka tartalmazhat összesítéseket (tipikusan SUM) a kocka oldalai szerint
 - ◆ A már említett példa: a sörök dimenzióját kicseréljük a gyártóra
- ◆ A kocka szélei tartalmazhatnak összesítést egy dimenzióban, két dimenzióban, stb.

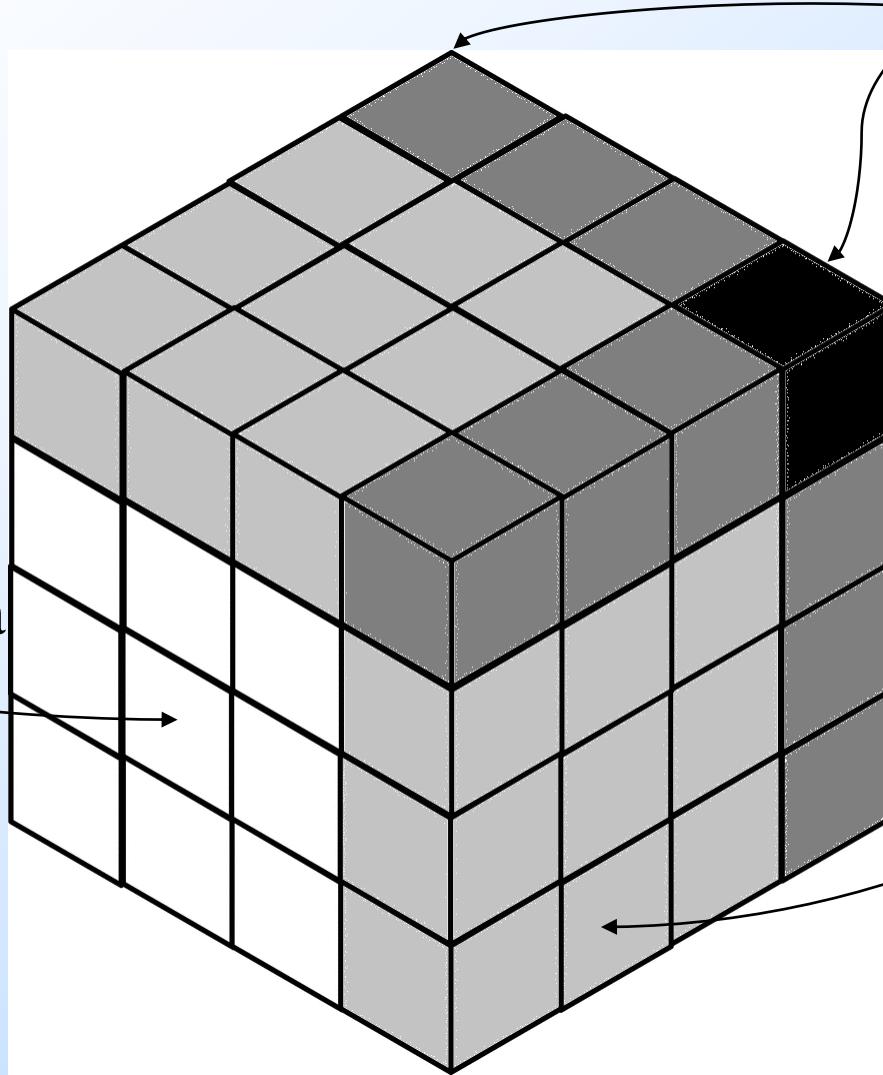
Vizualizáció --- Adatkocka összesítéssel



Példa

- ◆ A példánk 4 –dimenziós adatnockája tartalmazza az *árak* összegét, minden kocsmára, sörre, alkeszre és idő egységre (pl. napra)
- ◆ De tartalmazhatná az értékesítési adatokat, az *árak* összegét minden kocsma-sör párosra, minden kocsma-alkesz-nap hármásra, stb.

Eredeti adatkocka



A kocka határainak bővítése a dim.-ok minden lehetséges kombinációján kiszámított összesítéseknek megfelelő módon

Az adatkocka szerkezete

- ◆ Képzeljük azt, hogy minden egyik dimenzió tartalmaz egy további értéket (*) [A szokásos vélemezett „tetszőleges” érték értelmezésével].
- ◆ Egy olyan pont, amelynek koordinátái között egy vagy több „*” szerepel azt jelenti, hogy azok fölött a dimenziók fölött, amelyeknek az értéke ebben a sorban (adatkocka pontban) „*” összegzést hajt végre.
- ◆ Pl. Értékesítések(“Joe’s Bar”, “Bud”, *, *, értékesítés), minden alkeszre vonatkozóan, az összes eltöltött időt figyelembe véve, amikor Bud-t fogyasztottak Joe kocsmájában (“Joe’s Bar”) a fogyasztás teljes összegét (értékesítés) adja meg.

Lefúrás

- ◆ Lefúrás= „finomabb összegzés” = az összegzéseket finomabb alkotórészeire bontjuk.
- ◆ Példa: Ha kiderült, hogy Joe kocsmája nagyon kevés Anheuser-Busch sört adott el.
- ◆ Bontsuk fel az értékesítési adatokat az egyes Anheuser-Busch sörfajták szerint.

Felgörgetés (felösszegzés)

- ◆ Felgörgetés = összegzés egy vagy több dimenzió mentén
- ◆ Példa: legyen egy olyan táblánk, amelyben minden alkeszről azt tároljuk, hogy mennyi *Bud sör* fogyasztott el az egyes kocsmákban, görgessük fel ezt egy olyan táblába, amelyik megadja minden alkeszre azt, hogy mennyi *Bud sör* fogyasztott el

Materializált adatkocka nézetek

- ◆ Adatkockák létrehozására olyan materializált nézeteket lehet használni, amelyek egy vagy több dimenzióban összegzéseket tartalmaznak.
- ◆ Az egyes dimenziókat nem kell teljes mértékben összegezni – az egyik lehetőség, hogy a dimenzió tábla bizonyos attribútumai szerint összegzünk.

Példa

- ◆ Az értékesítés materializált nézete (*Értékesítések*), adatkocka a következő lehet:
 - ◆ Alkeszek alapján teljes összegzés.
 - ◆ A sör fajták alapján nincs összegzés
 - ◆ Idő tekintetében hetek szerinti összegzés
 - ◆ A kocsmák városa szerinti összegzés.

Kockaművelet SQL-ben

- ◆ Az SQL támogatja a kockaműveletet: WITH CUBE (illetve PostgreSQL-ben, Oracle-ben: GROUP BY CUBE (A, B))
- ◆ Így nem csak az egyes csoportokra vonatkozó sorokat kapjuk vissza, hanem azokat is, amelyek összesítéseket fejeznek ki a csoportosításban résztvevő egy/több dimenzióra nézve
- ◆ A soroknál a *-ot a NULL jelzi

Példa

Materializált nézet:
Értékesítések

kocsma	sör	ár
Joe's bar	Bud	2.5
Joe's bar	Bud Lite	3.5
Joe's bar	Michelob	3.0
Joe's bar	Miller Lite	2.75
Joe's bar	Miller Genuine Draft	4.5
Joe's bar	Miller High Life	3.25
Sue's bar	Bud	2.0
Sue's bar	Bud Lite	2.25
Sue's bar	Miller High Life	3.75
Sue's bar	Miller Lite	2.75
Sue's bar	Dreher	4.0

```

SELECT kocsma, sört, SUM(ár)
FROM Értékesítések
GROUP BY kocsma, sört WITH CUBE

```

kocsma	sör	SUM
Joe's bar	Bud	2.5
Joe's bar	Bud Lite	3.5
Joe's bar	Michelob	3.0
Joe's bar	Miller Lite	2.75
Joe's bar	Miller Genuine Draft	4.5
Joe's bar	Miller High Life	3.25
Sue's bar	Bud	2.0
Sue's bar	Bud Lite	2.25
Sue's bar	Miller High Life	3.75
Sue's bar	Miller Lite	2.75
Sue's bar	Dreher	4.0
Joe's bar	NULL	14.75
Sue's bar	NULL	19.5
NULL	Bud	4.5
NULL	Bud Lite	5.75
NULL	Michelob	3.0
NULL	Miller Lite	5.5
NULL	Miller Genuine Draft	4.5
NULL	Miller High Life	7.0
NULL	Dreher	4.0
NULL	NULL	19.5

Adatbányászat

- ◆ Az adatbányászat az olyan adatfeldolgozásokat jelenti, amelyek nagy adathalmazokat összegeznek valamilyen szempontból hasznos módon.
- ◆ Példák:
 1. Az összes Web lap klaszterezése témák szerint.
 2. A bankkártyák csalásra történő használatának jellemzőinek feltárása

Bevásárló kosár

- ◆ A relációs adatforrásokból az egyik tipikus adatbányászati feladat a *bevásárló kosár* = olyan bevásárlási tételek lista, amiket egy fogyasztó megvásárolt.
- ◆ A bevásárlási adatok egyik összegzése a *gyakran előforduló tételhalmazok* = olyant áru tételek, amelyek gyakran fordulnak elő együtt.

Példa

- ◆ Ha valaki gyakran vásárol hamburgert és ketchup-ot együtt és szalma krumplit vesz még hozzá.
- ◆ Akkor csinálunk egy árleszállítást a hamburgerre, viszont emeljük meg a ketchup árát.

Gyakori párosítások feltárása

- ◆ Az egyszerű eset, amikor csak a gyakori párosításokat akarjuk megtalálni.
- ◆ Tegyük fel, hogy az adatok a *Baskets(basket, item)* relációban vannak.
- ◆ Az *s támogató küszöbérték* az olyan bevásárló kosarak minimális száma, amelyben egyrészt a minket érdeklő termék párosítások jelennek meg, és másrészt ha ezt a küszöb értéket túllépi a bevásárló kosarak száma, akkor érdekesek lesznek számunkra.

Gyakori párosítások SQL-ben

```
SELECT b1.item, b2.item
```

```
FROM Baskets b1, Baskets b2
```

```
WHERE b1.basket = b2.basket
```

```
AND b1.item < b2.item
```

```
GROUP BY b1.item, b2.item
```

```
HAVING COUNT(*) >= s;
```

Dobjuk el azokat a termékeket, amelyek nem jelennek meg legalább s -szer

Keressük azokat a bevásárló kosár párosokat, amelyek ugyanarra a kosárra vonatkoznak, de az árutételek különböznek

Az első tételek meg kell előznie a másikat azért, hogy ne számoljuk kétszer ugyanazt a pár.

Hozunk létre egy csoportot minden olyan pár termékre, amelyik legalább az egyik kosárban megjelenik

A-Priori Trükk --- 1

- ◆ Az egyszerű megvalósítást a *Baskets* reláció önmagával történő összekapcsolásával lehetne megoldani
- ◆ Az *a-priori algorithmus* felgyorsítja a lekérdezést, a következőképpen, egy $\{i, j\}$ csak akkor éri el a támogató küszöbértéket, $s - t$, ha mind $\{i\}$ minden $\{j\}$ vagyis mindenketten eléri.

A-Priori Trükk --- 2

- ◆ Használunk egy materializált nézetet a gyakori termékek nyilvántartására.

```
INSERT INTO Baskets1(basket, item)
SELECT * FROM Baskets
WHERE item IN (
    SELECT ITEM FROM Baskets
    GROUP BY item
    HAVING COUNT(*) >= s
);
```

Termékek, amelyek legalább s kosárban megjelennek

A-Priori Algoritmus

1. Készítsünk egy *Baskets1* materializált nézetet.
2. A nyilvánvaló lekérdezést futtassuk a *Baskets1-en*, *Baskets* helyett.
 - *Baskets1* lekérdezése olcsó, mivel nincs benne összekapcsolás
 - *Baskets1* –nek kevesebb sora van (val.szeg) mint *Baskets*-nek
 - A futási idő az összekapcsolásban érintett sorok számának négyzetével arányosan csökken.

Példa

◆ Tegyük fel:

1. Egy szupermarket 10,000 terméket árusít.
2. Az átlagos kosár tartalma 10 tétel.
3. A támogatási küszöbérték kosarak számának 1%-a

◆ Ezért legfeljebb a tételek 1/10 tekinthető gyakorinak.

◆ Valószínűleg, egy kosárban a tételek kisebb része tekinthető gyakorinak -> 4-széres gyorsuláshoz vezet ez a feltételezés.

Objektum-relációs adatbázisok

Felhasználói típusok (User-Defined Types)

Objektum ID-k

Beágynazott táblák (Nested Tables)

Relációs és az O-O modell egyesítése

- Az O-O modell több érdekes adattípushoz támogat – nem csak egyszerű állományokat
 - Térkép, multimédia, stb.
- A relációs modell magas szintű lekérdezésekhez támogat
- Objektum-relációs adatmodell egy olyan kísérlet, amely minden világból a legjobbat szeretné nyújtani

Az adatbázis-kezelő rendszerek (DBMS) fejlődése

- Az O-O adatbáziskezelő rendszerek sokáig nem mutattak olyan hatékonyságot a jól bevált relációsokkal szemben, amely lehetővé tette volna az elterjedésüket.
- A relációs DBMS-ek objektum-relációs kiterjesztése az O-O megközelítés több előnyös tulajdonságát megragadja, mégis megtartja a relációt mint az alapvető absztrakciós mechanizmust és adatszerkezetet

SQL-99 és az ORACLE szolgáltatásai

- SQL-99 több objektum relációs szolgáltatás leírását tartalmazta.
- Azonban mivel viszonylag új gondolat és szabvány volt abban az időben, minden gyártó a saját megközelítését és megvalósítását használta.
 - A példákban általában az ORACLE szolgáltatásait és szintaxisát használjuk

Felhasználó által definiált adattípus

- **Felhasználó által definiált adattípus (UDT rövidítés)**, egy O-O osztály definíciója, amely egy adatszerkezet és metódusai.
 - Azonos „típusú” objektumok egy osztály definiálnak
 - Viselkedés: metódusok halmzával kifejezve, amelyek az osztályhoz tartozó objektumokon hajthatóak végre

Felhasználó által definiált adattípus

- Két használati módja van:
 1. *Sortípus*, vagyis egy relációt, mint adattípust kezelünk.
 2. Egy reláció attribútumának a típusa.

Felhasználó által definiált adattípus

```
CREATE TYPE <typename> AS (
    <list of attribute-type pairs>
);
```

- **ORACLE-ben:** CREATE TYPE <typename>
AS OBJECT
- Utána lehet a típust eltárolni.

Példa: UDT létrehozásra

```
CREATE TYPE BarType AS (
    name CHAR(20),
    addr CHAR(20)
) ;
CREATE TYPE BeerType AS (
    name CHAR(20),
    manf CHAR(20)
) ;
```

Példa: UDT létrehozásra Oracle-n belül

```
CREATE TYPE SDO_POINT_TYPE AS OBJECT
(
    X      NUMBER,
    Y      NUMBER,
    Z      NUMBER
) ;
```

Hivatkozások

- Ha T egy UDT, akkor $\text{REF } T$ a T -re történő hivatkozás típusa, vagyis egy mutató egy T típusú objektumra.
- Ezt „objektum azonosítónak” (OID) is hívják O-O rendszerekben.
- Gyakorlatilag az OID élete végéig azonosít egy objektumot, függetlenül a komponenseinek/mezőinek értékeitől

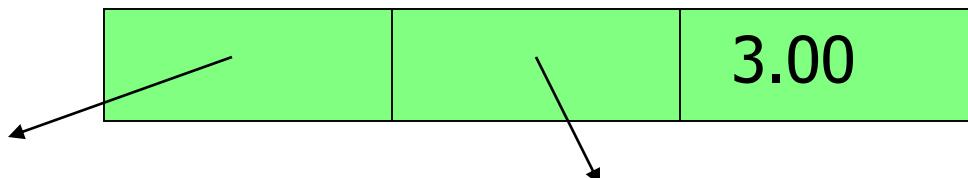
Hivatkozások

- Azonban az *O/D*-től eltérően – amelyek alapértelmezésben nem láthatók -, *REF* látható, bár általában nehezen értelmezhető.

Példa: REF

```
CREATE TYPE MenuType AS (
    bar      REF BarType,
    beer     REF BeerType,
    price    FLOAT
);
```

- **MenuType** objektum valahogy így néz ki:



Egy *BarType*
objektumra hivatkozás

Egy *BeerType*
objektumra hivatkozás

UDT-k, mint sortípusok

- Egy relációs táblát egy *sortípus* segítségével mint sémával lehet definiálni, az elemeinek felsorolása helyett
- Szintaxis:

```
CREATE TABLE <table name> OF  
<type name>;
```

Példa: Egy reláció készítése

```
CREATE TABLE Bars OF BarType;
```

```
CREATE TABLE Beers OF BeerType;
```

```
CREATE TABLE Sells OF MenuType;
```

Sortípusú relációk értékei

- A *kocsmák (Bars)* relációt lehet, úgy definiálni, hogy a típusa a *KocsmaTípus (BarType)* , ez egy unáris reláció - nem párok halmaza -, amelynek a sorai két komponenst/mezőt tartalmaznak: *név* és *cím*.
- Mindegyik *UDT*-nek van egy típus konstruktora, amely összefogja ehhez a típushoz tartozó objektumokat.

Példa: típuskonstruktor

- Lekérdezés

```
SELECT * FROM Bars;
```

- Eredmény sora:

BarType('Joe''s Bar', 'Maple St.')

Sortípus értékeinek elérése

- ORACLE-ben a pont („.”) az elvártaknak megfelelően működik.
 - Azonban az ORACLE-ben kötelező minden relációra egy aliaset használni akkor, amikor az O-R szolgáltatásokkal kezeljük (pl. amikor az objektum mezőire hivatkozunk)
- Példa:

```
SELECT bb.name, bb.addr  
FROM Bars bb;
```

SQL-99 jellegű megközelítés

- SQL-99-ben, minden UDT-nek vannak *generátorai* (vedd ki az értéket) és *mutátorai* (változtasd meg az értéket), amelyeknek mint metódusoknak a nevei megegyeznek a mezők neveivel.
 - Pl . Az A mező *generátorának* nincs argumentuma A().
 - Az A mező *mutátorának* az új érték az argumentuma pl. A(v).

Példa: SQL-99 jellegű adatelérés

- Az előbbi lekérdezés SQL-99-ben:

```
SELECT bb.name() , bb.addr()  
FROM Bars bb;
```

Sortípusú érték beillesztése

- ORACLE-ben a szabványos INSERT-et használják
 - De ne feledjük, hogy egy sortípusú reláció unáris, és ezért szükség van a típuskonstruktorkra.
- Példa:

```
INSERT INTO Bars VALUES (
    BarType('Joe''s Bar', 'Maple
St.')
);
```

Értékek beszúrása SQL-99 stílusban

- Egy alkalmas típusú X változót hozunk létre, használva e típus típuskonstruktorát, mint metódust.
- Használjuk a *mutátor* metódust az attribútumokra azért, hogy az X változó mezőinek értékét megadhassuk.
- Illesszük be az X változó értékeit a relációba

SQL-99 beillesztés példa

- Ez a lekérdezés egy *eljárás* része lehet, ezért van egy új változó, *newBar*.
- A *mutátor* metódusok megváltoztatják a név és cím komponenst.

```
SET newBar = BarType();
    newBar.name('Joe''s Bar');
    newBar.addr('Maple St.');
    INSERT INTO Bars VALUES(newBar);
```

UDT-k, mint oszloptípusok

- UDT lehet egy attribútum típusa.
- Akár egy UDT deklarációban, vagy egy CREATE TABLE utasításban, az UDT típus neve úgy használható mint az attribútum típusa.

Példa: oszloptípus

```
CREATE TYPE AddrType AS (
    street    CHAR(30),
    city      CHAR(20),
    zip       INT
);
```

```
CREATE TABLE Drinkers (
    name     CHAR(30),
    addr     AddrType,
    favBeer  BeerType
);
```

Az *addr* és
favBeer attribútumok
értékei objektumok,
3 illetve
2 mezővel

Mező elérés problematikája az ORACLE-ben

- Egy objektum F mezőjét $A.F$ kifejezéssel elérhetjük, amelynek ez az értéke
- Azonban egy aliaset kell használni, pl. rr, R relációra, annak A attribútumára mint pl. $rr.A.F$

Példa: Oracle-ben mezők elérése

- Rossz:

```
SELECT favBeer.name  
FROM Drinkers;
```

- Rossz :

```
SELECT Drinkers.favBeer.name  
FROM Drinkers;
```

- Jó:

```
SELECT dd.favBeer.name  
FROM Drinkers dd;
```

A REF-k (hivatkozások) követése: SQL-99 stílus

- $A \rightarrow B$ csak akkor értelmes ha:
 1. Ha A egy REF T típusú.
 2. A T típusú objektum mezője (komponense) B .
- Az A által hivatkozott, mutatott objektum B **mezőjének** értékét jelöli

Példa: REF-k (hivatkozások) követése

- Emlékezzünk rá, hogy az **Sells (Értékesítés)** egy olyan reláció egy olyan sortípussal ahol **MenuType(bar, beer, price)**, és ahol bar (kocsma) és beer (sör) REF-ek, hivatkozások a **BarType** és **BeerType** típusú típusú objektumokra.
- ◆ Keresd meg a Joe által felszolgál söröket:
- ```
SELECT ss.beer()->name
FROM Sells ss
WHERE ss.bar()->name = 'Joe''s Bar';
```
- A nyilat követve kapjuk meg a hivatkozott „*kocsma*”-t és „*sör*”-t

Először használjuk a generátor metódust, hogy hozzáférjünk a *kocsma* és *sör* komponenshez

# Oracle stílusban REF (hivatkozás) követése

- REF követése implicit a pontban.
- A REF-t nyomon követni: egy „elem” után egy pont, majd a megjelölt objektum mezőjének, amire hivatkozik, követésével kapjuk meg az értéket
- Példa:

```
SELECT ss.beer.name
 FROM Sells ss
 WHERE ss.bar.name = 'Joe''s Bar';
```

# Oracle DEREF művelete - motiváció

- Ha a Joe által értékesített sörökre mint sór objektumok halmazára van szükségünk, megpróbálhatjuk az alábbit:

```
SELECT ss.beer
FROM Sells ss
WHERE ss.bar.name = 'Joe''s Bar';
```

- Legális SQL, de *ss.beer* maga egy hivatkozás, ezért egy zagyvaság.

# Deref használata

- Ahhoz, hogy a **BeerType** objektumait láthassuk:

```
SELECT Deref(ss.beer)
 FROM Sells ss
 WHERE ss.bar.name = 'Joe''s Bar';
```

- Egy ilyen értéket állít elő:

**BeerType('Bud', 'Anheuser-Busch')**

# Metódusok – ORACLE szintaxis

- Az osztályok többek mint adatszerkezetek; lehetnek metódusaik.
- Tanulmányozni fogjuk az Oracle szintaxisát

# Metódus definicíó (Oracle)

- A metódusok deklarálhatjuk a CREATE TYPE-ban
- Definiálhatjuk a CREATE TYPE BODY utasításban
  - Használva a PL/SQL szintaxisát a metódusokra
  - SELF változó arra az objektumra vonatkozik, amelyre a metódust alkalmazni kívánjuk.

# Példa: metódus deklaráció

Adjuk hozzá *priceInYen*-t, *MenuType*-hoz.

```
CREATE TYPE MenuType AS OBJECT (
 bar REF BarType,
 beer REF BeerType,
 price FLOAT,
 MEMBER FUNCTION pricelnYen(rate IN FLOAT) RETURN
 FLOAT,
 PRAGMA RESTRICT_REFERENCES(pricelnYen, WNDS)
); /
```

Oracle ezt nevezi  
metódusnak

Vagyis *priceInYen* nem fogja  
módosítani az adatbázis állapotát

# Metódus definíció – Oracle stílusban

- ◆ A *create-body* utasítás formája:

```
CREATE TYPE BODY <type name> AS
 <method definitions = PL/SQL procedure definitions, using
 “MEMBER FUNCTION” in place of
 “PROCEDURE”>
END;
/
```

# Példa: Metódus definíció

```
CREATE TYPE BODY MenuType AS
```

```
 MEMBER FUNCTION
```

```
 pricelnYen(rate FLOAT) RETURN FLOAT IS
```

```
 BEGIN
```

```
 RETURN rate * SELF.price;
```

```
 END;
```

```
 END;
```

```
/
```

Az (IN)  
nincs a „body”-ban,  
csak a deklarációban

*Csak akkor használjunk  
zárójelet, ha legalább  
egy argumentum van*

# Metódus használata

- ◆ Az objektum neve után legyen egy pont, majd a metódus neve, és végül az argumentumok, ha egyáltalán vannak.

- Példa:

```
SELECT ss.beer.name,
 ss.priceInYen(114.0)
FROM Sells ss
WHERE ss.bar.name = 'Joe''s Bar';
```

# Rendező metódusok: SQL-99

- Mindegyik  $T$  UDT két metódust definiálhat **EQUAL** és **LESSTHAN**.
  - Mindegyik metódus egy  $T$  típus argumentumot kap bemenetként és egy másik  $T$  típusú objektumra alkalmazza.
  - TRUE értéket ad vissza akkor és csak akkor ha a cél objektum = (vagy  $<$ ) mint az az argumentumban szereplő objektum.
- Lehetővé teszi, hogy  $T$  típusú objektumokat hasonlítsunk össze  $=$ ,  $<,>=$ , stb. segítségével a WHERE záradékban és a rendezésben (ORDER BY).

# Rendező metódusok: Oracle

- Bármilyen UDT típusra bármelyik metódust *rendező metódusnak* deklarálhatjuk.
- A rendező metódusok visszatérő értéke  $<0$ ,  $=0$ , vagy  $>0$  lehet, ahogy a SELF objektumhoz viszonyítva az argumentum értéke  $<$ ,  $=$ , vagy  $>$

# Példa: Rendező metódusok deklarálás

- ◆ Rendezd a BarType objektumokat név szerint:

```
CREATE TYPE BarType AS OBJECT (
 name CHAR(20),
 addr CHAR(20),
 ORDER MEMBER FUNCTION before(
 bar2 IN BarType) RETURN INT,
 PRAGMA RESTRICT_REFERENCES(before,
 WNDS, RNDS, WNPS, RNPS)
);
```

Nincs adatbázis/csomag állapot változás. Egy „csomag” eljárások és változók gyűjteménye.

# Példa: Rendező metódusok definiálás

```
CREATE TYPE BODY BarType AS
 ORDER MEMBER FUNCTION
 before(bar2 BarType) RETURN INT IS
 BEGIN
 IF SELF.name < bar2.name THEN RETURN -1;
 ELSIF SELF.name = bar2.name THEN RETURN 0;
 ELSE RETURN 1;
 END IF;
 END;
END;
/
```

# Oracle beágyazott táblák

- Megengedi, hogy a sorok egyes komponensei teljes relációk legyenek.
- Ha  $T$  egy UDT, létrehozhatunk egy  $S$  típust, amelynek az értékei relációk, amelyeknek a sortípusa viszont  $T$ :

```
CREATE TYPE S AS TABLE OF T;
```

# Példa: beágyazott tábla típusok létrehozása

```
CREATE TYPE BeerType AS OBJECT (
 name CHAR(20),
 kind CHAR(10),
 color CHAR(10)
) ;
/
CREATE TYPE BeerTableType AS
 TABLE OF BeerType;
/
```

# Példa -- folytatása

- ◆ BeerTableType-t használjuk Manfs relációban, amelyik a söröket gyártótól tárolja, minden gyártó egy sorban.

```
CREATE TABLE Manfs (
 name CHAR (30) ,
 addr CHAR (50) ,
 beers beerTableType
) ;
```



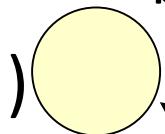
Ez így még nem lesz jó!  
Ld. később a helyes szintaxist!

# A beágyazott relációk eltárolása

- Oracle valójában nem tárolja el a beágyazott relációkat külön relációkként – még ha így is tűnik.
- Ehelyett, egy  $R$  reláció van, amelyben egy A attribútumra az összes beágyazott táblázatot és azok összes sorát eltárolja.
- Deklaráció a CREATE TABLE:  
NESTED TABLE A STORE AS  $R$

# Példa: Beágyazott táblák tárolása

```
CREATE TABLE Manfs (
 name CHAR(30),
 addr CHAR(50),
 beers beerTableType
```



NESTED TABLE beers STORE AS BeerTable;



A pontosvessző (;) vessző használatára figyelni!

# Beágyazott táblák lekérdezése

- Bármely beágyazott táblázat ugyanúgy jeleníthető meg, nyomtatható ki mint bármilyen más érték.
- Azonban ennek az alábbi két értéknek van két típuskonstruktora:
  1. A tábláknak-
  2. A soroknak a táblákban

# Példa: Beágyazott táblák lekérdezése

- Anheuser-Busch söreit keressük ki:

```
SELECT beers FROM Manfs
WHERE name = 'Anheuser-Busch';
```

- Egy értéket eredményez:

BeerTableType(

    BeerType('Bud', 'lager', 'yellow'),

    BeerType('Lite', 'malt', 'pale'),...

)

# Beágyazott táblán belüli lekérdezés

- Egy beágyazott táblát hagyományos relációvá lehet konvertálni a TABLE() alkalmazásával
- Ezt a relációt, ugyanúgy mint bármely másikat, a FROM záradékban lehet alkalmazni.

# Példa: TABLE() használata

- ◆ Keresd meg Anheuser-Busch által gyártott „ale”-ket:

```
SELECT bb.name
```

```
FROM TABLE(
```

```
 SELECT beers
```

```
 FROM Manfs
```

```
 WHERE name = 'Anheuser-Busch'
```

```
) bb
```

```
WHERE bb.kind = 'ale';
```

Beágyazott tábla  
Anheuser-Busch  
sörökre

Alias a névnélküli  
beágyazott táblára

# Még egy példa

```
CREATE TYPE cim_t AS OBJECT (
 utca VARCHAR2(30),
 varos VARCHAR2(20),
 ir_szam CHAR(4));
/
CREATE TYPE cim_tab IS TABLE OF cim_t;
/
CREATE TABLE vevok (
 vevo_id NUMBER,
 cimek cim_tab)
NESTED TABLE cimek STORE AS vevo_cimek;
```

# Még egy példa

```
INSERT INTO vevok VALUES (1,
 cim_tab(
 cim_t('Malom utca 2.', 'Varos', '9999'),
 cim_t('Nagy utca 1.', 'Falu', '0000')
));
INSERT INTO vevok VALUES (2,
 cim_tab(
 cim_t('Pajta utca 1.', 'Varos', '9999')));
```

# Még egy példa

```
SELECT * FROM vevok;
```

```
vevo_id cimek
```

```

1 CIM_TAB(CIM_T('Malom utca 2.', 'Varos', '9999'),
 CIM_T('Nagy utca 1.', 'Falu', '0000'))
2 CIM_TAB(CIM_T('Pajta utca 1.', 'Varos', '9999'))
```

```
SELECT v.vevo_id, u.* FROM vevok v, TABLE(v.cimek) u;
```

```
vevo_id utca varos ir_szam
```

```

1 Malom utca 2. Varos 9999
1 Nagy utca 1. Falu 0000
2 Pajta utca 1. Varos 9999
```

# Relációk beágyazott táblává alakítása

- Bármely reláció megfelelő számú attribútummal és azok illeszkedő adattípusaival egy beágyazott tábla értékei lehetnek.
- Használjuk a CAST(MULTISET(...) AS <type> ) utasítást a reláción azért, hogy a helyes adattípussal rendelkező értékeivel egy beágyazott táblázattá alakítsuk.

## Példa: CAST --- 1

- Tegyük fel, hogy a `Beers(beer, manf)` olyan reláció, hogy a `sör (beer)` egy `BeerType` típusú objektum és `manf` pedig egy string – a sör gyártója.
- Egy új sort akarunk beilleszteni a `Manfs` –ba , „Pete’s Brewing Co.” -t, mint (gyártó) nevet, és Pete által gyártott sörök halmazát.

## Példa: CAST --- 2

```
INSERT INTO Manfs VALUES (
```

```
 'Pete''s', 'Palo Alto',
```

```
 CAST(
```

```
 MULTISET(
```

```
 SELECT bb.beer
 FROM Beers bb
 WHERE bb.manf = 'Pete''s'
```

```
) AS BeerTableType
```

```
)
);
```

Pete söreire  
BeerType  
objektumok  
halmaza

Az objektumok halmazát  
beágyazott relációvá alakítjuk

# Fájlszervezés

Adatbázisok tervezése, megvalósítása  
és menedzselése

# Kezdetek

- **Célok:**
  - gyors lekérdezés,
  - gyors adatmódosítás,
  - minél kisebb tárolási terület.
- **Nincs általánosan legjobb optimalizáció.** Az egyik cél a másik rovására javítható (például indexek használatával csökken a keresési idő, nő a tárméret, és nő a módosítási idő).

# Költségek

- **Hogyan mérjük a költségeket?**
- Memória műveletek nagyságrenddel gyorsabbak, mint a háttértárolóról beolvasás, kiírás.
- Az író-olvasó fej nagyobb adategységeket (blokkokat) olvas be.
- A blokkméret függhet az operációs rendszertől, hardvertől, adatbázis-kezelőtől.
- A blokkméretet fixnek tekintjük. Oracle esetén 8K az alapértelmezés.
- **Feltételezzük, hogy a beolvasás, kiírás költsége arányos a háttértároló és memória között mozgatott blokkok számával.**

# Blokkok

- A **blokkok** tartalmaznak:
  - leíró fejlécet (rekordok száma, struktúrája, fájlok leírása, belső/külső mutatók (hol kezdődik a rekord, hol vannak üres helyek, melyik a következő blokk, melyik az előző blokk, statisztikák (melyik fájlból hánny rekord szerepel a blokkban)),
  - rekordokat (egy vagy több fájlból),
  - üres helyeket.
- Feltesszük, hogy **B** darab blokkunk van.

# Rekordok

- A fájl rekordokból áll.
- A **rekordok szerkezete** eltérő is lehet.
- A rekord tartalmaz:
  - **leíró fejlécet** (rekordstruktúra leírása, belső/külső mutatók, (hol kezdődik egy mező, melyek a kitöltetlen mezők, melyik a következő rekord, melyik az előző rekord), törlési bit, statisztikák),
  - **mezőket**, melyek üresek, vagy adatot tartalmaznak.
- A rekordhossz lehet:
  - állandó,
  - változó (változó hosszú mezők, ismétlődő mezők miatt).
- Az egyszerűség kedvéért **feltesszük, hogy állandó hosszú rekordokból áll a fájl**, melyek hossza az átlagos rekordméretnek felel.

# Milyen lekérdezések?

- Milyen lekérdezéseket vizsgáljunk?
- A relációs algebrai kiválasztás felbontható atomi kiválasztásokra, így elég ezek költségét vizsgálni.
- A legegyszerűbb kiválasztás:
  - $A=a$  ( $A$  egy keresési mező,  $a$  egy konstans)
- Az esetek vizsgálatánál az is számít, hogy az  $A=a$  feltételnek megfelelő rekordokból lehet-e több, vagy biztos, hogy csak egy lehet.
- Fel szoktuk tenni, hogy az  $A=a$  feltételnek eleget tevő rekordokból nagyjából egyforma számú rekord szerepel különböző  $a$ -akra. (Ez az **egyenletességi feltétel**.)

# Mivel foglalkozunk majd?

- A következő fájlszervezési módszereket fogjuk megvizsgálni:
  - B<sup>+</sup>-fa, B<sup>\*</sup>-fa
  - bitmap index.
- Módosítási műveletek:
  - beszúrás (insert)
  - frissítés (update)
  - törlés (delete)
- Itt nem foglalkozunk azzal, hogy a beolvasott rekordokat bent lehet tartani a memóriában, későbbi keresések céljára.

# Indexek

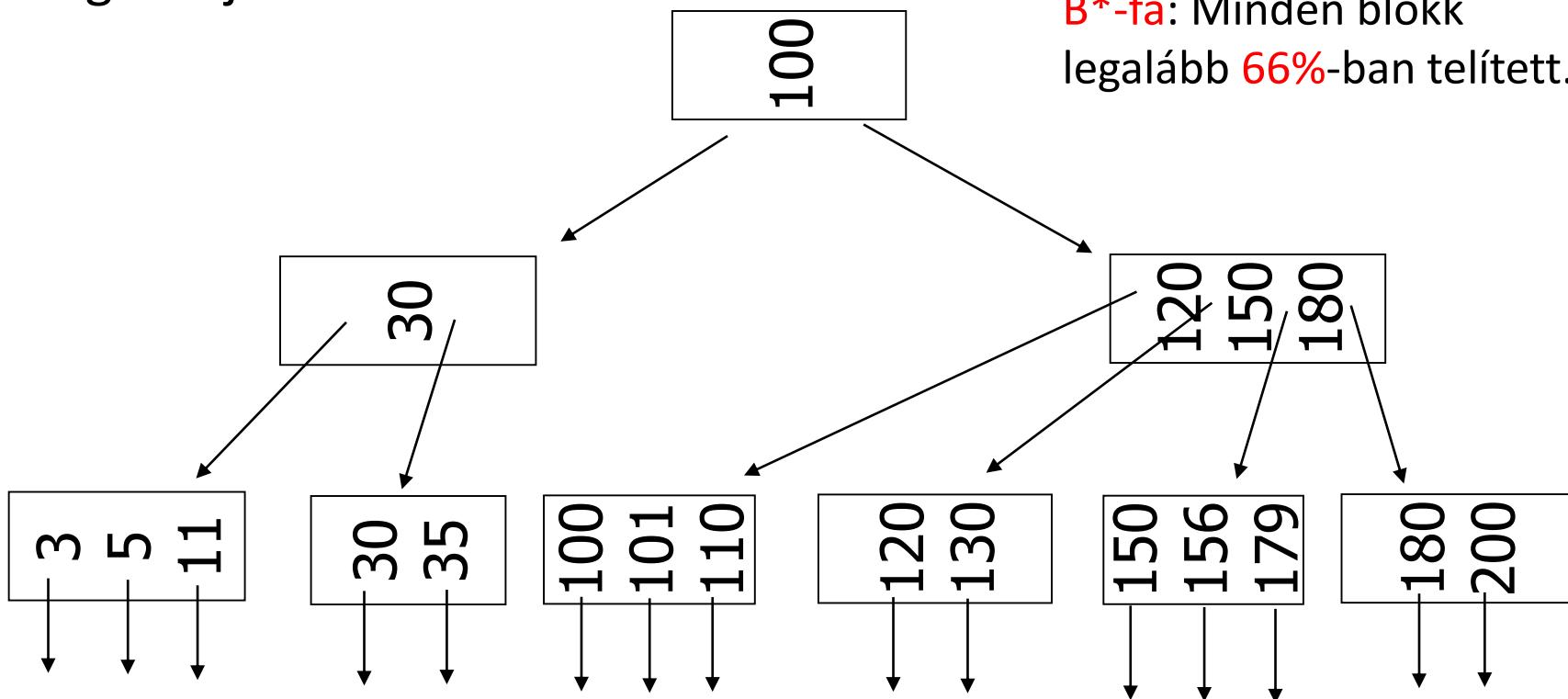
- Indexek használata:
  - keresést gyorsító segédstruktúra,
  - több mezőre is lehet indexet készíteni,
  - az index tárolása növeli a tármejet,
  - nem csak a főfájlt, hanem az indexet is karban kell tartani, ami plusz költséget jelent.
- Az indexrekordok szerkezete:
  - **(a,p)**, ahol a egy érték az indexelt oszlopban, p egy blokkmutató, arra a blokkra mutat, amelyben az A=a értékű rekordot tároljuk (vagy többszintű index esetén esetleg egy másik indexblokkra).
  - az index minden rendezett az indexértékek szerint.

# B-fák

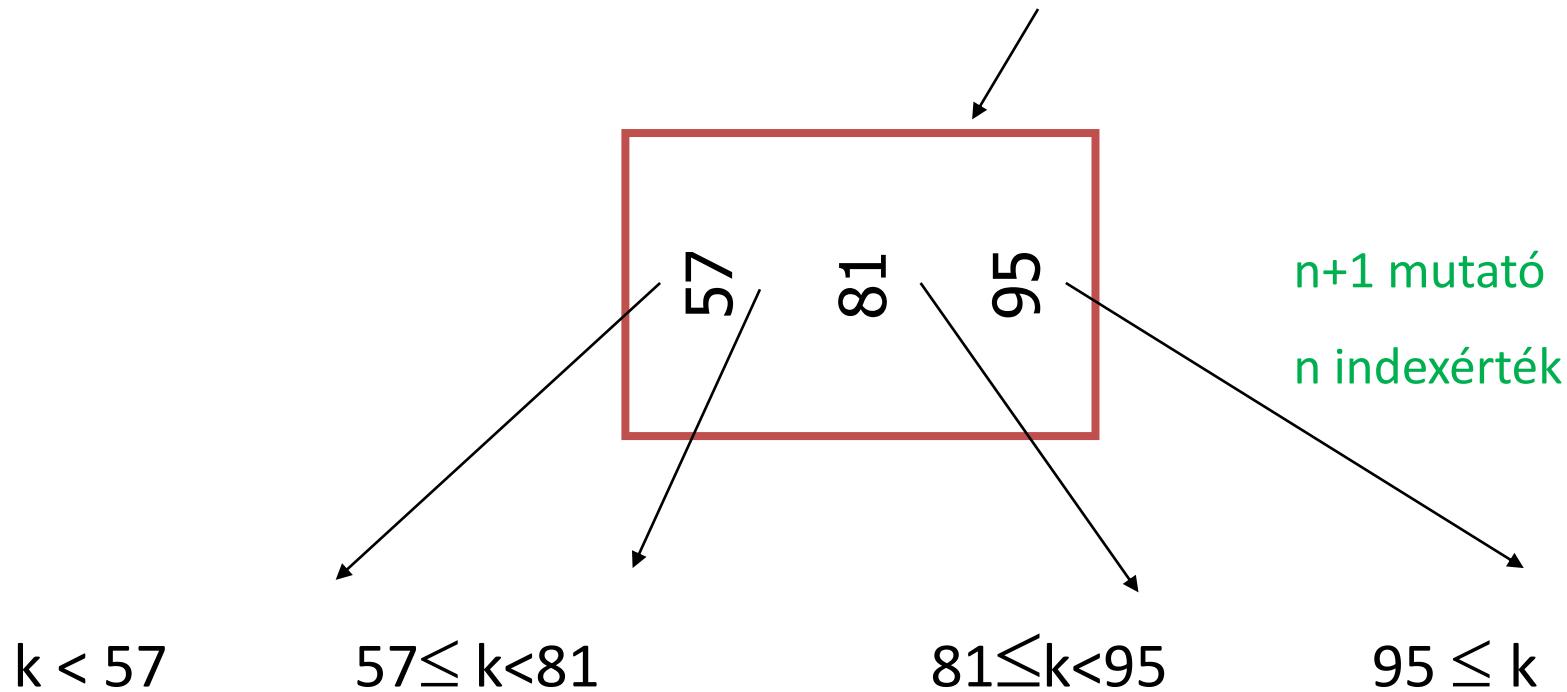
A többszintű indexek közül  
a **B<sup>+</sup>-fák**, **B<sup>\*</sup>-fák** a  
legelterjedtebbek.

**B<sup>+</sup>-fa:** minden blokk legalább 50%-ban telített.

**B<sup>\*</sup>-fa:** minden blokk  
legalább 66%-ban telített.

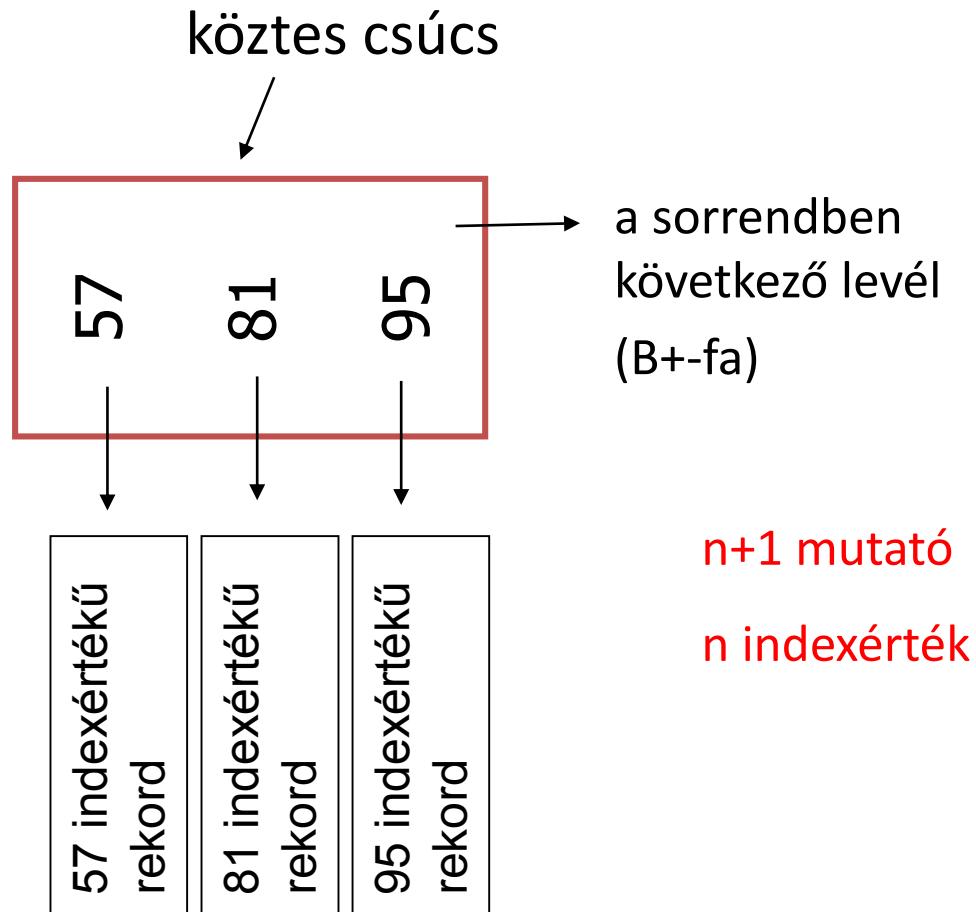


# Köztes (nem-levél) csúcs szerkezete



Ahol **k** a mutató által meghatározott részgráfban szereplő tetszőleges indexérték.

# Levél csúcs szerkezete



# Felépítésre vonatkozó szabályok

- Szabályok (felte tesszük, hogy egy csúcs  $n$  értéket és  $n+1$  mutatót tartalmazhat):
  - a gyökérben legalább két mutatónak kell lennie (kivéve, ha a B-fa ( $B+$ -fa) egyetlen bejegyzést tartalmaz),
  - a levelekben az utolsó mutató a következő (jobboldali) levére mutat, legalább  $\lfloor(n+1)/2\rfloor$  mutatónak használatban kell lennie,
  - köztes pontokban minden mutató a B-fa következő szintjére mutat, közülük legalább  $\lceil(n+1)/2\rceil$  darabnak használatban kell lennie,
  - ha egy mutató nincs használatban úgy vesszük, mintha NILL mutató lenne.

# Beszúrás I.

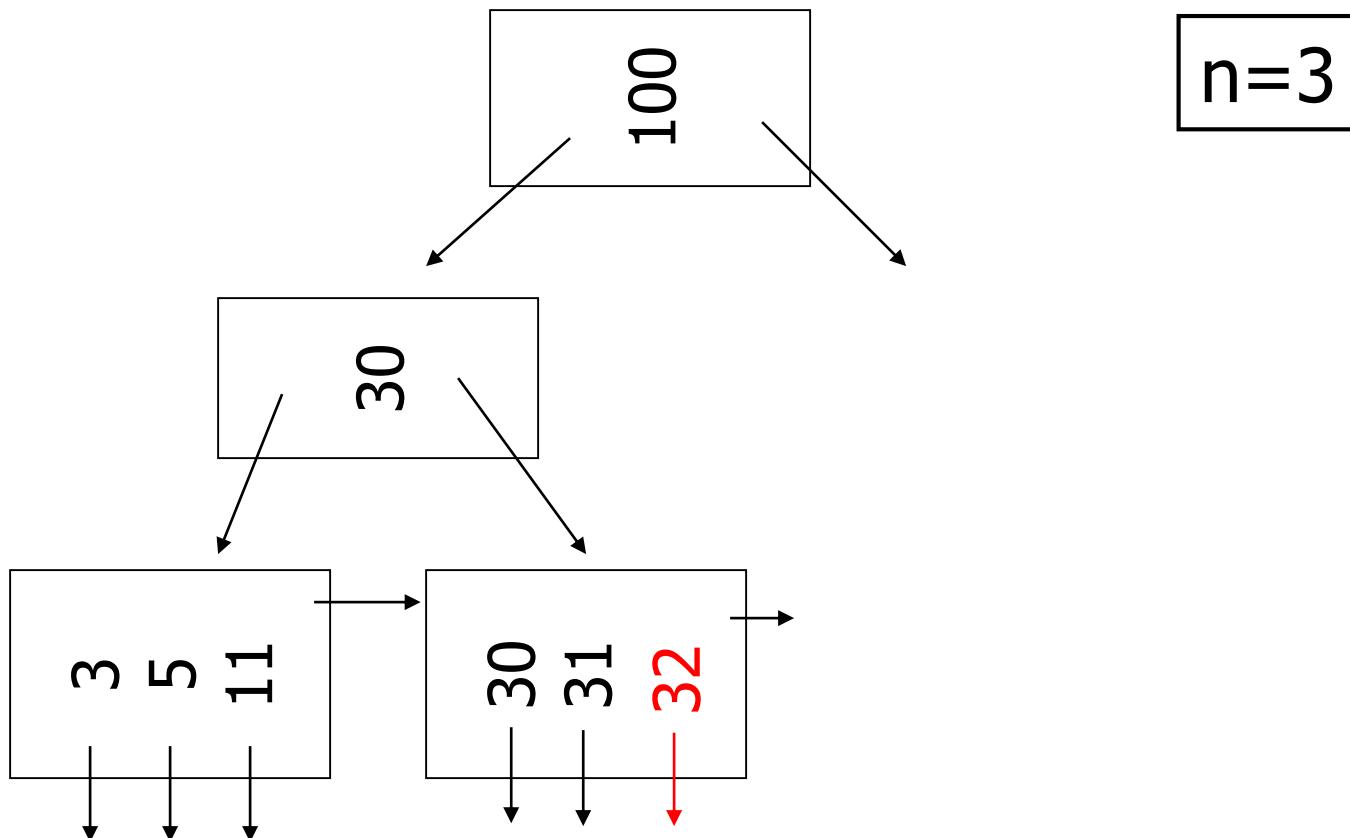
- Ha van szabad hely az új kulcs számára a megfelelő levélben, beszúrjuk.
- Ha nincs, kettévágjuk a levelet, és szétosztjuk a kulcsokat a két új levél között, így minden kettő félre lesz telítve, vagy éppen csak egy kicsit jobban.
- Az eggyel feljebb lévő szint megfelelő csúcsát ennek megfelelően kell kiigazítani.

# Beszúrás II.

- Egy csúcs szétvágása tehát hatással lehet a fölötté lévő szintre is. Itt az előbbi két pontban megadott stratégiát alkalmazzuk rekurzívan.
- Itt viszont: ha N olyan belső csúcs, aminek kapacitása  $n$  kulcs,  $n+1$  mutató és most az  $(n+2)$ . mutatót illesztenénk be, akkor szintén létrehozunk egy új M pontot. N-ben most marad  $\lceil n/2 \rceil$  kulcs, M-ben lesz  $\lfloor n/2 \rfloor$  kulcs, a "középen lévő" kulcs pedig az eggyel fentebbi szintre kerül, hogy elválassza N-t és M-t egymástól.
- Ha a gyökérbe nem tudunk beszúrni, mert nincs hely, akkor szétvágjuk a gyökeret két új csúcsra, és „fölöttük” létrehozunk egy új gyökeret, aminek két bejegyzése lesz.

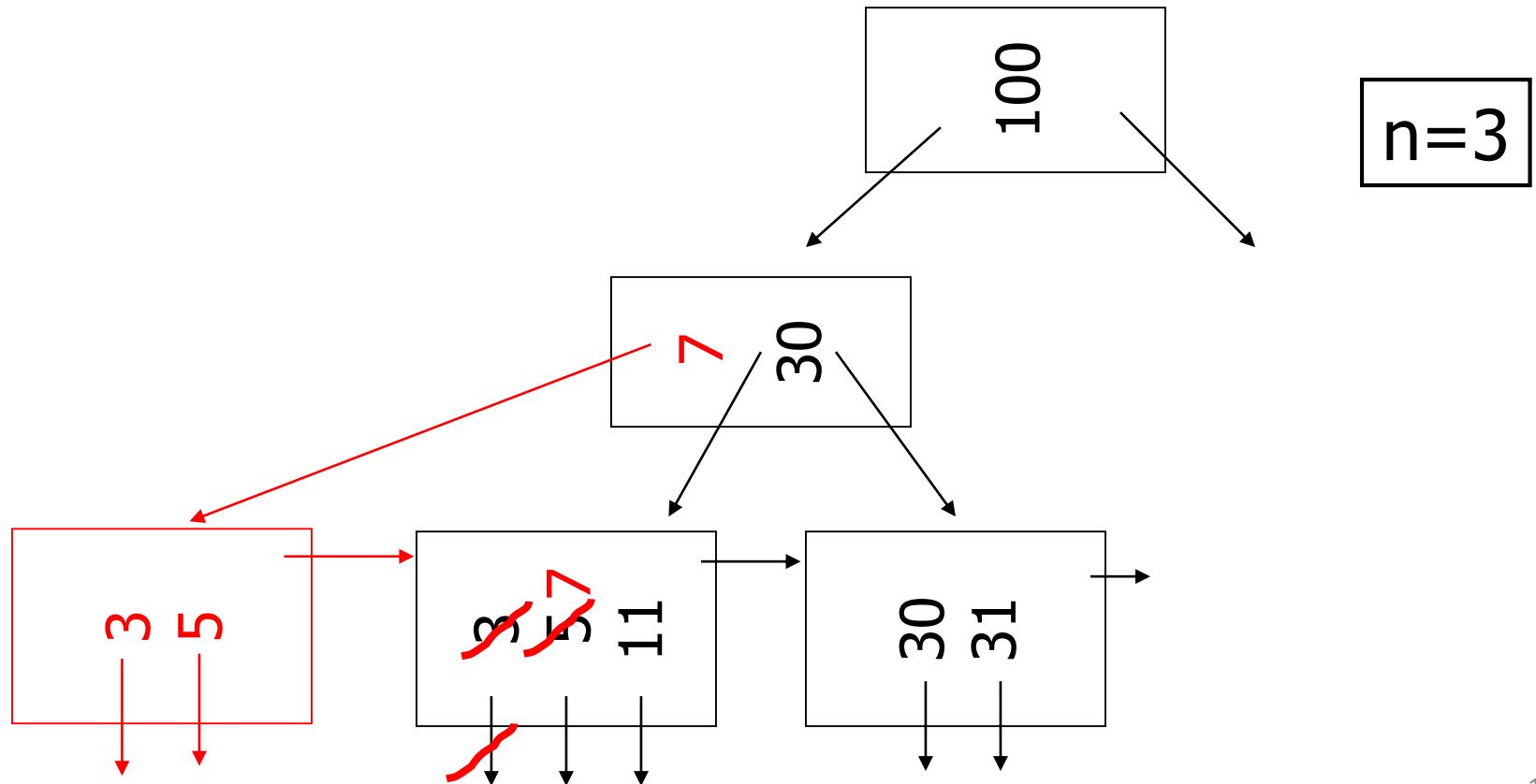
# Példa beszúrásra I.

Szúrjuk be a 32-es indexértékű rekordot!



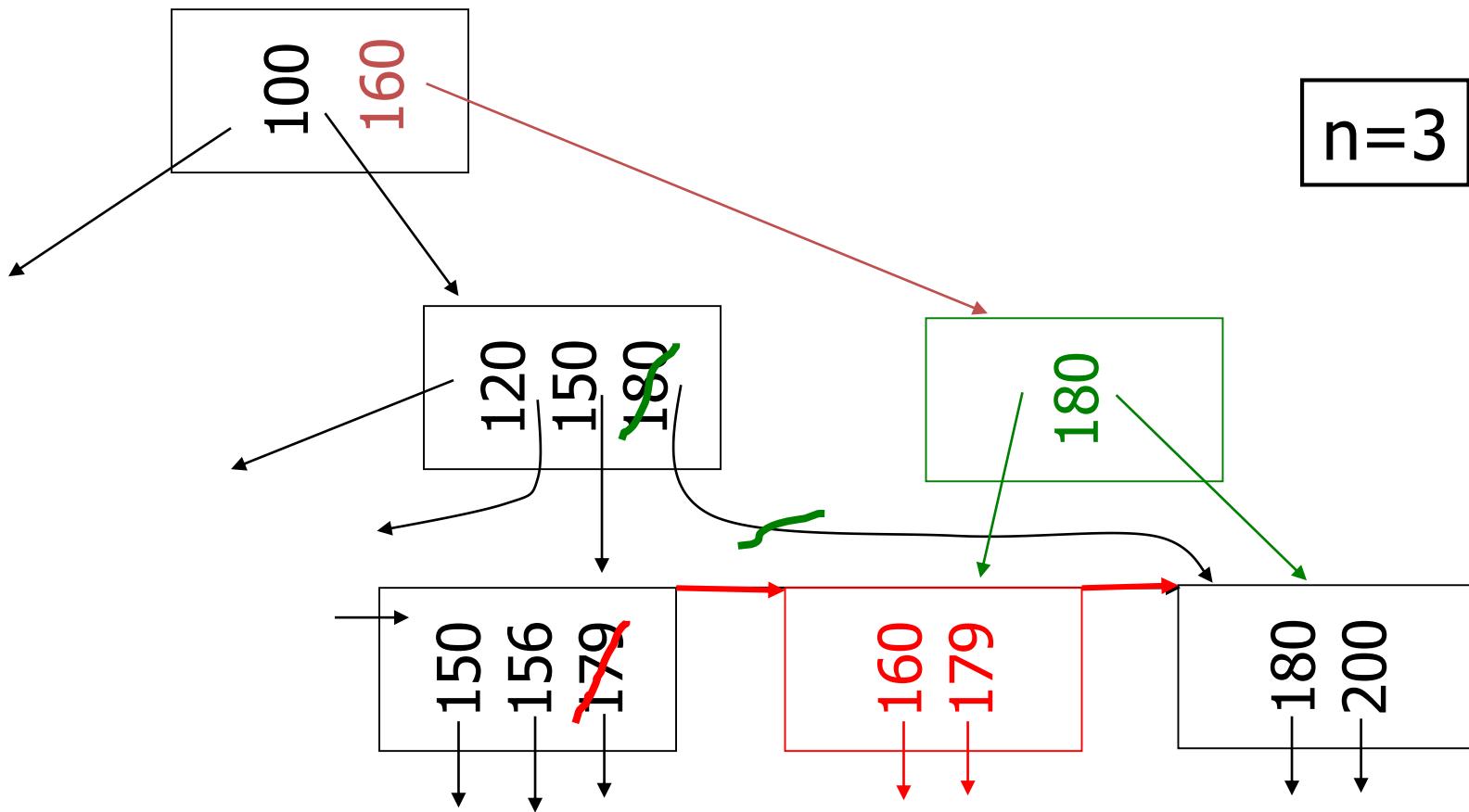
# Példa beszúrásra II.

Szúrjuk be a 7-es indexértékű rekordot!



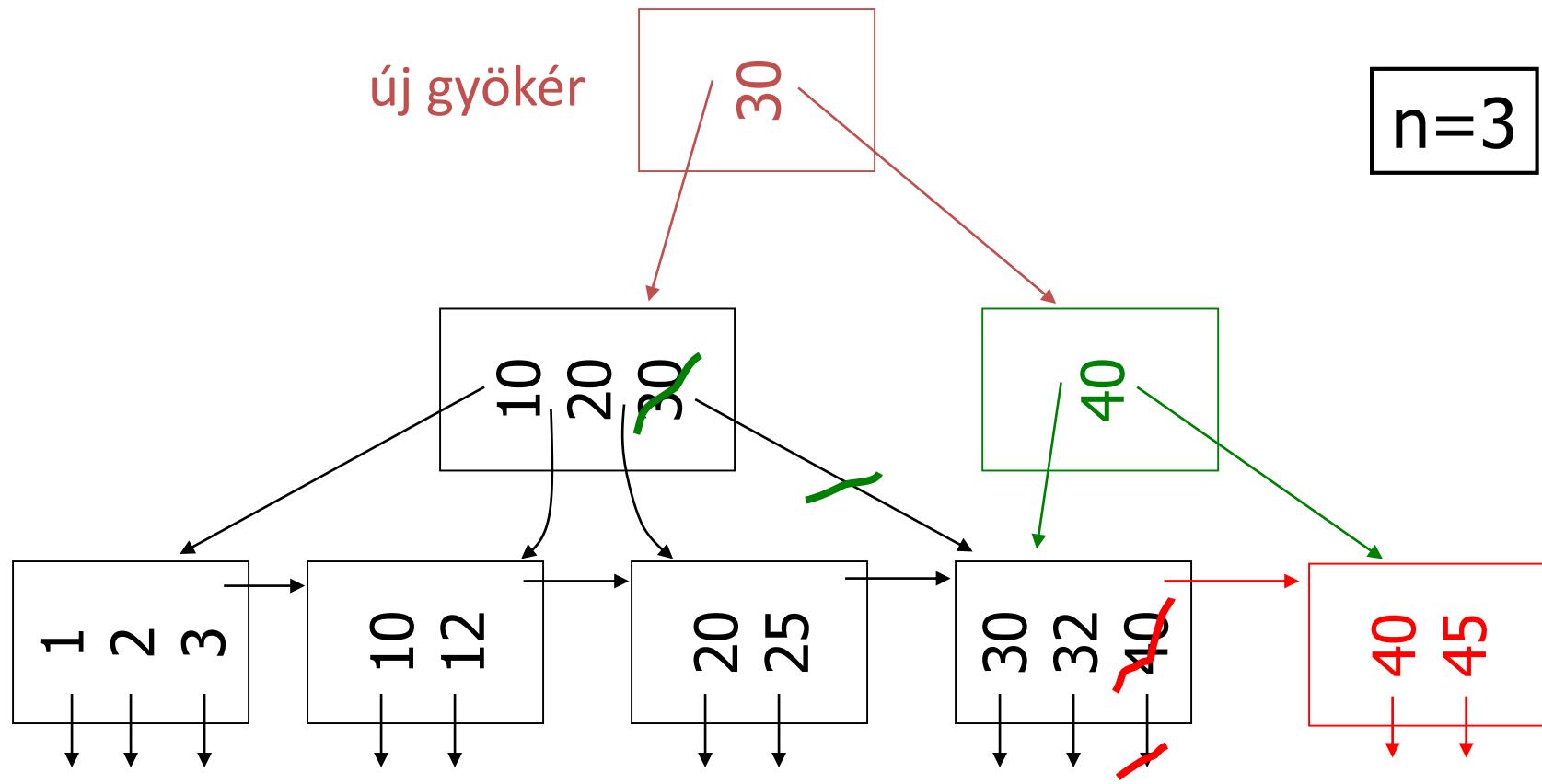
# Példa beszúrásra III.

Szúrjuk be a 160-as indexértékű rekordot!



# Példa beszúrásra IV.

Szúrjuk be a 45-ös indexértékű rekordot!

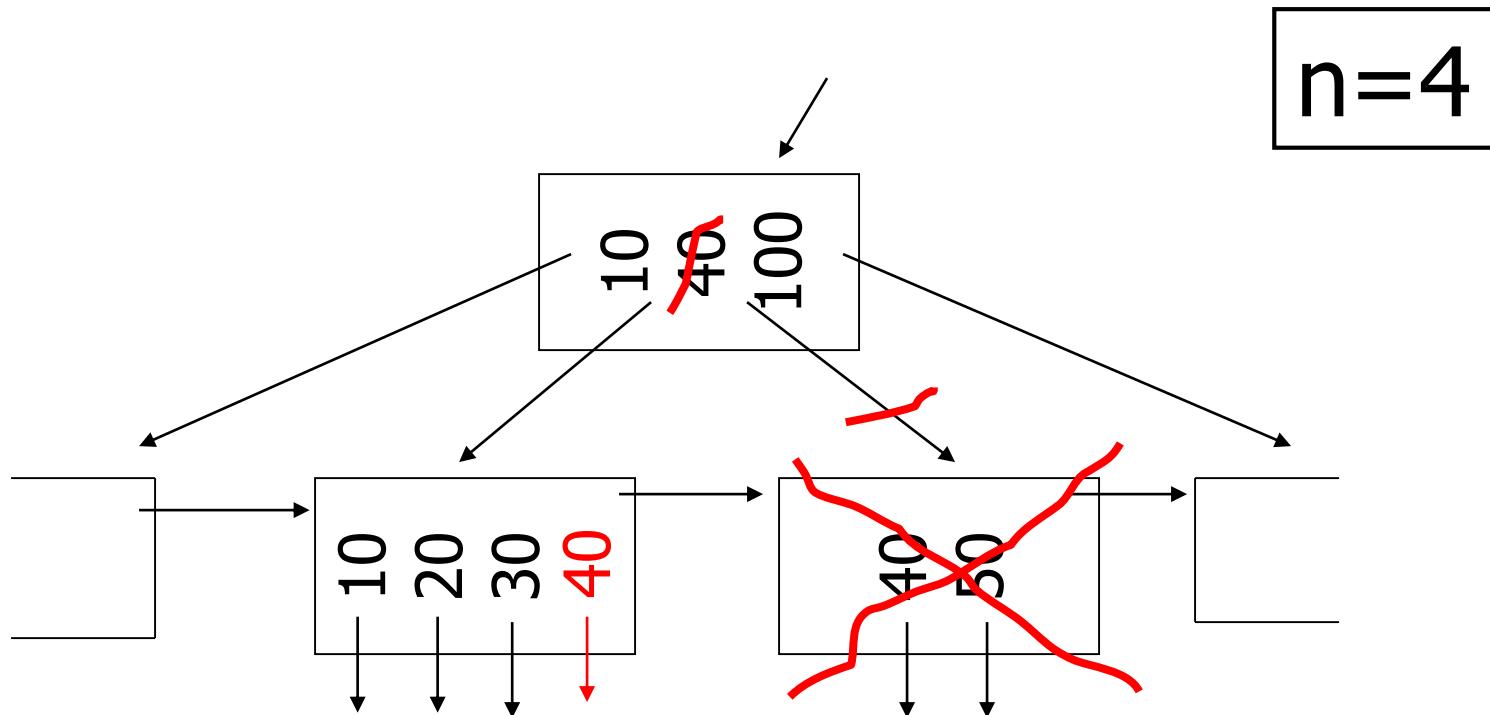


# Törlés

- Ha a törlés megtörtént az  $N$  pontban, és  $N$  még mindig megfelelően telített, akkor készen vagyunk.
- Ha  $N$  már nem tartalmazza a szükséges minimum számú kulcsot és mutatót, akkor:
  - ha  $N$  valamelyik testvérével összevonható, akkor vonjuk össze. A szülőcsúcsban törlődik ekkor egy elem, s ez további változtatásokat eredményezhet.
  - Ha nem vonható össze, akkor  $N$  szomszédos testvérei több kulcsot és mutatót tartalmaznak, mint amennyi a minimumhoz szükséges, akkor egy kulcs-mutató párt áttehetünk  $N$ -be. (Baloldali testvér esetén a legutolsó, jobboldali testvér esetén a legelső kulcs-mutató párt.) A változást a szülő csúcson is „regisztrálni kell”.

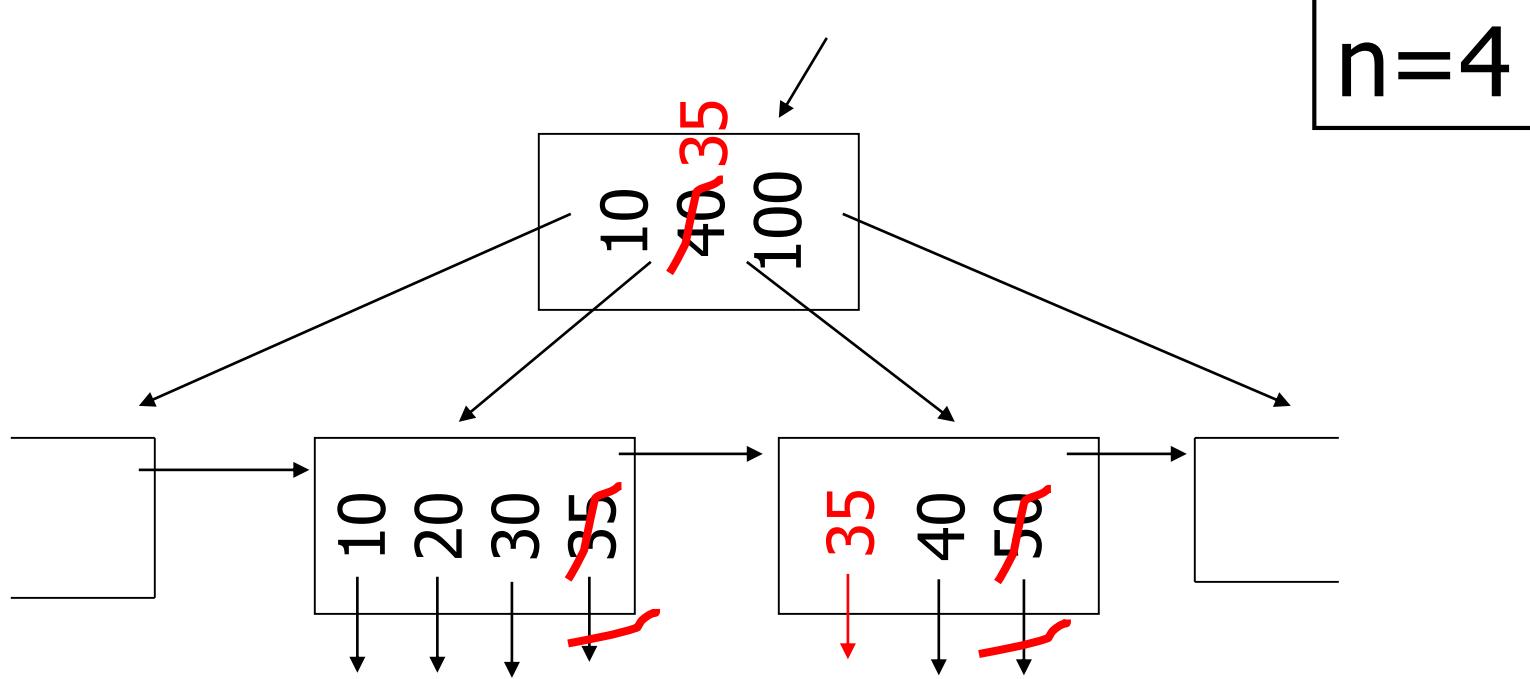
# Példa törlésre I.

Töröljük az 50-es indexértékű rekordot!



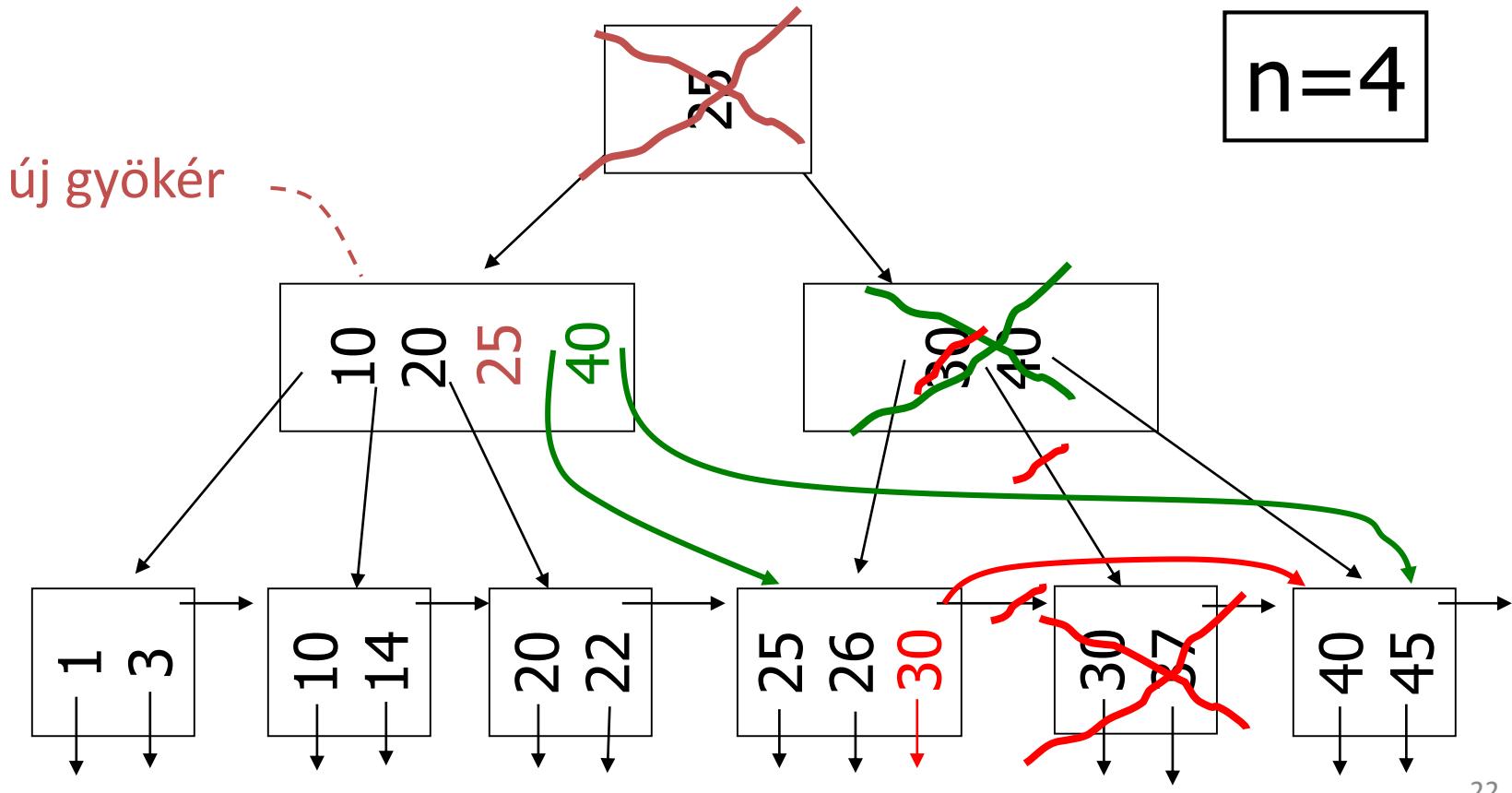
# Példák törlésre II.

Töröljük az 50-es indexértékű rekordot!



# Törlés III.

Töröljük a 37-es indexértékű rekordot!



# Bitmap indexek

## Személy

| név       | nem   | kor | kereset |
|-----------|-------|-----|---------|
| Péter     | férfi | 57  | 350000  |
| Dóra      | nő    | 25  | 30000   |
| Salamon   | férfi | 36  | 350000  |
| Konrád    | férfi | 21  | 30000   |
| Erzsébet  | nő    | 20  | 30000   |
| Zsófia    | nő    | 35  | 160000  |
| Zsuzsanna | nő    | 35  | 160000  |

| érték | vektor  |
|-------|---------|
| férfi | 1011000 |
| nő    | 0100111 |

| érték  | vektor  |
|--------|---------|
| 30000  | 0101100 |
| 160000 | 0000011 |
| 350000 | 1010000 |

# Bitmap indexek haszna

```
SELECT COUNT(*)
```

```
FROM személy
```

```
WHERE nem='nő' and kereset = 160000;
```

- 0100111 AND 0000011 = 0000011, az eredmény: 2.

```
SELECT név
```

```
FROM személy
```

```
WHERE kereset > 100000;
```

- 0000011 (160000) OR 1010000 (350000) = 1010011, azaz az 1., 3., 6. és 7. rekordokat tartalmazó blokko(ka)t kell beolvasni.

# Tömörítés I.

- Ha a táblában **n** rekord van, a vizsgált attribútum pedig **m** különböző értéket vehet fel, ekkor, ha m nagy, a bitmap index **túl naggyá is nőhet** ( $n*m$  méret). Ebben az esetben viszont a bitmap indexben **az egyes értékekhez tartozó rekordokban kevés az 1-es**. A **tömörítési technikák** általában csak ezeknek az 1-eseknek a helyét határozzák meg.
- Tegyük fel, hogy i db 0-t követ egy 1-es. Legegyszerűbb megoldásnak tűnik, ha i-t binárisan kódoljuk. Ám **ez a megoldás még nem jó**: (a 000101 és 010001 vektorok kódolása is 111 lenne).
- Tegyük fel, hogy **i** binárisan ábrázolva **j** bitből áll. Ekkor először írunk le  $j-1$  db 1-est, **majd egy 0-t**, és csak ez után i bináris kódolását.
- **Példa:** a 000101 kódolása: 101101, a 010001 kódolása: 011011.

# Tömörítés II.

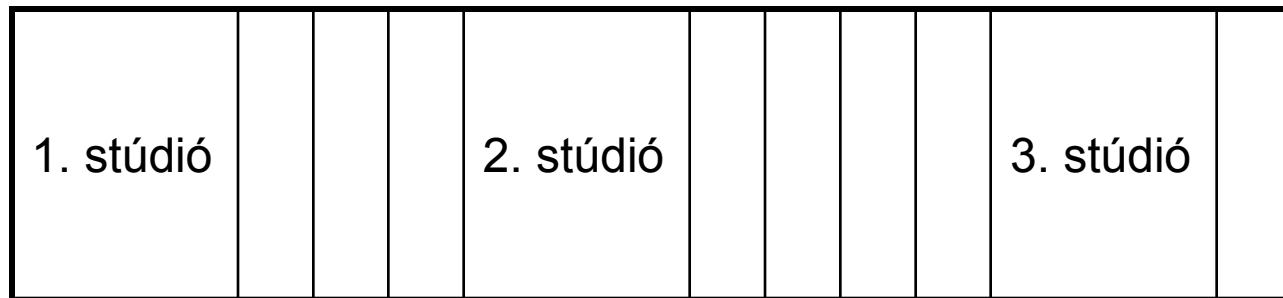
- Állítás: a kód így egyértelművé válik.
- Tegyük fel, hogy  $m=n$ , azaz minden rekordérték különböző a vizsgált attribútumban.
- Ekkor, mivel a bitmap indexekben  $n$  hosszú rekordokról van szó, egy rekord kódolása nagyságrendileg legfeljebb  $2\log_2 n$ . Az indexet alkotó teljes bitek száma pedig nagyságrendileg  $2n\log_2 n$ , az  $n^2$  helyett.

# Klaszter

- **Klaszterszervezés** esetén a **két tábla** közös oszlopain megegyező sorok egy blokkban, vagy fizikailag egymás utáni blokkokban helyezkednek el.
- **CÉL:** összekapcsolás esetén az összetartozó sorokat soros beolvasással megkaphatjuk.

# Példa klaszterre

- Film (cím, év, hossz, stúdiónév)
- Stúdió (név, studiócím, elnök)



1. stúdióban  
készült filmek.



2. stúdióban  
készült filmek.



3. stúdióban  
készült filmek.

Pl. gyakori a: **SELECT cím, év  
FROM film, stúdió  
WHERE cím LIKE '%Moszkva%' AND stúdiónév = név;** jellegű  
lekérdezés.

# SQL jogosultság-kezelés

Privilégiumok

Grant és Revoke

Grant Diagrammok

# Jogosultság-kezelés

- Egy fájlrendszer általában jogosultságokat rendel az általa kezelt objektumokhoz.
  - Tipikusan olvasható, írható, végrehajtási jogosultságokról van szó.
- Ugyanakkor bizonyos „résznevőkhöz” sorolja ezeket a jogosultságokat.
  - Például rendszergazda, egy korlátozott jogosultságokkal rendelkező felhasználó stb.

## Jogosultságok – (1)

- Az SQL-ben több fajta jogosultság és adatobjektum (pl. relációs táblák) létezik, mint egy tipikus fájlrendszerben.
- Összességében 9 jogosultság, ezek némelyike egy reláció egyetlen attribútumára is megadható.

## Jogosultságok – (2)

- Néhány relációra vonatkozó jogosultság:
  1. **SELECT** = a reláció lekérdezésének joga.
    - Lehet, hogy egyetlen attribútumra vonatkozik.
  2. **INSERT** = sorok beszúrásának joga.
    - ◆ Lehet, hogy egyetlen attribútumra vonatkozik.
  3. **DELETE** = sorok törlésének joga.
  4. **UPDATE** = sorok módosításának a joga.
    - ◆ Szintén lehet, hogy egy attribútumra vonatkozik.

## Példa: jogosultságok

- Az alábbi utasítás esetében:

INSERT INTO sörok(név)

SELECT sör FROM felszolgál

WHERE NOT EXISTS

(SELECT \* FROM sörok  
WHERE név = sör);

azok a sörok, amelyek még nincsenek benne a sörok táblában. A beszúrás után a gyártó értéke NULL.

- Az utasítás végrehajtásához szükséges: SELECT jogosultság a felszolgál és sörok táblába és INSERT jog a sörok tábla név attribútumára vonatkozóan.

# Adatbázis objektumok

- Jogosultságokat nézetekre és materializált nézetekre vonatkozóan is megadhatunk.
- Egy másik fajta jogosultság lehet pl. adatbázis objektumok létrehozásának a joga: pl. táblák, nézetek, triggerek.
- A nézettáblák segítségével tovább finomíthatjuk az adatokhoz való hozzáférést.

## Példa: nézettáblák és jogosultságok

- Tegyük fel, hogy nem szeretnénk SELECT jogosultságot adni az Dolgozók(név, cím, fizetés) táblában.
- Viszont a BiztDolg nézettáblán már igen:

```
CREATE VIEW BiztDolg AS
SELECT név, cím FROM Dolgozók;
```

- A BiztDolg nézettáblára vonatkozó kérdésekhez nem kell SELECT jog a Dolgozók táblán, csak a BiztDolg nézettáblán.

# Jogosultsági azonosítók

- A felhasználókat egy *jogosultsági azonosító (authorization ID)* alapján azonosítjuk, általában ez a bejelentkezési név.
- Külön jogosultsági azonosító a PUBLIC.
  - A PUBLIC jogosultság minden felhasználó számára biztosítja az adott jogot.

# Jogosultságok megadása

- A magunk készítette objektumok esetében az összes jogosultsággal rendelkezünk.
- Másoknak is megadhatunk jogosultságokat, például a PUBLIC jogosultsági azonosítót is használhatjuk.
- A WITH GRANT OPTION utasításrész lehetővé teszi, hogy aki megkapta a jogosultságot, tovább is adhassa azt.

# A GRANT utasítás

- Jogosultságok megadásának szintaktikája:
  - GRANT <jogosultságok listája>
  - ON <reláció vagy másféle objektum>
  - TO <jogosultsági azonosítók listája>;
- Ehhez hozzáadható:
  - WITH GRANT OPTION

## Példa: GRANT

- GRANT SELECT, UPDATE (ár)  
ON Felszolgál  
TO sally;
- Ez után Sally kérdéseket adhat meg a Felszolgál táblára vonatkozóan és módosíthatja az ár attribútumot.

## Példa: Grant Option

```
GRANT UPDATE ON Felszolgál TO sally
WITH GRANT OPTION;
```

- Ez után Sally módosíthatja a Felszolgál táblát és tovább is adhatja ezt a jogosultságot.
  - Az UPDATE jogosultságot korlátozottan is továbbadhatja: UPDATE (ár) ON Felszolgál.

# Jogosultságok visszavonása

REVOKE <jogosultságok listája>

ON <reláció vagy más objektum>

FROM <jogosultsági azonosítók listája>;

- Az általunk kiadott jogosultságok ez által visszavonódnak.
  - De ha máshonnan is megkapták ugyanazt a jogosultságot, akkor az még megmarad.

# REVOKE opciói

- A REVOKE utasításhoz a két opció közül valamelyiket még hozzá kell adnunk:
  1. **CASCADE**. Azok a jogosultságok, melyeket az adott ki a visszavonandó jogosultságon keresztül, akitől éppen visszavonjuk az adott jogosultságot, szintén visszavonódnak.
  2. **RESTRICT**. A visszavonás nem hajtódik végre, amíg a visszavonandó jogosultságtól függő jogosultságok is vannak. Először ezeket kell megszüntetni.

# Grant diagramok

- Pontok = felhasználó/jogosultság/grant option?/tulajdonosa?
- UPDATE ON R, UPDATE(a) ON R, és UPDATE(b) ON R három különböző pontot adnak.
- SELECT ON R és SELECT ON R WITH GRANT OPTION szintén.
- Az  $X \rightarrow Y$  él jelentése: az X pontot használtuk az Y pont jogosultságának megadására.

# Hogy néznek ki a pontok?

- Az  $AP$  pont az  $A$  felhasználói azonosítójú felhasználó  $P$  jogát jelenti.
  - $P^* = P$  jogosultság grant option opcióval.
  - $P^{**} =$  a  $P$  jog tulajdonosi viszonyból származik.
    - A  $^{**}$  jelölésből a grant option opció is következik.

## Élek kezelése – (1)

- Amikor A megadja a P jogot B-nek, AP \* vagy AP \*\*-ből húzunk egy élt BP -be.
  - Vagy BP \* ha grant option is szerepel.
- Ha A a P jogosultságának egy részét adja meg, legyen ez Q [például UPDATE(a) on R, ahol P : UPDATE ON R], akkor az él BQ vagy BQ \* pontba lesz.

## Élek kezelése – (2)

- **Alapvető szabály:** a  $C$  felhasználó rendelkezik a  $Q$  jogosultsággal egészen addig, amíg létezik út az  $XP^{**}$  pontból  $CQ$ ,  $CQ^*$  vagy  $CQ^{**}$ -ba és  $P$  egy  $Q$ -t magában foglaló jogosultság.
  - Itt  $P$  lehet  $Q$  és  $X$  lehet  $C$ .

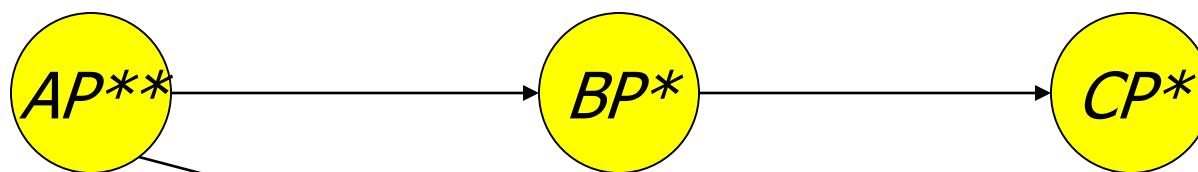
## Élek kezelése – (3)

- Ha  $A$  visszavonja a  $P$  jogot  $B$ -től a CASCADE opcióval, ki kell törölni az  $AP - BP$  élt.
- De, ha  $A$  RESTRICT opciót használ és a  $BP$  pontból indul ki él, akkor vissza kell utasítani a kérést a gráfot változatlanul hagyva.

## Élek kezelése – (4)

- A gráf frissítésekor, minden pontra meg kell nézni, hogy elérhető-e egy \*\* pontból.
- Ha nincs ilyen út, az adott pont egy visszavont jogosultságot reprezentál, és ki kell törölni.

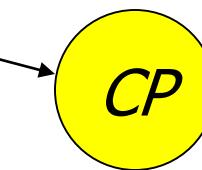
## Példa: Grant diagram



*A* tulajdonosa  
annak az  
objektumnak,  
amire *P*  
vonatkozik

A: GRANT P  
TO B WITH  
GRANT OPTION

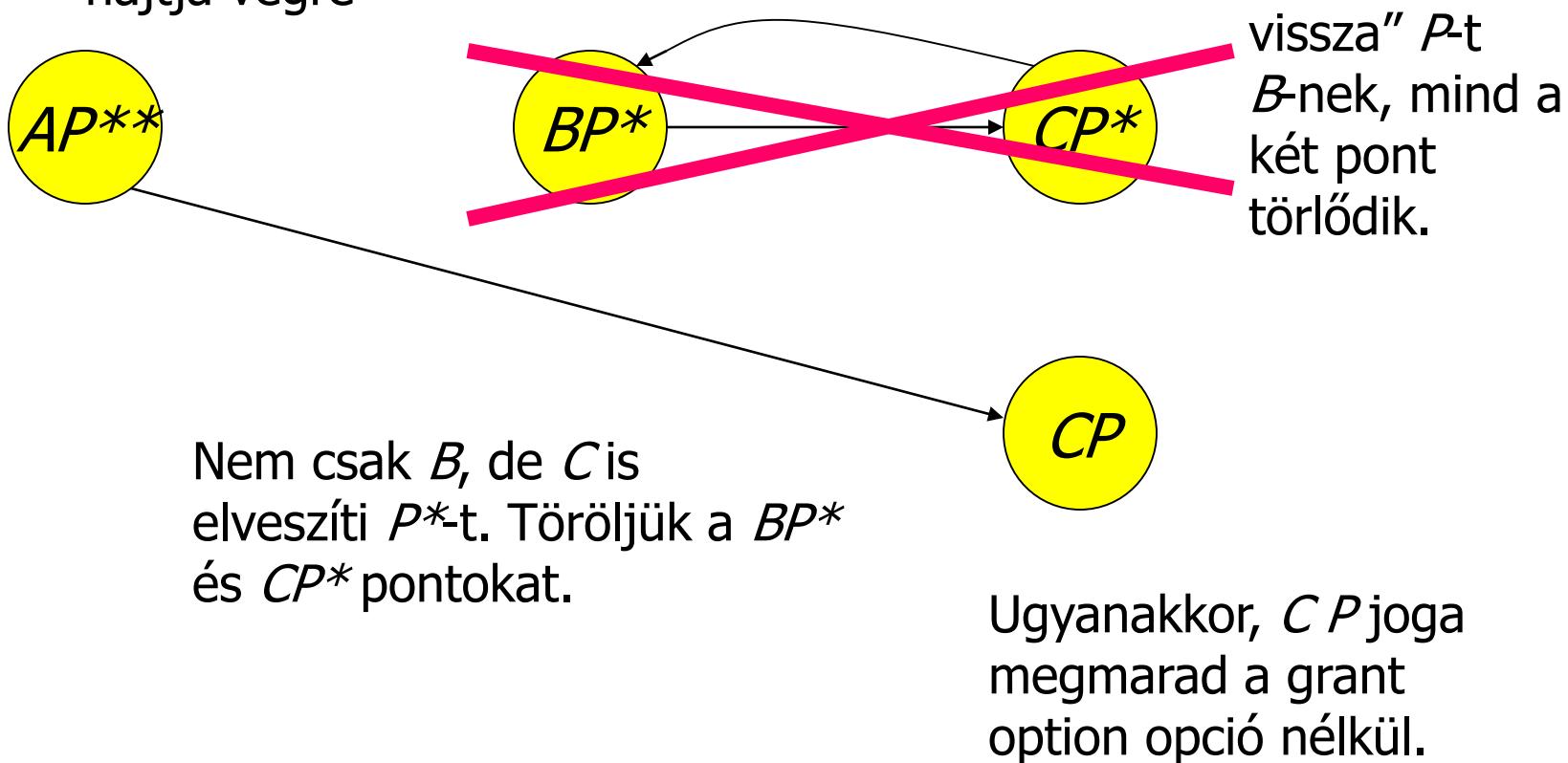
B: GRANT P  
TO C WITH  
GRANT OPTION



A: GRANT P  
TO C

## Példa: Grant Diagram

A a REVOKE P FROM B CASCADE; utasítást hajtja végre



# Kiegészítések

Ismétlések

Kiegészítések az eddigiekhez

# Információk az írásbeli vizsgáról

- Vizsgaalkalmak:
  - Időpontok: 2018. dec. 20., 2019. jan. 03., jan. 16., jan. 24. 17:00-19:00 óráig
  - Vizsga helye: Adatbázis labor (LD-00-807)
- Időkorlát: 80 perc
- Várhatóan 40 kérdés lesz, várhatóan a max. pontszám 100 lesz (a kérdések nem egyforma pontozásúak)
- Egyszerre egy kérdést fog a rendszer megjeleníteni, de a korábbi kérdésre, válaszra vissza lehet térni
- A kérdéssor befejezése és beadása után már nincs lehetőség módosításra

# Információk az írásbeli vizsgáról

- Várhatóan a kérdések többségénél egyszeres/többszörös választási lehetőség lesz, de előfordulhatnak „kifejtőς”/definíciókra rákérdezések is (rövid 1-2 mondatos válasszal)
- A többszörös választási lehetőségnél minden válasz lehetőség mellett négyzet jelenik majd meg, itt ki kell majd választani minden olyan választ, ami helyes megoldásnak van véve (de lehet, hogy csak egy válasz lesz valójában helyes)
- Definíciókat érteni kell, előfordulhatnak olyan egyszerű feladatok, amelyek a definíció egyszerű alkalmazásával nagyon könnyen megoldhatók

# Információk az írásbeli vizsgáról

- Vizsgatematika:
  - A relációs adatmodell
  - A relációs algebrai kifejezések optimalizációja
  - SQL
  - Megszorítások
  - Relációs adatbázisok tervezésének elmélete
  - Többértékű függőség
  - E/K modell
  - Tranzakciók
  - XML
  - OLAP
  - Objektum-relációs adatbázisok
  - Indexek
  - Jogosultságok
- Jelen előadás csak kiegészítés az eddigiekhez

# Relációs adatmodell

- $R | X| S$  ugyanazt jelöli, mint  $R \bowtie S$
- **Természetes összekapcsolás:**  $R(A_1, \dots, A_n)$ ,  $S(B_1, \dots, B_m)$  sémájú táblák esetén  $R | X| S$  azon sorokat tartalmazza  $R$ -ből illetve  $S$ -ből, amelyek  $R$  és  $S$  azonos attribútumain megegyeznek.

| A  | B |
|----|---|
| a1 | b |
| a2 | b |
| a3 | b |

$|X|$

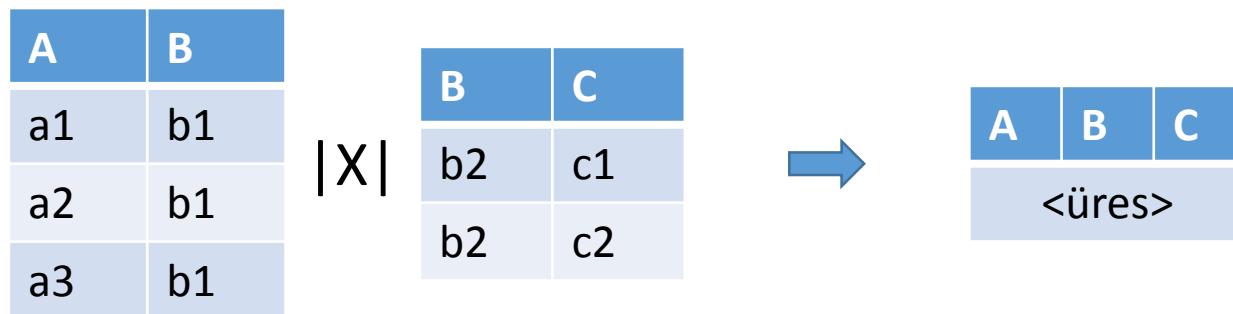
| B | C  |
|---|----|
| b | c1 |
| b | c2 |

$\rightarrow$

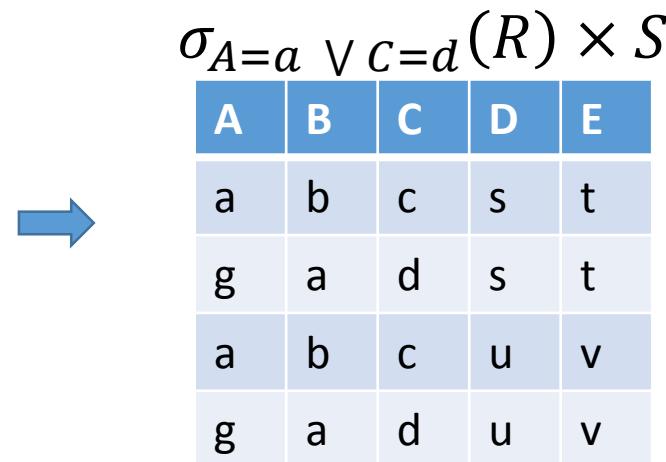
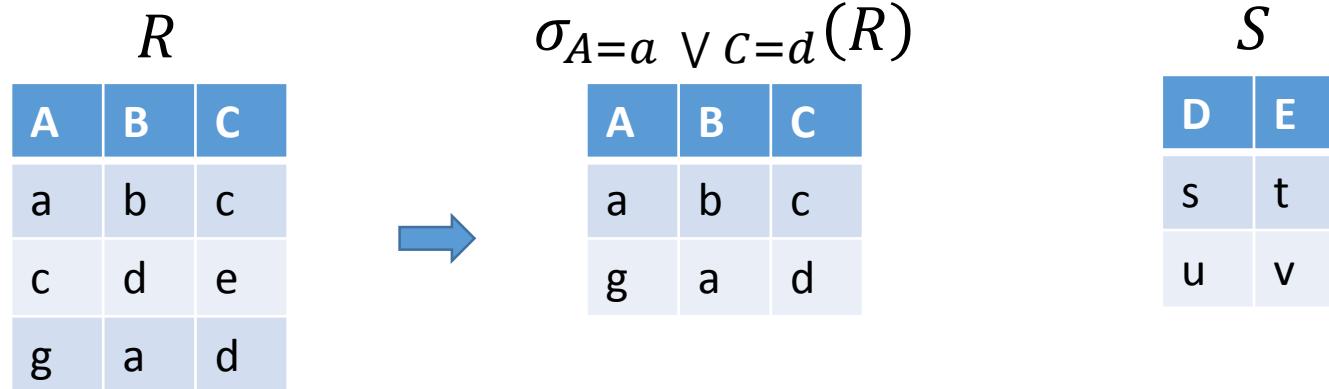
| A  | B | C  |
|----|---|----|
| a1 | b | c1 |
| a1 | b | c2 |
| a2 | b | c1 |
| a2 | b | c2 |
| a3 | b | c1 |
| a3 | b | c2 |

3\*2 sor

# Relációs adatmodell

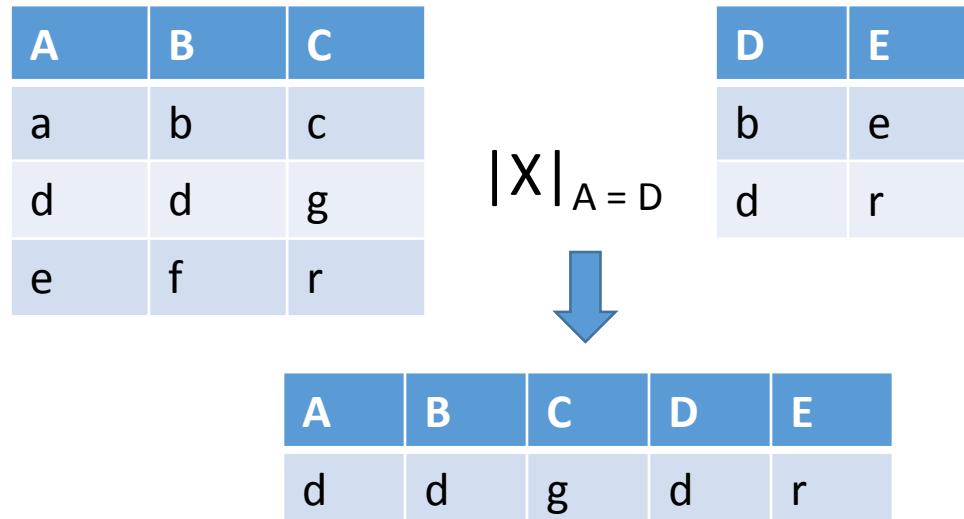


# Relációs adatmodell



# Relációs adatmodell

- A műveletek kifejezhetők a többi alapművelettel
- Például:
- Természetes összekapcsolás:  $R |X| S \equiv \pi_L(\sigma_C(R \times S))$ , itt: C a közös attribútumok egyenlőségét írja elő, L pedig csak egyszer veszi a közös attribútumokat.
- Théta-összekapcsolás:  $R |X|_F S = \sigma_F(R \times S)$  teljesül, itt F valamelyen feltételel



| A | B | C | D | E |
|---|---|---|---|---|
| a | b | c |   |   |
| d | d | g |   |   |
| e | f | r |   |   |

# Relációs adatmodell

- Mivel sorok halmazáról van szó, így értelmezhetők a szokásos halmazműveletek: az **unió**, a **metszet** és a **különbség**. A műveletek alkalmazásának feltétele, hogy az operandusok attribútumai megegyezzenek és azonos sorrendben szerepeljenek.

| A | B | C | D |
|---|---|---|---|
| a | b | c | t |
| c | d | e | v |
| g | a | d | u |



$\pi_{A,B,C}(R)$

| A | B | C |
|---|---|---|
| a | b | c |
| c | d | e |
| g | a | d |

S

| A | B | C |
|---|---|---|
| a | b | c |
| t | u | v |
| c | d | e |
| g | a | d |

$\pi_{A,B,C}(R) - S$



| A      | B | C |
|--------|---|---|
| <üres> |   |   |

# Relációs adatmodell

$$\pi_{A,B,C}(R)$$

| A | B | C |
|---|---|---|
| a | b | c |
| c | d | e |
| g | a | d |

S

| A | B | C |
|---|---|---|
| t | u | v |

$$\pi_{A,B,C}(R) - S$$

| A | B | C |
|---|---|---|
| a | b | c |
| c | d | e |
| g | a | d |



# Relációs adatmodell

$$\pi_{A,B,C}(R)$$

| A | B | C |
|---|---|---|
| a | b | c |
| c | d | e |
| g | a | d |

$$S$$

| A | B | C |
|---|---|---|
| t | u | v |
| g | a | d |

$$\pi_{A,B,C}(R) - S$$

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



# Relációs adatmodell

$R$

| A | B | C |
|---|---|---|
| a | b | c |
| c | d | e |
| g | a | d |

$S$

| A | B | C |
|---|---|---|
| t | u | v |
| g | a | d |

$R \cap S$



| A | B | C |
|---|---|---|
| g | a | d |

# Relációs adatmodell

$R$

| A | B | C |
|---|---|---|
| a | b | c |
| c | d | e |
| g | a | d |
| t | u | v |

$S$

| A | B | C |
|---|---|---|
| t | u | v |
| g | a | d |

$R \cap S$



| A | B | C |
|---|---|---|
| g | a | d |
| t | u | v |



# SQL

- Tegyük fel, hogy az alábbi táblát hozták létre:

```
CREATE TABLE Hallgatók (
 név CHAR(30) PRIMARY KEY,
 cím CHAR(50)
);
```

# SQL

Írassuk ki azokat a hallgatókat, akiknek a nevük második karaktere ,a', és a nevük ,y'-ra végződik!

```
SELECT * FROM Hallgatók WHERE név LIKE '_a%y';
```

Hány különböző nevű hallgató van?

```
SELECT COUNT(DISTINCT név) FROM Hallgatók;
```

Hogyan tudjuk meghatározni, hogy az azonos nevű hallgatókból hány darab van?

```
SELECT név, COUNT(név) FROM Hallgatók
GROUP BY név
HAVING COUNT(név) > 0;
```

# Funkcionális függőségek

- „ $A_1, A_2, \dots, A_n$  funkcionálisan meghatározza  $B_1, B_2, \dots, B_m$ -et” akkor mondjuk, hogy ha két sor megegyezik  $A_1, A_2, \dots, A_n$  attribútumokon, akkor  $B_1, B_2, \dots, B_m$  attribútumokon is meg kell egyezniük. Ha  $X=\{A_1, A_2, \dots, A_n\}$  és  $Y=\{B_1, B_2, \dots, B_m\}$  két attribútum halmazt jelöl, akkor ezt úgy jelölhetjük, hogy  $X \rightarrow Y$ .

# Egyed/kapcsolat modell

- Tervezési alapelvek:
  - A modell minél pontosabban tükrözze a valóságot.
  - Attribútum vs. új egyedosztály? Általában: ha az ábrázolni kívánt „valóságdarab” több, számunkra fontos tulajdonsággal rendelkezik, akkor érdemes egy új egyedosztályt készíteni, különben elegendő felvenni egy új attribútumot.
  - Példa: Film egyedosztállynál a színnesznév új attribútumként szerepeljen vagy inkább hozzunk létre egy új egyedosztályt.
  - **A kettőt egyszerre semmi esetre se tegyük.**

# Egyed/kapcsolat modell

- E-K modell átírása adatbázissémává
  - Egy egyedhalmaznak egy reláció felel meg, melynek neve megegyezik az egyedhalmaz nevével, attribútumai az egyedhalmaz attribútumai.
  - Egy kapcsolatnak szintén egy relációt feleltetünk meg, melynek neve a kapcsolat neve, attribútumai pedig a kapcsolatban részt vevő egyedhalmazok kulcsai. Amennyiben két attribútum neve megegyezne, egyiket értelemszerűen át kell neveznünk.
  - Gyenge egyedhalmazok esetében a kapott relációhoz hozzá kell még venni azokat az attribútumokat, amelyek egyértelműen azonosítják az egyedhalmazt.

# Egyed/kapcsolat modell

Forrás: <https://spot.colorado.edu/~moonhawk/technical/C1912567120/E1121736573/Media/rdbdesign.pdf>

- A kalózokat az egyedi becenevükkel azonosítják, amelyet a Központi Kalóz Hivatal jegyez. Kalózok nem változtathatták meg a becenevüket. Ráadásként minden kalóznak van egy keresztnéve, egy vadság minősítése és az első kalózkodásának dátuma.
- A kalózhajók szintén az egyedi kalózhajó névvel azonosíthatók. Ezeket szintén a Központi Kalóz Hivatal jegyzi. Fontos információk egy kalózhajóról az árbocok száma, ágyúk száma és a megbízás dátuma.
- Egy kereskedelmi hajót egy egyedi kereskedelmi hajó név azonosít, amelyet a Londoni Potenciális Áldozatok Szövetsége jegyez. Fontos jellemzői egy kereskedelmi hajónak a súlya és az árbocok száma. A Központi Kalóz Hivatal számon tartja a kereskedelmi hajókról, hogy melyik lett kifosztva vagy elsüllyesztve.

# Egyed/kapcsolat modell

Forrás: <https://spot.colorado.edu/~moonhawk/technical/C1912567120/E1121736573/Media/rdbdesign.pdf>

- A kalózokat az egyedi becenevükkel azonosítják, amelyet a Központi Kalóz Hivatal jegyez. Kalózok nem változtathatják meg a becenevüket. Ráadásként minden kalóznak van egy keresztneve, egy vadság minősítése és az első kalózkodásának dátuma.
- A kalózhajók szintén az egyedi kalózhajó névvel azonosíthatók. Ezeket szintén a Központi Kalóz Hivatal jegyzi. Fontos információk egy kalózhajóról az árбocok száma, ágyúk száma és a megbízás dátuma.
- Egy kereskedelmi hajót egy egyedi kereskedelmi hajó név azonosít, amelyet a Londoni Potenciális Áldozatok Szövetsége jegyez. Fontos jellemzői egy kereskedelmi hajónak a súlya és az árбocok száma. A Központi Kalóz Hivatal számon tartja a kereskedelmi hajókról, hogy melyik lett kifosztva vagy elsülyesztve.

# Egyed/kapcsolat modell

Forrás: <https://spot.colorado.edu/~moonhawk/technical/C1912567120/E1121736573/Media/rdbdesign.pdf>

- Egy adott kalóz egy adott kalózhajó legénységéhez tartozik egy adott időben. Egy kalóz lehet munkanélküli is néha. Egy és csak egy kapitánya lehet egy és csak egy kalózhajónak egy adott időben. (Tegyük fel, hogy a múltbeli foglalkoztatás nem lényeges.)
- A Központi Kalóz Hivatal minden rablás esetet számon tart. Egy feljegyzés azonosítja a kalózhajót, a kereskedelmi hajót, a kifosztás dátumát, azt, hogy vajon a kereskedelmi hajó elsüllyedt vagy sem, valamint azt, hogy hány ártatlan tengerész lett kivégezve, Lehetséges, hogy ugyanaz a kalózhajó többször is kifosztott egy kereskedelmi hajót.

# Egyed/kapcsolat modell

Forrás: <https://spot.colorado.edu/~moonhawk/technical/C1912567120/E1121736573/Media/rdbdesign.pdf>

- Egy adott kalóz egy adott kalózhajó legénységéhez tartozik egy adott időben. Egy kalóz lehet munkanélküli is néha. Egy és csak egy kapitánya lehet egy és csak egy kalózhajónak egy adott időben. (Tegyük fel, hogy a múltbeli foglalkoztatás nem lényeges.)
- A Központi Kalóz Hivatal minden rablás esetet számon tart. Egy feljegyzés azonosítja a kalózhajót, a kereskedelmi hajót, a kifosztás dátumát, azt, hogy vajon a kereskedelmi hajó elsüllyedt vagy sem, valamint azt, hogy hány ártatlan tengerész lett kivégezve, Lehetséges, hogy ugyanaz a kalózhajó többször is kifosztott egy kereskedelmi hajót.

# Egyed/kapcsolat modell

Forrás: <https://spot.colorado.edu/~moonhawk/technical/C1912567120/E1121736573/Media/rdbdesign.pdf>

- **Azonosítsuk az összes funkcionális függőséget!**
- **Készítsünk egy egyed/kapcsolat diagramot a struktúrához!**
- **Tervezzünk egy normalizált adatbázis struktúrát!**
- **Készítsünk egy teljes egyed/attribútum listát az összes táblához!**

# Egyed/kapcsolat modell

Forrás: <https://spot.colorado.edu/~moonhawk/technical/C1912567120/E1121736573/Media/rdbdesign.pdf>

- **Funkcionális függőségek:**

- Kalózbecenév → Kalóz keresztnév
- Kalózbecenév → Kalóz vadság minősítés
- Kalózbecenév → Kalóz első kalózkodásának dátuma
- Kalózbecenév → Kalózhajó név
- Kalózhajónév → Kalózhajó árbocok száma
- Kalózhajónév → Kalózhajó ágyúk száma
- Kalózhajónév → Kalózhajó megbízás dátuma
- Kalózhajónév → Kalózhajó kapitány
- Kereskedelmi hajónév → Kereskedelmi hajó súly
- Kereskedelmi hajónév → Kereskedelmi hajó árbocok száma
- Kereskedelmi hajónév + Kalózhajónév + Kifosztás dátuma → Kereskedelmi hajó süllyedése
- Kereskedelmi hajónév + Kalózhajónév + Kifosztás dátuma → Kivégzések száma

# Egyed/kapcsolat modell

Forrás: <https://spot.colorado.edu/~moonhawk/technical/C1912567120/E1121736573/Media/rdbdesign.pdf>



# Egyed/kapcsolat modell

Forrás: <https://spot.colorado.edu/~moonhawk/technical/C1912567120/E1121736573/Media/rdbdesign.pdf>

- Normalizált adatstruktúra:

Kalóz

| Oszlopnevek                      | Adattípus   | Kulcsok |
|----------------------------------|-------------|---------|
| Kalóz neve                       | Varchar(80) | PK      |
| Kalóz keresztneve                | Varchar(80) |         |
| Kalóz vadság minősítés           | Integer     |         |
| Kalóz első kalózkodásának dátuma | Date        |         |
| Kalózhajó név                    | Varchar(80) | FK1     |

# Egyed/kapcsolat modell

Forrás: <https://spot.colorado.edu/~moonhawk/technical/C1912567120/E1121736573/Media/rdbdesign.pdf>

- Normalizált adatstruktúra:

## Kalózhajó

| Oszlopnevek               | Adattípus   | Kulcsok |
|---------------------------|-------------|---------|
| Kalózhajó név             | Varchar(80) | PK      |
| Kalózhajó árbocok száma   | Integer     |         |
| Kalózhajó ágyúk száma     | Integer     |         |
| Kalózhajó megbízás dátuma | Date        |         |
| Kalózhajó kapitány        | Varchar(80) | FK1     |

# Egyed/kapcsolat modell

Forrás: <https://spot.colorado.edu/~moonhawk/technical/C1912567120/E1121736573/Media/rdbdesign.pdf>

- Normalizált adatstruktúra:

## Kereskedelmi hajó

| Oszlopnevek             | Adattípus   | Kulcsok |
|-------------------------|-------------|---------|
| Kereskedelmi hajó név   | Varchar(80) | PK      |
| Kalózhajó árbocok száma | Integer     |         |

# Egyed/kapcsolat modell

Forrás: <https://spot.colorado.edu/~moonhawk/technical/C1912567120/E1121736573/Media/rdbdesign.pdf>

- Normalizált adatstruktúra:

## Kifosztás

| Oszlopnevek                  | Adattípus   | Kulcsok |
|------------------------------|-------------|---------|
| Kereskedelmi hajó név        | Varchar(80) | PK1 FK1 |
| Kalózhajó név                | Varchar(80) | PK2 FK2 |
| Kifosztás dátuma             | Date        | PK3     |
| Kereskedelmi hajó süllyedése | Char(1)     |         |
| Kivégzések száma             | Integer     |         |

# Tranzakciók

- Az SQL elkülönítési szintjeinek tulajdonságai

| Elkülönítési szint | Piszkes adat olvasása | Nem ismételhető olvasás | Fantomadatok    |
|--------------------|-----------------------|-------------------------|-----------------|
| READ UNCOMMITTED   | Megengedett           | Megengedett             | Megengedett     |
| READ COMMITTED     | Nem Megengedett       | Megengedett             | Megengedett     |
| REPEATABLE READ    | Nem Megengedett       | Nem Megengedett         | Megengedett     |
| SERIALIZABLE       | Nem Megengedett       | Nem Megengedett         | Nem Megengedett |

# XPath

- Legyen adva az alábbi XML:

```
<?xml version="1.0" encoding="UTF-8"?>

<könyvesbolt>

<könyv kategória="főzés">
 <cím nyelv="hu">Leonardo lakomái - Az olasz konyha titkos története</cím>
 <szerző>Dave DeWitt</szerző>
 <év>2009</év>
 <ár>3000</ár>
</könyv>

<könyv kategória="gyerek">
 <cím nyelv="hu">Harry Potter</cím>
 <szerző>J. K. Rowling</szerző>
 <év>1999</év>
 <ár>3300</ár>
</könyv>
```

...

## Folytatás

# XPath

...

```
<könyv kategória="web">
 <cím nyelv="en">XQuery: The XML Query Language</cím>
 <szerző>Michael Brundage</szerző>
 <szerző>Paul Peterson</szerző>
 <év>2004</év>
 <ár>22000</ár>
</könyv>

<könyv kategória="web">
 <cím nyelv="hu">Az XML-kézikönyv</cím>
 <szerző>Neil Bradley</szerző>
 <év>2000</év>
 <ár>4000</ár>
</könyv>

<könyv kategória="valami">
 <cím nyelv="hu">Akármiból</cím>
 <szerző>
 <név>X Y</név>
 <cím>Valahol</cím>
 </szerző>
 <év>2000</év>
 <ár>4000</ár>
</könyv>

</könyvesbolt>
```

# XPath

- Nézzünk egy-két példa kérdést!

/könyvesbolt/könyv/cím

Eredmény:

<cím nyelv="hu">Leonardo lakomái - Az olasz konyha titkos története</cím>

<cím nyelv="hu">Harry Potter</cím>

<cím nyelv="en">XQuery: The XML Query Language</cím>

<cím nyelv="hu">Az XML-kézikönyv</cím>

<cím nyelv="hu">Akármí</cím>

# XPath

/könyvesbolt/könyv/cím/@nyelv

Eredmény:

nyelv=hu

nyelv=hu

nyelv=en

nyelv=hu

nyelv=hu

# XPath

/könyvesbolt/könyv[ár>3500]/szerző

Eredmény:

<szerző>Michael Brundage</szerző>

<szerző>Paul Peterson</szerző>

<szerző>Neil Bradley</szerző>

<szerző>

  <név>X Y</név>

  <cím>Valahol</cím>

</szerző>

# XPath

/könyvesbolt/könyv/ár[.>3500]

Eredmény:

<ár>22000</ár>

<ár>4000</ár>

<ár>4000</ár>

# XPath

//cím

Eredmény:

```
<cím nyelv="hu">Leonardo lakomái - Az olasz konyha
titkos története</cím>
<cím nyelv="hu">Harry Potter</cím>
<cím nyelv="en">XQuery: The XML Query Language</cím>
<cím nyelv="hu">Az XML-kézikönyv</cím>
<cím nyelv="hu">Akármí</cím>
<cím>Valahol</cím>
```

# XPath

```
/*/*[cím[@nyelv='hu']]
```

Eredmény:

```
<cím nyelv="hu">Leonardo lakomái - Az olasz konyha
titkos története</cím>
```

```
<cím nyelv="hu">Harry Potter</cím>
```

```
<cím nyelv="hu">Az XML-kézikönyv</cím>
```

```
<cím nyelv="hu">Akármí</cím>
```

# XPath

/könyvesbolt/könyv/cím[ .= 'Az XML-kézikönyv' ]

Eredmény:

<cím nyelv="hu">Az XML-kézikönyv</cím>