

Bevezetés: Relációs adatmodell

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009

-
- 2.1. Adatmodellek áttekintése
 - 2.2. A relációs modell alapjai
 - 2.3. SQL: Relációsémák definiálása
 - 2.4. Rel.algebra: vetítés, kiválasztás
 - 6.1. SQL: Egyszerű egytáblás lekérdezések az SQL-ben
 - A SQL gyakorlat felépítése miatt a Tk. 2.fejezettel kezdünk,
 - Később lesz az 1.fejezet Az adatbázisrendszerek világáról.
-



Bevezető példa (reláció = tábla)

- Naponta találkozunk adatbázisokkal
 - 1960-as évektől a korai DBMS: banki rendszerek, repülőgép-helyfoglalás, vállalati nyilvántartások
 - Napi szinten: Google, Yahoo, Amazon.com, egyetemi tanulmányi rendszerek (ETR, Neptun)
- **1.példa:** A jelentkezési adatok egyeztetésére táblázat (lásd papíron **a jelenléti ív** /Ez az első előadás példája/)
- Reláció = tábla (a jelenléti ív)
- Séma = a reláció szerkezetének leírása (tábla fejléce)
- Előfordulás v. példány = a tábla sorai, adott időpontban a tábla aktuális tartalma dinamikusan változik (példában szereplő tábla tartalma a tantárgyfelvétel időszakában)

Bevezetés: Ki ismeri az SQL-t?

- **Ki ismeri az SQL-t? Van-e különbség?**

Tetszőleges táblát lekérdezve
megegyezik-e az eredmény?

(1) `SELECT B FROM R`

`WHERE A<10 OR A>=10;`

(2) `SELECT B FROM R;`

R	A	B
	5	20
	10	30
	20	40

- Itt mi a helyzet ezzel?

(3) `SELECT A FROM R, S`

`WHERE R.B = S.B;`

(4) `SELECT A FROM R`

`WHERE B IN (SELECT B FROM S);`

Bevezetés: Mi az adatbázis?

- Olyan adatok együttese, amit egy az adatbázis-kezelő rendszer kezel. **Mit várunk az ABKR-től? (DBMS-től?)**
 - Támogassa nagy méretű (több terabyte mennyisége) adat hosszú időn keresztül való tárolását, és tegye lehetővé a hatékony hozzáférést a lekérdezések és adatbázis-módosítások számára.
 - Biztosítsa a tartósságot, az adatb. helyreállíthatóságát, biztonságos (konzisztens állapot biztosítsa, védve legyen a hardware, software és felhasználói hibáktól).
 - Felügyelje a több felhasználó által egy időben történő adathozzáféréseket úgy, hogy ezek a műveletek ne legyenek hatással a többi felhasználóra számára (konkurencia-vezérlés)

Mit várunk egy ABKR-től?

- (folyt.) Mit várunk az ABKR-től? (DBMS-től?)
 - Tegye lehetővé a felhasználók számára, hogy új adatbázisokat hozhassanak létre, és azok sémaját, vagyis az adatok logikai struktúráját egy speciális nyelven adhassák meg: **Adatdefiníciós nyelv (DDL)**
 - Tegye lehetővé a felhasználóknak, hogy az adatokat egy megfelelő nyelv segítségével lekérdezhessék vagy módosíthassák: **Adatkezelő nyelv (DML)**
 - Kényelmes (fizikai adatfüggetlenség, magas szintű deklaratív nyelv, mint például az SQL szabvány)
 - Hatékony legyen a megvalósítás.
- Erre (Tk.1.fejezetére) később a lekérdezések (relációs algebra és SQL SELECT utasítás) után fogunk visszatérni.

Mi az adatmodell?

- Az adatmodell a valóság fogalmainak, kapcsolatainak, tevékenységeinek magasabb szintű ábrázolása
- **Kettős feladat:** az adatmodell megadja, hogy a számítógép számára és a felhasználó számára hogy néznek ki adatok.
- **Az adatmodell:** adatok leírására szolgáló jelölés.
Ez a leírás általában az alábbi három részből áll:
 1. **Az adat struktúrája** (struktúrák, tömbök, rekordok, fizikai és fogalmi adatstruktúrák, hamarosan példákat is mutatunk)
 2. **Az adaton végezhető műveletek** (lekérdezések, módosítások, feldolgozások legyenek megfogalmazhatók és hatékonyan legyenek implementálva)
 3. **Az adatokra tett megszorítások** (milyen adatokat engedélyezünk, milyen megszorításokat teszünk?)

A fontosabb adatmodellek

- Hálós, hierarchikus adatmodell (gráf-orientált, fizikai szintű, ill. apa-fiú kapcsolatok gráfja, hatékony keresés)
- Relációs adatmodell (táblák rendszere, könnyen megfogalmazható műveletek), magában foglalja az objektumrelációs kiterjesztést is (strukturált típusok, metódusok), SQL/Object, SQL/CLI, SQL/PSM (PL/SQL)
- Objektum-orientált adatmodell (az adatbázis-kezelés funkcionálitásainak biztosítása érdekében gyakran relációs adatmodellre épül), ODMG: ODL és OQL
- Logikai adatmodell (szakértői rendszerek, tények és következtetési szabályok rendszere)
- Dokumentum típusú adatok, félig-strukturált adatmodell (XML-dokumentum), további modellek: gráf adatbázisok

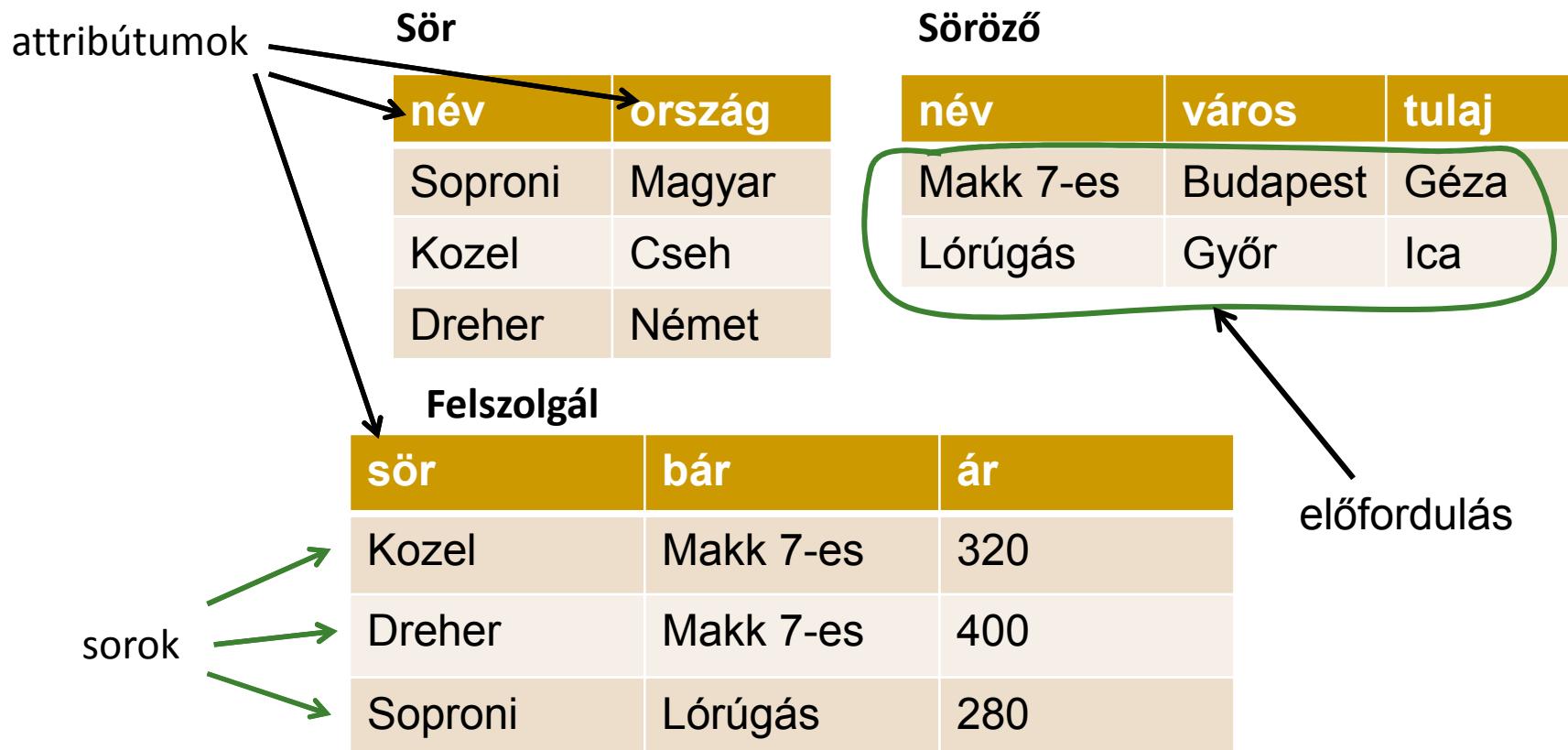
Példa féligstrukturált adatra (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<söröző 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>
</söröző>
<söröző típus="kocsma">
    <név>Lórúgás</név>
    <város>Eger</város>
    <telefon>+36-30-451-1894</telefon>
</söröző>
</xml>
```

Példa relációs adatmodellre

Egy reláció sémája: **Sör (név, ország)**.

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



Relációs adatmodell története

- E.F. Codd 1970-ben publikált egy cikket
A Relational Model of Data for Large Shared Data Banks
Link: <http://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf>
amelyben azt javasolta, hogy az adatokat táblázatokban,
relációkban tárolják. Az elméletére alapozva jött létre a
relációs adatmodell, és erre épülve jöttek létre a relációs
adatmodellen alapuló relációs adatbázis-kezelők.
- **Relációs** (objektum-relációs) **adatbázis-kezelők** például:
ORACLE , INFORMIX , SYSBASE , INGRES, DB2, stb
- Adatbázisok-1 gyakorlaton **ORACLE** adatbázis-kezelő
rendszeret használunk.

Relációs adatmodell előnyei

Miért ez a legelterjedtebb és legkifinomultabb?

- Az adatmodell egy egyszerű és könnyen megérthető **strukturális részt** tartalmaz. A természetes táblázatos formát nem kell magyarázni, és jobban alkalmazható.
- A relációs modellben a fogalmi-logikai-fizikai szint teljesen szétválik, nagyfokú **logikai és fizikai adatfüggetlenség**. A felhasználó magas szinten, hozzá közel álló fogalmakkal dolgozik (implementáció rejtve).
- Elméleti megalapozottság, több absztrakt kezelő nyelv létezik, például relációs algebra (ezen alapul az SQL automatikus és **hatékony lekérdezés optimalizálása**).
- **Műveleti része** egyszerű kezelői felület, szabvány SQL.

Relációs lekérdező nyelvek

Három nyelv szerepel (Adatbázisok-1 gyakorlaton is lesz)

- **Relációs algebra:** algebrai megközelítés , megadjuk a kiértékelési eljárásokat, többféle lehetőség összevetése, hatékonyiségi vizsgálatok.
- **SQL szabvány relációs lekérdező nyelv:** SQL története, szabványok, felépítése. Lekérdezések SELECT-utasítás, SQL DDL, DML, DCL, és SQL/PSM ill. PL/SQL (Oracle)
- **Datalog:** logika alapú megközelítés (korábban elsőrendű logikának megfelelő kalkulus típusú lekérdezések voltak, erről az MSc-n lesz majd bővebben). Itt az Adatbázisok-1 keretében a Datalog az összetett lekérdéseknek segítség, például a rekurzív lekérdezéseknek.

Relációs adatmodell: relációs séma

- Adatok gyűjteményét kezeli (gyűjtemény azonosítása: név)
A gyűjtemény - **R reláció** (tábla, táblázat) megadása
- A gyűjtemény milyen típusú adatokat gyűjt?
adattípus: **sor-típus**. A sor-típus (egy n-es) megadása:
 $\langle \text{Attribútumnév}_1: \text{értéktípus}_1, \dots, \text{Attrnév}_n: \text{értéktípus}_n \rangle$
röviden $\langle A_1, \dots, A_n \rangle$
- **Relációséma:** Relációnév (sortípus) (itt: kerek zárójelben!)
vagyis $R(A_1: \text{értéktípus}_1, \dots, A_n: \text{értéktípus}_n)$
röviden $R(A_1, \dots, A_n)$ ill. $U = \{A_1, \dots, A_n\}$ jelöléssel $R(U)$
- **PÉLDA:** jelenléti ív fejléce – relációséma: megadjuk
a tábla szerkezetét, oszlopnevek és típusuk. Milyen
megszorításokat (pl. kulcs) tudunk megadni a sémán?

Relációs adatmodell: előfordulás

- Mit jelent egy konkrét sor? sor $\langle A_1: \text{érték}_1, \dots, A_n: \text{érték}_n \rangle$
- **Reláció előfordulás (példány, instance)**
 - A sor-típusnak megfelelő véges sok sor (sorok halmaza).
 $\{t_1, \dots, t_m\}$ ahol t_i (tuple, sor, rekord) $i = 1, \dots, m$ (véges sok)
 - $t_i = \langle v_{i1}, \dots, v_{in} \rangle$ (vagyis egy sor n db értékből áll)
 - m - számosság (sorok száma)
 - n - dimenzió (attribútumok száma)
- **Értéktartományok:** A reláció minden attribútumához tartozik egy értéktartomány (adott értéktípusú értékek halmaza) (1normálforma, 1NF feltétel: atomi típusú)
- **PÉLDA:** Jelenléti ív táblázat (a táblázatban az adatok)

Szemléltetés: táblázatos forma

- Szemléltetése: a táblázatos forma (reláció, tábla)

R	A ₁	...	A _j	...	A _n
t ₁	v ₁₁	v _{1n}
...	
t _i	...		v _{ij}	...	
...	
t _m	v _{m1}	v _{mn}

A_j - attribútumnév

t_i - itt csak szimbolikusan
vezetem be, hogy tudjak
a sorokra hivatkozni
(Oracle-ben: rowid)

Szemléltetés: függvényel

- A táblázatos szemléltetésből átérhetünk a sorok egy másik szemléltetésére:
 - $t \in R$ sor felfogható **függvényként** is
 - $t_i : U \rightarrow \text{értékek}$, ahol $U = \{A_1, \dots, A_n\}$
Ezzel a jelöléssel $t_i(A_j) = v_{ij}$ ekvivalens jelöléssel $t_i[A_j]$ vagy $t_i . A_j$ (objektum-orientált/metódus típusú jelöléssel)
 - Előfordulás: sor-függvények véges halmaza
 - Korlátozom a függvényt: $X \subseteq U = \{A_1, \dots, A_n\}$
Ha $X = \{A_{j_1}, \dots, A_{j_k}\}$ attr.halmaz, akkor $t[X]$ típusa:
$$t[X] = \langle A_{j_1} : t(A_{j_1}), \dots, A_{j_k} : t(A_{j_k}) \rangle$$
 - **Függvény szemléltetéssel** könnyen tudom képezni a sorok egy részét és így állítom elő a megfelelő sort.

Logikai szinten: táblázatos szemléltetés

- A relációk táblákban jelennek meg. A tábláknak egyedi neve van.
A relációk oszlopait az attribútumok címzik. A tábla sorait tetszőlegesen megcserélhetjük, sorok sorrendje lényegtelen (a halmazszemlélet miatt)

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

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

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

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

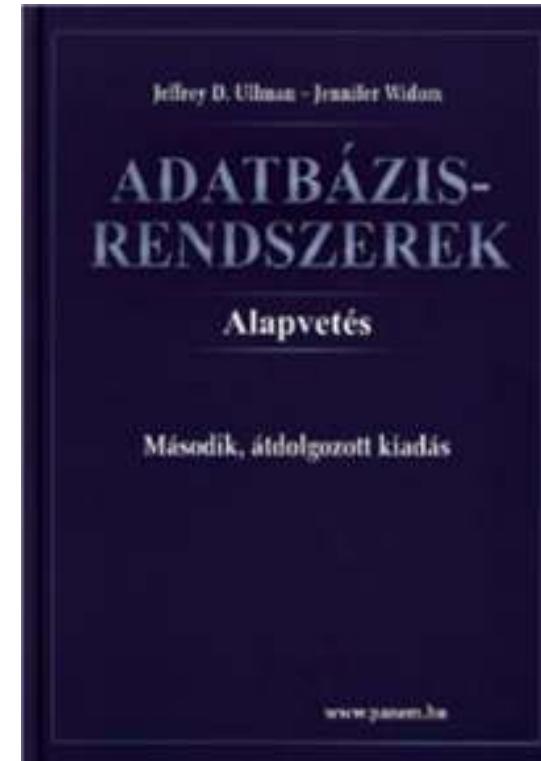
B	C	A
b	c	a
a	a	d
b	d	c

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

Ebben a modellben Példa 4 nem reláció, de a valóságban megengedünk mutihalmazokat lásd később SQL

Relációsémák definiálása SQL-ben

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009



2.3. SQL: Egyszerű tábla-
létrehozások, adattípusok,
alapértelmezés szerinti értékek
egyszerű és összetett kulcsok
megadása

SQL története, szabványok

- Szabvány adatbázis-kezelő nyelv: SQL
- SQL (angol kiejtésben SEQUEL) uis az SQL előfutára
IBM fejlesztette ki a 70-es években: SEQUEL → SQL
más is volt pl. Ingres : QUEL (ez kalkulus alapú lekérdezés)
- Szabványok (ANSI, ISO)
SQL86, SQL89, SQL92 (SQL2), **SQL:1999** (SQL3),
SQL: 2003, SQL:2006, SQL:2008
- Nyelvjárások (**Oracle**, Sybase, DB2, Progress, MSSQL,
mySQL, SQL Server, PostgreSQL, Access,...)
- Az SQL megvalósítások között vannak különbségek,
gyakorlatokon az **Oracle SQL**-t nézzük meg részletesen.

SQL fő komponensei

- Sémaleíró nyelv, DDL (Data Definition Language)
CREATE, ALTER, DROP
- Adatkezelő nyelv, DML (Data Manipulation Language)
INSERT, UPDATE, DELETE, SELECT
 - Az SQL elsődlegesen lekérdező nyelv (Query Language)
- SELECT utasítás (az adatbázisból információhoz jussunk)
- Adatvezérlő nyelv, DCL (Data Control Language)
GRANT, REVOKE
- Tranzakció-kezelés
COMMIT, ROLLBACK, SAVEPOINT
- Procedurális kiterjesztések
Oracle PL/SQL (Ada alapján), SQL/PSM (PL/SQL alapján)

Adatbázis relációsémák definiálása

- Az SQL tartalmaz **adatleíró részt (DDL)**, az adatbázis **objektumainak** a leírására és megváltoztatására.
Objektumok leíró parancsa a **CREATE** utasítás.
- A relációt az SQL-ben táblának (TABLE) nevezik,
az SQL alapvetően háromféle táblát kezel:
 - Alaptáblák (permanens) CREATE TABLE
 - Nézetttáblák CREATE VIEW
 - Átmeneti munkatáblák (WITH utasítás)
- **Alaptáblák** megadása: **CREATE TABLE**

Tábla/reláció sémák SQL-ben

- A legegyszerűbb formája:

```
CREATE TABLE relációnév (
    Attribútum deklarációk listája,
    További kiegészítések
);
```

- Az attribútum deklaráció legalapvetőbb elemei:

Attribútumnév típus [kiegészítő lehetőségek]

-- itt: a **típus** olyan, amit az SQL konkrét megvalósítása támogat (gyakorlaton Oracle környezetben nézzük meg),
Típusok, pl: INTEGER, REAL, CHAR, VARCHAR, DATE

-- A **kiegészítő lehetőségek** például [PRIMARY KEY] vagy
[DEFAULT érték] (köv.lapon példa)

Az előadás példa: sörivók adatbázis

Sörök(név, gyártó)

Sörözők(név, város, tulaj, engedély)

Sörivók(név, város, tel)

Szeret(név, sör)

Felszolgál(söröző, sör, ár)

Látogat(név, söröző)

- Az aláhúzás jelöli a **kulcsot** (a sorok a kulcs összes attribútumán nem vehetik fel ugyanazt az értékeket).

Egyszerű példák táblák létrehozására

```
CREATE TABLE Sörözők (
    név CHAR(20) ,
    város VARCHAR2(40) ,
    tulaj CHAR(30) ,
    engedély DATE DEFAULT SYSDATE
) ;
```



```
CREATE TABLE Felszolgál (
    söröző CHAR(20) ,
    sör      VARCHAR2(20) ,
    ár       NUMBER(10,2) DEFAULT 100
) ;
```

Az SQL értékekről (bővebben gyakorlaton)

- INTEGER, REAL, stb, a szokásos értékek, számok.
- STRING szintén, de itt egyes-aposztróf közé kell tenni a 'szöveget' (vagyis nem „macskaköröm” közé).
Két egyes-aposztróf = egynek felel meg, például
'Joe' 's Bar' megfelel a Joe's Bar szövegnek.
- DATE és TIME típusok is vannak az SQL-ben.
- A dátum formátumát meg kell adni DATE 'yyyy-mm-dd'
Például: DATE '2007-09-30' (2007. szept. 30)
- Az idő formátumát is meg kell adni TIME 'hh:mm:ss'
Például: TIME '15:30:02.5' (délután fél 4 múlt két és fél másodperccel)
- Bármely érték lehet **NULL** hiányzó érték:

Hiányzó értékek: NULL

- Az SQL lehetővé teszi a táblákban **a hiányzó értékeket**, vagyis a relációk soraiban az attribútum értéke ne legyen megadva, hanem egy speciális **NULL** nullérték legyen.
- **A nullérték értelmezésére** több lehetőségünk is van:
 - **Nem-ismert érték**: például tudom, „Joe’s Bár”-jának van valamilyen címe, de nem tudom, hogy mi az.
 - **Nem-definiált érték**: például a házastárs attribútumnak egyedülálló embereknél nincs olyan értéke, aminek itt értelme lenne, nincs házastársa, ezért nullérték.
 - stb (van olyan cikk, amely több százféle okot felsorol)

Táblák létrehozása után a táblák feltöltése adatokkal

Eddig láttuk, hogy a CREATE TABLE utasítással hogyan tudunk létrehozni táblákat és megadni a kulcsokat:

- **SQL DDL: sémaleíró nyelv** (Data Definition Language)
CREATE TABLE, ALTER TABLE, DROP TABLE

Most nézzük meg a táblák tartalmának módosítását, hogyan tudjuk INSERT utasítással a táblát feltölteni adatsorokkal:

- **SQL DML: adatkezelő nyelv** (Data Manipulation Language)
INSERT, UPDATE, DELETE, SELECT

Adatbázis tartalmának módosítása

- Lekérdező utasítás **SELECT** - lekérdezés
- A módosító utasítások nem adnak vissza eredményt, mint a lekérdezések, hanem az adatbázis tartalmát változtatják meg.
- 3-féle módosító utasítás létezik:
 - INSERT** - sorok beszúrása
 - DELETE** – sorok törlése
 - UPDATE** – sorok komponensei értékeinek módosítása

Beszúrás (insert into)

- Két alakja van: 1.) ha egyetlen sort szúrunk be:

INSERT INTO <reláció>

VALUES (<konkrét értékek listája>);

- 2.) ezt majd a lekérdezések után nézzük meg, hogyan tudunk több sort beolvasni a táblába, egy lekérdezés eredményét alkérdés segítségével:

INSERT INTO <reláció>

(<alkérdés>);

- **INSERT INTO 1.) alakjára példa:** a **Szeret(név, sör)** táblában rögzítjük, hogy Zsu szereti a Bud sört.

INSERT INTO Szeret

VALUES('Zsu', 'Bud');

Attribútumok megadása

- A reláció neve után megadhatjuk az attribútumait.
- Ennek alapvetően két oka lehet:
 1. Nem emlékszünk, 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:

```
INSERT INTO Szeret(sör, név)  
VALUES('Bud', 'Zsu');
```

Default 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.

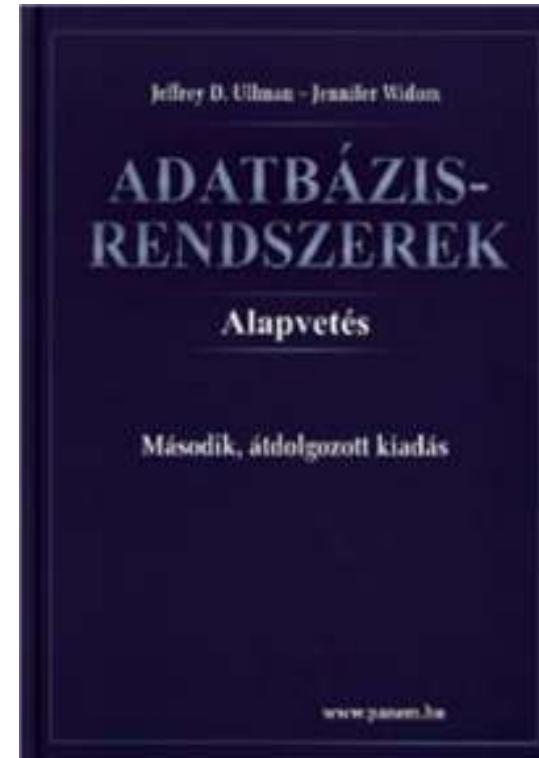
```
CREATE TABLE Sörivók (
    név CHAR(30) PRIMARY KEY,
    cím CHAR(50) DEFAULT 'Sesame St'
    telefon CHAR(16) );
INSERT INTO Sörivók(név)
    VALUES ('Zsu');
```

Az eredmény sor:

név	cím	telefon
Zsu	Sesame St	NULL

SQL lekérdezések

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009



2.4. Relációs algebra
egytáblás műveletei:

- vetítés és
- kiválasztás

6.1. SQL: Egyszerű egytáblás
lekérdezések az SQL-ben

Vetítés (project, jelölése pí: Π)

- **Vetítés** (projekció). Adott relációt vetít le az alsó indexben szereplő attribútumokra (attribútumok számát csökkentik)
- $\Pi_{\text{lista}}(R)$ ahol lista: $\{A_{i_1}, \dots, A_{i_k}\}$ R-sémájában levő attribútumok egy részhalmazának felsorolása
eredmény típusa $\langle A_{i_1}: \text{értéktípus}_{i_1}, \dots, A_{i_k}: \text{értéktípus}_{i_k} \rangle$
 $\Pi_{\text{lista}}(R) := \{ t.A_{i_1}, t.A_{i_2}, \dots, t.A_{i_k} \mid t \in R \} = \{ t[\text{lista}] \mid t \in R \}$
- Reláció soraiból kiválasztja az attribútumoknak megfelelő A_{i_1}, \dots, A_{i_k} -n előforduló értékeket, ha többször előfordul akkor a duplikátumokat kiszűrjük (hogy halmazt kapunk)

➤ **Példa:**

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

$\Pi_{A, B}(R)$



A	B
a	b
c	d

Kiválasztás (select, jelölése szigma: σ)

- Kiválasztás (szűrés). Kiválasztja az argumentumban szereplő reláció azon sorait, amelyek eleget tesznek az alsó indexben szereplő feltételnek.
- $\sigma_{\text{Feltétel}}(R)$ és R sémája megegyezik
- $\sigma_{\text{Feltétel}}(R) := \{ t \mid t \in R \text{ és } t \text{ kielégíti az } F \text{ feltételt}\}$
- $R(A_1, \dots, A_n)$ séma feletti reláció esetén a σ_F kiválasztás F feltétele a következőképpen épül fel:
 - elemi feltétel: $A_i \theta A_j$, $A_i \theta c$, ahol c konstans, θ pedig $=, \neq, <, >, \leq, \geq$
 - összetett feltétel: ha B_1, B_2 feltételek, akkor $\neg B_1, B_1 \wedge B_2, B_1 \vee B_2$ és zárójelezésekkel is feltételek

➤ Példa:

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

$\sigma_{A=a \vee C=d}(R)$



A	B	C
a	b	c
g	a	d

Áttekintés: Relációs algebra (2EA lesz)

Relációs algebrai kifejezések formális felépítése

➤ 1.) Elemi kifejezések

- (i) $R_i \in \mathbb{R}$ (az adatbázis-sémában levő relációnevek)
- (ii) konstans reláció (véges sok, konstansból álló sor)

➤ 2.) Összetett kifejezések

➤ Ha E_1, E_2 kifejezések, akkor a következő E is kifejezés

- $E := \prod_{\text{lista}} (E_1)$ vetítés (típus a lista szerint)
 - $E := \sigma_{\text{Feltétel}} (E_1)$ kiválasztás (típus nem változik)
 - $E := E_1 \cup E_2$ unió, ha azonos típusúak (és ez a típusa)
 - $E := E_1 - E_2$ különbség, ha E_1, E_2 azonos típusúak (típus)
 - $E := E_1 \bowtie E_2$ term. összekapcsolás (típus attr-ok uniója)
 - $E := \rho_{S(B_1, \dots, B_k)}(E_1(A_1, \dots, A_k))$ átnevezés (típ.új attr.nevek)
 - $E := (E_1)$ kifejezést zárójelezve is kifejezést kapunk
- Ezek és csak ezek a kifejezések, amit így meg tudunk adni

Példa – Sörivók adatbázisséma

- Az előadások SQL lekérdezései az alábbi Sörivók adatbázissémán alapulnak
(aláhúzás jelöli a kulcs attribútumokat)

Sörök(név, gyártó)

Sörözők(név, város, tulaj, engedély)

Sörivók(név, város, tel)

Szeret(név, sör)

Felszolgál(söröző, sör, ár)

Látogat(név, söröző)

Egyszerű példa Select-From-Where-re

- Használjuk Sörök(név, gyártó) relációsémát, mely söröket gyártja a Dreher?

```
SELECT név  
FROM Sörök  
WHERE gyártó = 'Dreher' ;
```

A lekérdezés eredménye

név
Arany Ászok
Dreher Classic
...

A lekérdezés eredménye egy reláció, amelynek egy attribútuma van (név) és a sorai az összes olyan sör neve, amelyet a Dreher gyárt.

Eltérés a relációs algebrától: Az SQL alapértelmezésben nem szűri ki a duplikátumokat, az eredmény multihalmaz.

Az egytáblás lekérdezés formális kiértékelése

- Kiindulunk a **FROM záradékból**, mely táblára vonatkozik a lekérdezés?
- Elvégezzük a **WHERE záradékban** szereplő feltételnek eleget tevő sorok kiválasztását
- Alkalmazzuk a **SELECT záradékban** jelölt kiterjesztett projekciót. Lényeges különbség a relációs algebra és SQL között, hogy **az SQL-ben az eredmény** alapértelmezés szerint **nem halmaz**, hanem **multihalmaz**, egy sor az eredményben többször is előfordulhat, ennek az oka, hogy az olcsóbb és hatékonyabb kiértékelést tekintjük az SQL-ben alapértelmezésnek.
- Ahhoz, hogy halmazt kapunk, azt külön kérni kell SELECT **DISTINCT** Lista FROM Táblanév

A műveletek szemantikája

név	gyártó
Arany Ászok	Dreher

2) Ha a feltétel teljesült, akkor
képezünk egy t eredménysort

1) Ellenőrizzük a feltételeit,
hogy a gyártó Dreher-e

3) Ebből a t sorból a
SELECT listának
megfelelő típusú sort
képezzük, példa: t.név

Az egytáblás SFW alapértelmezése

```
SELECT [DISTINCT] kif1 [[AS] onév1], … , kifn [onévn]  
FROM táblanév [sorváltozó]  
[WHERE feltétel]
```

Alapértelmezés (a műveletek szemantikája – általában)

- A FROM záradékban levő relációhoz tekintünk egy **sorváltozót**, amely a reláció minden sorát bejárja
- minden egyes „aktuális” sorhoz **kiértékeljük** a WHERE záradékot
- Ha helyes (vagyis **igaz**) választ kaptunk, akkor képezünk egy sort a SELECT záradékban szereplő kifejezéseknek megfelelően.

SELECT záradékban * jelentése

- Amikor csak egy reláció van a FROM záradékban, akkor a SELECT záradékban levő * jelentése: „a reláció minden attribútuma”
- Példa: Keressük a Sörök(név, gyártó) tábla alapján a Dreher-sörök adatait.
- A lekérdezés eredménye

```
SELECT *  
FROM Sörök  
WHERE gyártó = 'Dreher' ;
```

- A lekérdezés eredménye a Sörök tábla összes attribútumát tartalmazza. Első lépésben (kezdő gyakorlásnál kicsi táblákra) minden lekérdezzük előbb a tábla tartalmát: **SELECT * FROM Táblanév;**

Attribútumok átnevezése

- Ha az eredményben (a fejlécben) más attribútumnevet szeretnénk használni, akkor “[AS] új_oszlopnév” segítségével tudunk más oszlopnevet kiírni.
(Oracle: másodnévben nem kell ‘AS’, csak szóköz)
- Listán azt értjük, hogy vesszővel vannak elválasztva az elemek (attribútumnevek), ha a másodnévben szóköz szerepel, akkor azt macskaköröm közé kell tenni: “...”
- Példa: Sörök(név, gyártó)

```
SELECT név sör, gyártó "Dreher gyártó"  
FROM Sörök  
WHERE gyártó = 'Dreher' ;
```

- A lekérdezés eredményében az új oszlopnevek lesznek.

SELECT záradékban levő kifejezések

- Az attribútumnevek helyett tetszőleges kifejezések állhatnak (amelyek megfelelnek az adott típusra) a SELECT záradék elemeként.
- Lásd bővebben majd **a gyakorlatok példáit, feladatait, felhasználjuk az Oracle DB SQL Language Reference** megfelelő fejezeteket: Operators, Functions, Expressions.
- Példa: Felszolgál(söröző, sör, ár)
**SELECT söröző, sör, ár*114 árYenben
FROM Felszolgál;**
- Konstansok a kifejezésekben **Szeret(név, sör)**:
**SELECT név DABkedvelő
FROM Szeret
WHERE upper(sör) = 'DAB' ;**

WHERE záradék (összetett feltételek)

- Hasonlóan, mint a relációs algebra kiválasztás (σ) feltételében **elemi feltételekből építkezünk**, ahol elemi feltételen két kifejezés =, $<>$, $<$, $>$, $<=$, $>=$ aritmetikai összehasonlítását, a theta műveletet értjük.
- **Logikai műveletek** AND, OR, NOT és zárójel () segítségével kapjuk **az összetett feltételeket**.
- Példa: **Felszolgál** (söröző, sör, ár) relációséma esetén keressük a „Joe’s Bar”-ban árult „DAB” söröök árát:

```
SELECT ár  
FROM Felszolgál  
WHERE söröző = 'Joe''s Bar' AND  
sör = 'DAB' ;
```

WHERE záradék (további lehetőségek)

SQL specialitások, amelyek könnyen átírhatóak relációs algebrai kifejezésre (összetett kiválasztási feltételre)

- **BETWEEN .. AND ..** intervallumba tartozás
- **IN (értékhalmaz)** egyszerű értékek halmaza

SQL specialitások, nem írhatók át relációs algebrába:

(--- ezek jönnek a köv.lapon...)

- Karakterláncok **LIKE** összehasonlítása mintákkal
- **IS NULL** összehasonlítás

LIKE

- Karakterláncok összehasonlítása mintákkal:
 - <attribútum> LIKE <minta> vagy
 - <attribútum> NOT LIKE <minta>
- Minta egy olyan karakterlánc, amelyben használhatjuk a speciális % és _ karaktereket. A mintában % megfelel bármilyen karakterláncnak és _ bármilyen karakternek.
- Példa: Azokat a sörözőket keressük, amelyik nevének a második betűje „a” vagy a nevében van „s”, mint ahogyan például a „Joe’s Bar” névben is szerepel:

```
SELECT név FROM Sörözők  
WHERE név LIKE '_a%' OR  
név LIKE '%s%';
```

NULL (hiányzó) értékek

- Az SQL lehetővé teszi, hogy a relációk soraiban az attribútum értéke egy speciális NULL nullérték legyen.
- **A nullérték értelmezésére** több lehetőségünk is van:
 - **Hiányzó érték**: például tudom, „Joe’s Bár”-jának van valamilyen címe, de nem tudom, hogy mi az.
 - **Nem-definiált érték**: például a házastárs attribútumnak egyedülálló embereknél nincs olyan értéke, aminek itt értelme lenne, nincs házastársa, ezért nullérték.
- **Where záradékban a nullérték vizsgálata:**
 - IS NULL
 - IS NOT NULL

NULL értékek használata

- **Where záradékban a nullérték használata:**
 - Amikor egy aritmetikai műveletben az egyik tag **NULL**, akkor az eredmény is **NULL**.
 - Amikor egy **NULL** értéket hasonlítunk össze bármely más értékkel (beleértve a NULL-t is) az összehasonlítási operátorok ($=$, $<$, $<=$, $>$, $>=$) segítségével, akkor az eredmény **UNKNOWN** (ismeretlen).

Az ismeretlen (unknown) igazságérték

- Az SQL-ben szereplő logikai feltételek valójában **háromértékű logika**: TRUE, FALSE, UNKNOWN (magyarban igaz, hamis, ismeretlen rövidítése miatt inkább meghagyjuk az angol T, F, U rövidítéseket).
- A WHERE záradékban szereplő logikai feltételt a rendszer minden egyes sorra ellenőrzi és a logikai érték TRUE, FALSE vagy UNKNOWN valamelyike lehet, de az eredménybe csak azok a sorok kerülnek, amelyeknek a feltétel kiértékelése TRUE értéket adott.

A háromértékű logika

- Hogyan működnek az AND, OR, és NOT logikai műveletek a 3-értékű logikában?
- A szabályt könnyű megjegyezni, ha úgy tekintjük, hogy TRUE = 1, FALSE = 0, és UNKNOWN = $\frac{1}{2}$.
- Ekkor AND = MIN, OR = MAX, NOT(x) = $1-x$.
- Példa:
TRUE AND (FALSE OR NOT(UNKNOWN)) =
 $\text{MIN}(1, \text{MAX}(0, (1 - \frac{1}{2}))) =$
 $\text{MIN}(1, \text{MAX}(0, \frac{1}{2})) =$
 $\text{MIN}(1, \frac{1}{2}) = \frac{1}{2} = \text{UNKNOWN}$
- A 3-értékű logika AND, OR és NOT igazságtáblázatát lásd a Tk. 6.2.ábráját (vagy kitöltése a fenti szabállyal)

A háromértékű logika (Tk.6.2. ábra)

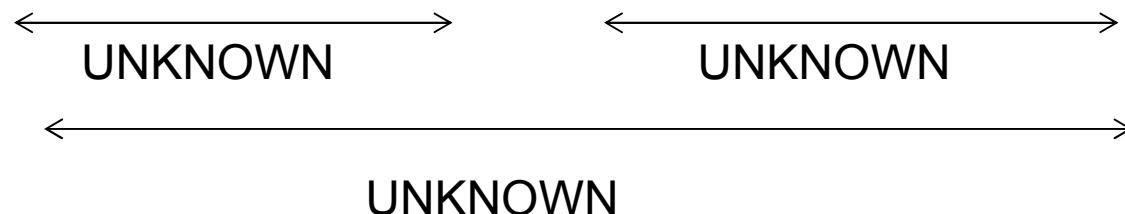
x	y	$x \text{ AND } y$	$x \text{ OR } y$	$\text{NOT } x$
T	T	T	T	F
T	U	U	T	F
T	F	F	T	F
U	T	U	T	U
U	U	U	U	U
U	F	F	U	U
F	T	F	T	T
F	U	F	U	T
F	F	F	F	T

Egy meglepő példa

- Példa: Felszolgál reláció legyen az alábbi:

söröző	sör	ár
Joe's Bar	Bud	NULL

```
SELECT söröző  
FROM Felszolgál  
WHERE ár < 2.00 OR ár >= 2.00;
```



Oka: a 2-értékű \neq 3-értékű szabályok

- Bizonyos általános szabályok, mint például, hogy az AND kommutatív érvényes a 3-értékű logikában is.
- Ellenben nem igaz, például a **kizáró szabály**, vagyis $p \text{ OR NOT } p = \text{TRUE}$ nem teljesül, ha $p = \text{UNKNOWN}$, mert ekkor a baloldal: $\text{MAX}(\frac{1}{2}, (1 - \frac{1}{2})) = \frac{1}{2} \neq 1$ vagyis a 3-értékű logikában baloldal értéke nem TRUE.
- Ezért az előző példában nem az eredeti egy soros táblát, hanem az üres táblát (amelynek egy sora sincs) kaptuk meg az eredménytáblaként.

Az eredmény rendezése

- SQL SELECT utasításban a záradékok
- Az SQL lehetővé teszi, hogy a lekérdezés eredménye bizonyos sorrendben legyen rendezve. Az első attribútum egyenlősége esetén a 2.attribútum szerint rendezve, stb, minden attribútumra lehet növekvő vagy csökkenő sorrend.
- Select-From-Where utasításhoz a következő záradékot adjuk, a WHERE záradék és minden más záradék (mint például GROUP BY és HAVING) után következik:

SELECT ... FROM ... [WHERE ...][...]
ORDER BY {attribútum [DESC], ...}

- Példa: **SELECT * FROM Felszolgál**
ORDER BY ár DESC, sör

Kérdés/Válasz

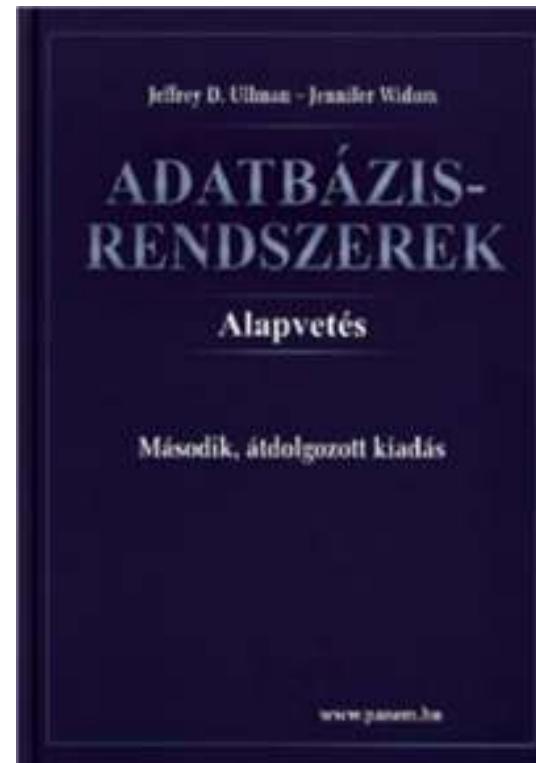
- Köszönöm a figyelmet! Kérdés/Válasz?
- Az első gyakorlaton: Példák táblák létrehozására:
- Az attribútumok típusának megadásakor az Oracle implementációban **milyen standard típusok** közül választhatunk, mi a különbség pl. CHAR és VARCHAR?
- A táblákat a létrehozásuk után feltöltjük adatsorokkal
- Egy táblára vonatkozó lekérdezésekre feladatok, példák, vetítés és kiválasztás művelete, függvények használata, a kiválasztott sorok rendezése.
- Gyakorlás: Oracle Példatár 1.fejezet feladatai:
<http://people.inf.elte.hu/sila/eduAB/Feladatok.pdf>

Több táblára vonatkozó lekérdezések

Relációs algebra és SQL SELECT

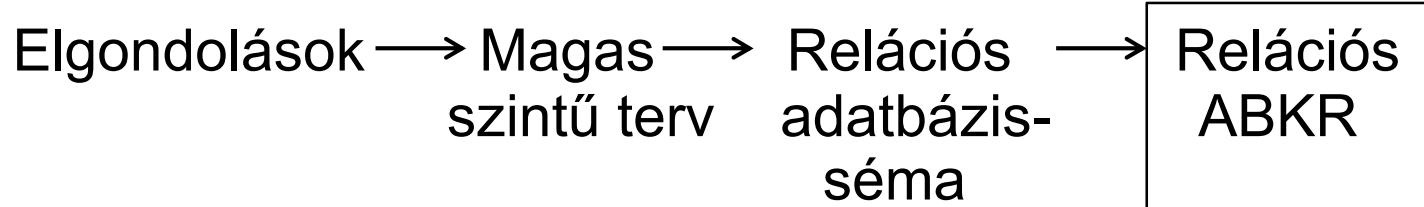
Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiad, 2009

-
- 4.fej. Tervezés: E/K-modell elemei
 - 2.3.+7.1. Megvalósítás: Több tábla és megszorítások megadása (SQL)
 - 2.4. Lekérdezések: Relációs algebra
 - 6.2. Több tábla lekérdezése SQL-ben
(mai órán: táblák szorzata, összekapcsolása,
folyt.köv.: halmaz- és multihalmaz-műveletek, példák)



Magas szintű adatbázismodellek

- Vizsgáljuk meg azt a folyamatot, amikor egy új adatbázist létrehozunk, vegyük példaként a sörivós adatbázist.
- Az adatbázis-modellezés és implementálás eljárása



- Modellezés
 - komplex valós világ leképezése, absztrakció
- **Tervezési fázis:**
 - Milyen információkat kell tárolni?
 - Mely információelemek kapcsolódnak egymáshoz?
 - Milyen megszorításokat kell figyelembe venni? stb...

Egyed-kapcsolat modell elemei

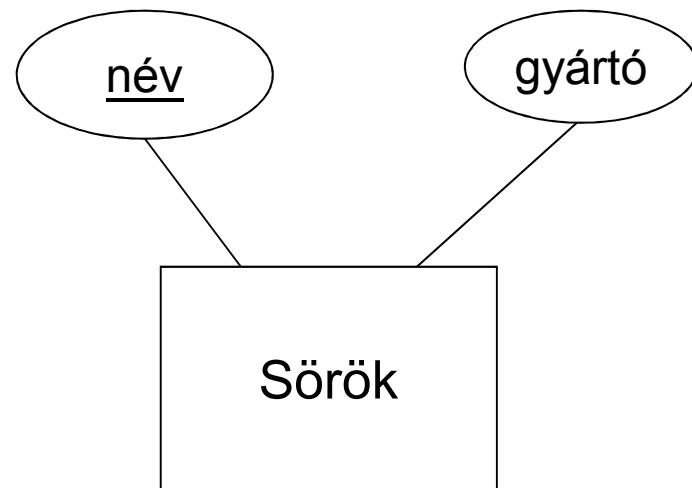
- Egyed-kapcsolat modell: E/K modell
(Entity-relationship ER) alapfogalmak:
- Egyedhalmazok (absztrakt objektumok osztálya)
 - Miről gyűjtünk adatokat?
 - Mit tegyünk egy gyűjteménybe? - hasonlóság
 - Hasonló egyedek összessége
- Attribútumok
 - Megfigyelhető tulajdonságok, megfigyelt értékek
 - Az egyedek tulajdonságait írják le
- Kapcsolatok
 - Más egyedhalmazokkal való kapcsolatuk

E/K modell elemei: Egyedhalmazok

- $E(A_1, \dots, A_n)$ egyedhalmaz **séma**:
 - E az egyedhalmaz neve,
 - A_1, \dots, A_n tulajdonságok,
 - $\text{DOM}(A_i)$ – lehetséges értékek halmaza.
 - például: tanár(név, tanszék).
- $E(A_1, \dots, A_n)$ sémájú egyedhalmaz **előfordulása**:
 - A konkrét egyedekből áll
 - $E = \{e_1, \dots, e_m\}$ egyedek (entitások) halmaza, ahol
 - $e_i(k) \in \text{DOM}(A_k)$,
 - semelyik két egyed sem egyezik meg minden attribútumban (léteznek és megkülönböztethetők)

E/K-diagram: Egyedhalmazok

- E/K diagram: séma-szinten grafikusan ábrázoljuk
- Egyedhalmazok: téglalap
- Tulajdonságok: ovális
 - az elsődleges kulcschoz tartozó tulajdonságokat aláhúzzuk.

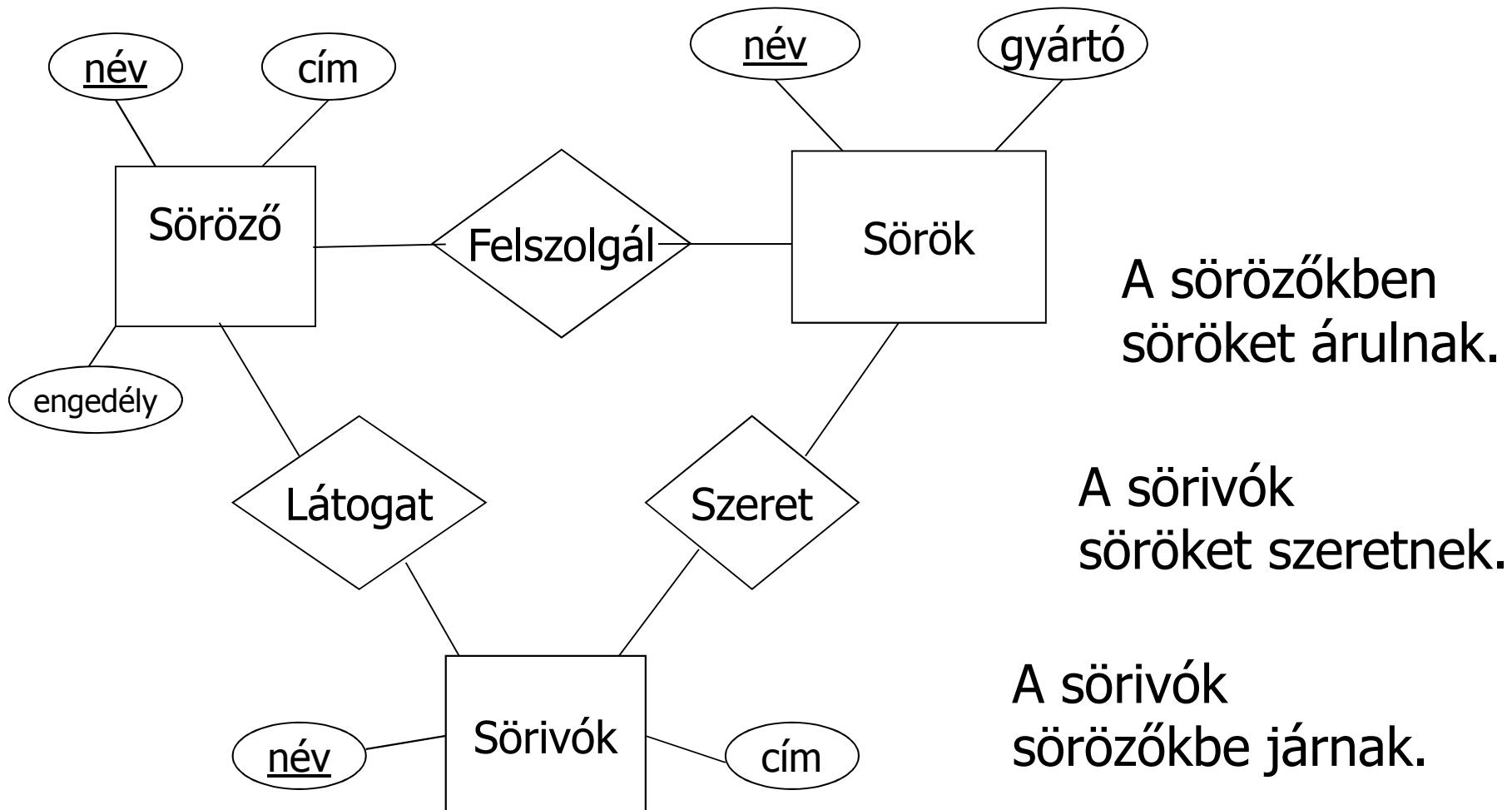


E/K modell elemei: Kapcsolatok

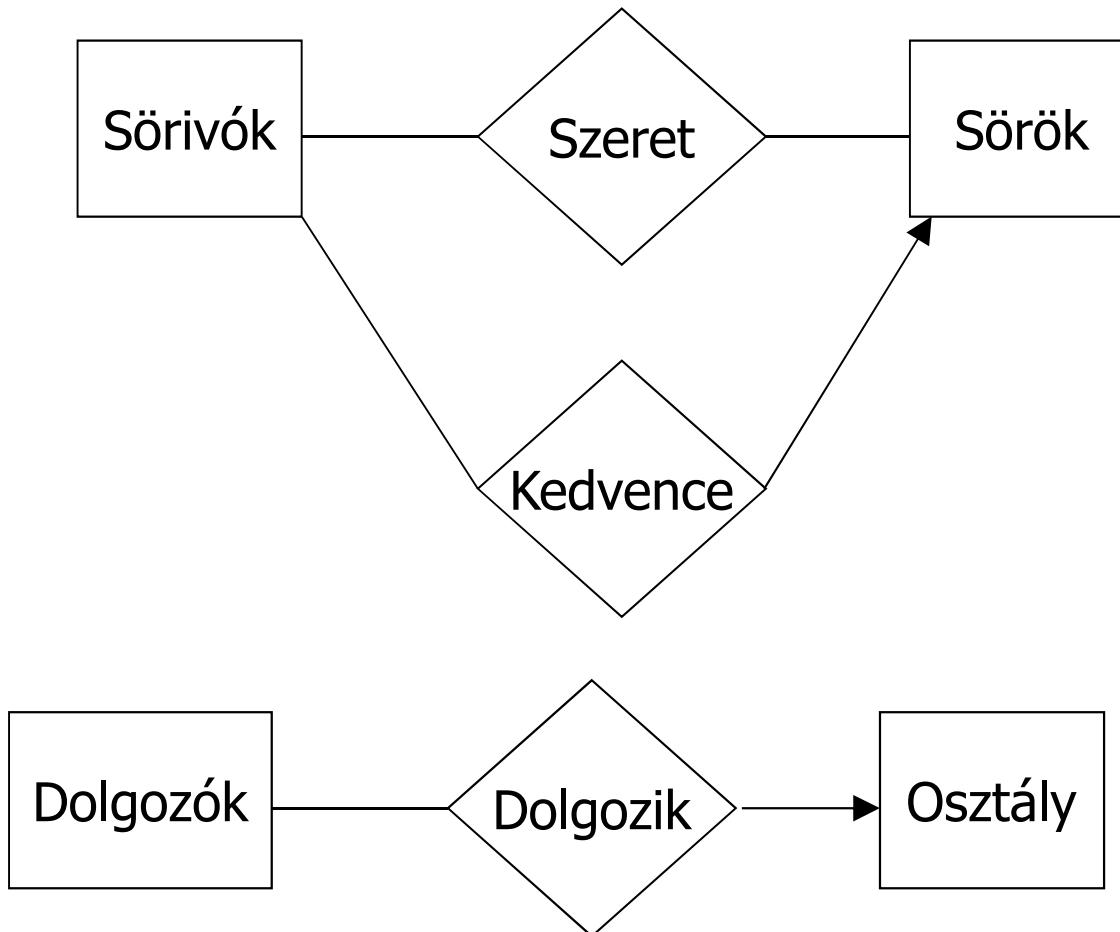
- K(E₁,...,E_p) a kapcsolat sémája,
 - K a kapcsolat neve,
 - E₁,...,E_p egyedhalmazok sémái,
 - p=2 bináris kapcsolat, p>2 többágú kapcsolat,
 - például: tanít(tanár,tárgy).
- K(E₁,...,E_p) sémájú kapcsolat előfordulása:
 - K = {(e₁,...,e_p)} egyed p-esek halmaza, ahol
 - e_i ∈ E_i,
 - a kapcsolat előfordulásaira tett megszorítások határozzák meg a kapcsolat típusát.

E/K-diagram: Kapcsolatok

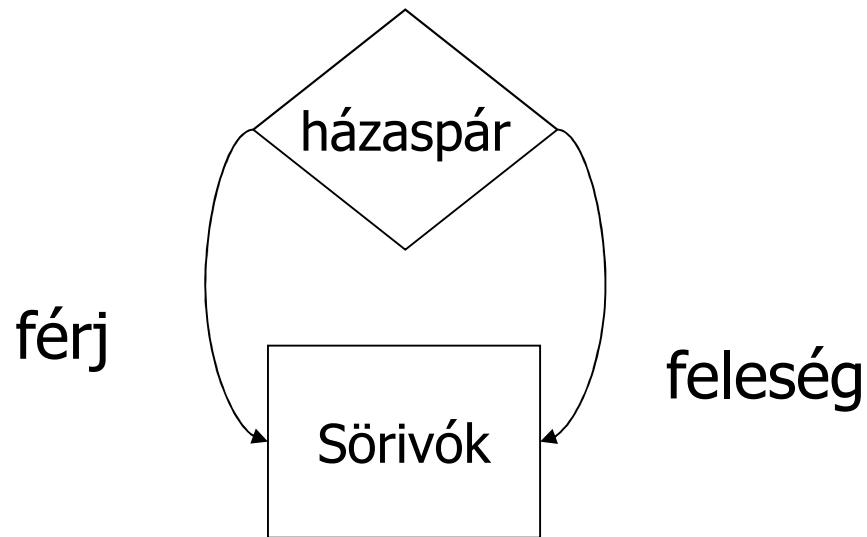
- A kapcsolatok jele: **rombusz**



Kapcsolatok típusai: sok-egy, sok-sok (két egyedhalmaz között több kapcsolat is lehet)



Egy egyedhalmaz önmagával is kapcsolódhat: Szerepek (Roles)

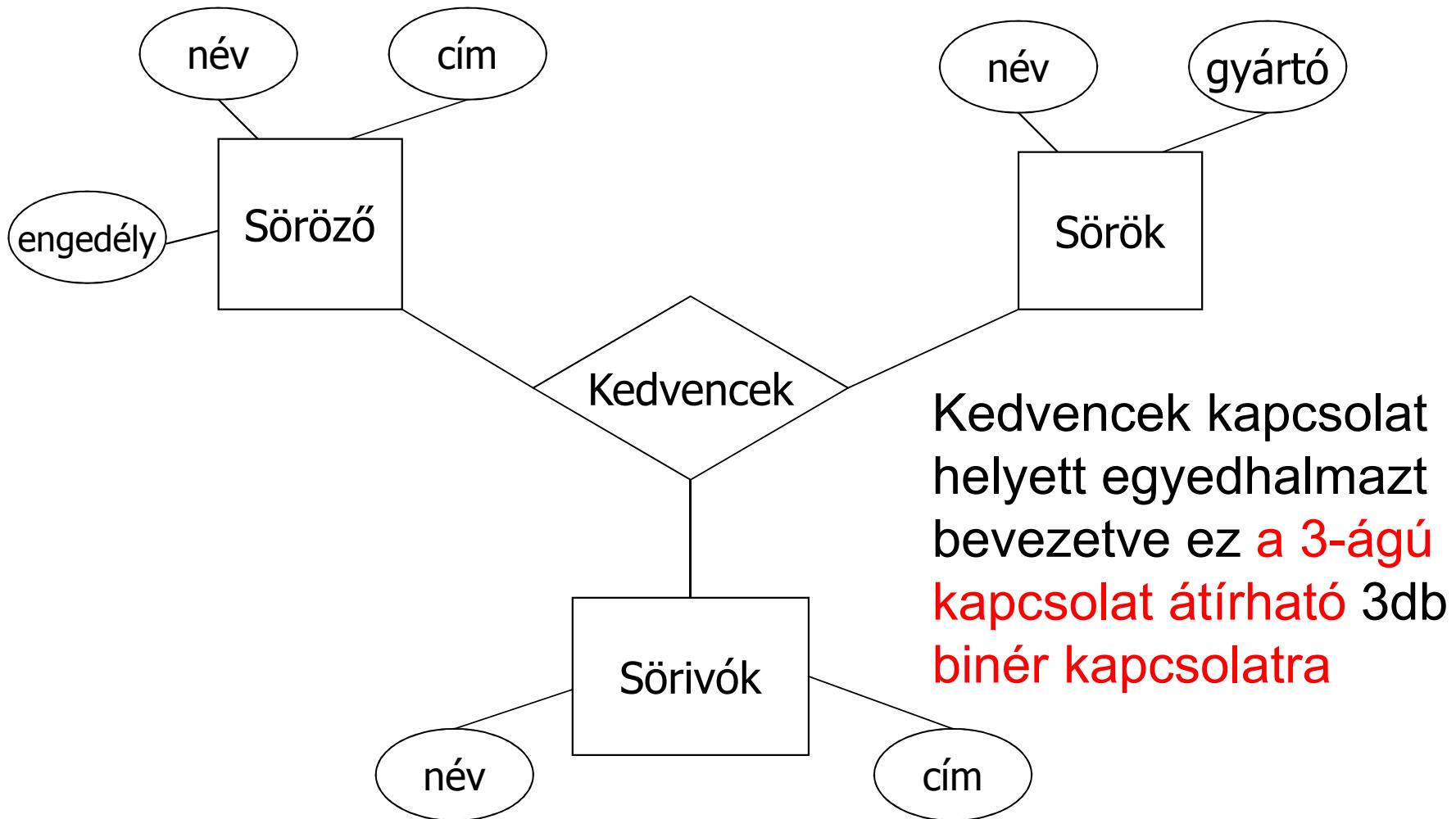


A kapcsolat előfordulása

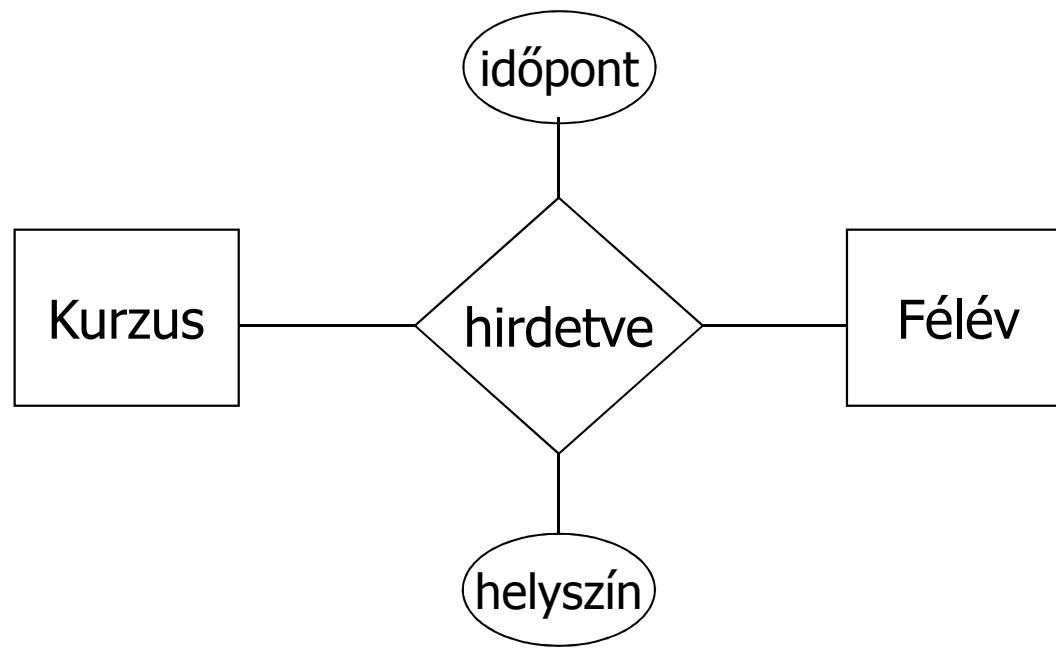
Férj	Feleség
Bob	Ann
Joe	Sue
...	...

Dolgozók egyedhalmaz is kapcsolódik önmagával:
ki kinek a főnöke (sok-egy kapcsolat)

Példa: Többágú (3-ágú) kapcsolatra



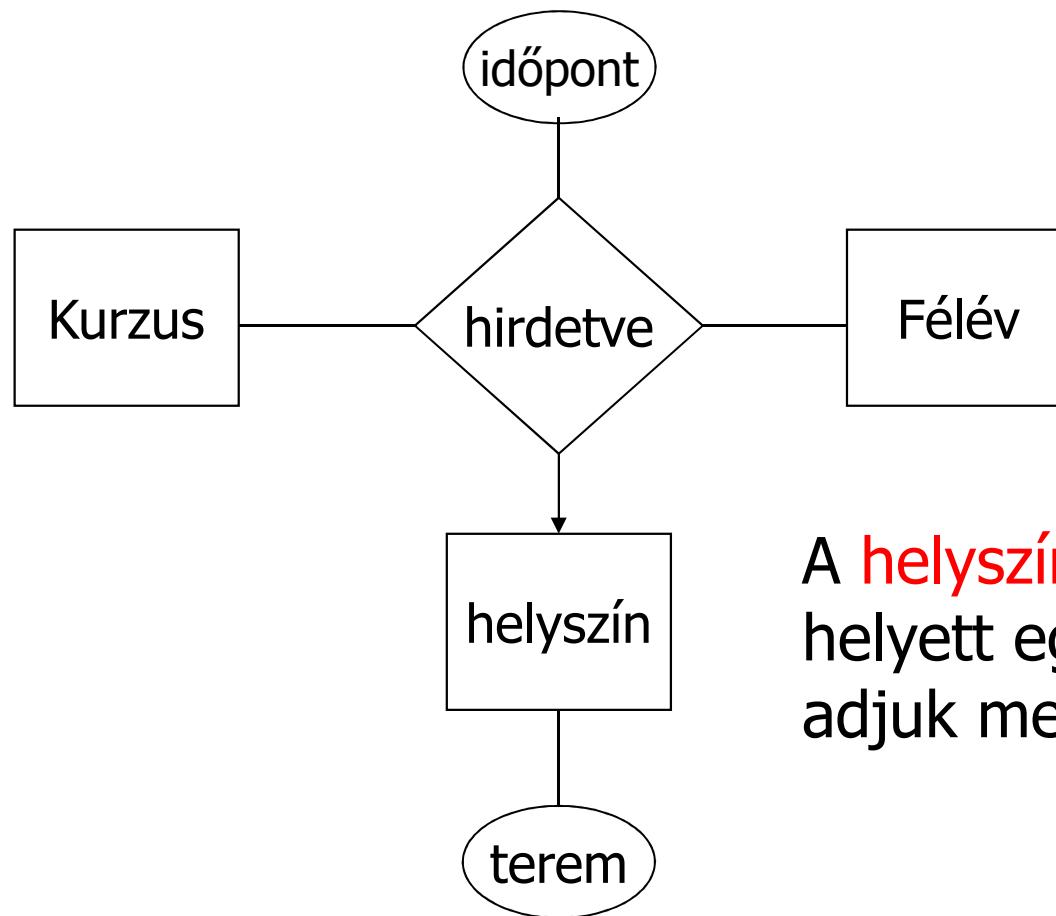
Kapcsolatnak is lehet attribútuma



Az **időpont** és **helyszín** a Kurzus és Félév együttes függvénye, de egyiké sem külön.

Tervezési kérdés:

Attribútum vagy egyedhalmaz?



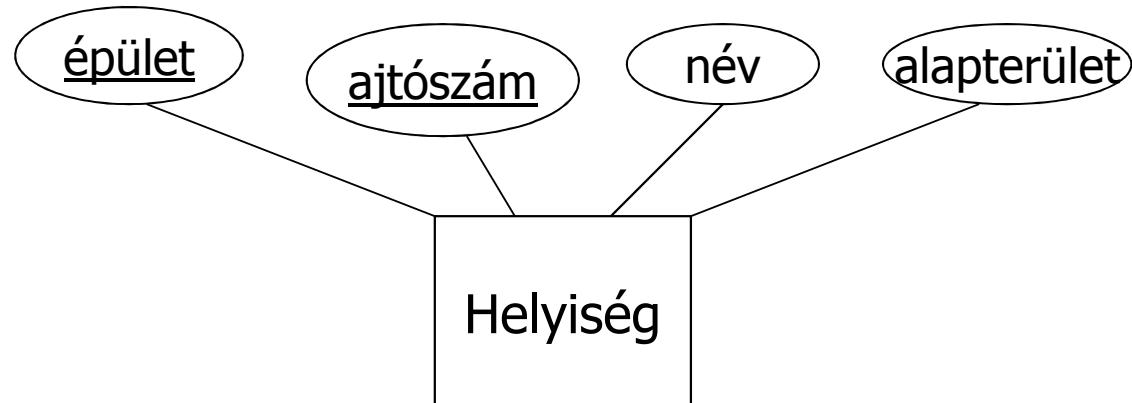
A **helyszínt** itt attribútum
helyett egyedhalmazként
adjuk meg

Kulcs megszorítás jele: aláhúzás

Példa egyszerű kulcsra: név a Sörök elsődleges kulcsa:



Példa összetett kulcsra: épület, ajtószám két-attribútumos
elsődleges kulcsa a Helyiség-nek:



E/K-diagram átírása

Tankönyv 4.5.-4.6. E/K-diagram átírása relációkká

- Egyedhalmazok átírása relációkká
 - E/K-kapcsolatok átírása relációkká
 - Egyszerűsítés, összevonások
- folyt.köv. későbbi előadáson:
- Gyenge egyedhalmazok kezelése
 - Osztályhierarchia átalakítása relációkká

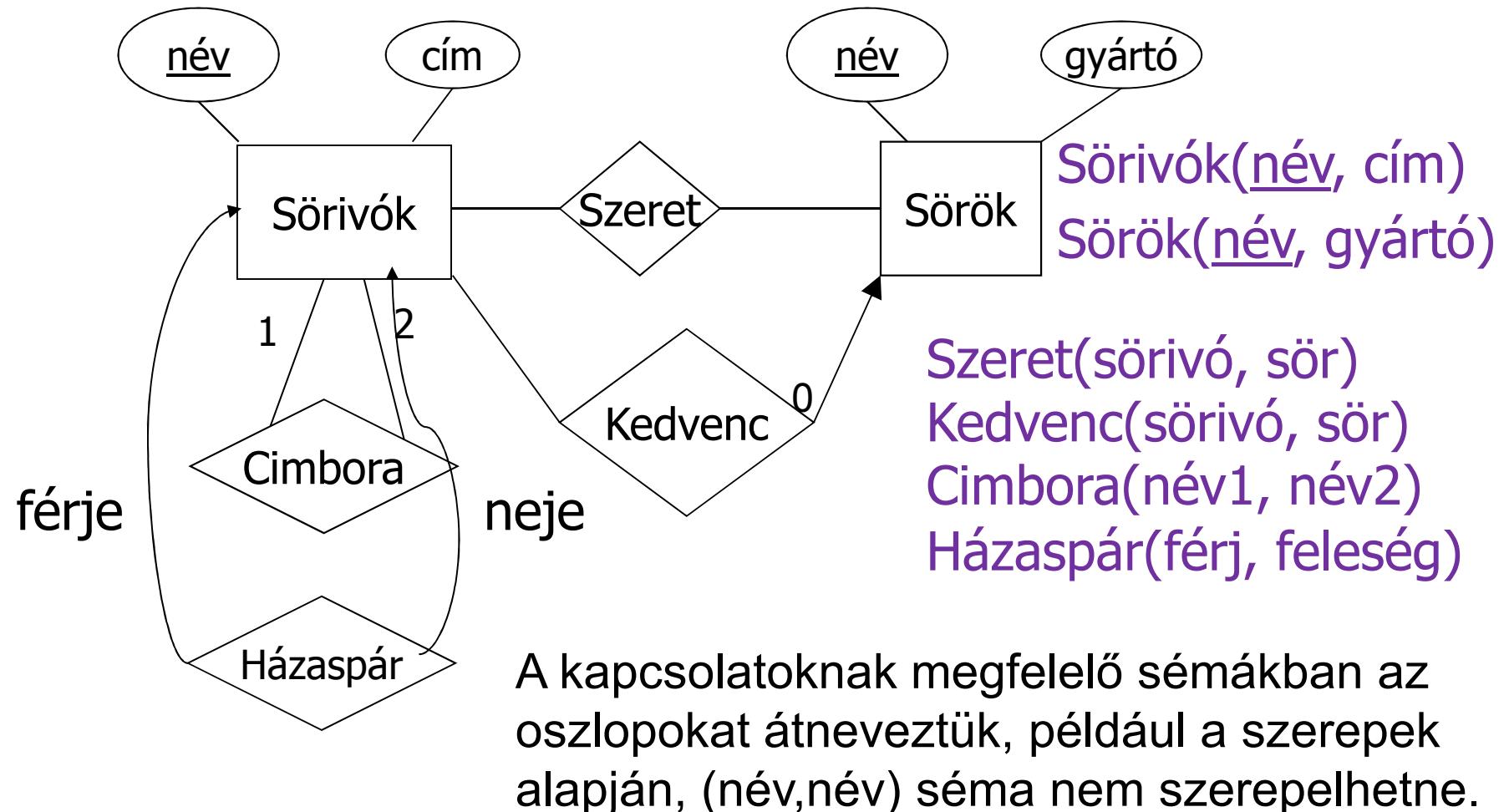
E/K diagram átírása relációs adatbázistervre

Mi minek felel meg:

➤ egyedhalmaz séma $E(A_1, \dots, A_n)$	↔ relációséma $E(A_1, \dots, A_n)$
➤ tulajdonságok	↔ attribútumok
➤ (szuper)kulcs	↔ (szuper)kulcs
➤ egyedhalmaz előfordulása	↔ reláció
➤ e egyed	↔ (e(A_1), \dots, e(A_n)) sor
➤ $R(E_1, \dots, E_p, A_1, \dots, A_q)$ kapcsolati séma, ahol Ei egyedhalmaz, Aj saját tulajdonság	↔ R(K_1, \dots, K_p, A_1, \dots, A_q) relációséma, ahol Ki az Ei (szuper)kulcsa
➤ E/K modell	↔ Relációs adatmodell

Példa: E/K diagram átírása relációkká

Az egyedek és a kapcsolatok átírása:



Relációk összevonása

- Összevonhatunk 2 relációt, ha az egyik egy **sok-egy** kapcsolatnak megfelelő reláció, a másik pedig a sok oldalon álló egyedhalmaznak megfelelő reláció.
- Példa:
Sörivók(név, cím) és **Kedvenc(ivó,sör)** összevonható, és kapjuk az **Sörivó1(név,cím,kedvencSöre)** sémát.
- Hasonlóan a Dolgozók táblába összevonható az Osztállyal illetve önmagával való kapcsolat

Példa – Sörivók adatbázisséma

- Az előadások SQL lekérdezései az alábbi Sörivók adatbázissémán alapulnak
(aláhúzás jelöli a kulcs attribútumokat)

Sörök(név, gyártó)

Sörözők(név, város, tulaj, engedély)

Sörivók(név, város, tel)

Szeret(név, sör)

Felszolgál(söröző, sör, ár)

Látogat(név, söröző)

Táblák és megszorítások megadása

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009

2.3.(folyt) és 7.1. Több táblára
vonatkozó megszorítások
megadása (SQL DDL)



Ismétlés: relációsémák definiálása

- Az SQL tartalmaz **adatleíró részt (DDL)**, az adatbázis **objektumainak** a leírására és megváltoztatására.
Objektumok leíró parancsa a **CREATE** utasítás.
 - CREATE – létrehozni, az objektumok leíró parancsa
 - DROP – eldobni, a teljes leírást és minden kapcsolódott hozzáférhetetlenné válik
 - ALTER – módosítani a leírást
- A relációt az SQL-ben táblának (TABLE) nevezik, az SQL alapvetően háromféle táblát kezel:
 - Alaptáblák [CREATE | ALTER | DROP] TABLE
 - Nézetttáblák [CREATE [OR REPLACE] | DROP] VIEW
 - Átmeneti munkatáblák (WITH záradéka a SELECT-nek)
- **Alaptáblák** megadása: **CREATE TABLE**

Tankönyv példa: Filmek séma

Filmek(

cím:string,
év:integer,
hossz:integer,
műfaj:string,
stúdióNév:string,
producerAzon:integer)

FilmSzínész(

név:string,
cím:string,
nem:char,
születésiDátum:date)

Stúdió(

név:string,
cím:string,
elnökAzon:integer)

Mit jelentenek az aláhúzások?

Tankönyv példája, hibás fordítás:
title=(film)cím és address=(lak)cím

Tervezéssel később foglalkozunk, ez a példa hibás, az elnevezések, de így jó lesz, hogy a lekérdezésekben megnézzük hogyan kezeljük.

SzerepelBenne(

filmCím:string,
filmÉv:integer,
színészNév:string)

GyártásIrányító(

név:string,
cím:string,
azonosító:integer,
nettóBevétel:integer)

Példa megszorításokra: Kulcs

- Előző példában: attribútumok aláhúzása mit jelent?
- Filmek: elvárjuk, hogy ne legyen a megengedett előfordulásokban két különböző sor, amelyek megegyeznek cím, év attribútumokon.
- Egyszerű kulcs egy attribútumból áll, de egy kulcs nem feltétlenül áll egy attribútumból, ez az összetett kulcs. Például a **Filmek** táblában a cím és év együtt alkotják a kulcsot, nem elég a cím, ugyanis van például (King Kong, 1933), (King Kong, 1976) és (King Kong, 2005).
- A kulcsot aláhúzás jelöli: **Filmek (cím, év, hossz, ...)**

Kulcsra vonatkozó megszorítások

- 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.
- Formális megadása:

$$R(U), X \subseteq U, U = \{A_1, \dots, A_n\}, X = \{A_{j_1}, \dots, A_{j_k}\}$$
$$t \in R, t[X] < A_{j_1} : t(A_{j_1}), \dots, A_{j_k} : t(A_{j_k}) >$$

ezzel a jelöléssel mit jelent, hogy X kulcs elvárás?

ha $t_1 \in R, t_2 \in R$ és $t_1[X] = t_2[X]$ akkor $t_1 = t_2$

Kulcs megadása

- PRIMARY KEY vagy UNIQUE
- Nincs a relációnak két olyan sora, amely a lista minden attribútumán megegyezne.
- Kulcs esetén nincs értelme a DEFAULT értéknek.
- Kulcsok megadásának két változata van:
 - Egyszerű kulcs (egy attribútum) vagy
 - Összetett kulcs (attribútumok listája)

Egyszerű kulcs megadása

- Ha a kulcs egyetlen attribútum, akkor ez az attribútum deklarációban megadható
 - <attribútumnév> <típus> **PRIMARY KEY**
 - vagy <attribútumnév> <típus> **UNIQUE**
- Példa:

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

Összetett kulcs megadása

- Ha a kulcs több attribútumból áll, akkor a CREATE TABLE utasításban az attribútum deklaráció után a kiegészítő részben meg lehet adni további tábla elemeket: **PRIMARY KEY (attrnév₁, ... attrnév_k)**
- Példa:

```
CREATE TABLE Felszolgál (
    söröző      CHAR(20) ,
    sör          VARCHAR2(20) ,
    ár           NUMBER(10,2) ,
    PRIMARY KEY (söröző, sör)
) ;
```

PRIMARY KEY vs. UNIQUE

- Csak egyetlen **PRIMARY KEY** lehet a relációban, viszont **UNIQUE** több is lehet.
- PRIMARY KEY egyik attribútuma sem lehet **NULL** érték egyik sorban sem. Viszont UNIQUE-nak deklarált attribútum lehet NULL értékű, vagyis a táblának lehet olyan sora, ahol a UNIQUE attribútum értéke **NULL vagyis hiányzó érték**.
- az SQL lekérdezésnél adjuk meg hogyan kell ezzel a speciális értékkel gázdálkodni, hogyan lehet NULL-t kifejezésekben és hogyan lehet feltételekben használni
- Következő héten visszatérünk a megszorítások és a hivatkozási épség megadására.

Idegen kulcsok megadása

- Az első előadáson a táblák létrehozásához veszünk kiegészítő lehetőségeket: **Kulcs és idegen kulcs (foreign key) hivatkozási épség megadása**
- Az egyik tábla egyik oszlopában szereplő értékeknek szerepelnie kell egy másik tábla bizonyos attribútumának az értékei között.
- A **hivatkozott attribútumoknak** a másik táblában kulcsnak kell lennie! (PRIMARY KEY vagy UNIQUE)
- Példa: **Felszolgál(söröző, sör, ár)** táblára megszorítás, hogy a sör oszlopában szereplő értékek szerepeljenek a **Sörök(név, gyártó)** táblában a név oszlop értékei között.

Idegen kulcs megadása: attribútumként

REFERENCES kulcsszó használatának két lehetősége:
attribútumként vagy sémaelemként lehet megadni.

1.) Attribútumonként (egy attribútumból álló kulcsra)

Példa:

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

```
CREATE TABLE Felszolgál (
    söröző CHAR(20),
    sör   CHAR(20) REFERENCES Sörök(név),
    ár     REAL );
```

Idegen kulcs megadása: sémaelemként

2.) Sémaelemként (egy vagy több attr.-ból álló kulcsra)

FOREIGN KEY (attribútum lista)

REFERENCES relációnév (attribútum lista)

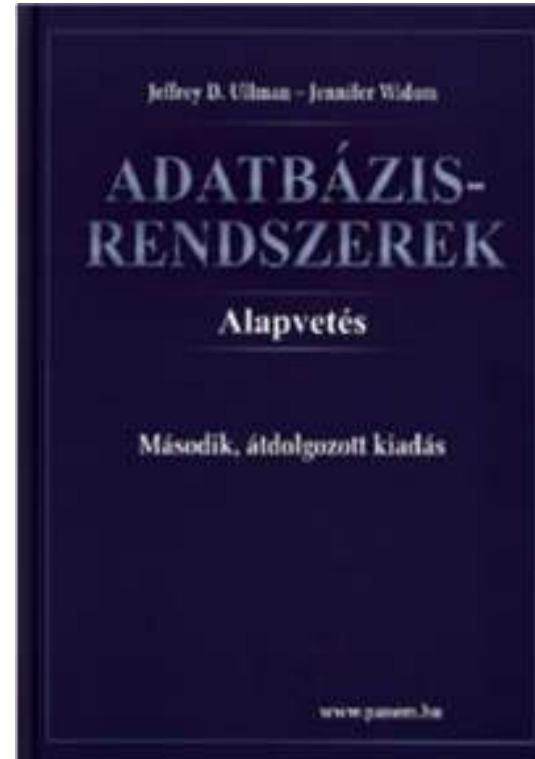
Példa:

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

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

Lekérdezések: Relációs algebra

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009



2.4. Relációs algebra, mint
lekérdező nyelv (később
ebben tudjuk értelmezni
az SQL SELECT utasítást)

Lekérdezések: Mi az algebra?

- Nyelv: a kérdés szintaktikai alakja és a kérdés kiértékelése (algoritmus) kiértékelési szemantika
- Algebra **műveleteket** és **atomi operandusokat** tartalmaz.
- **Relációs algebra:** az atomi operandusokon és az algebrai kifejezéseken végzett műveletek alkalmazásával kapott relációkon műveleteket adunk meg, kifejezéseket építünk (a kifejezés felel meg a kérdés szintaktikai alakjának).
- Fontos tehát, hogy **minden művelet végeredménye reláció**, amelyen további műveletek adhatók meg.
- A relációs algebra atomi operandusai a következők:
 - a relációhoz tartozó **változók**,
 - **konstansok**, amelyek véges relációt fejeznek ki.

Relációs algebrai lekérdező nyelv ---1

Relációs algebrai kifejezés, mint lekérdező nyelv

Lekérdező nyelv: L -nyelv

Adott az adatbázis sémája: $\mathbb{R} = \{R_1, \dots, R_k\}$

$q \in L \quad q: R_1, \dots, R_k \rightarrow V$ (eredmény-reláció)

E - relációs algebrai kifejezés: $E(R_1, \dots, R_k) = V$ (output)

Relációs algebrai kifejezések formális felépítése

- **Elemi kifejezések** (alapkifejezések)
 - (i) $R_i \in \mathbb{R}$ (az adatbázis-sémában levő relációnevek)
 R_i kiértékelése: az aktuális előfordulása
 - (ii) konstans reláció (véges sok, konstansból álló sor)
- **Összetett kifejezések** (folyt. köv.oldalon)

Relációs algebrai lekérdező nyelv ---2

(folyt.) Relációs algebrai kifejezések felépítése

- Összetett kifejezések
- Ha E_1, E_2 kifejezések, akkor a következő E is kifejezés
 - $E := \Pi_{\text{lista}} (E_1)$ vetítés (típus a lista szerint)
 - $E := \sigma_{\text{Feltétel}} (E_1)$ kiválasztás (típus nem változik)
 - $E := E_1 \cup E_2$ unió, ha azonos típusúak (és ez a típusa)
 - $E := E_1 - E_2$ különbség, ha E_1, E_2 azonos típusúak (típus)
 - $E := E_1 \bowtie E_2$ term. összekapcsolás (típus attr-ok uniója)
 - $E := \rho_{S(B_1, \dots, B_k)}(E_1(A_1, \dots A_k))$ átnevezés (típ.új attr.nevek)
 - $E := (E_1)$ kifejezést zárójelezve is kifejezést kapunk
- Ezek és csak ezek a kifejezések, amit így meg tudunk adni

1ea: Vetítés (project, jelölése pí: Π)

- **Vetítés** (projekció). Adott relációt vetít le az alsó indexben szereplő attribútumokra (attribútumok számát csökkentik)
- $\Pi_{\text{lista}}(R)$ ahol lista: $\{A_{i_1}, \dots, A_{i_k}\}$ R-sémájában levő attribútumok egy részhalmazának felsorolása
eredmény típusa $\langle A_{i_1}: \text{értéktípus}_{i_1}, \dots, A_{i_k}: \text{értéktípus}_{i_k} \rangle$
 $\Pi_{\text{lista}}(R) := \{ t.A_{i_1}, t.A_{i_2}, \dots, t.A_{i_k} \mid t \in R \} = \{ t[\text{lista}] \mid t \in R \}$
- Reláció soraiból kiválasztja az attribútumoknak megfelelő A_{i_1}, \dots, A_{i_k} -n előforduló értékeket, ha többször előfordul akkor a duplikátumokat kiszűrjük (hogy halmazt kapunk)

➤ **Példa:**

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

$\Pi_{A, B}(R)$



A	B
a	b
c	d

1ea: Kiválasztás (select, jelölése szigma: σ)

- Kiválasztás (szűrés). Kiválasztja az argumentumban szereplő reláció azon sorait, amelyek eleget tesznek az alsó indexben szereplő feltételnek.
- $\sigma_{\text{Feltétel}}(R)$ és R sémája megegyezik
- $\sigma_{\text{Feltétel}}(R) := \{ t \mid t \in R \text{ és } t \text{ kielégíti az } F \text{ feltételt}\}$
- $R(A_1, \dots, A_n)$ séma feletti reláció esetén a σ_F kiválasztás F feltétele a következőképpen épül fel:
 - elemi feltétel: $A_i \theta A_j$, $A_i \theta c$, ahol c konstans, θ pedig $=, \neq, <, >, \leq, \geq$
 - összetett feltétel: ha B_1, B_2 feltételek, akkor $\neg B_1, B_1 \wedge B_2, B_1 \vee B_2$ és zárójelezésekkel is feltételek

➤ Példa:

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

$\sigma_{A=a \vee C=d}(R)$



A	B	C
a	b	c
g	a	d

1ea: Átnevezés (rename, jelölése ró: ρ)

- Miért van erre szükség? Nem tudjuk a reláció saját magával való szorzatát kifejezni, $R \bowtie R = R$ lesz.
- Láttuk, hogy egyes esetekben szükség lehet relációnak vagy a reláció attribútumainak **átnevezésére**:

$$\rho_{T(B_1, \dots, B_k)}(R(A_1, \dots A_k))$$

- Ha az attribútumokat nem szeretnénk átnevezni, csak a relációt, ezt $\rho_T(R)$ -rel jelöljük. Ha ugyanazt a táblát használjuk többször, akkor a táblának adunk másik hivatkozási (alias) nevet.
- Az attribútumok átnevezése helyett alternatíva: R.A (vagyis relációnév.attribútumnév hivatkozás) amivel meg tudjuk különböztetni a különböző táblákban származó azonos nevű attribútumokat.

Halmazműveletek (jelölése a szokásos)

- Reláció előfordulás véges sok sorból álló halmaz. Így értelmezhetők a szokásos halmazműveletek: az **unió** (az eredmény halmaz, csak egyszer szerepel egy sor) értelmezhető a **metszet** és a **különbség**. Milyen művelet van még halmazokon? Értelmezhető-e relációkon?
- R, S és azonos típusú, $R \cup S$ és $R - S$ típusa ugyanez
 $R \cup S := \{t \mid t \in R \vee t \in S\}$, $R - S := \{t \mid t \in R \wedge t \notin S\}$
- Az alapműveletekhez az **unió** és **különbség** tartozik, **metszet** műveletet származtatjuk $R \cap S = R - (R - S)$

➤

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

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

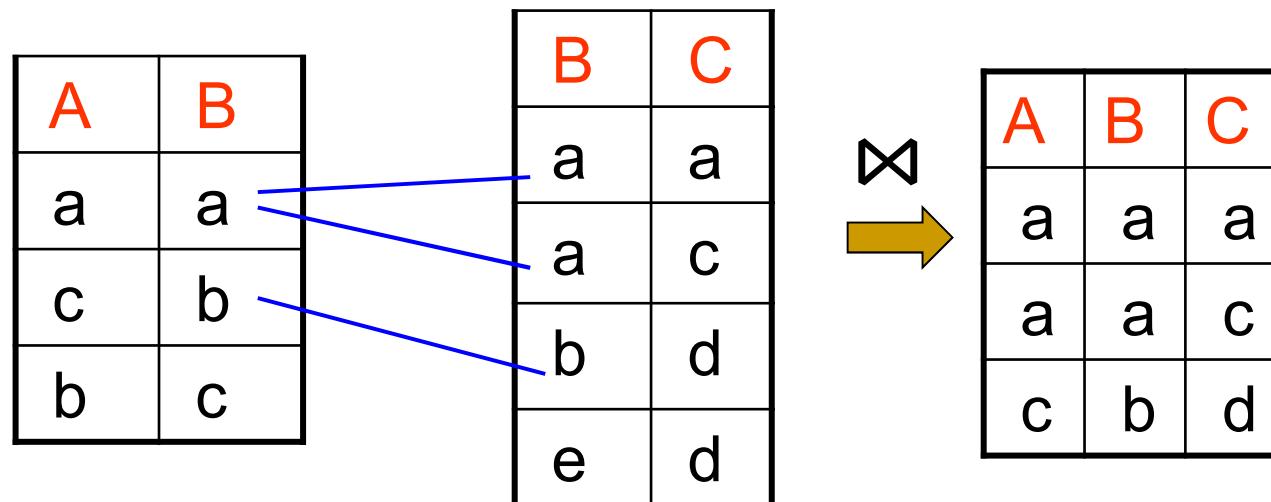
Példa: különbségre

$R - S$ 

A	B	C
g	a	d

Természetes összekapcsolás ---1

- Szorzás jellegű műveletek (attribútumok számát növeli) többféle lehetőség, amelyekből csak egyik alapművelet:
- Angolul: Natural Join (jelölése: „csokornyakkendő”)
- Természetes összekapcsolás: közös attribútum-nevekre épül. $R \bowtie S$ azon sorpárokat tartalmazza R-ből illetve S-ből, amelyek R és S azonos attribútumain megegyeznek.



Természetes összekapcsolás ---2

- Természetes összekapcsolás:
- Legyen $R(A_1, \dots, A_k, B_1, \dots, B_n)$, illetve $S(B_1, \dots, B_n, C_1, \dots, C_m)$
- $R \bowtie S$ típusa $(A_1, \dots, A_k, B_1, \dots, B_n, C_1, \dots, C_m)$ vagyis a két attribútum-halmaz uniója
- $R \bowtie S = \{ \langle A_1: t(A_1), \dots, A_k: t(A_k), B_1: t(B_1), \dots, B_n: t(B_n), C_1: s(C_1), \dots, C_m: s(C_m) \rangle \mid t \in R, s \in S, t(B_i) = s(B_i) \text{ } i=1, \dots, n \}$
- $R \bowtie S$ elemei $v \in R \bowtie S$
$$R \bowtie S = \{ v \mid \exists t \in R, \exists s \in S: t[B_1, \dots, B_n] = s[B_1, \dots, B_n] \wedge \wedge v[A_1, \dots, A_k] = t[A_1, \dots, A_k] \wedge v[B_1, \dots, B_n] = t[B_1, \dots, B_n] \wedge \wedge v[C_1, \dots, C_m] = s[C_1, \dots, C_m] \}$$

Természetes összekapcsolás ---3

- Példákban: két azonos nevű attribútumot úgy tekintünk, hogy ugyanazt jelenti és a közös érték alapján fűzzük össze a sorokat.
- Milyen problémák lehetnek?
- Filmek adatbázisban ugyanarra a tulajdonságra más névvel hivatkozunk: Filmek.év és SzerepelBenne.filmÉv, illetve FilmSzínész.név és SzerepelBenne.színészNév
- Termékek adatbázisban pedig ugyanaz az azonosító mást jelent: Termék.típus más, mint Nyomtató.típus
- Emiatt a Filmek és a Termékek adatbázisokban ahoz, hogy jól működjön az összekapcsolás **szükségünk van** egy technikai műveletre, és ez: **az átnevezés (rename)**

Szorzás jellegű műveletek ---1

- Szorzás jellegű műveletek többféle lehetősége közül csak az egyiket vesszük alapműveletnek: **join** vagy **természetes összekapcsolást**, amely közös attribútumnevekre épül.
 $R \bowtie S$ azon sorpárokat tartalmazza R -ből illetve S -ből, amelyek R és S azonos attribútumain megegyeznek.
- Egy másik lehetőség: **direkt-szorzat (Descartes-szorzat)**
Ez is tekinthető alapműveletnek (és bizonyos esetekben egyszerűbb ezt venni alapműveletnek) az ennél sokkal gyakrabban használt **természetes összekapcsolás** helyett.
- **$R \times S$** : az R és S minden sora párból összefűződik, az első tábla minden sorához hozzáfüzzük a második tábla minden sorát

$$R \times S := \{ t \mid t[R] \in R \text{ és } t[S] \in S \}$$

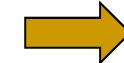
Szorzás jellegű műveletek ---2

- A **direkt-szorzat** (vagy szorzat, **Descartes-szorzat**) esetén természetesen nem fontos az attribútumok egyenlősége. A két vagy több reláció azonos nevű attribútumait azonban meg kell különböztetni egymástól.
Hivatkozás séma: oszlopok átnevezése illetve azonos nevű oszlop esetén: $R.A_1, \dots, R.A_k, S.A_1, \dots, S.A_k$
- **Példa:**

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

B	D
b	r
q	s

$R \times S$



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

Szorzás jellegű műveletek ---3

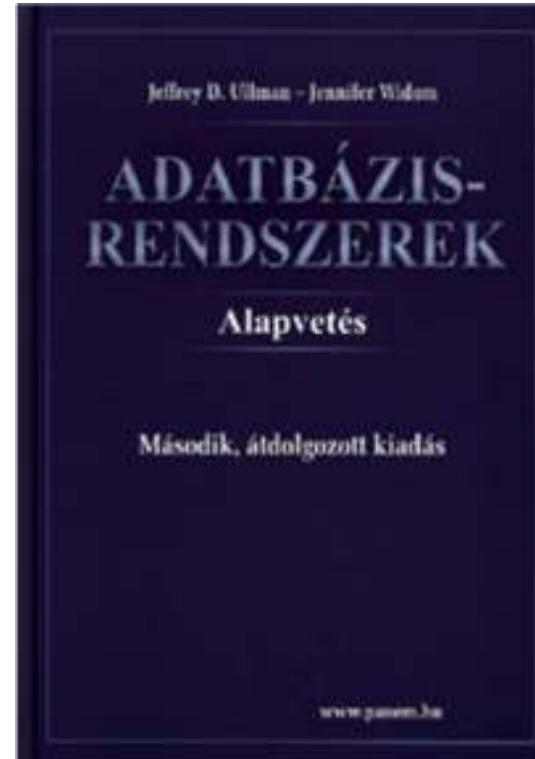
- Ha R, S sémái megegyeznek, akkor $R \bowtie S = R \cap S$.
- Ha R, S sémáiban nincs közös attribútum, akkor $R \bowtie S = R \times S$.
- Később nézünk még további szorzás jellegű műveletet:
Théta összekapcsolás \bowtie_θ , félig összekapcsolás \bowtie , és a rel.algebra kiterjesztésénél külső összekapcsolásokat.
- Hogyan fejezhető ki az $R \times S$ direkt szorzat relációs algebrában? (ha a természetes összekapcsolást tekintjük alapműveletnek, ebből és az átnevezés segítségével felírható a direkt szorzat).
- Hogyan fejezhető ki a természetes összekapcsolás, ha a direkt szorzatot soroljuk az alapműveletek közé?

Lekérdezések kifejezése algebrában

- Következő előadáson folytatjuk (példák)

Több táblára vonatkozó lekérdezések

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009



6.2. Több táblára vonatkozó
lekérdezések az SQL-ben,
lekérdezések értelmezése

Select-From-Where (SFW) utasítás

- Gyakran előforduló relációs algebrai kifejezés $\Pi_{\text{Lista}} (\sigma_{\text{Felt}} (R_1 \times \dots \times R_n))$ típusú kifejezések
- Szorzat és összekapcsolás az SQL-ben
- SELECT s-lista -- milyen típusú sort szeretnénk az eredményben látni?
FROM f-lista -- relációk (táblák) összekapcsolása, illetve szorzata
WHERE felt -- milyen feltételeknek eleget tevő sorokat kell kiválasztani?
- FROM f-lista elemei (ezek ismétlődhetnek)
táblanév [[AS] sorváltozó, ...]
Itt: a from lista elemei a táblák direkt szorzatát jelenti, az összekapcsolási feltételt where-ben adjuk meg, később bevezetünk majd tovább lehetőségeket a különböző összekapcsolásokra az SQL from záradékában.

Attribútumok megkülönböztetése ---1

- Milyen problémák merülnek fel?
- (1) Ha egy attribútumnév több sémában is előfordul, akkor nem elég az attribútumnév használata, mert ekkor nem tudjuk, hogy melyik sémához tartozik.
- Ezt a problémát az SQL úgy oldja meg, hogy megengedi egy relációnévnek és egy pontnak a használatát egy attribútum előtt: **R.A** (az R reláció A attribútumát jelenti).
- Természetes összekapcsolás legyen $R(A, B)$, $S(B,C)$

```
SELECT A, R.B B, C  
FROM R, S  
WHERE R.B=S.B;
```

Attribútumok megkülönböztetése ---2

- Milyen problémák merülnek még fel?
- (2) Ugyanaz a reláció többször is szerepelhet, vagyis szükség lehet arra, hogy ugyanaz a relációnév többször is előforduljon a FROM listában.
- Ekkor a FROM listában a táblához másodnevet kell megadni, erre **sorváltozóként** is szoktak hivatkozni, megadjuk, h. melyik sorváltozó melyik relációt képviseli:
FROM R₁ [t₁], ..., R_n [t_n]
Ekkor a SELECT és WHERE záradékok kifejezésekben a hivatkozás: **t_i.A** (vagyis sorváltozó.attribútumnév)

SFW szabvány alapértelmezése ---1

- Kiindulunk a **FROM záradékból**: a FROM lista minden eleméhez **egy beágyazott ciklus**, végigfut az adott tábla sorain a ciklus minden lépésénél az n darab sor változónak lesz egy-egy értéke
- ehhez kiértékeljük a WHERE feltételt, vagyis elvégezzük a **WHERE záradékban** szereplő feltételnek eleget tevő sorok kiválasztását (csak a helyesek, ahol TRUE=igaz választ kapunk), azok a sorok kerülnek az eredménybe.
- Alkalmazzuk a **SELECT záradékban** jelölt kiterjesztett projekciót. Az **SQL-ben az eredmény alapértelmezés szerint** itt sem halmaz, hanem **multihalmaz**.
Ahhoz, hogy halmazt kapunk, azt külön kérni kell:
SELECT DISTINCT Lista

SFW szabvány alapértelmezése ---2

FOR t_1 sorra az R_1 relációban DO

FOR t_2 sorra az R_2 relációban DO

...

FOR t_n sorra az R_n relációban DO

IF a where záradék igaz, amikor az attribútumokban
 t_1, t_2, \dots, t_n megfelelő értékei találhatóak

THEN

t_1, t_2, \dots, t_n -nek megfelelően kiértékeljük a
select záradék attribútumait
és az értékekből alkotott sort
az eredményhez adjuk

SFW szabvány alapértelmezése ---3

SELECT [DISTINCT] kif₁ [[AS] onév₁], ..., kif_n [[AS] onév_n]
FROM R₁ [t₁], ..., R_n [t_n]
WHERE feltétel (vagyis logikai kifejezés)

Alapértelmezés (a műveletek szemantikája -- általában)

- A FROM záradékban levő relációkhöz tekintünk egy-egy **sor változót**, amelyek a megfelelő reláció minden sorát bejárják (beágyazott ciklusban)
- minden egyes „aktuális” sorhoz kiértékeljük a WHERE záradékot
- Ha helyes (vagyis igaz) választ kaptunk, akkor képezünk egy sort a SELECT záradékban szereplő kifejezéseknek megfelelően.

Megj.: konverzió relációs algebrába

SELECT [DISTINCT] kif₁ [[AS] onév₁], ..., kif_n [[AS] onév_n]

FROM R₁ [t₁], ..., R_n [t_n]

WHERE feltétel (vagyis logikai kifejezés)

- 1.) A FROM záradék sor változóból indulunk ki, és tekintjük a hozzájuk tartozó relációk Descartes-szorzatát. Átnevezéssel valamint R.A jelöléssel elérjük, hogy minden attribúumnak egyedi neve legyen.
- 2.) A WHERE záradékot átalakítjuk egy kiválasztási feltétellel, melyet alkalmazunk az elkészített szorzatra.
- 3.) Végül a SELECT záradék alapján létrehozzuk a kifejezések listáját, a (kiterjesztett) vetítési művelethez.

$$\Pi_{\text{onév}1, \dots, \text{onévn}} (\sigma_{\text{feltétel}} (R_1 \times \dots \times R_n))$$

Példa: Két tábla összekapcsolása ---1

- Mely söröket szeretik a Joe's Bárba járó sörivók?

```
SELECT sör
      FROM Szeret, Látogat
     WHERE bár = 'Joe''s Bar'
       AND Látogat.név = Szeret.név;
```

- Kiválasztási feltétel: **bár = 'Joe''s Bar'**
- Összekapcsolási feltétel: **Látogat.név = Szeret.név**
- Alapértelmezése a következő oldalon a mai órán
- Összekapcsolások szintaxisát később nézzük majd

Példa: Két tábla összekapcsolása ---2

Látogat

név	bár
Sally	Joe's

t1

Szeret

név	sör
Sally	Bud

t2

Ellenőrzés
Joe's bárja

output

Ellenőrizzük, hogy
megegyeznek-e

Tábla önmagával való szorzata ---1

- Bizonyos lekérdezésekben arra van szükségünk, hogy ugyanannak a relációnak több példányát vegyük.
- Ahhoz, hogy meg tudjuk különböztetni a példányokat a relációkat átnevezzük, másodnevét adunk, vagyis **sorváltozókat** írunk mellé a FROM záradékban.
- A relációkat mindenkor átnevezhetjük ily módon, akkor is, ha egyébként nincs rá szükség (csak kényelmesebb).
- **Példa:** **R(Szülő, Gyerek)** séma feletti relációban adott szülő-gyerek adatpárokból állítsuk elő a megállapítható Nagyszülő-Unoka párokat!

```
SELECT t1.Szülő NagySzülő, t2.Gyerek Unoka
FROM R t1, R t2
WHERE t1.Gyerek = t2.Szülő;
```

Tábla önmagával való szorzata ---2

- Példa: Sörök(név, gyártó) tábla felhasználásával keressük meg az összes olyan sörpárt, amelyeknek ugyanaz a gyártója.
 - Ne állítsunk elő (Bud, Bud) sörpárokat.
 - A sörpárokat ábécé sorrendben képezzük, például ha (Bud, Miller) szerepel az eredményben, akkor (Miller, Bud) ne szerepeljen.

```
SELECT s1.név, s2.név  
FROM Sörök s1, Sörök s2  
WHERE s1.gyártó = s2.gyártó  
AND s1.név < s2.név;
```

Halmazműveletek az SQL-ben

- Mi hiányzik még, hogy a relációs algebra alapműveleteit minden az SQL-ben vissza tudjuk adni?
- A relációs algebrai halmazműveletek: **unió, különbség** mellett az **SQL-ben ide soroljuk a metszetet is** (ugyanis fontos a metszet és az SQL-ben is implementálva van).
- Az SQL-ben a halmazműveleteket úgy vezették be, hogy azt mindig két lekérdezés között lehet értelmezni, vagyis nem relációk között, mint $R \cup S$, hanem lekérdezem az egyiket is és a másikat is, majd a lekérdezések unióját veszem.

(SFW-lekérdezés1)

[UNION | INTERSECT | {EXCEPT | MINUS}]

(SFW-lekérdezés2);

Példa: Intersect (metszet)

- Szeret(név, sör), Felszolgál(bár, sör, ár) és Látogat(név, bár) táblák felhasználásával keressük

Trükk: itt ez az alkérdezés valójában az adatbázisban tárolt tábla

azokat a sörvíköt és söröket, amelyekre a sörvíró szereti az adott sört **és** a sörvíró látogat olyan bárt, ahol felszolgálják a sört.

```
(SELECT * FROM Szeret)
INTERSECT
(SELECT név, sör
FROM Felszolgál, Látogat
WHERE Látogat.bár = Felszolgál.bár);
```

(név, sör) párok, ahol a sörvíró látogat olyan bárt, ahol ezt a sört felszolgálják

Kérdés/Válasz

- Köszönöm a figyelmet! Kérdés/Válasz?
- A 2.gyakorlat az 1.előadáshoz kapcsolódik, egy táblára vonatkozó lekérdezések: az Oracle SQL leggyakrabban használt sorfüggvényeit nézzük meg az egyszerű egy táblára vonatkozó lekérdezések SELECT és WHERE záradékaiban, lásd Oracle 3.lecke sorfüggvények
- Numerikus, karakteres, dátum, konverziós, általános, például NULL értéket megadott értékkel helyettesítő NVL és COALESCE sorfüggvényeket

Alkérdések az SQL SELECT-ben

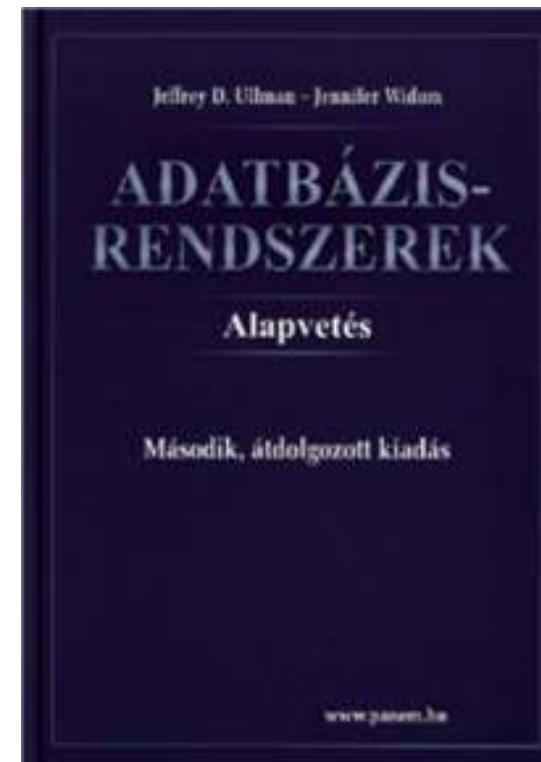
Relációs algebrai lekérdezések (példák)

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009

6.3. Alkérdések (SQL SELECT)

Folyt.2.4. Relációs algebra, mint
lekérdező nyelv

Példák: Tk. Termékek feladatai



(2EA) ismétlés: SFW alapértelmezése

- Tk.6.2.fej.: **2.EA: Több táblára vonatkozó lekérdezések:**

SELECT [DISTINCT] kif₁ [[AS] onév₁], ..., kif_n [[AS] onév_n]
FROM R₁ [t₁], ..., R_n [t_n]

WHERE feltétel (vagyis logikai kifejezés)

Alapértelmezés (a műveletek szemantikája -- általában)

- A FROM záradékban levő relációkhöz tekintünk egy-egy **sor változót**, amelyek a megfelelő reláció minden sorát bejárják (beágyazott ciklusban)
- minden egyes „aktuális” sorhoz kiértékeljük a WHERE záradékot (csak az igaz sorok kerülnek az eredménybe)
- A SELECT záradékban szereplő kifejezéseknek megfelelően képezzük a sorokat. Ha van DISTINCT, akkor az ismétlődő sorokat elhagyjuk.

(2EA) ismétlés: Halmazműveletek az SQL-ben

- A relációs algebrai halmazműveletek: **unió**, **különbség** és **metszet**, ebből csak az unió és különbség alapművelet, az SQL-ben mindenkor használható, implementálva van
- A **SELECT-FROM-WHERE** utasítások általában multihalmaz szemantikát használnak, külön kell kérni **DISTINCT**-tel ha halmazt szeretnénk kapni, viszont a **halmazműveleteknél** mégis a **halmaz szemantika** az érvényes, itt a multihalmaz szemantikát kell kérni: **ALL**
- Az SQL-ben a halmazműveleteket úgy vezették be, hogy azt mindig két lekérdezés között lehet értelmezni:

(SFW-lekérdezés1)

[**UNION [ALL]** |
INTERSECT [ALL] |
{EXCEPT | MINUS} [ALL]]

(SFW-lekérdezés2);

Alkérdések

- Zárójelezett SFW SELECT-FROM-WHERE utasításokat (**alkérdéseket**) is használhatunk a WHERE záradékban, HAVING záradékban (később lesz) és a FROM listán is.
- Szintaktikus alakja: zárójelbe kell tenni a lekérdezést
- Hol használható? Ott, ahol relációnevet használunk:
 - (1) WHERE és HAVING záradékban: kifejezésekben, feltételekben
 - (2) FROM listában: új listaelem (rel.név változó SQL-ben)
(lekérdezés) [AS] sorváltozó
 Ez felel meg annak, ahogyan a relációs algebrában tetsz.helyen használhattuk a lekérdezés eredményét.

Alkérdések a WHERE záradékban

WHERE záradékban:

- (i) Az alkérdés eredménye egyetlen **skalárérték**, vagyis az alkérdés olyan, mint a konstans, ami egy új elemi kifejezésként tetszőleges kifejezésben használható.
- (ii) **Skalár értékekből álló multihalmaz** logikai kifejezésekben használható: [NOT] EXISTS (lekérdezés)
kifejezés [NOT] IN (lekérdezés)
kifejezés Θ [ANY | ALL] (lekérdezés)
- (iii) **Teljes, többdimenziós tábla** a visszatérő érték:
[NOT] EXISTS (lekérdezés)
 (kif_1, \dots, kif_n) [NOT] IN (lekérdezés)

Alkérdések a WHERE záradékban

- Milyen változók szerepelhetnek egy alkérdésben?
 - Lokális saját változói a saját FROM listáról
 - Külső kérdés változói: ekkor az alkérdés korrelált.

Szemantikája

- Ha az alkérdés **nem korrelált**, önállóan kiértékelhető és ez az eredmény a külső kérdés közben nem változik, a külső kérdés szempontjából ez egy konstanstabla, akkor a kiértékelés minden esetben a legelsőből halad kifelé.
- **Korrelált alkérdés**, amely többször kerül kiértékelésre, minden egyes kiértékelés megfelel egy olyan értékkalának, amely az alkérdésen kívüli sorváltozóból származik (ezt később, példákon keresztül mutatjuk be)

Skalár értéket visszaadó alkérdések

- Ha egy alkérdés biztosan egy attribútumon egy sort ad vissza eredményként (egyelemű), akkor úgy használható, mint egy konstans érték.
 - az eredmény sornak egyetlen oszlopa van.
 - Futásidőjű hiba keletkezik, ha az eredmény nem tartalmaz sort, vagy több sort tartalmaz.
- **Példa:** Felszolgál(bár, sör, ár) táblában keressük meg azokat a bárokat, 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?

Skalár értéket visszaadó alkérdések

Példa: Felszolgál(bár, sör, ár) táblában keressük meg azokat a bárokat, ahol a Miller ugyanannyiba kerül, mint Joe bárjában a Bud.

SELECT bár

FROM Felszolgál

WHERE sör = 'Miller' AND

ár = (SELECT ár

FROM Felszolgál

WHERE bár = 'Joe''s bar'

AND sör = 'Bud') ;

Ennyit kér
Joe a Budért.

Tk.példa: Skalár értéket adó alkérdések

- Csillagok háborúja film gyártásirányítója:

```
SELECT név
```

```
FROM GyártásIrányító
```

```
WHERE azonosító =
```

```
(SELECT producerAzon
```

```
FROM Filmek
```

```
WHERE cím = 'Csillagok háborúja'
```

```
) ;
```

Skalár értékekből álló multihalmazt visszaadó alkérdések: ANY művelet

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

Skalár értékekből álló multihalmazt visszaadó alkérdések: 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: $x \geq \text{ALL}(\text{alkérdés})$ x az alkérdés eredményének maximum értékével azonos.

Példa: ALL

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

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

Az IN művelet a WHERE záradékban

- sor IN (alkérdés) akkor és csak akkor **igaz**, ha a sor eleme az alkérdés eredményének (itt a sor egy sor/tuple, nem sör)
 - Tagadás: sor NOT IN (alkérdés).
- Az IN-kifejezések a WHERE záradékban jelenhetnek meg
- Példa:

SELECT *

FROM Sörök

WHERE név IN (SELECT sör

A sörök,
melyeket
Fred szeret.

FROM Szeret

WHERE név = 'Fred') ;

Tk.példa: Sorokat tartalmazó feltételek

- Harrison Ford filmjeinek gyártásirányítója:

```
SELECT név
FROM GyártásIrányító
WHERE azonosító IN
    (SELECT producerAzon
     FROM Filmek
     WHERE (cím, év) IN
         (SELECT filmCím, filmév
          FROM SzerepelBenne
          WHERE színész = 'Harrison Ford' )
    );
```

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 (SELECT b FROM S);
```

Egy ciklus R sorai
fölött.

a	b	b	c
1	2	2	5
3	4	2	6

R 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
1	2
3	4

R

b	c
2	5
2	6

S

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

Az EXISTS művelet a WHERE-ben

- EXISTS (alkérdés) akkor és csak akkor igaz, ha az alkérdés eredménye nem üres.
 - Tagadása: NOT EXISTS (alkérdés)
- Példa: A Sörök(név, gyártó) táblában keressük meg azokat a söröket, amelyeken kívül a gyártójuk nem gyárt másikat.
- Ez korrelált alkérdés, többször kerül kiértékelésre, a külső tábla minden sorára kiértékeljük az alkérdést.
- A korrelált lekérdezések használata közben figyelembe kell vennünk a nevek érvényességi körére vonatkozó szabályokat.

Példa: EXISTS

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

Azon b1
sörtől
különböző
sörök,
melyeknek
ugyanaz
a gyártója.

→ WHERE gyártó = b1.gyártó
AND név <> b1.név);

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

A „nem
egyenlő”
művelet
SQL-ben.

Tk.példa: Korrelált alkérdés

- A több, mint egyszer előforduló filmcímek megkeresése:

```
SELECT DISTINCT cím
  FROM Filmek AS Régi
 WHERE év < ANY
       (SELECT év
  FROM Filmek
 WHERE cím = Régi.cím
      ) ;
```

Alkérdések a FROM záradékban

4.EA: ALKÉRDÉSEK WHERE feltételben

Folytatása következik:

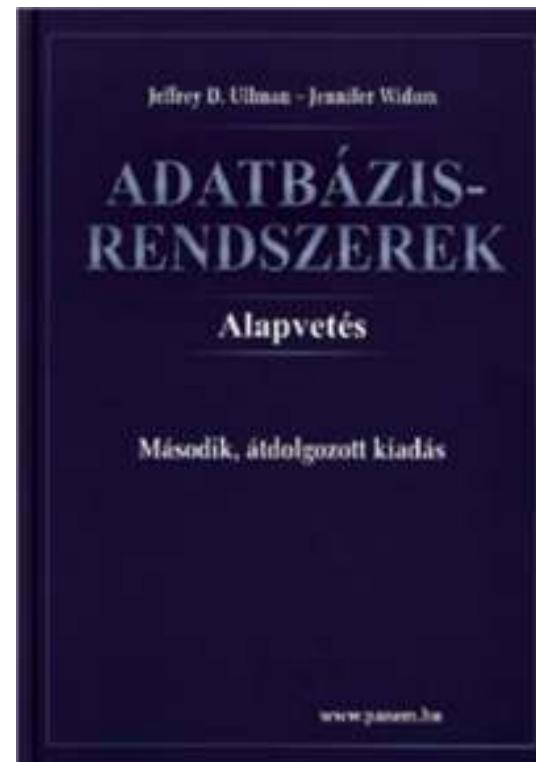
5.EA: ALKÉRDÉSEK HAVING feltételben

6.EA: ALKÉRDÉSEK FROM záradékban

Relációs algebrai lekérdezések (példák)

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009

Példák: Relációs algebra és SQL
(Tankönyv Termékek feladatai)



(2ea) ismétlés: Relációs algebra ---1

Relációs algebrai kifejezés, mint lekérdező nyelv

Lekérdező nyelv: L -nyelv

Adott az adatbázis sémája: $\mathbb{R} = \{R_1, \dots, R_k\}$

$q \in L \quad q: R_1, \dots, R_k \rightarrow V$ (eredmény-reláció)

E - relációs algebrai kifejezés: $E(R_1, \dots, R_k) = V$ (output)

Relációs algebrai kifejezések formális felépítése

- **Elemi kifejezések** (alapkifejezések)
 - (i) $R_i \in \mathbb{R}$ (az adatbázis-sémában levő relációnevek)
 R_i kiértékelése: az aktuális előfordulása
 - (ii) konstans reláció (véges sok, konstansból álló sor)
- **Összetett kifejezések** (folyt. köv.oldalon)

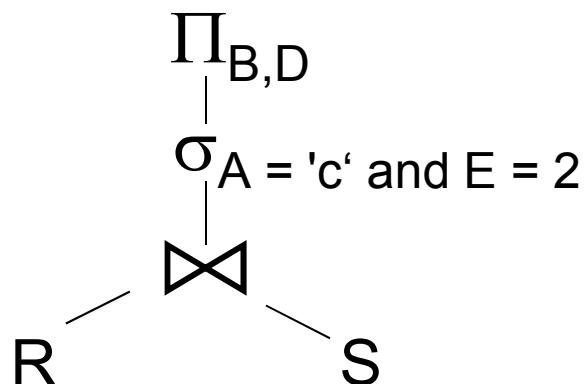
(2ea) ismétlés: Relációs algebra ---2

(folyt.) Relációs algebrai kifejezések felépítése

- Összetett kifejezések
- Ha E_1, E_2 kifejezések, akkor a következő E is kifejezés
 - $E := \Pi_{\text{lista}} (E_1)$ vetítés (típus a lista szerint)
 - $E := \sigma_{\text{Feltétel}} (E_1)$ kiválasztás (típus nem változik)
 - $E := E_1 \cup E_2$ unió, ha azonos típusúak (és ez a típusa)
 - $E := E_1 - E_2$ különbség, ha E_1, E_2 azonos típusúak (típus)
 - $E := E_1 \bowtie E_2$ term. összekapcsolás (típus attr-ok uniója)
 - $E := \rho_{S(B_1, \dots, B_k)}(E_1(A_1, \dots A_k))$ átnevezés (típ.új attr.nevek)
 - $E := (E_1)$ kifejezést zárójelezve is kifejezést kapunk
- Ezek és csak ezek a kifejezések, amit így meg tudunk adni

Lekérdezések kifejezése algebrában ---1

- Kifejezés kiértékelése: összetett kifejezést kívülről befelé haladva átírjuk kiértékelő fává, levelek: elemi kifejezések.
- A relációs algebra procedurális nyelv, vagyis nemcsak azt adjuk meg, hogy **mit** csinálunk, hanem azt is **hogyan**.
- Legyen R, S az $R(A, B, C), S(C, D, E)$ séma feletti reláció $\Pi_{B,D} \sigma_A = 'c' \text{ and } E = 2 (R \bowtie S)$
- Ehhez **a kiértékelő fa**: (kiértékelése alulról felfelé történik)



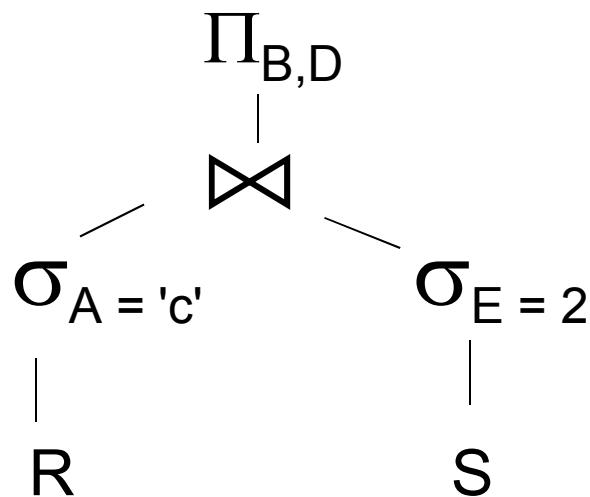
- Tudunk-e ennél jobb, hatékonyabb megoldást találni?

Lekérdezések kifejezése algebrában ---2

- Ekvivalens átalakítási lehetőségekkel, relációs algebrai azonosságokkal át tudjuk alakítani a fentivel ekvivalens másik relációs algebrai kifejezésre. Hatékonyabb-e?

$$\Pi_{B,D} (\sigma_{A = 'c'}(R) \bowtie \sigma_{E = 2}(S))$$

- Ehhez is felrajzolva a kiértékelő fát:



Lekérdezések kifejezése algebrában ---3

- **Ekvivalens átalakítás:** oly módon alakítjuk át a kifejezést, hogy az adatbázis minden lehetséges előfordulására (vagyis bármilyen is a táblák tartalma) minden esetben ugyanazt az eredményt (vagyis ugyanazt az output táblát) adja az eredeti és az átalakított kiértékelő fa.
- **Adatbázisok-2** tárgyból lesznek az **ekvivalens átalakítási szabályok**, a szabály alapú optimalizálás első szabálya például, hogy a kiválasztási műveletet minél előbb kell végrehajtani (közbülső táblák lehetőleg kicsik legyenek)
- Ha egy-egy részkifejezést, ha gyakran használjuk, akkor új változóval láthatjuk el, **segédváltozót vezethetünk be**:
 $T(C_1, \dots, C_n) := E(A_1, \dots, A_n)$, de a legvégén a bevezetett változók helyére be kell másolni a részkifejezést.

Feladatok ---1

- Először relációs algebrában táblákkal gondolkodva nézzük meg, hogy milyen műveletekkel tudjuk megkapni a kívánt eredményt, írjuk fel lineáris módon és kifejezőfákkal, majd a kifejezőfákat átírva SQL lekérdezésekre többféle megoldási lehetőséget vizsgálunk meg, vessünk össze
- A mai előadáson: Tankönyv -- Termékek feladatai:
- http://people.inf.elte.hu/sila/AB1ea/Feladatok_Termek.pdf
create table: http://people.inf.elte.hu/sila/eduAB/create_termek.txt
- További feladatok: Tankönyv -- Csatahajós feladatai:
- http://people.inf.elte.hu/sila/AB1ea/Feladatok_Csatahajok.pdf
create table: http://people.inf.elte.hu/sila/eduAB/create_csatahajok.txt

Feladatok ---2

Legyen adott az alábbi **relációs sémák** feletti relációk:

Termék (gyártó, modell, típus)

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

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

Nyomtató (modell, színes, típus, ár)

Feladatok Tk.2.4.1.feladat (ezeket a kérdéseket konkrét táblák alapján természetes módon meg lehet válaszolni, majd felírjuk relációs algebrában)

- a) Melyek azok a PC modellek, amelyek sebessége legalább 3.00
 - b) Mely gyártók készítenek legalább száz gigabájt méretű merevlemezzel rendelkező laptopot?
 - c) Adjuk meg a B gyártó által gyártott összes termék modellszámát és árát!
stb...
- !! i) Melyik gyártó gyártja a leggyorsabb számítógépet (laptopot vagy PC-t)?
- !! k) Melyek azok a gyártók, akik pontosan három típusú PC-t forgalmaznak?

Példák relációs algebrai lekérdezésekre ---1

- Relációs algebra kifejezések ilyen bevezetése valóban használható a lekérdezések megadására?
- Tk.2.4.1.feladat

➤ **Példa:** Adottak az alábbi relációs sémák feletti relációk

Termék (gyártó, modell, típus)

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

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

Nyomtató (modell, színes, típus, ár)

➤ Jelölje: T(gy, m, t)

PC(m, s, me, ml, ár)

L(m, s, me, ml, k, ár)

Ny(m, sz, t, ár)

Megj.: a két típus attr.név nem ugyanazt fejezi ki és így $T \bowtie Ny$ természetes összekapcsolásnál „zűr”

Példák relációs algebrai lekérdezésekre ---2

a.) Melyek azok a PC modellek, amelyek sebessége legalább 3.00?

Példák átírásokra ---2

a.) Melyek azok a PC modellek, amelyek sebessége legalább 3.00?

$\Pi_m(\sigma_{s \geq 3.00} (PC))$

SELECT modell
FROM PC
WHERE sebesség>=3.00;

$$\begin{array}{c} \Pi_m \\ | \\ \sigma_{s \geq 3.00} \\ | \\ PC \end{array}$$

Példák átírásokra ---2

a.) Melyek azok a PC modellek, amelyek sebessége legalább 3.00?

$\Pi_m(\sigma_{s \geq 3.00} (PC))$

SELECT modell
FROM PC
WHERE sebesség>=3.00;

$$\begin{array}{c} \Pi_m \\ | \\ \sigma_{s \geq 3.00} \\ | \\ PC \end{array}$$

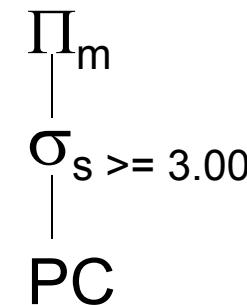
b.) Mely gyártók készítenek legalább száz gigabájt méretű merevlemezzel rendelkező laptopot?

Példák átírásokra ---2

a.) Melyek azok a PC modellek, amelyek sebessége legalább 3.00?

$$\Pi_m(\sigma_{s \geq 3.00} (PC))$$

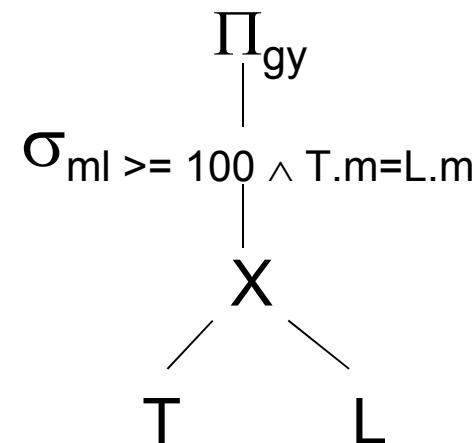
SELECT modell
FROM PC
WHERE sebesség>=3.00;



b.) Mely gyártók készítenek legalább száz gigabájt méretű merevlemezzel rendelkező laptopot?

$$\Pi_{gy} (\sigma_{ml \geq 100} (T \bowtie L))$$

SELECT gyarto
FROM Termek T, Laptop L
WHERE merevlemez>=100
AND T.modell=L.modell;



Példák átírásokra ---2

b.) Mely gyártók készítenek legalább száz gigabájt méretű merevlemezzel rendelkező laptopot?

$$\Pi_{gy} (\sigma_{ml \geq 100} (T \bowtie L))$$

SELECT gyarto

FROM Termek natural join Laptop

WHERE merevlemez >= 100

SELECT gyarto

FROM Termek T, Laptop L

WHERE merevlemez >= 100

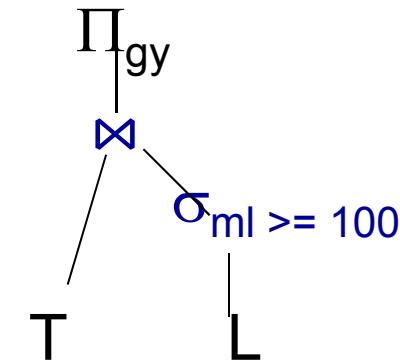
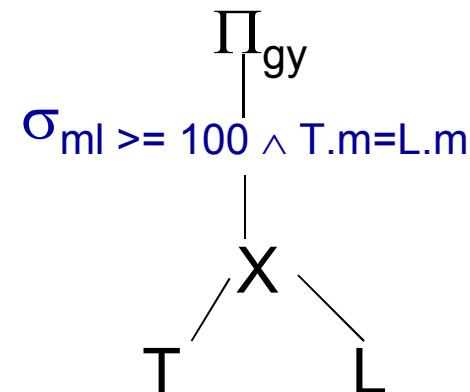
AND T.modell=L.modell;

SELECT gyarto FROM Termek

WHERE modell IN

(SELECT modell FROM PC

WHERE merevlemez >= 100);



Példák relációs algebrai lekérdezésekre ---2

a.) Melyek azok a PC modellek, amelyek sebessége legalább 3.00?

$$\Pi_m(\sigma_{s \geq 3.00}(PC))$$

b.) Mely gyártók készítenek legalább száz gigabájt méretű merevlemezzel rendelkező laptopot?

$$\Pi_{gy} (\sigma_{ml \geq 100}(T \bowtie L)) \text{ vagy ekv. } \Pi_{gy}(T \bowtie (\sigma_{ml \geq 100}(L)))$$

Példák relációs algebrai lekérdezésekre ---2

a.) Melyek azok a PC modellek, amelyek sebessége legalább 3.00?

$$\Pi_m(\sigma_{s \geq 3.00}(PC))$$

b.) Mely gyártók készítenek legalább száz gigabájt méretű merevlemezzel rendelkező laptopot?

$$\Pi_{gy}(\sigma_{ml \geq 100}(T \bowtie L)) \text{ vagy ekv. } \Pi_{gy}(T \bowtie (\sigma_{ml \geq 100}(L)))$$

c.) Adjuk meg a B gyártó által gyártott összes termék modellszámát és árát típustól függetlenül!

Példák relációs algebrai lekérdezésekre ---2

a.) Melyek azok a PC modellek, amelyek sebessége legalább 3.00?

$$\Pi_m(\sigma_{s \geq 3.00}(PC))$$

b.) Mely gyártók készítenek legalább száz gigabájt méretű merevlemezzel rendelkező laptopot?

$$\Pi_{gy}(\sigma_{ml \geq 100}(T \bowtie L)) \text{ vagy ekv. } \Pi_{gy}(T \bowtie (\sigma_{ml \geq 100}(L)))$$

c.) Adjuk meg a B gyártó által gyártott összes termék modellszámát és árát típustól függetlenül!

három részből áll (Nyomtató táblánál vigyázni, uis term.összekapcsolásnál a típus attr. itt mást jelent!)

-- segédváltozót vezetek be, legyen **BT** := $\Pi_m \sigma_{gy=B}(T)$

Példák relációs algebrai lekérdezésekre ---2

a.) Melyek azok a PC modellek, amelyek sebessége legalább 3.00?

$$\Pi_m(\sigma_{s \geq 3.00}(PC))$$

b.) Mely gyártók készítenek legalább száz gigabájt méretű merevlemezzel rendelkező laptopot?

$$\Pi_{gy}(\sigma_{ml \geq 100}(T \bowtie L)) \text{ vagy ekv. } \Pi_{gy}(T \bowtie (\sigma_{ml \geq 100}(L)))$$

c.) Adjuk meg a B gyártó által gyártott összes termék modellszámát és árát típustól függetlenül!

három részből áll (Nyomtató táblánál vigyázni, uis term.összekapcsolásnál a típus attr. itt mást jelent!)

$$\begin{aligned} \text{-- segédváltozót vezetek be, legyen } BT := \Pi_m \sigma_{gy='B'}(T) \\ \Pi_{m, \text{ár}}(BT \bowtie PC) \cup \Pi_{m, \text{ár}}(BT \bowtie Laptop) \cup \\ \cup \Pi_{m, \text{ár}}(BT \bowtie Ny) \end{aligned}$$

c.) SQL-ben kifejezve

```
select modell, ar from pc
```

```
where modell in
```

```
(select modell from termek  
where gyarto='B')
```

union

```
select modell, ar from laptop
```

```
where modell in
```

```
(select modell from termek  
where gyarto='B')
```

union

```
select modell, ar from nyomtato
```

```
where modell in
```

```
(select modell from termek  
where gyarto='B');
```

c.) --- mint az előző, egyszerűbben,
„with” még nem kell, visszatérünk

with

Btermek as

(select modell from termek where gyarto='B')

select modell, ar from pc natural join Btermek
union

select modell, ar from laptop natural join Btermek
union

select modell, ar from nyomtato natural join Btermek;

Példák relációs algebrai lekérdezésekre ---3

d.) Adjuk meg valamennyi színes lézernyomtató modellszámát

Példák relációs algebrai lekérdezésekre ---3

d.) Adjuk meg valamennyi színes lézernyomtató

modellszámát: $\Pi_m(\sigma_{sz='i'}(Ny)) \cap \Pi_m(\sigma_{t='lézer'}(Ny))$

-- elvégezhető más módon is: $\Pi_m(\sigma_{sz='i' \wedge t='lézer'}(Ny)) =$
 $= \Pi_m(\sigma_{sz='i'} \sigma_{t='lézer'}(Ny)) = \Pi_m(\sigma_{t='lézer'} \sigma_{sz='i'}(Ny))$

e) Melyek azok a gyártók, amelyek laptopot árulnak,

PC-t viszont nem? (ha laptop gyártó több pc-t gyárt, akkor
az eredménytábla csökken, **nem monoton** művelet: **R - S**)

Példák relációs algebrai lekérdezésekre ---3

d.) Adjuk meg valamennyi színes lézernyomtató

modellszámát: $\Pi_m(\sigma_{sz='i'} (Ny)) \cap \Pi_m(\sigma_{t='lézer'} (Ny))$

-- elvégezhető más módon is: $\Pi_m(\sigma_{sz='i' \wedge t='lézer'} (Ny)) =$
 $= \Pi_m(\sigma_{sz='i'} \sigma_{t='lézer'} (Ny)) = \Pi_m(\sigma_{t='lézer'} \sigma_{sz='i'} (Ny))$

e) Melyek azok a gyártók, amelyek laptopot árulnak,

PC-t viszont nem? (ha laptop gyártó több pc-t gyárt, akkor
az eredménytábla csökken, **nem monoton** művelet: **R - S**)

$\Pi_{gy}(T \bowtie L) - \Pi_{gy}(T \bowtie PC)$

! f) Melyek azok a merevlemez méretek, amelyek legalább
két PC-ben megtalálhatók? (**táblát önmagával szorozzuk**)

Példák relációs algebrai lekérdezésekre ---3

d.) Adjuk meg valamennyi színes lézernyomtató

modellszámát: $\Pi_m(\sigma_{sz='i'} (Ny)) \cap \Pi_m(\sigma_{t='lézer'} (Ny))$

-- elvégezhető más módon is: $\Pi_m(\sigma_{sz='i' \wedge t='lézer'} (Ny)) =$
 $= \Pi_m(\sigma_{sz='i'} \sigma_{t='lézer'} (Ny)) = \Pi_m(\sigma_{t='lézer'} \sigma_{sz='i'} (Ny))$

e) Melyek azok a gyártók, amelyek laptopot árulnak,

PC-t viszont nem? (ha laptop gyártó több pc-t gyárt, akkor
az eredménytábla csökken, **nem monoton** művelet: **R - S**)

$\Pi_{gy}(T \bowtie L) - \Pi_{gy}(T \bowtie PC)$

! f) Melyek azok a merevlemez méretek, amelyek legalább
két PC-ben megtalálhatók? (**táblát önmagával szorozzuk**)

Példák relációs algebrai lekérdezésekre ---3

d.) Adjuk meg valamennyi színes lézernyomtató

modellszámát: $\Pi_m(\sigma_{sz='i'} (Ny)) \cap \Pi_m(\sigma_{t='lézer'} (Ny))$

-- elvégezhető más módon is: $\Pi_m(\sigma_{sz='i' \wedge t='lézer'} (Ny)) =$
 $= \Pi_m(\sigma_{sz='i'} \sigma_{t='lézer'} (Ny)) = \Pi_m(\sigma_{t='lézer'} \sigma_{sz='i'} (Ny))$

e) Melyek azok a gyártók, amelyek laptopot árulnak,

PC-t viszont nem? (ha laptop gyártó több pc-t gyárt, akkor
az eredménytábla csökken, **nem monoton** művelet: **R - S**)

$\Pi_{gy}(T \bowtie L) - \Pi_{gy}(T \bowtie PC)$

f) Melyek azok a merevlemez méretek, amelyek legalább

két PC-ben megtalálhatók? (**táblát önmagával szorozzuk**)

-- segédváltozót vezetek be, legyen **PC₁ := PC**

$\Pi_{PC.ml}(\sigma_{PC_1.m \neq PC.m \wedge PC_1.ml=PC.ml} (PC_1 \times PC))$

Példák relációs algebrai lekérdezésekre ---4

! g) Adjuk meg azokat a PC-modell párokat, amelyek ugyanolyan gyorsak és a memóriájuk is ugyanakkora. Egy pár csak egyszer jelenjen meg, azaz ha már szerepel az (i, j) , akkor a (j, i) ne jelenjen meg.

Példák relációs algebrai lekérdezésekre ---4

! g) Adjuk meg azokat a PC-modell párokat, amelyek ugyanolyan gyorsak és a memóriájuk is ugyanakkora. Egy pár csak egyszer jelenjen meg, azaz ha már szerepel az (i, j) , akkor a (j, i) ne jelenjen meg.

$$\Pi_{\text{PC}_1.m, \text{PC}.m} (\sigma_{\text{PC}_1.m < \text{PC}.m \wedge \text{PC}_1.s = \text{PC}.s \wedge \text{PC}_1.me = \text{PC}.me} (\text{PC}_1 \times \text{PC}))$$

!! h) Melyek azok a gyártók, amelyek gyártanak legalább két, egymástól különböző, legalább 2.80 gigahertzen működő számítógépet (PC-t vagy laptopot)

Példák relációs algebrai lekérdezésekre ---4

! g) Adjuk meg azokat a PC-modell párokat, amelyek ugyanolyan gyorsak és a memóriájuk is ugyanakkora. Egy pár csak egyszer jelenjen meg, azaz ha már szerepel az (i, j) , akkor a (j, i) ne jelenjen meg.

$$\Pi_{\text{PC}_1.m, \text{PC}.m} (\sigma_{\text{PC}_1.m < \text{PC}.m \wedge \text{PC}_1.s = \text{PC}.s \wedge \text{PC}_1.me = \text{PC}.me} (\text{PC}_1 \times \text{PC}))$$

!! h) Melyek azok a gyártók, amelyek gyártanak legalább két, egymástól különböző, legalább 2.80 gigahertzen működő számítógépet (PC-t vagy laptopot)

-- segédváltozó: **Gyors** := $\Pi_m (\sigma_{s \geq 2.8} (\text{PC})) \cup \Pi_m (\sigma_{s \geq 2.8} (\text{L}))$

-- és ezzel legyen: $T_1 := T \bowtie \text{Gyors}$ és $T_2 := T \bowtie \text{Gyors}$

$$\Pi_{T_1.gy} (\sigma_{T_1.gy = T_2.gy \wedge T_1.m \neq T_2.m} (T_1 \times T_2))$$

Példák relációs algebrai lekérdezésekre ---5

!! i) Melyik gyártó gyártja a leggyorsabb PC-t?

(„elhagyás” típusú lekérdezések, nincs nála gyorsabb PC)

Példák relációs algebrai lekérdezésekre ---5

!! i) Melyik gyártó gyártja a leggyorsabb PC-t?

(az „elhagyás” típusú lekérdezések, lásd maximum kif.)

Kiválasztjuk azokat a PC-ket, amelyiknél van gyorsabb, ha ezt kivonjuk a PC-ékből megkapjuk a leggyorsabbat:

EnnélVanNagyobb = $\prod_{PC.m} (\sigma_{PC.s < PC_1.s} (PC \times PC1))$

Leggyorsabb: $\prod_m (PC) - EnnélVanNagyobb$

-- Ehhez rajzoljuk fel a kiértékelő fát is:

Példák relációs algebrai lekérdezésekre ---5

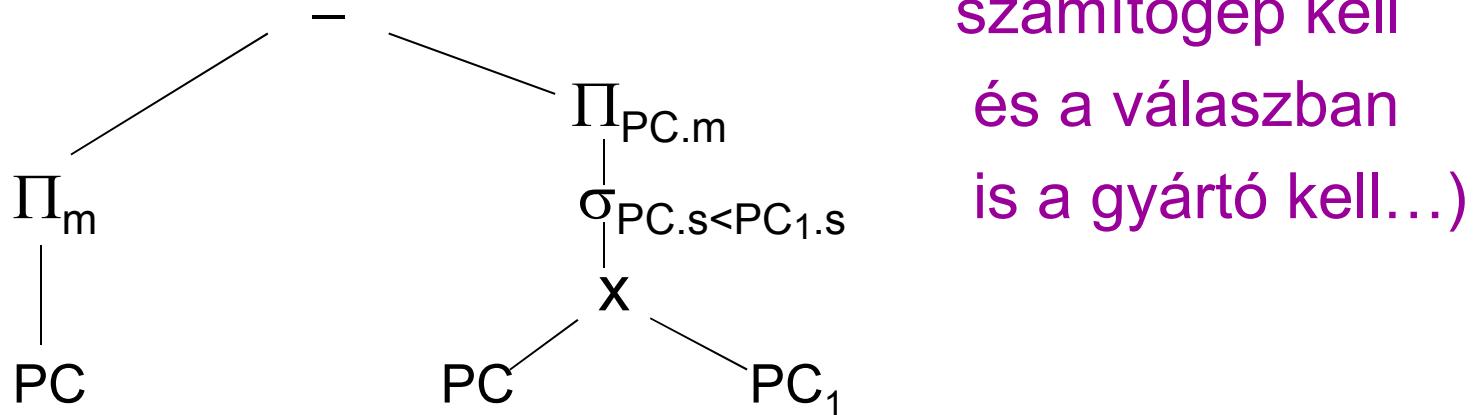
!! i) Melyik gyártó gyártja a leggyorsabb számítógépet
(PC-t vagy laptopot)? Lásd még az „elhagyás” típusú
lekérdezéseket (köv.oldalon pl. maximum kifejezése)

Kiválasztjuk azokat a PC-ket, amelyknél van gyorsabb,
ha ezt kivonjuk a PC-ékből megkapjuk a leggyorsabbat:

EnnélVanNagyobb = $\Pi_{PC.m}(\sigma_{PC.s < PC_1.s}(PC \times PC_1))$

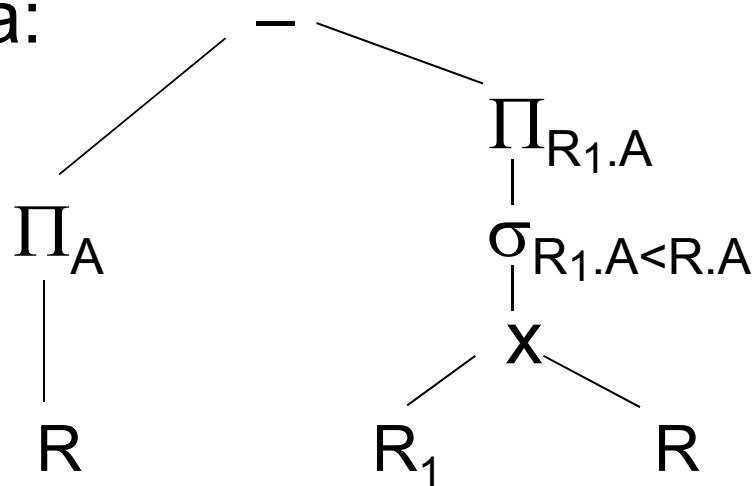
Leggyorsabb: $\Pi_m(PC) - EnnélVanNagyobb$

Ehhez rajzoljuk fel a kiértékelő fát is: (folyt.: PC helyett
számítógép kell
és a válaszban
is a gyártó kell...)



MAX előállítása relációs algebrában

- Nézzük meg a maximum előállításának a kérdését! Legyen $R(A,B)$. **Feladat:** Adjuk meg $\text{MAX}(A)$ értékét! (Ez majd átvezet az új témára, aggregáló függvényekre, illetve csoportosításra).
 - $\pi_A(R) - \pi_{R1.A}(\sigma_{R1.A < R.A}(\rho_{R1}(R) \times R))$
 - Kiértékelő fa:



Folyt. Rel.alg. kifejezés átírása SQL-re

- Előző oldal folyt.max előállítás átírása SQL-re:
- Kiértékelő fa szerinti átírás SQL-be:

```
(SELECT A FROM R)
    EXCEPT
(SELECT R1.A AS A
    FROM R R1, R R2
    WHERE R1.A < R2.A);
```

- Nézzük meg korrelált (függő) alkérdéssel is:

```
SELECT A FROM R MAXA
    WHERE NOT EXISTS
        (SELECT A FROM R
            WHERE A > MAXA.A);
```

Példák relációs algebrai lekérdezésekre ---6

!! j) Melyik gyártó gyárt legalább három, különböző sebességű PC-t? mint a legalább kettő, csak ott $2x$, itt $3x$ kell a táblát önmagával szorozni. Legyenek $S, S_1, S_2 := T \bowtie \Pi_{m,s}(PC)$

$\Pi_{S.gy}(\sigma_{S_1.gy=S.gy} \wedge S_2.gy=S.gy \wedge S_1.s \neq S.s \wedge S_2.s \neq S.s \wedge S_1.s \neq S_2.s) (S \times S_1 \times S_2)$

!! k) Melyek azok a gyártók, amelyek pontosan három típusú PC-t forgalmaznak? legalább 3-ból - legalább 4-t kivonni

-
- Mire érdemes felhívni a figyelmet?
Mi a leggyakrabban előforduló típus, amiből építkeztek?

$\Pi_{\text{lista}}(\sigma_{\text{feltételek}}(\text{táblák szorzata}))$

Ezt a komponenst támogatja legerősebben majd az SQL:
SELECT s-lista FROM f-lista WHERE feltétel;

Kérdés/Válasz

- Köszönöm a figyelmet! Kérdés/Válasz?
- Először relációs algebrában táblákkal gondolkodva felírjuk kifejezőfákkal, majd átírva SQL lekérdezésekre többféle megoldási lehetőséget vizsgálunk meg, vessünk össze!
- Tk.2.4.14. (54-57.o.) **2.4.1.feladata Termékek feladatai**
http://people.inf.elte.hu/sila/AB1ea/Feladatok_Termek.pdf
create table: http://people.inf.elte.hu/sila/eduAB/create_termekek.txt
- Tk.2.4.14. (57-60.o.) **2.4.3.feladata Csatahajók feladatai**
http://people.inf.elte.hu/sila/AB1ea/Feladatok_Csatahajok.pdf
create table: http://people.inf.elte.hu/sila/eduAB/create_csatahajok.txt

Kiterjesztett relációs algebra

SQL: csoportosítás és összesítések

Tankönyv: Ullman-Widom:

Adatbázisrendszerek Alapvetés

Második, átdolgozott kiadás,

Panem, 2009

[Oracle gyak.] gépes lekérdezések:

SQL függvények használata

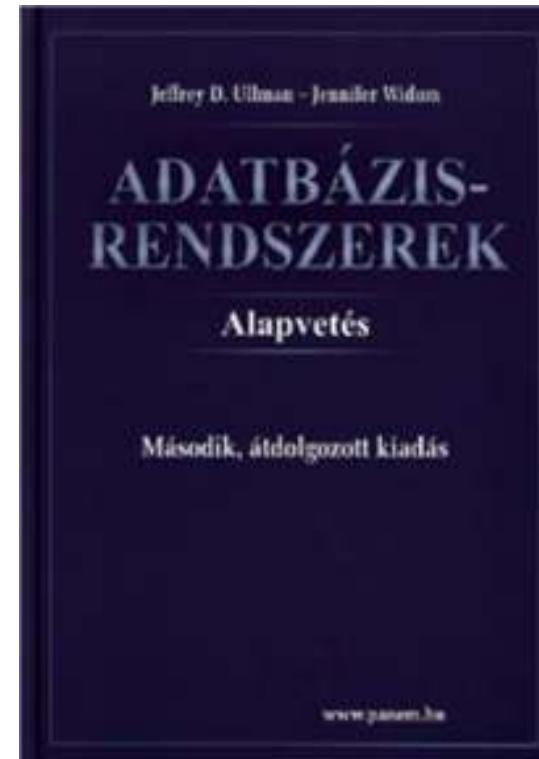
5.1.- 5.2. Kiterjesztett műveletek

6.4.1. Ismétlődések kezelése

6.4.2. Halmazműveletek SQL-ben

6.4.3.-6.4.8. Csoportosítás és összesítések az SQL-ben

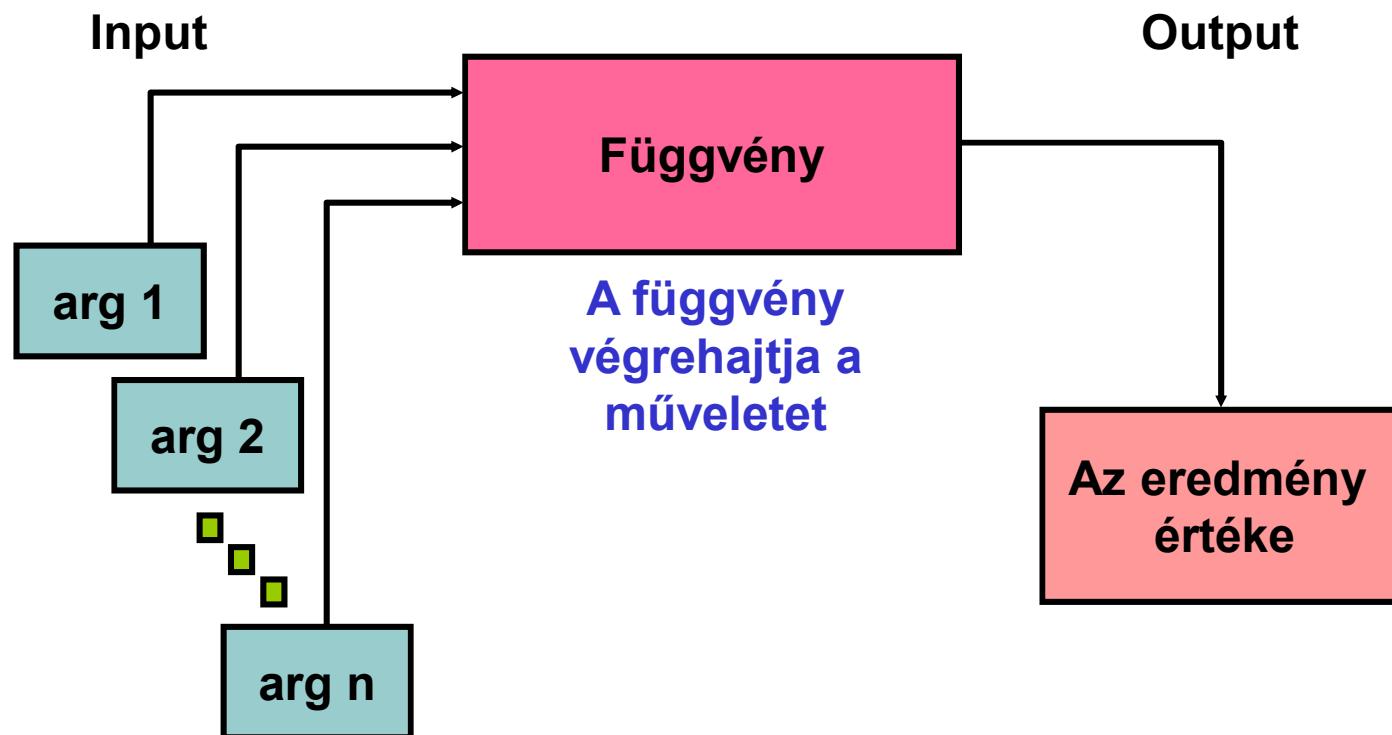
[Oracle gyak.] 6.3.6.-6.3.9. Összekapcsolások az SQL-ben



Egytáblás lekérdezések az SQL-ben

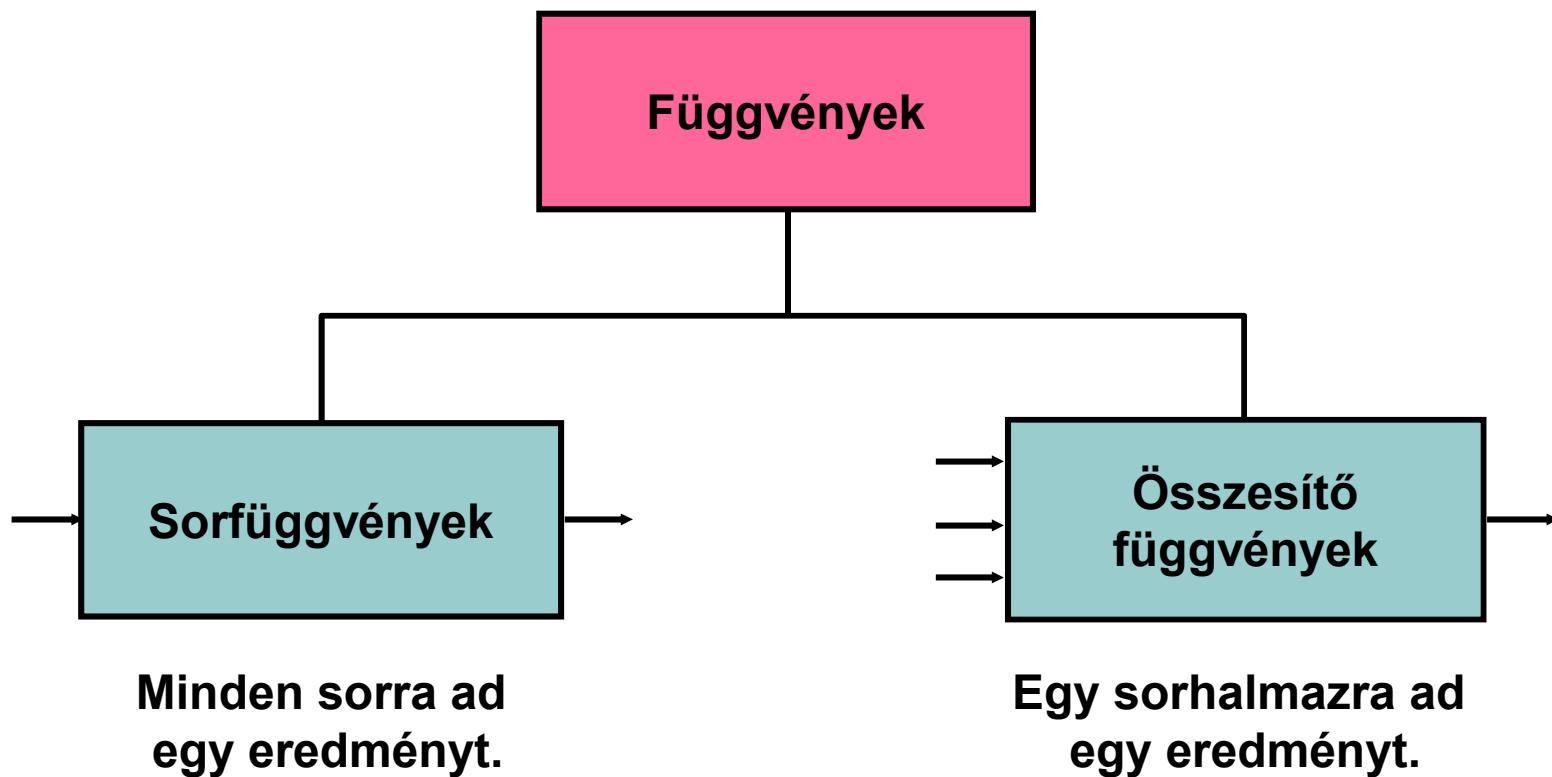
- Az SQL-ben halmazok helyett **multihalmazokat** használunk (egy sor többször is előfordulhat)
- $\Pi_{\text{Lista}} \sigma_{\text{Feltétel}}(R)$ kifejezésben a vetítés L-listája és a kiválasztás F-feltétele az attribútumnevek helyén **kifejezések** állnak, amely függvényeket és műveleti jeleket is tartalmazhat
- SQL leggyakrabban használt sorfüggvények:
- Numerikus, karakteres, dátum, konverziós, általános, például NULL értéket megadott értékkel helyettesítő NVL, COALESCE sorfüggvényeket lásd a gyakorlaton néztük meg Oracle SQL sorfüggvényekre a 3.leckét
- 2.gyak: Oracle Példatár 1.fejezet 1.1-1.20 feladatai
- 3.gyak: Oracle Példatár 2.fejezet 2.1-2.24 feladatai

Oracle GYAK: SQL-függvények

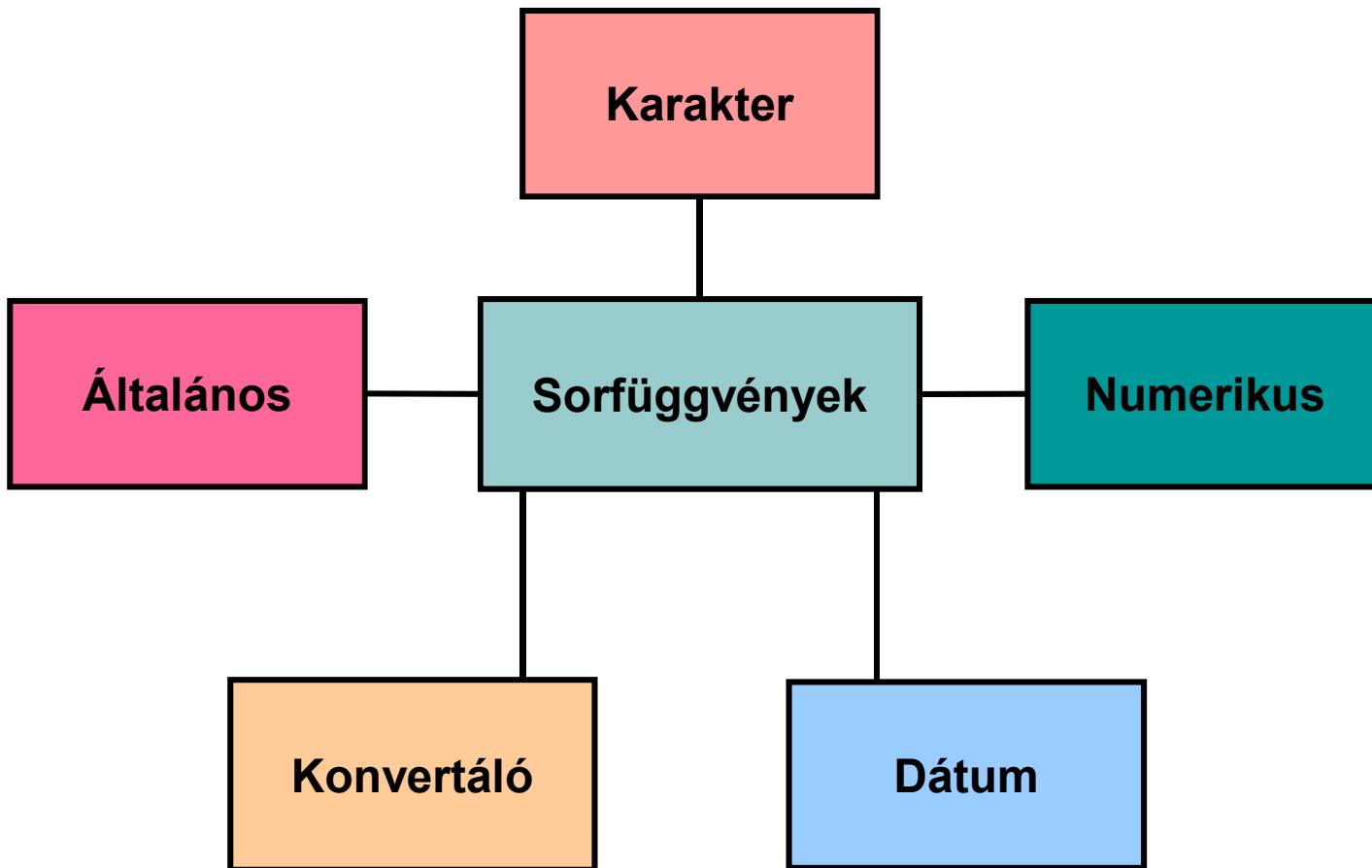


A gyakorlaton bemutatott függvények többsége Oracle-specifikus.

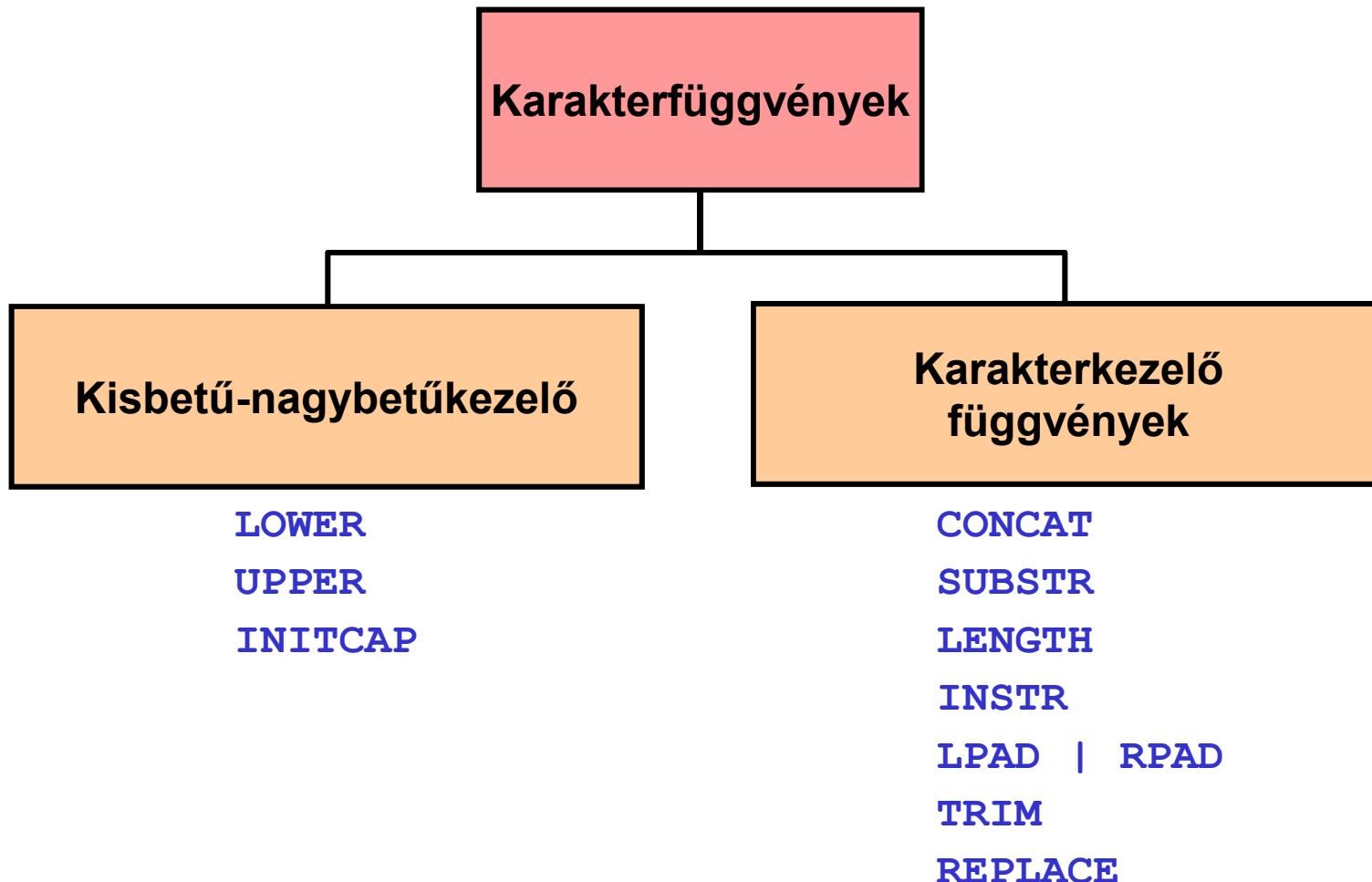
Az SQL-függvények két típusa



Sorfüggvények



Karakterfüggvények



Dátumfüggvények

Függvény	Eredmény
<code>MONTHS_BETWEEN(date1, date2)</code>	A dátumok közti hónapok száma
<code>ADD_MONTHS(date, n)</code>	n hónappal növeli a dátumot
<code>NEXT_DAY(date, 'char')</code>	A következő adott nevű nap dátuma.
<code>LAST_DAY(date)</code>	A dátum hónapjának utolsó napja.
<code>ROUND(date[, 'fmt'])</code>	A dátum kerekítése
<code>TRUNC(date[, 'fmt'])</code>	A dátum levágása

A dátumfüggvények

Függvény	Eredmény
MONTHS_BETWEEN ('01-SEP-95', '11-JAN-94')	19.6774194
ADD_MONTHS ('11-JAN-94', 6)	'11-JUL-94'
NEXT_DAY ('01-SEP-95', 'FRIDAY')	'08-SEP-95'
LAST_DAY ('01-FEB-95')	'28-FEB-95'

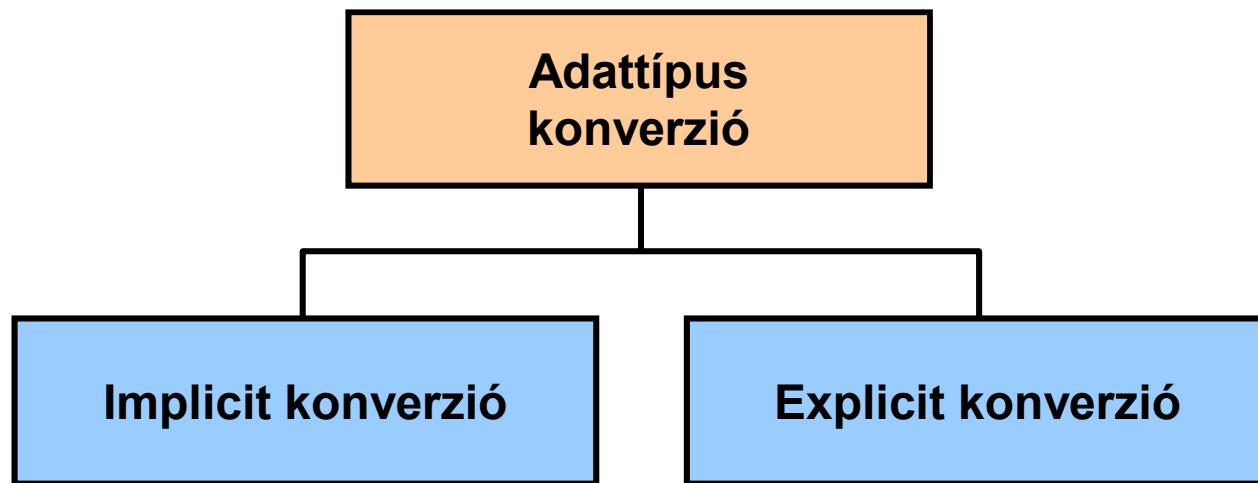
```

SELECT employee_id, hire_date,
       MONTHS_BETWEEN (SYSDATE, hire_date) TENURE,
       ADD_MONTHS (hire_date, 6) REVIEW,
       NEXT_DAY (hire_date, 'FRIDAY'),
       LAST_DAY(hire_date)
  FROM employees
 WHERE MONTHS_BETWEEN (SYSDATE, hire_date) < 70;

```

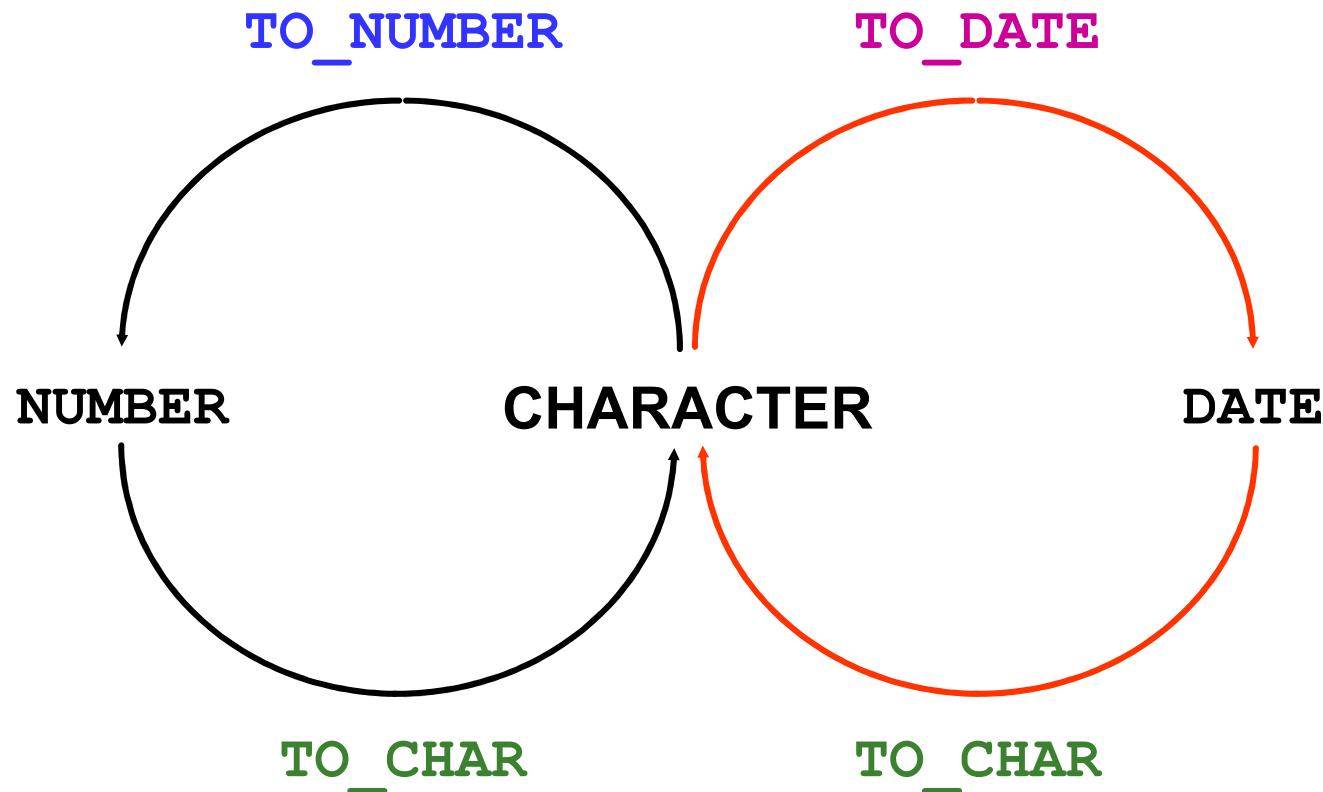
EMPLOYEE_ID	HIRE_DATE	TENURE	REVIEW	NEXT_DAY(LAST_DAY(
107	07-FEB-99	31.6982407	07-AUG-99	12-FEB-99	28-FEB-99
124	16-NOV-99	22.4079182	16-MAY-00	19-NOV-99	30-NOV-99
149	29-JAN-00	19.9885633	29-JUL-00	04-FEB-00	31-JAN-00
178	24-MAY-99	28.1498536	24-NOV-99	28-MAY-99	31-MAY-99

Konvertáló függvények



A hasonló adattípusok konverzióját az Oracle szerverre is bíthatjuk (**implicit**), de ajánlott inkább konvertáló függvényeket használni (**explicit**).

Explicit adattípus-konverzió

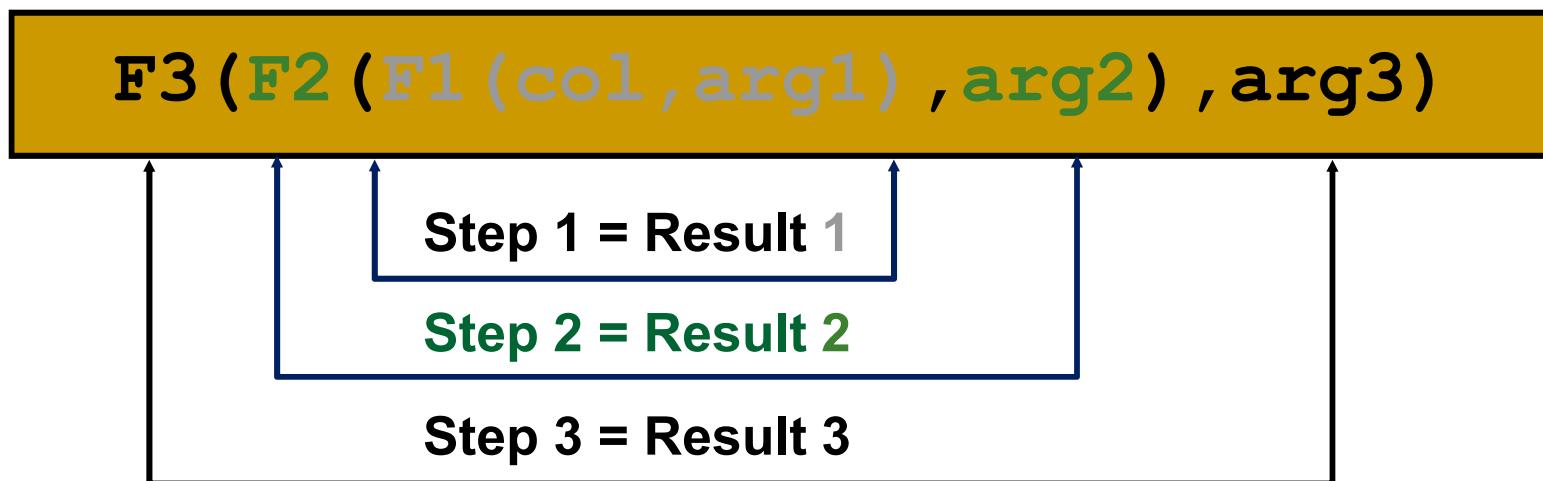


Explicit adattípus-konverzió

Függvény	Leírás
TO_CHAR(number date,[fmt], [nlsparams])	A VARCHAR2 karakter formátumát az <i>fmt</i> modellel lehet megadni. Az <i>nlsparams</i> paraméter mondja meg, hogy milyen tízesvesszöt, ezres csoportosítót, pénznemeket használunk.
TO_CHAR(number date,[fmt], [nlsparams])	Dátumkonverzió esetén az <i>nlsparams</i> paraméter mondja meg, hogy milyen nyelven adtuk meg a napok, hónapok nevét, vagy miként rövidítettük a neveket.
TO_NUMBER(char,[fmt], [nlsparams])	Az <i>fmt</i> és <i>nlsparams</i> opcionális paraméterek értelme a fentiek szerint.
TO_DATE(char,[fmt],[nlsparams])	Az <i>fmt</i> és <i>nlsparams</i> opcionális paraméterek értelme a fentiek szerint.

Függvények egymásba ágyazása

- › A sorfüggvények tetszőleges mélységiig egymásba ágyazhatók.
- › A kiértékelés belülről kifele történik.



Az NVL függvény

A nullértéket a megadott értékkel helyettesíti:

- Az adattípus lehet dátum, karakter, szám.
- Az argumentumok adattípusának egyezőknek kell lenniük:

NVL(fizetés, 0)

NVL(dátum, to_date('1997.01.07', 'YYYY.MM.DD'))

NVL(foglalkozás, 'Tanár')

A COALESCE függvény használata

- A COALESCE függvény esetében - az NVL függvénnnyel szemben - több helyettesítő értéket is megadhatunk.
- Ha az első kifejezés nem nullértéket ad vissza, akkor ez a függvény értéke, különben a COALESCE függvényt alkalmazza a maradék kifejezésekre.

Példa : COALESCE függvény

```
SELECT product_id, list_price, min_price,  
       COALESCE(0.9*list_price, min_price, 5) "Sale"  
FROM oe.product_information  
WHERE supplier_id = 102050  
ORDER BY product_id;
```

PRODUCT ID	LIST PRICE	MIN PRICE	Sale
1769	48		43.2
1770		73	73
2378	305	247	274.5
2382	850	731	765
3355			5

Oracle GYAK: Összesítő függvények

- az összesítő függvény csoportosított sorok halmazain működik, és egyetlen eredményt ad vissza csoportonként.

EMPLOYEES

DEPARTMENT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
50	5800
50	3500
50	3100
50	2600
50	2500
80	10500
80	11000
80	8600
	7000
10	4400

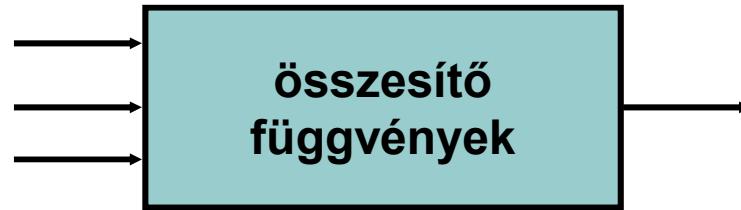
20 rows selected.

A legmagasabb fizetés az EMPLOYEES táblában

MAX(SALARY)
24000

Az aggregáló függvények típusai

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE



Az aggregáló függvények típusai

Függvény	Leírása
<code>AVG ([DISTINCT <u>ALL</u>] <i>n</i>)</code>	<i>n</i> átlagértéke (nullértékeket kihagyva)
<code>COUNT ({ * [DISTINCT <u>ALL</u>] <i>expr</i> })</code>	Azon sorok száma, amelyekre <i>expr</i> kiértékelése nem null (DE: * esetén az összes sorok száma, beleértve az ismétlődőket és a null értéket tartalmazókat is)
<code>MAX ([DISTINCT <u>ALL</u>] <i>expr</i>)</code>	<i>Expr</i> legnagyobb értéke (nullértékeket kihagyva)
<code>MIN ([DISTINCT <u>ALL</u>] <i>expr</i>)</code>	<i>Expr</i> legkisebb értéke (nullértékeket kihagyva)
<code>STDDEV ([DISTINCT <u>ALL</u>] <i>x</i>)</code>	<i>n</i> szórása (nullértékeket kihagyva)
<code>SUM ([DISTINCT <u>ALL</u>] <i>n</i>)</code>	<i>n</i> értékeinek összege (nullértékeket kihagyva)
<code>VARIANCE ([DISTINCT <u>ALL</u>] <i>x</i>)</code>	<i>n</i> szórásnégyzete (nullértékeket kihagyva)

Az aggregáló függvények használata

```
SELECT      [column,] oszlop_függvény(column) , . .
FROM        table
[WHERE      condition]
[GROUP BY  column]
[ORDER BY   column];
```

Az oszlopfüggvények a nullértéket tartalmazó sorokat kihagyják, a nullérték helyettesítésére használhatók az NVL, NVL2 és COALESCE függvények.

AVG és SUM összesítő függvény

- Az AVG és SUM csak numerikus adatokra használható (a VARIANCE és STDDEV szintén).

```
SELECT AVG(salary) , MAX(salary) ,  
       MIN(salary) , SUM(salary)  
FROM   employees  
WHERE  job_id LIKE '%REP%';
```

Avg(Salary)	Max(Salary)	Min(Salary)	Sum(Salary)
8150	11000	6000	32600

MIN és MAX összesítő függvény

- A MIN és MAX numerikus, karakteres és dátum típusú adatokra használható (LOB és LONG típusokra nem).

```
SELECT MIN(hire_date), MAX(hire_date)  
FROM employees;
```

MIN(HIRE)	MAX(HIRE)
17-JUN-87	29-JAN-00

COUNT összesítő függvény

- COUNT(*) visszaadja a sorok számát a táblában:

1

```
SELECT COUNT (*)
FROM employees
WHERE department_id = 50;
```

COUNT()

5

- COUNT(expr) azoknak a soroknak a számát adja vissza, amelyekben expr nem nullérték:

2

```
SELECT COUNT(commission_pct)
FROM employees
WHERE department_id = 80;
```

COUNT(COMMISSION_PCT)

3

A DISTINCT kulcsszó használata

- COUNT(DISTINCT expr) azoknak a soroknak a számát adja vissza, amelyekben expr értéke különböző és nem nullérték
- Pl. a különböző (nem null) osztályazonosítók száma az EMPLOYEES táblában:

```
SELECT COUNT(DISTINCT department_id)  
FROM employees;
```

COUNT(DISTINCTDEPARTMENT_ID)
7

Összesítő függvények és a nullértékek

- Az összesítő függvények általában ignorálják a nullértéket tartalmazó sorokat:

1

```
SELECT AVG(commission_pct)  
FROM employees;
```

Avg(COMMISSION_PCT)

.2125

- Az NVL függvényel kikényszeríthető a nullértéket tartalmazó sorok figyelembe vétele:

2

```
SELECT AVG(NVL(commission_pct, 0))  
FROM employees;
```

Avg(NVL(COMMISSION_PCT,0))

.0425

Adatcsoportok létrehozása

EMPLOYEES

DEPARTMENT_ID	SALARY
10	4400
20	13000
20	6000
50	5800
50	3500
50	3100
50	2500
50	2600
60	9000
60	6000
60	4200
80	10500
80	8600
80	11000
90	24000
90	17000

...

20 rows selected.

4400
9500
3500
6400
10033

Az
EMPLOYEES
tábla
osztályai
és azokon az
átlagfizetések

DEPARTMENT_ID	Avg(Salary)
10	4400
20	9500
50	3500
60	6400
80	10033.3333
90	19333.3333
110	10150
	7000

Adatcsoportok létrehozása: a GROUP BY rész szintaxisa

```
SELECT      column, group_function(column)
FROM        table
[WHERE      condition]
[GROUP BY  group_by_expression]
[ORDER BY  column];
```

- Egy tábla sorai csoportosíthatóak a GROUP BY rész használatával.
- A GROUP BY részben oszlop másodnevek nem szerepelhetnek.

A GROUP BY rész használata

- A SELECT lista minden olyan oszlopnevének, amely nem összesítő függvényekben fordul elő, szerepelnie kell a GROUP BY részben.

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id ;
```

DEPARTMENT_ID	AVG(SALARY)
10	4400
20	9500
50	3500
60	6400
80	10033.3333
90	19333.3333
110	10150
	7000

8 rows selected.

A GROUP BY rész használata

- A GROUP BY oszlopneveknek nem kötelező szerepelni a SELECT listában.

```
SELECT      AVG(salary)
FROM        employees
GROUP  BY department_id ;
```

Avg(Salary)
4400
9500
3500
6400
10033.3333
19333.3333
10150
7000

összesítő függvény szerepelhet az ORDER BY részben is.

Csoportosítás több oszlopnév alapján

EMPLOYEES

DEPARTMENT_ID	JOB_ID	SALARY
90	AD_PRES	24000
90	AD_VP	17000
90	AD_VP	17000
60	IT_PROG	9000
60	IT_PROG	6000
60	IT_PROG	4200
50	ST_MAN	5800
50	ST_CLERK	3500
50	ST_CLERK	3100
50	ST_CLERK	2600
50	ST_CLERK	2500
80	SA_MAN	10500
80	SA_REP	11000
80	SA_REP	8600
...		
20	MK_REP	6000
110	AC_MGR	12000
110	AC_ACCOUNT	8300

20 rows selected.

Az
EMPLOYEES
tábla
osztályain
az egyes
beosztások
átlagfizetései

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	11700
50	ST_MAN	5800
60	IT_PROG	19200
80	SA_MAN	10500
80	SA_REP	19600
90	AD_PRES	24000
90	AD_VP	34000
110	AC_ACCOUNT	8300
110	AC_MGR	12000
	SA_REP	7000

13 rows selected.

A GROUP BY rész használata több oszlopnév esetén

```
SELECT      department_id dept_id, job_id, SUM(salary)
FROM        employees
GROUP BY    department id, job id ;
```

DEPT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	11700
50	ST_MAN	5800
60	IT_PROG	19200
80	SA_MAN	10500
80	SA_REP	19600
90	AD_PRES	24000
90	AD_VP	34000
110	AC_ACCOUNT	8300
110	AC_MGR	12000
	SA_REP	7000

13 rows selected.

Csoportok korlátzása

EMPLOYEES

DEPARTMENT_ID	SALARY
90	24000
	17000
	17000
60	9000
	6000
	4200
50	5800
	3500
	3100
	2600
	2500
80	10500
	11000
	8600
...	
20	6000
110	12000
110	8300

20 rows selected.

A legmagasabb fizetés osztályonként, ha az nagyobb mint \$10,000

DEPARTMENT_ID	MAX(SALARY)
20	13000
80	11000
90	24000
110	12000

HAVING

- A HAVING rész használata esetén az Oracle szerver az alábbiak szerint korlátozza a csoportokat:
 1. Csoportosítja a sorokat.
 2. Alkalmazza Az összesítő függvényeket a csoportokra.
 3. A HAVING résznek megfelelő csoportokat megjeleníti.

```
SELECT      column, group_function
FROM        table
[WHERE      condition]
[GROUP BY  group_by_expression]
[HAVING    group_condition]
[ORDER BY  column];
```

A HAVING rész használata

```
SELECT      department_id, MAX(salary)
FROM        employees
GROUP BY    department_id
HAVING      MAX(salary)>10000 ;
```

DEPARTMENT_ID	MAX(SALARY)
20	13000
80	11000
90	24000
110	12000

A HAVING rész használata

```
SELECT      job_id, SUM(salary) PAYROLL
FROM        employees
WHERE       job_id NOT LIKE '%REP%'
GROUP BY    job_id
HAVING     SUM(salary) > 13000
ORDER BY    SUM(salary);
```

JOB_ID	PAYROLL
IT_PROG	19200
AD_PRES	24000
AD_VP	34000

Összesítő függvények egymásba ágyazása

- Az osztályonkénti legmagasabb átlagfizetés megjelenítése:

```
SELECT MAX(AVG(salary))  
FROM employees  
GROUP BY department_id;
```

MAX(AVG(SALARY))
19333.3333

Az összesítő függvények csak kétszeres mélységgig ágyazhatóak egymásba!

Oracle GYAK: Összefoglalás

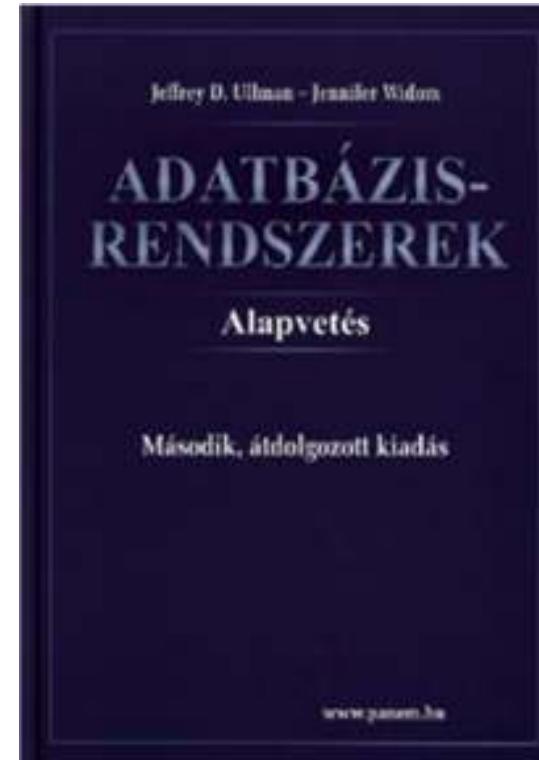
- Ebben a részben megtanultuk:
 - a COUNT, MAX, MIN, SUM és AVG összesítő függvények használatát,
 - hogyan írunk GROUP BY részt tartalmazó lekérdezéseket,
 - hogyan írunk HAVING részt tartalmazó lekérdezéseket.

```
SELECT      column, group_function
FROM        table
[WHERE      condition]
[GROUP BY  group_by_expression]
[HAVING    group_condition]
[ORDER BY  column];
```

Relációs algebra kiterjesztése SQL: csoportosítás, összesítések, külső összekapcsolások

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009

- 5.1. Relációs műveletek multihalmazokon
- 5.2. Kiterjesztett műveletek a relációs algebrában



Új anyag: Relációs algebra kibővítése

- Az eddig tanult műveleteket: **vetítés** (Π), **kiválasztás** (σ), **halmazműveletek**: **unió** (\cup), **különbség** ($-$), **metszet** (\cap), **szorzás**: **természetes összekapcsolás** (\bowtie), **direkt-szorzat** (\times), stb. **multihalmazok fölött** értelmezzük, mint az SQL-ben, egy reláció nem sorok halmazából, hanem **multihalmazából** áll, vagyis megengedett a sorok ismétlődése.
- Ezeken kívül a SELECT kiegészítéseinek és záradékaival megfeleltetett új műveletekkel is kibővíjtük a rel. algebrát:
 - Ismétlődések megszüntetése (δ) - select distinct ..
 - Összesítő műveletek és csoportosítás (γ_{lista}) - group by..
 - Vetítési művelet kiterjesztése (Π_{lista}) - select kif [as onev]..
 - Rendezési művelet (τ_{lista}) - order by..
 - Külső összekapcsolások ($\overset{\circ}{\bowtie}$) - [left | right | full] outer join

Multihalmazok egyesítése, különbsége

- **Unió:** $R \cup S$ -ben egy t sor annyiszor fordul elő ahányszor előfordul R-ben, plusz ahányszor előfordul S-ben: $n+m$
- **Metszet:** $R \cap S$ -ben egy t sor annyiszor fordul elő, amennyi az R-ben és S-ben lévő előfordulások minimuma: $\min[n, m]$
- **Különbség:** $R - S$ -ben egy t sor annyiszor fordul elő, mint az R-beli előfordulások minusz az S-beli előfordulások száma, ha ez pozitív, egyébként pedig 0, vagyis $\max[0, n-m]$
- $(R \cup S) - T =? (R - T) \cup (S - T)$ (Ez hz: igen, multihz:nem)

R		S			
A	B	A	B	A	B
1	3	1	3	1	3
1	2	2	5	2	5

∪ ➔

A „többi művelet” multihalmazok fölött

- A projekció, szelekció, Descartes-szorzat, természetes összekapcsolás és Théta-összekapcsolás végrehajtása során nem küszöböljük ki az ismétlődéseket.

R		$\Pi_A(R)$	
A	B	A	
1	2	1	
1	5	1	
2	3	2	

Új műveletek: Ismétlődések megszüntetése (duplikátumok kiszűrése)

- **Ismétlődések megszüntetése:** $R1 := \delta(R2)$
- A művelet jelentése: $R2$ multihalmazból $R1$ halmazt állít elő, vagyis az $R2$ -ben egyszer vagy többször előforduló sorok csak egyszer szerepelnek az $R1$ -ben.
- A **DISTINCT** reprezentálására szolgál (jele: δ kis-delta)
- A δ speciális esete lesz az általánosabb γ műveletnek

$$R = (\begin{array}{|c|c|} \hline A & B \\ \hline \end{array})$$

A	B
1	2
3	4
1	2

$$\delta(R) =$$

A	B
1	2
3	4

Összesítő (aggregáló) függvények

- Miért hívják **aggregáló** függvényeknek? Ha kiszámoltuk az összeget a tábla bizonyos soraira, akkor újabb sorok figyelembe vételével felhasználhatjuk az eddigi összeget.

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

$$\text{SUM}(A) = 7$$

$$\text{COUNT}(A) = 3$$

$$\text{MIN}(B) = 2$$

$$\text{MAX}(B) = 4$$

$$\text{AVG}(B) = 3$$

Csoportosítás és összesítés $\gamma_L(R)$

- A csoportosítást (GROUP BY), a csoportokon végezhető összesítő függvényeket (AVG, SUM, COUNT, MIN, MAX, stb...) reprezentálja.
- A művelet jele: γ_L gamma
- Itt az L lista valamennyi eleme a következők egyike:
 - R olyan attribútuma, amely szerepel a GROUP BY záradékban, egyike a csoportosító attribútumoknak.
 - R egyik attribútumára (ez az összesítő attribútum) alkalmazott összesítő operátor.
 - Ha az összesítés eredményére névvel szeretnénk hivatkozni, akkor nyilat és új nevet használunk.

Csoportosítás és összesítés $\gamma_L(R)$

- Osszuk R sorait csoportokba. Egy csoport azokat a sorokat tartalmazza, amelyek az L listán szereplő csoportosítási attribútumokhoz tartozó értékei megegyeznek
 - Vagyis ezen attribútumok minden egyes különböző értéke egy csoportot alkot.
- minden egyes csoporthoz számoljuk ki az L lista összesítési attribútumaira vonatkozó összesítéseket
- Az eredmény minden egyes csoportra egy sor:
 - Eredmény: a csoportosítási attribútumok és
 - az összesítési attribútumra vonatkozó összesítések (az adott csoport összes sorára)

Példa: Csoportosításra és összesítésre

$R = (A \quad B \quad C)$

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

$$\gamma_{A,B,\text{AVG}(C) \rightarrow X} (R) = ??$$

Először csoportosítunk

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

majd csoportonként összesítünk:

A	B	X
1	2	4
4	5	6

A vetítési művelet kiterjesztése

- $\Pi_L(R)$ kiterjesztett vetítés L listájában szerepelhetnek:
- Az R reláció attribútuma
- $E \rightarrow z$ kifejezés, ahol E az R reláció attribútumaira vonatkozó (konstansokat, aritmetikai műveleteket, függvényeket tartalmazó kifejezés), z pedig az

R	
A	B
1	2
1	5
2	3

 $\Pi_{A+B \rightarrow z}(R)$ 

Z
3
6
5

E kifejezés által számolt, az eredményekhez tartozó új attribútum nevét jelöli

Kiválasztott sorok rendezése

- Rendezés: $\tau_{A_1, \dots, A_n}(R)$
- Először A_1 attribútum szerint rendezzük R sorait. Majd azokat a sorokat, amelyek értéke megegyezik az A_1 attribútumon, A_2 szerint, és így tovább.
- Az **ORDER BY** reprezentálására szolgál (jele: τ tau)
- Ez az egyetlen olyan művelet, amelynek az eredménye nem halmaz és nem multihalmaz, hanem rendezett lista.

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

$$\tau_B(R) = [(5,2), (1,2), (3,4)]$$

Külső összekapcsolások

- Ez **nem relációs algebrai művelet**, uis kilép a modellből.
- Lehet baloldali, jobboldali, teljes külső összekapcsolás.
- R, S sémái $R(A_1, \dots, A_n, B_1, \dots, B_k)$, ill. $S(B_1, \dots, B_k, C_1, \dots, C_m)$
- $R \bowtie S = R \bowtie S$ relációt kiegészítjük az R és S soraival, a hiányzó helyekre **NULL** értéket írva megőrzi a „lógó sorokat”
- Van teljes, baloldali és jobboldali külső összekapcsolás attól függően, hogy melyik oldalon szereplő reláció sorait adjuk hozzá az eredményhez (a lógó sorokat kiegészítve **NULL** értékkel) \perp szimbólummal.

Példák külső összekapcsolásokra

A	B	C
1	2	3
4	5	6
7	8	9

R reláció

B	C	D
2	3	10
2	3	11
6	7	12

S reláció

A	B	C	D
1	2	3	10
1	2	3	11
4	5	6	⊥
7	8	9	⊥
⊥	6	7	12

$R \circledast S$ eredmény

A	B	C	D
1	2	3	10
1	2	3	11
4	5	6	⊥
7	8	9	⊥

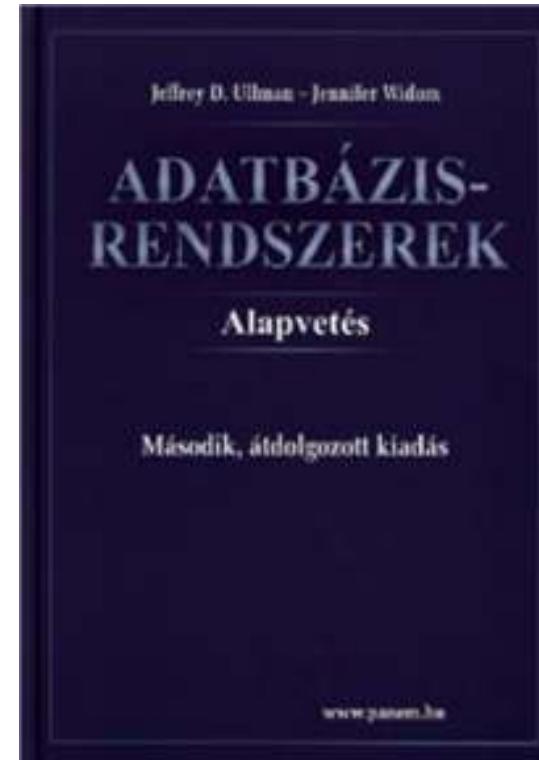
$R_L^o \bowtie S$ eredmény

A	B	C	D
1	2	3	10
1	2	3	11
⊥	6	7	12

$R \overset{o}{\bowtie}_R S$ eredmény

SQL lekérdezések

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009



- Ismétlés: 6.2.5. Halmazműveletek (egyesítés, metszet és különbség)
 - 6.4.1. Ismétlődések megszüntetése
 - 6.4.2. Ismétlődések kezelése a halmazműveletek során
 - 6.4.3.-6.4.7. Csoportosítás és összesítések az SQL-ben
 - group by és having záradékok
 - 6.3.6.-6.3.8. Összekapcsolások az SQL-ben
-

Emlékeztető az előadás példa: Sörivók

- Az előadások SQL lekérdezései az alábbi Sörivók adatbázissémán alapulnak
(aláhúzás jelöli a kulcs attribútumokat)

Sörök(név, gyártó)

Sörözők(név, város, tulaj, engedély)

Sörivók(név, város, tel)

Szeret(név, sör)

Felszolgál(söröző, sör, ár)

Látogat(név, söröző)

Halmazműveletek az SQL-ben

- A relációs algebrai halmazműveletek: **unió**, **különbség** és **metszet**, ebből csak az unió és különbség alapművelet, az SQL-ben mindenkor használható, implementálva van
- A **SELECT-FROM-WHERE** utasítások általában multihalmaz szemantikát használnak, külön kell kérni **DISTINCT**-tel ha halmazt szeretnénk kapni, viszont a **halmazműveleteknél** mégis a **halmaz szemantika** az érvényes, itt a multihalmaz szemantikát kell kérni: **ALL**
- Az SQL-ben a halmazműveleteket úgy vezették be, hogy azt mindig két lekérdezés között lehet értelmezni:

(SFW-lekérdezés1)

[UNION [ALL] |
INTERSECT [ALL] |
{EXCEPT | MINUS} [ALL]]

(SFW-lekérdezés2);

Halmaz-multihalmaz szemantika

- A **SELECT-FROM-WHERE** állítások multihalmaz szemantikát használnak, a **halmazműveleteknél** mégis a **halmaz szemantika** az érvényes.
 - Azaz sorok nem ismétlődnek az eredményben.
- 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.
- **Motiváció:** hatékonyság, minimális költségek

SQL: Ismétlődések megszüntetése

- SELECT **DISTINCT** ... FROM ...
- A δ művelet SQL-beli megfelelője, amellyel az eredményben kiszűrjük a duplikátumokat, vagyis multihalmazból halmazt állítunk elő.

SQL: Ismétlődések kezelése halmazművelet során

- (SELECT ... FROM ...)
 {UNION | INTERSECT | EXCEPT} [**ALL**]
 (SELECT ... FROM ...)
- Alapértelmezésben a halmaz-szemantika
(duplikátumok szűrése)
- Az **ALL** kulcsszóval ezek a műveletek
multihalmaz-szemantika szerint működnek.

Példa: Intersect (metszet)

- Szeret(név, sör), Felszolgál(bár, sör, ár) és Látogat(név, bár) táblák felhasználásával keressük

Trükk: itt ez az alkérdezés valójában az adatbázisban tárolt tábla

azokat a sörvíköt és söröket, amelyekre a sörvíró szereti az adott sört **és** a sörvíró látogat olyan bárt, ahol felszolgálják a sört.

```
(SELECT * FROM Szeret)
INTERSECT
(SELECT név, sör
FROM Felszolgál, Látogat
WHERE Látogat.bár = Felszolgál.bár);
```

(név, sör) párok, ahol a sörvíró látogat olyan bárt, ahol ezt a sört felszolgálják

Példa: ALL (multihalmaz szemantika)

- Látogat(név, söroző) és Szeret(név, sór) táblák felhasználásával kilistázzuk azokat a sörvíköt, akik több sörozőt látogatnak, mint amennyi sórt szeretnek, és annyival többet, mint ahányszor megjelennek majd az eredményben

(SELECT név FROM Látogat)

EXCEPT ALL

(SELECT név FROM Szeret) ;

- Megj.: ORACLE-ben EXCEPT helyett MINUS-t használunk, illetve UNION és UNION ALL lehet

SQL: Az eredmény rendezése

- SQL SELECT utasítás utolsó záradéka: **ORDER BY**
- Az SQL lehetővé teszi, hogy a lekérdezés eredménye bizonyos sorrendben legyen rendezve. Az első attribútum egyenlősége esetén a 2.attribútum szerint rendezve, stb, minden attribútumra lehet növekvő vagy csökkenő sorrend.
- Select-From-Where utasításhoz a következő záradékot adjuk, a WHERE záradék és minden más záradék (mint például GROUP BY és HAVING) után következik:

SELECT ... FROM ... [WHERE ...] [...]

ORDER BY {attribútum [DESC], ...}

- **Példa:** **SELECT * FROM Felszolgál**

ORDER BY ár DESC, sör

SQL: Összesítések (aggregálás)

- SELECT listán:
<Aggregáló művelet>(kifejezés) [[AS] onév], ...
SUM, COUNT, MIN, MAX aggregáló műveleteket,
AVG (bevezették ezt is, mivel gyakran kell AVG) a
SELECT záradékban alkalmazhatjuk egy oszlopra.
- COUNT(*) az eredmény sorainak számát adja meg.
- Itt is fontos a halmaz, multihalmaz megkülönböztetés.
SUM(DISTINCT R.A) csak a különböző értékűeket
veszi figyelembe. NULL értékek használata, SUM
nem veszi figyelembe (implementáció függő,
ellenőrizzük le a COUNT-ra, lásd a gyakorlaton)

Példa: Összesítő függvények

- A **Felszolgál(bár, söör, ár)** tábla segítségével adjuk meg a Bud átlagos árát:

```
SELECT AVG (ár)  
FROM Felszolgál  
WHERE söör = '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** nem számít a SUM, AVG, COUNT, MIN, MAX függvények kiértékelésekor.
- De ha nincs NULL értéktől különböző érték az oszloban, akkor az összesítés eredménye NULL.
- **Kivétel:** COUNT az üres halmazon 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';
```

A Bud sört ismert áron árusító kocsmák száma.

SQL: Csoportosítás

- SELECT ...
 FROM ...
 [WHERE ...]
 [GROUP BY $kif_1, \dots kif_k$]
- Egy SELECT-FROM-WHERE kifejezést **GROUP BY** záradékkal folytathatunk, melyet attribútumok lista követ.
- A SELECT-FROM-WHERE eredménye a megadott attribútumok értékei szerint csoportosítódik, az összesítéseket ekkor minden csoportra külön alkalmazzuk.

Példa: Csoportosítás

- A Felszolgál(bár, söör, ár) tábla segítségével adjuk meg a söörök átlagos árát.

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

söör	AVG(ár)
Bud	2.33
Miller	2.45

A SELECT lista és az összesítések

- Ha összesítés is szerepel a lekérdezésben, a SELECT-ben felsorolt attribútumok
 1. vagy egy összesítő függvény paramétereként szerepelnek,
 2. vagy a GROUP BY attribútumlistájában is megjelennek.

Csoportok szűrése: HAVING záradék

- A GROUP BY záradékot egy HAVING <feltétel> záradék követheti.
- HAVING feltétel az egyes csoportokra vonatkozik, ha egy csoport nem teljesíti a feltételt, nem lesz benne az eredményben.
- csak olyan attribútumok szerepelhetnek, amelyek:
 1. vagy csoportosító attribútumok,
 2. vagy összesített attribútumok.

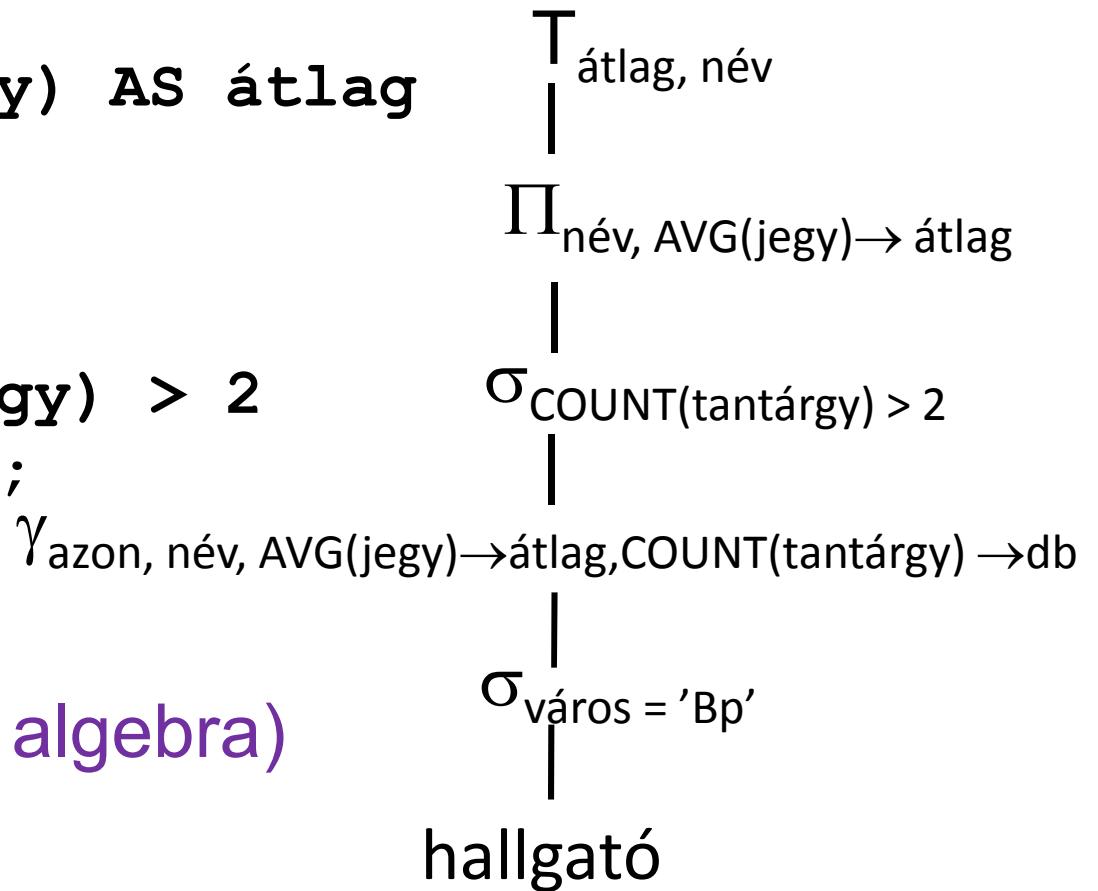
(vagyis ugyanazok a szabályok érvényesek, mint a SELECT záradéknál).

Példa: group by, having és order by

Példa: hallgató (azon, név, város, tantárgy, jegy)

```
SELECT név, AVG(jegy) AS átlag  
FROM hallgató  
WHERE város = 'Bp'  
GROUP BY azon, név  
HAVING COUNT(tantárgy) > 2  
ORDER BY átlag, név;
```

(Kiterjesztett relációs algebra)



Összefoglalás: SELECT utasítás záradékai

- Teljes SELECT utasítás(a záradékok sorrendje adott)

SELECT [DISTINCT] Lista1	-- 5 és 6
FROM R t	-- 1
[WHERE Felt1]	-- 2
[GROUP BY csopkif	-- 3
[HAVING Felt2]]	-- 4
[ORDER BY Lista2]	-- 7

$$T_{Lista2} \ \delta \ \Pi_{Lista1} \ \sigma_{Felt2} (\gamma_{csopkif, \dots, AGGR(kif) \rightarrow onev} \ \sigma_{Felt1} (R))$$

Oracle GYAK: Összekapcsolások

- Az SQL-ben összekapcsolások számos változata megtalálható: **Természetes összekapcsolás**
- **USING** utasítással történő összekapcsolás
- **Teljes** (vagy két oldali) külső összekapcsolás
- **Tetszőleges feltételen** alapuló külső összekapcsolás
- **Direktszorzat** (kereszt-összekapcsolás).

```
SELECT tábla1.oszlop, tábla2.oszlop FROM tábla1  
[NATURAL JOIN tábla2] |  
[JOIN tábla2 USING (oszlopnév)] |  
[JOIN tábla2 ON (tábla1.oszlopnév = tábla2.oszlopnév)]  
[{LEFT | RIGHT | FULL} OUTER JOIN tábla2  
    ON (tábla1.oszlopnév = tábla2.oszlopnév)]  
[CROSS JOIN tábla2]
```

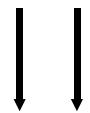
Példa: Többtáblás lekérdezések

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
...		
202	Fay	20
205	Higgins	110
206	Gietz	110

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700



EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
200	10	Administration
201	20	Marketing
202	20	Marketing
...		
102	90	Executive
205	110	Accounting
206	110	Accounting

Természetes összekapcsolás

- A NATURAL JOIN utasítás a benne szereplő két tábla azonos nevű oszlopain alapul.
- A két tábla azon sorait eredményezi, ahol az azonos nevű oszlopokban szereplő értékek megegyeznek.
- Ha az azonos nevű oszlopok adattípusa eltérő, akkor hibával tér vissza az utasítás.

Példa a természetes összekapcsolásra

```
SELECT department_id, department_name,  
       location_id, city  
FROM   departments  
NATURAL JOIN locations ;
```

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
60	IT	1400	Southlake
50	Shipping	1500	South San Francisco
10	Administration	1700	Seattle
90	Executive	1700	Seattle
110	Accounting	1700	Seattle
190	Contracting	1700	Seattle
20	Marketing	1800	Toronto
80	Sales	2500	Oxford

8 rows selected.

- A WHERE használható további megszorítások megfogalmazására. Például, ha csak a 20-as illetve 50-es department_id-kra vagyunk kíváncsiak, akkor:

```
SELECT department_id department_name,  
       location_id city  
FROM   departments  
NATURAL JOIN locations  
WHERE  department_id IN (20, 50);
```

Összekapcsolás USING kulcsszóval

- Ha több oszlopnak azonos a neve, de az adattípusa eltérő, akkor a USING segítségével megadható, hogy mely oszlopokat lehet használni az egyenlőségen alapuló összekapcsoláshoz.
- Használunk USING-ot, ha csak egy oszlop egyezik meg.
- Ne használjuk a tábla eredeti vagy alias nevét a kiválasztott oszlopok megadásánál.
- A NATURAL JOIN és a USING kulcsszavak együttes használata nem megengedett.

Oszlopnevek összekapcsolása

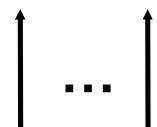
EMPLOYEES

EMPLOYEE_ID	DEPARTMENT_ID
200	10
201	20
202	20
124	50
141	50
142	50
143	50
144	50
103	60
104	60
107	60
149	80
174	80
176	80

...

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
20	Marketing
50	Shipping
60	IT
60	IT
60	IT
80	Sales
80	Sales
80	Sales



Foreign key

Primary key

- Az osztályok dolgozóinak meghatározásához a Departments tábla és az Employees tábla DEPARTMENT_ID oszlopaikban szereplő értékeinek összehasonlítása kell. Ez egy egyenlőségen alapuló összekapcsolás lesz. Az ilyen típusú összekapcsolásban általában az elsődleges- és az idegen kulcs komponensei szerepelnek.

A USING kulcsszó használata lekérdezésben

```
SELECT employees.employee_id, employees.last_name,  
       departments.location_id, department_id  
FROM   employees JOIN departments  
USING (department_id);
```

EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
200	Whalen	1700	10
201	Hartstein	1800	20
202	Fay	1800	20
124	Mourgos	1500	50
141	Rajs	1500	50
142	Davies	1500	50
144	Vargas	1500	50
143	Matos	1500	50
...			

19 rows selected.

Azonos nevű oszlopok megkülönböztetése

- Használjuk a táblaneveket előtagként az azonos nevű oszlopok megkülönböztetésére
- A előtagok használata javítja a hatékonyságot is.
- Használhatunk alias neveket az olyan oszlopokra, amelyeket megkülönböztetünk a többi táblában lévő azonos nevűtársaiktól.
- Ne használunk alias nevet azon oszlopokra, amelyeket a USING kulcsszó után adtunk meg és az SQL utasításban még más helyen is szerepelnek.

Sorváltozók használata tábláknál

- A lekérdezések átláthatósága miatt használhatunk sorváltozót (tábla alias neveket).
- A sorváltozók használata javítja a lekérdezés teljesítményét.
- A sorváltozók maximum 30 karakter hosszúak lehetnek (minél rövidebb annál jobb)
- A sorváltozók csak az aktuális SELECT utasítás során lesznek használhatóak!

Összekapcsolások az ON kulcsszó segítségével

- A természetes összekapcsolás alapvetően az azonos nevű oszlopok egyenlőségvizsgálatán alapuló összekapcsolása volt.
- Az ON kulcsszót használhatjuk az összekapcsolás tetszőleges feltételének vagy oszlopainak megadására.
- Az összekapcsolási feltétel független a többi keresési feltételtől.
- Az ON használata áttekinthetőbbé teszi a kódot

Lekérdezés az ON kulcsszó használatával

```
SELECT e.employee_id, e.last_name, e.department_id,  
       d.department_id, d.location_id  
FROM   employees e JOIN departments d  
ON     (e.department_id = d.department_id);
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500
...				

19 rows selected.

- Az ON segítségével különböző nevű oszlopok is összekapcsolhatóak

Önmagával való összekapcsolás (self-join) az ON kulcsszóval 1.

EMPLOYEES (WORKER)

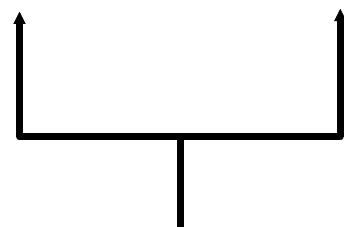
EMPLOYEE_ID	LAST_NAME	MANAGER_ID
100	King	
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103
107	Lorentz	103
124	Mourgos	100

...

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
124	Mourgos

...



A WORKER tábla Manager_ID mezője megfelel
a MANAGER tábla EMPLOYEE_ID mezőjével

Önmagával való összekapcsolás (self-join) az ON kulcssqlal 2

```
SELECT e.last_name emp, m.last_name mgr
FROM   employees e JOIN employees m
ON     (e.manager_id = m.employee_id);
```

EMP	MGR
Hartstein	King
Zlotkey	King
Mourgos	King
De Haan	King
Kochhar	King

19 rows selected.

További feltételek megadása egy összekapcsoláshoz

```
SELECT e.employee_id, e.last_name, e.department_id,  
       d.department_id, d.location_id  
FROM   employees e JOIN departments d  
ON     (e.department_id = d.department_id)  
AND    e.manager_id = 149;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
174	Abel		80	2500
176	Taylor		80	2500

- Ugyanezt érhetjük el a WHERE feltétellel is, azaz:

```
SELECT e.employee_id, e.last_name, e.department_id,  
       d.department_id, d.location_id  
FROM   employees e JOIN departments d  
ON     (e.department_id = d.department_id)  
WHERE  e.manager_id = 149;
```

Három-utas összekapcsolás ON segítségével

```
SELECT employee_id, city, department_name  
FROM   employees e  
JOIN   departments d  
ON     d.department_id = e.department_id  
JOIN   locations l  
ON     d.location_id = l.location_id;
```

EMPLOYEE_ID	CITY	DEPARTMENT_NAME
103	Southlake	IT
104	Southlake	IT
107	Southlake	IT
124	South San Francisco	Shipping
141	South San Francisco	Shipping
142	South San Francisco	Shipping
143	South San Francisco	Shipping
144	South San Francisco	Shipping

- Három tábla összekapcsolását nevezük három-utas összekapcsolásnak
- Az SQL 1999-es szintaxis szerint az ilyen összekapcsolások balról jobbra haladva hajtódnak végre (DEPARTMENTS – EMPLOYEES) – LOCATION

Nem egyenlőségvizsgálaton alapuló összekapcsolás

EMPLOYEES

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Ernst	6000
Lorentz	4200
Mourgos	5800
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500
Zlotkey	10500
Abel	11000
Taylor	8600
...	

20 rows selected.

JOB_GRADES

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000



Az EMPLOYEES tábla fizetés mezőjének értéke a JOBS_GRADE tábla legmagasabb illetve legalacsonyabb fizetés közötti kell legyen.

Példa a nem egyenlőségvizsgálaton alapuló összekapcsolás

```
SELECT e.last_name, e.salary, j.grade_level
FROM employees e JOIN job_grades j
ON e.salary
BETWEEN j.lowest_sal AND j.highest_sal;
```

LAST NAME	SALARY	GRA
Matos	2600	A
Vargas	2500	A
Lorentz	4200	B
Mourgos	5800	B
Rajs	3500	B
Davies	3100	B
Whalen	4400	B
Hunold	9000	C
Ernst	6000	C

...

Külső összekapcsolás

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

8 rows selected.

EMPLOYEES

DEPARTMENT_ID	LAST_NAME
90	King
90	Kochhar
90	De Haan
60	Hunold
60	Ernst
60	Lorentz
50	Mourgos
50	Rajs
50	Davies
50	Matos
50	Vargas
80	Zlotkey

...
20 rows selected.

A 190-es számú osztályon
nincs alkalmazott

Belső vagy külső összekapcsolás?

- SQL-1999: Belső összekapcsolásnak nevezzük azokat az összekapcsolásokat, amelyek két tábla megegyező soraival térnek vissza.
- Két tábla olyan összekapcsolását, amely a belső összekapcsolás eredményéhez hozzáveszi a bal (vagy jobboldali) tábla összes sorát, baloldali (vagy jobboldali) külső összekapcsolásnak nevezzük.
- Teljes külső összekapcsolásnak hívjuk azt az esetet, amikor a külső összekapcsolás egyszerre bal- és jobboldali.

Baloldali külső összekapcsolás

```
SELECT e.last_name, e.department_id, d.department_name  
FROM employees e LEFT OUTER JOIN departments d  
ON (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
...		
De Haan	90	Executive
Kochhar	90	Executive
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		

20 rows selected.

Jobboldali külső összekapcsolás

```
SELECT e.last_name, e.department_id, d.department_name  
FROM employees e RIGHT OUTER JOIN departments d  
ON (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
Davies	50	Shipping
...		
Kochhar	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
	190	Contracting

20 rows selected.

Teljes külső összekapcsolás

```
SELECT e.last_name, d.department_id, d.department_name  
FROM employees e FULL OUTER JOIN departments d  
ON (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
...		
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		
	190	Contracting

21 rows selected.

A direkt szorzat

- A direkt-szorzat a következőként kapható:
 - az összekapcsolási feltétel elhagyásával,
 - nem megengedett összekapcsolási feltétellel,
 - az első tábla összes sorának összekapcsolása a másik tábla összes sorával.
- A direkt szorzatok elkerülése érdekében, mindig kell legalább egy megengedett összekapcsolási feltétel legyen.

A direkt szorzat

EMPLOYEES (20 rows)

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
...		
202	Fay	20
205	Higgins	110
206	Gietz	110

20 rows selected.

DEPARTMENTS (8 rows)

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700

8 rows selected.

Direkt-szorzat :
 $20 \times 8 = 160$ sor

EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
100	90	1700
101	90	1700
102	90	1700
103	60	1700
104	60	1700
107	60	1700

...

A direkt szorzat

- A CROSS JOIN kulcsszó előállítja két tábla keresztszorzatát (vagyis a direkt szorzatát)

```
SELECT last_name, department_name  
FROM employees CROSS JOIN departments ;
```

LAST_NAME	DEPARTMENT_NAME
King	Administration
Kochhar	Administration
De Haan	Administration
Hunold	Administration
...	

160 rows selected.

Kérdés/Válasz

- **Köszönöm a figyelmet! Kérdés/Válasz?**
- Több táblára (DEPT és EMP tábla) vonatkozó lekérdezésekre példák, összekapcsolások.
- **Házi feladat:** Oracle Példatár 2.fejezet feladatai, összesítések és csoportosítás, sorok rendezése.
- Oracle Példatár 3.fejezet feladatai, természetes összekapcsolások és alkérdések használata, de a hierarchikus és rekurzív lekérdezések még nem:
<http://people.inf.elte.hu/sila/eduAB/Feladatok.pdf>

Adatbázis tartalmának módosítása

SQL DML utasítások

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009



6.3. Alkérdések a záradékokban
(folyt) (where, having és group by)

6.5. Változtatások az adatbázisban:
SQL DML adatkezelő utasítások:
INSERT, DELETE, UPDATE

1. fejezet: Adatbázis-kezelő rendszerek (alapfogalmak)

Alkérdések

- A FROM listán és a WHERE záradékban (valamint a GROUP BY HAVING záradékában) zárójelezett SFW SELECT-FROM-WHERE utasításokat (**alkérdéseket**) is használhatunk.
- Szintaktikus alakja: zárójelbe kell tenni a lekérdezést
- Hol használható? Ott, ahol relációnevet használunk:
 - (1) WHERE és HAVING záradékban: kifejezésekben, feltételekben
 - (2) FROM listában: új listaelem (rel.név változó SQL-ben)
(lekérdezés) [AS] sorváltozó

Ez felel meg annak, ahogyan a relációs algebrában tetsz.helyen használhattuk a lekérdezés eredményét.

Alkérdések használata FROM listán

- **FROM záradékban** alkérdéssel létrehozott ideiglenes táblát is megadhatunk. Ilyenkor a legtöbb esetben meg kell adnunk a sorváltozó nevét. **Szintaktikus alakja:**
(lekérdezés) [AS] sorváltozó
- **Szemantikája:** A FROM záradékban kiértékelődik az alkérdés, utána a sorváltozót ugyanúgy használjuk, mint a közönséges adatbázis relációkat.
- **Példa:** Keressük meg a Joe's bár vendégei által kedvelt söröket (a feladatnak sok megoldása van)

Alkérdések használata FROM listán

- **FROM záradékban** alkérdéssel létrehozott ideiglenes táblát is megadhatunk. Ilyenkor a legtöbb esetben meg kell adnunk a sorváltozó nevét.
- **Példa:** Keressük meg a Joe's bár vendégei által kedvelt söröket.

SELECT sör

Sörivók, akik látogatják
Joe's bárját.

FROM Szeret, (SELECT név

FROM Látogat

WHERE bár = 'Joe''s bar') JD

WHERE Szeret.név = JD.név;

Ismétlés: Alkérdések a WHERE záradékban

Fontos! Ugyanezt használjuk a mai anyagban

SQL DML utasítások WHERE záradékában!

WHERE és HAVING záradékban:

- (i) Az alkérdés eredménye egyetlen **skalárérték**, vagyis az alkérdés olyan, mint a konstans, ami egy új elemi kifejezésként tetszőleges kifejezésben használható.
- (ii) **Skalár értékekből álló multihalmaz** logikai kifejezésekben használható: [NOT] EXISTS (lekérdezés)
kifejezés [NOT] IN (lekérdezés)
kifejezés Θ [ANY | ALL] (lekérdezés)
- (iii) **Teljes, többdimenziós tábla** a visszatérő érték:
[NOT] EXISTS (lekérdezés)
 $(kif_1, \dots kif_n)$ [NOT] IN (lekérdezés)

Emlékeztető: Példa csoportosításra

```
SELECT onev, AVG(fizetes) + 100 emelt  
FROM dolgozo d, osztaly o  
WHERE d.oazon=o.oazon AND telephely='Bp'  
GROUP BY o.oazon, onev  
HAVING COUNT(dkod) > 3  
ORDER BY onev;
```

$\tau_{onev}(\pi_{onev, \text{átlagfiz}+100 \rightarrow \text{emelt}}($
 $(\sigma_{COUNT(dkod)>3}(\gamma_{o.oazon, onev, AVG(fizetes), COUNT(dkod)}($
 $(\sigma_{telephely='Bp'}(d \bowtie o))))))$

--- HF: Az operátorok egymás utáni alkalmazását
kifejezésfa formájában is rajzolhatjuk fel!

Példa alkérdésre a HAVING-ben --1

- Felszolgál(söröző, sör, ár) és Sörök(név, gyártó) táblák felhasználásával adjuk meg az átlagos árat azon söroknek, melyeket
 - legalább három sörözőben felszolgálnak,
 - vagy Pete a gyártójuk.

Példa alkérdésre a HAVING-ben --2

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

HAVING COUNT(söröző) >= 3 OR

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

(HAVING...) Sör csoportok,
Melyeket legalább három
nem-NULL bárban árulnak,
Vagy Pete a gyártójuk.

(SELECT...)
Sörök, melyeket
Pete gyárt (ez az
Ullman mo., de
H.F.: más mo-okkal

Adatbázis tartalmának módosítása

Tankönyv 6.5. Változtatások az adatbázisban

- A módosító **utasítások** nem adnak vissza eredményt, mint a lekérdezések, hanem az adatbázis tartalmát változtatják meg.
- 3-féle módosító utasítás létezik:
 - INSERT** - sorok beszúrása
 - DELETE** – sorok törlése
 - UPDATE** – sorok komponensei értékeinek módosítása

Beszúrás (insert into)

- Két alakja van:
 - 1.) ha egyetlen sort szűrunk be:

INSERT INTO <reláció>

VALUES (<konkrét értékek lista>);

- 2.) ha több sort, egy lekérdezés eredményét visszük fel alkérdés segítségével:

INSERT INTO <reláció>

(<alkérdés>);

Beszúrás, attribútumok megadása

- Példa: A Szeret táblába beírjuk, Zsu szereti a Bud sört.

```
INSERT INTO Szeret  
VALUES('Zsu', 'Bud');
```

- A reláció neve után megadhatjuk az attribútumait.
- Ennek alapvetően 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: `INSERT INTO Szeret(sör, név)
VALUES('Bud', 'Zsu');`

Default 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.

```
CREATE TABLE Sörivók (
    név CHAR(30) PRIMARY KEY,
    cím CHAR(50) DEFAULT 'Sesame St'
    telefon CHAR(16) );
INSERT INTO Sörivók(név)
    VALUES ('Zsu');
```

Az eredmény sor:

név	cím	telefon
Zsu	Sesame St	NULL

Több sor beszúrása

- Egy lekérdezés eredményét is beszúrhatjuk:
INSERT INTO <reláció>
(<alkérdés>);
- A Látogat(név, söröző) tábla felhasználásával adjuk hozzá a LehetBarát(név) táblához Zsu „lehetséges barátait”, vagyis azokat a sörvíköt, akik legalább egy olyan sörözőt látogatnak, ahova Zsu is szokott járni.

Megoldás: Több sor beszúrása

```
INSERT INTO LehetBarát  
(SELECT I2.név  
    FROM Látogat I1, Látogat I2  
    WHERE I1.név = 'Zsu' AND  
        I2.név <> 'Zsu' AND  
        I1.söröző = I2.söröző  
);
```

(SELECT) a másik sörivő

(FROM) névpárok:
az első Zsu,
a második nem Zsu,
de van olyan bár,
amit mindketten
látogatnak.

Tk.Példa INSERT INTO utasításra

- A lekérdezést teljesen ki kell értékelni, mielőtt a sorokat beszúrnánk.
- Tankönyv 6.36 példa: új stúdiók beszúrása

```
INSERT INTO Stúdió (név)
  (SELECT DISTINCT stúdióNév
   FROM Filmek
   WHERE stúdióNév NOT IN
     (SELECT név FROM Stúdió));
```

Törlés (delete)

- A törlendő sorokat egy WHERE feltétel segítségével adjuk meg:

DELETE FROM <reláció>

WHERE <feltétel>;

- Példa:

DELETE FROM Szeret

WHERE nev = 'Zsu' AND
sör = 'Bud';

- Az összes sor törlése:

DELETE FROM Szeret;

Példa: Több sor törlése

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

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);
```

(WHERE) azok a sörök, amelyeknek ugyanaz a gyártója, mint az s éppen aktuális sorának, a nevük viszont különböző.

A törlés szemantikája

- 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 alkérdez nem üres, a későbbi Bud Lite sor miatt, így a Bud törlődik.
- **Kérdés:** a Bud Lite sor törlődik-e?

A törlés szemantikája

- **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 (update)

- Bizonyos sorok bizonyos attribútumainak módosítása.

UPDATE <reláció>

SET <attribútum értékkadások lista>

WHERE <sorokra vonatkozó feltétel>;

- Fecó telefonszámát 555-1212-re változtatjuk (Fecó itt egy sörivő neve):

UPDATE Sörivők

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;

- Olcsó sörök árát duplázzuk

UPDATE Felszolgál

SET ár = 2 * ár

WHERE ár < 1.00;

Tk.Példa UPDATE utasításra

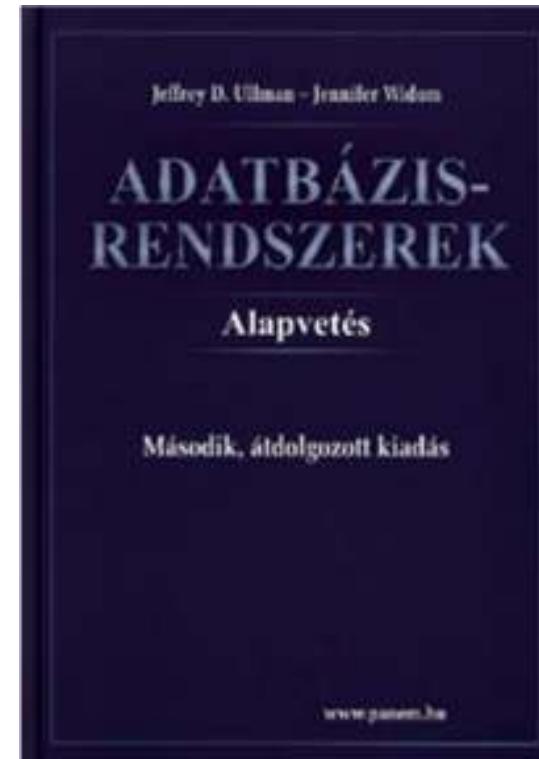
- Tankönyv 6.39 példa:

```
UPDATE GyártásIrányító  
SET név = 'Ig.' || név  
WHERE azonosító IN  
(SELECT elnökAzon FROM Stúdió)
```

Adatbázis-kezelő rendszerek felépítése

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009

1. fejezet: Az adatbázis-kezelő rendszerek (DBMS) felépítése, alapfogalmak, ACID tranzakciók



Az adatbázisrendszerek világa

Tk.1.fejezete Az adatbázis-kezelő rendszerek áttekintése

- Adatbázisok-1 kurzuson mit láttunk eddig és mit fogunk venni az adatbázisrendszerek világából?
 - Adatbázist, adatok gyűjteményét kezeli, relációs modell típus: sortípus, gyűjtemény: reláció
 - Hogyan tervezzük meg milyen gyűjteményünk legyen? Lesz majd tervezés: E/K modell, UML diagramok, Relációs adatbázis sématervezés (FF, TÉF, NF)
 - Metaadatok kezelése: DDL sémaleíró nyelv
 - Táblák tartalmának lekérdezése (select) és módosítása: insert-delete-update: DML adatkezelő nyelv
 - Lekérdezések feldolgozása: alap és kiterjesztett relációs algebra, SQL: SELECT, program (SQL/PSM, PL/SQL)

(1) Adatbázis-kezelés

Adatbázis-kezelés:

- (1) Háttértárolón tárolt, nagy adatmennyiségek hatékony kezelése (lekérdezése, módosítása)
- (2) Adatmodell támogatása
- (3) Adatbázis-kezelő nyelvek támogatása
- (4) Több felhasználó támogatása
- (5) Adatvédelem, adatbiztonság
- (6) Tranzakció-kezelés
- (7) Konkurenencia-kezelés
- (8) Naplózás és helyreállíthatóság
- (9) Lekérdezések végrehajtásának optimalizálása

(2) Adatmodell támogatása

- Az adatmodell a valóság fogalmainak, kapcsolatainak, tevékenységeinek magasabb szintű ábrázolása
 - File-kezelés indexekkel együtt, ezt váltotta fel a
 - CODASYL szabvány, hálós adatmodell (hatékony keresés)
 - Hierarchikus adatmodell (apa-fiú kapcsolatok gráfja)
 - Ted Codd - Relációs adatmodell (táblák rendszere, könnyen megfogalmazható műveletek)
 - Objektum-orientált adatmodell (az adatbázis-kezelés funkcionalitásainak biztosítása érdekében gyakran relációs adatmodellre épül), + Objektum-relációs adatmodell
 - Logikai adatmodell (szakértői rendszerek, tények és következtetési szabályok rendszere)
 - Dokumentumok - Félig strukturált adatmodell, az XML (szabvány adatcsereformaként jelent meg)

(3) Adatbázis-kezelő nyelvek támogatása

- **SQL** – relációs (és objektum-relációs) adatbázis-kezelő szabvány nyelv, fontosabb szabványok:
SQL86, SQL89, SQL92 (SQL2), **SQL:1999** (SQL3),
SQL: 2003, SQL:2006, SQL:2008
- **DDL** (Data Definition Language) adatdefiniáló (sémaleíró) nyelv: sémák, adatstruktúrák megadása, objektumok létrehozása, módosítása, törlése: CREATE, ALTER, DROP
- **DML** (Data Manipulation Lang.) adatkezelő és lekérdező nyelv: INSERT, DELETE, UPDATE és SELECT
- **DCL** (Data Control Lang.) adatvezérlő nyelv, jogosultságok kiosztása és visszavonása: GRANT, REVOKE
- **Tranzakció-kezelés**: COMMIT, ROLLBACK

(4) Több felhasználó támogatása

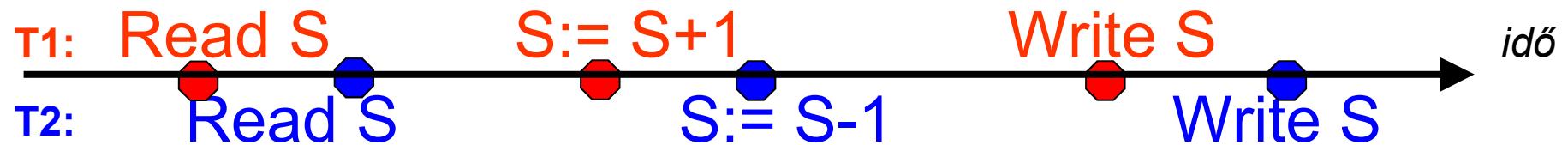
- **Felhasználói csoportok. Kulcsemberek:**
 - **DBA** adatbázis-rendszergazda
 - felügyeli az adatbázis-példányokat és adatbázis-szervereket
 - felépíti a rendszert, implementálja és optimális adatbázis-megoldást biztosít
 - Adatbázis-tervező (sématervezés)
 - Alkalmazás-fejlesztő, programozó (kódolás)
 - Felhasználók (akik használják a rendszert)

(5) Adatvédelem, adatbiztonság

- **Jogosultságok** (objektumok olvasása, írása, módosítása, készítése, törlése, jogok továbbadása, jogok visszavonása)
- GRANT és REVOKE utasítás
- Jogosultságok tárolása rendszertáblákban történik
- **Jogosultságok kezelése**, felhasználók, jelszavak, hozzáférési jogok
- Adatbázissémák korlátozása (virtuális) nézettáblák segítségével
- Tárolt adatok, hálózati adatforgalmak titkosítása (nagy prímszámok, RSA, DES)

(6) Tranzakció-kezelés

- **Tranzakció:** adatkezelő műveletekből (adategység írása, olvasása) álló sorozat
- Cél: tranzakciók párhuzamos végrehajtása



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

(6) 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 adatbázis rendszereknek el kell különíteniük a folyamatokat.

(6) Példa: rossz interakció

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

(6) Tranzakciók

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

(6) A tranzakciók ACID tulajdonságai

- Atomiság (atomicity): a tranzakció egységesen lefut vagy nem, vagy az összes vagy egy utasítás sem hajtódi végre.
- Konzisztencia (consistency): a tranzakció futása után konzisztens legyen az adatbázis, megszorításokkal, triggerekkel biztosítjuk.
- Elkülönítés (isolation): párhuzamos végrehajtás eredménye egymás utáni végrehajtással egyezzen meg
- Tartósság (durability): a befejezett tranzakció eredménye rendszerhiba esetén sem veszhet el

(6) COMMIT és ROLLBACK

- A COMMIT utasítás a tranzakció sikeres befejeződését eredményezi. Egy sikeresen befejeződött tranzakció a kezdete óta végrehajtott utasításainak módosításait tartósan rögzíti az adatbázisban
 - vagyis a módosítások *vélegesítődnek*.
- A ROLLBACK utasítás megszakítja a tranzakció végrehajtását, és annak sikertelen befejeződését eredményezi. Az így befejezett tranzakció SQL utasításai által végrehajtott módosításokat a rendszer meg nem történtekké teszi
 - Vagyis az összes utasítás *visszagörgetésre kerül*, a módosítások nem jelennék meg az adatbázisban.

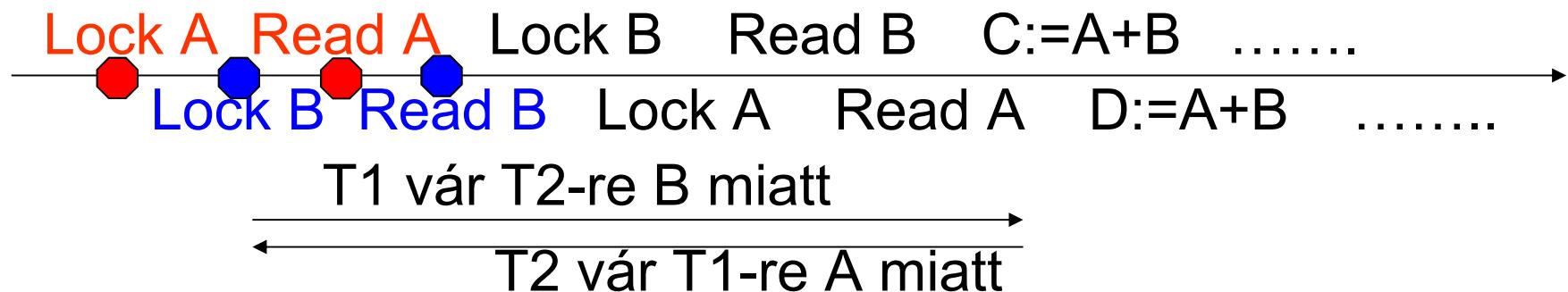
(7) Konkurencia-kezelés

➤ Zárolások (Lock, Unlock)

T1: (Lock S, Read S, $S:=S+1$, Write S, Unlock S)

T2: (Lock S, Read S, $S:=S-1$, Write S, Unlock S)

- A zár kiadásához meg kell várni a zár feloldását.
- Csökken a párhuzamosíthatóság
- Zárak finomsága (zárolt adategység nagysága, zárolás típusa) növeli a párhuzamosíthatóságot
- **Holtpont probléma:**



(8) Naplózás és helyreállítás

- Szoftver- vagy hardverhiba esetén az **utolsó konzisztens állapot visszaállítása**
- Rendszeres **mentések**
 - Statikus adatbázis (módosítás nem gyakori)
 - Dinamikus adatbázis (módosítás gyakori)
- **Naplóállományok**
- Összefügg a tranzakció-kezeléssel

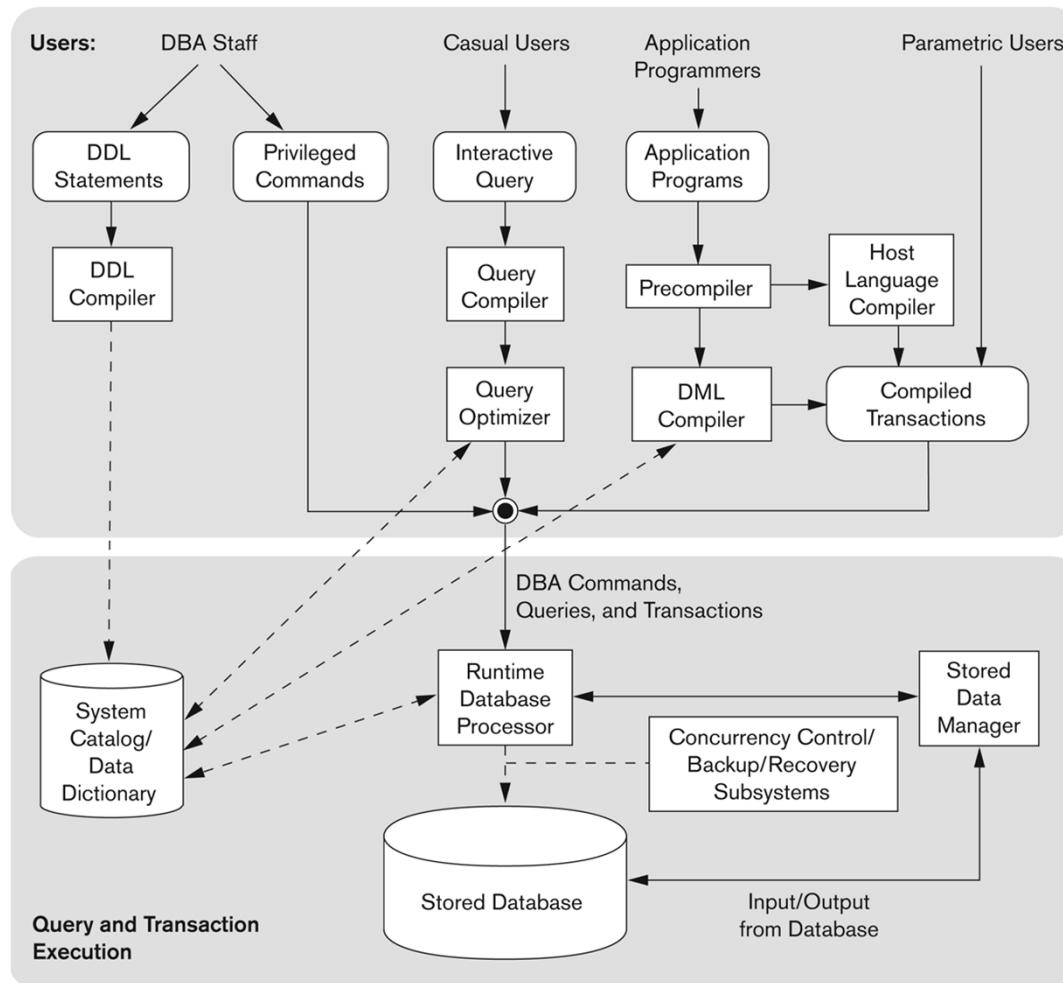
(9) Lekérdezések végrehajtása optimalizálás



Adatbázis-kezelők részei

- **Lekérdezés-feldolgozó**
 - Lekérdezés szintaktikai ellenőrzése
 - Adatbázis-objektumok létezésének, és a hozzáférési jogoknak az ellenőrzése (metaadatbázis, rendszertáblák)
 - Lekérdezés optimális átfogalmazása
 - Végrehajtási tervezés készítése
 - Az adatstruktúrák, méretek statisztikái alapján várhatóan minimális költségű végrehajtási terv kiválasztása
 - Az optimális végrehajtási terv lefuttatása
- **Tranzakció-kezelő:**
 - Tranzakciók párhuzamos végrehajtásának biztosítása (atomosság, következetesség, elkülönítés, tartósság)
- **Tárkezelő és pufferkezelő**
 - fizikai adatstruktúrák, táblák, indexek, pufferek kezelése

Adatbázis-kezelő rendszer felépítése



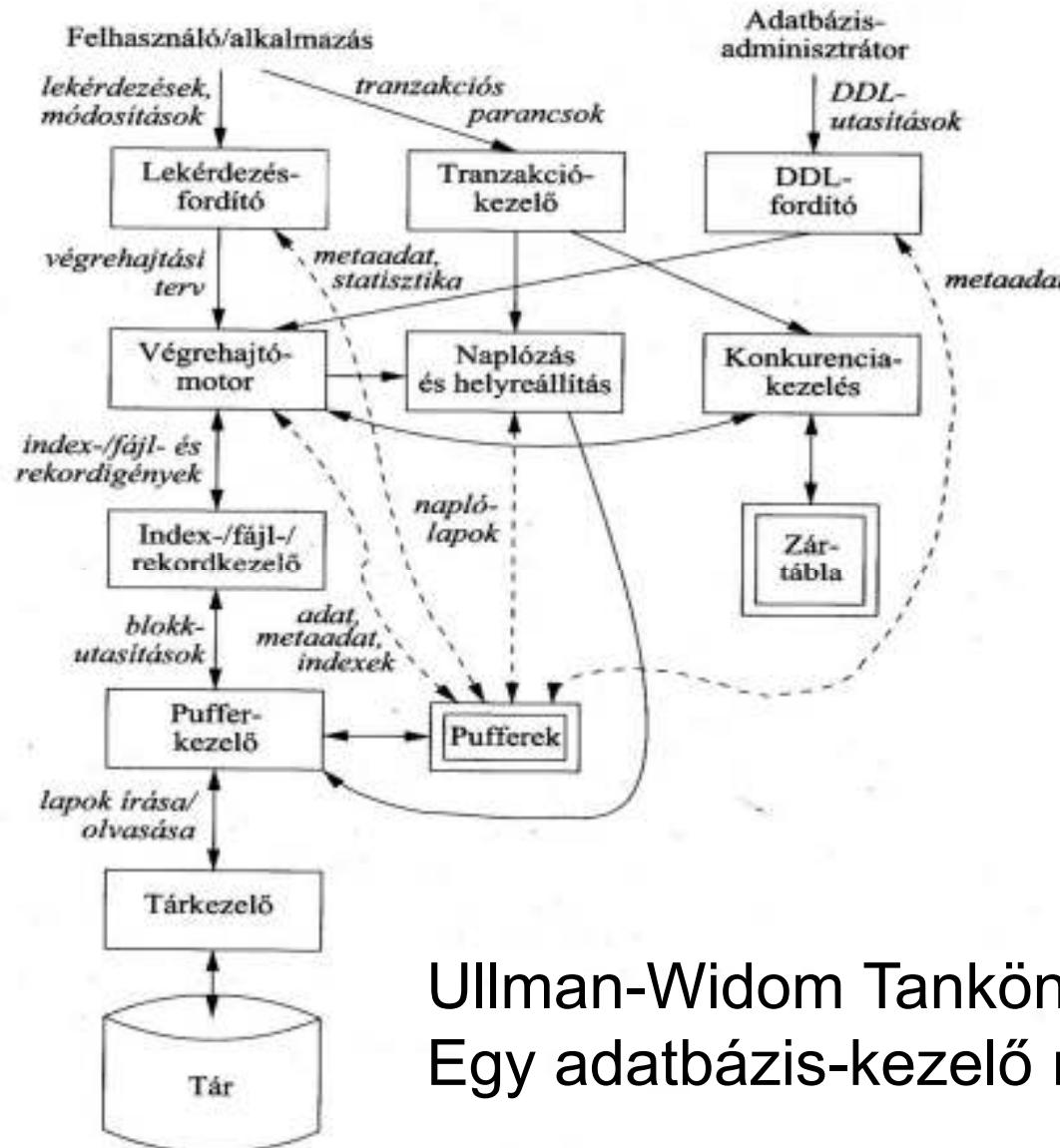
Forrás:

Elmasri-Navathe: Fundamentals of Database Systems

Figure 2.3

Component modules of a DBMS and their interactions.

Adatbázis-kezelő rendszer felépítése



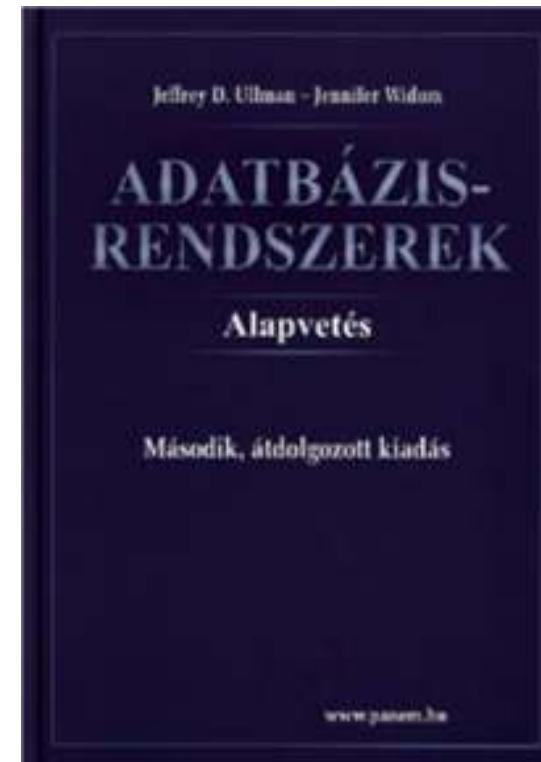
Ullman-Widom Tankönyv 1.1. ábra
Egy adatbázis-kezelő rendszer részei

Kérdés/Válasz

- Köszönöm a figyelmet! Kérdés/Válasz?
- Gyakorlás a 5EA-hoz: Több táblára (DEPT és EMP tábla) vonatkozó lekérdezésekre példák, összekapcsolások.
- Házi feladat: Oracle Példatár 3.fejezet feladatai, összekapcsolások és alkérdések használata, de a hierarchikus és rekurzív lekérdezések még nem:
<http://people.inf.elte.hu/sila/eduAB/Feladatok.pdf>

Logikai lekérdező nyelv: Datalog Rekurzió a Datalogban és az SQL-ben

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009



5.3. Logika a relációkhöz: Datalog

5.4. Relációs algebra és
nem-rekurzív biztonságos Datalog
kifejező erejének összehasonlítása

10.2. Az Eljut-feladat megoldása: (monoton és lineáris)
rekurzió a Datalogban és az SQL-ben: with recursion

Következik...

- Relációs algebra korlátai: bizonyos típusú lekérdezéseket nem tudunk relációs algebrával kifejezni...
- Nézzünk meg olyan logikai felépítést, amivel az ilyen rekurzív jellegű lekérdezések könnyen megoldhatók.
- Ez a nyelv: a Datalog (Tankönyv 5.3-5.4)
- Rekurzió (Tankönyv 10.2)

Milyen fontos rekurzív feladatok vannak?

I. Hierarchiák bejárása

- Leszármazottak-ősök ParentOf(parent,child)
 - Find all of Mary's ancestors
- Vállalati hierarchia felettes-beosztott
 - Employee(ID,salary)
 - Manager(mID,eID)
 - Project(name,mgrID)
 - Find total salary cost of project 'X'
- Alkatrész struktúra (mely alkatrésznek mely alkatrész része)

Milyen fontos rekurzív feladatok vannak?

II. Gráf jellegű bejárások

➤ Repülőgép járatok, eljut-feladat

Flight(orig,dest,airline,cost)

➤ Find cheapest way to fly from ‘A’ to ‘B’

➤ Közösségi hálók

Ki-kinek az ismerőse, Twitterben ki-kit követ

Kiegészítés a gráf adatbázisokról

➤ Gráfok könnyen megadhatók relációs táblával, a gráf lekérdezések egyre gyakoribb feladatok, ezek relációs megoldása hatékonyiségi kérdés. Vannak kimondottan gráf-adatbázisok.

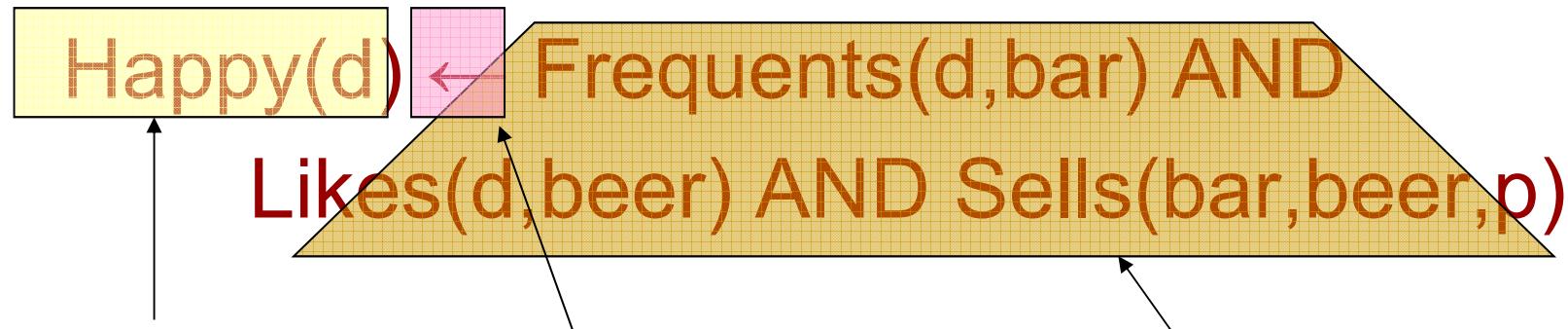
Logika, mint lekérdező nyelv

- Abstract Query Languages:
 - Relational Algebra (procedural → optimization)
 - Logical QL: Datalog, Rel.Calculus (declarative)
- **Datalog** = ‘Data’- Database, ‘log’- logic, Prolog
- Logikai alapú nyelv, szabály alapú nyelv. If-then logical rules have been used in many systems.
- Nonrecursive rules are equivalent to the core relational algebra.
- Recursive rules extend relational algebra and appear in SQL-99.

Datalog szabályok és lekérdezések

- Datalog: logikai alapú, szabály-alapú nyelv
szabály: IF feltétel THEN eredmény, ahol a
feltétel relációkkal legyen megadható és az
eredmény az output tábla sorait eredményezze
- Our first example of a rule uses the relations
*Frequents(drinker, bar),
Likes(drinker, beer),
Sells(bar, beer, price).*
- The rule is a query asking for “happy” drinkers
--- those that frequent a bar that serves a beer
that they like.

Datalog szabályok felépítése



Head = consequent,
a single subgoal

Body = antecedent =
AND of *subgoals*.

Read this
symbol “if”

Atomi formulák-1: Relációs formulák

- An *atom* is a *predicate*, or relation name with variables or constants as arguments.
- The head is an atom; the body is the AND of one or more atoms.
- **Convention:** Predicates begin with a capital, variables begin with lower-case.

Példa relációs atomi formulára

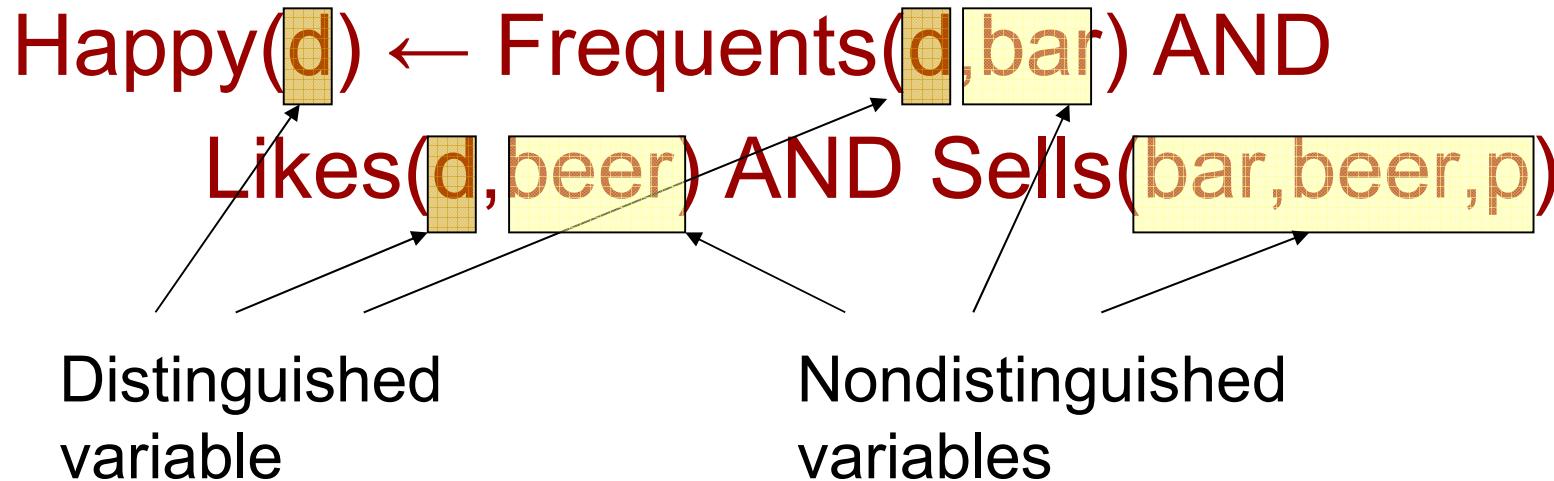
(1) **Sells**(bar, beer, p)

The predicate
= name of a
relation

Arguments are
variables (or constants).

(2) Proba(x, x, y, 5, 'alma')

Example: Interpretation



Interpretation: drinker d is happy if there exist a bar, a beer, and a price p such that d frequents the bar, likes the beer, and the bar sells the beer at price p .

Datalog szabályok kiértékelése ---1

- Approach 1: consider all combinations of values of the variables.
- If all subgoals are true, then evaluate the head.
- The resulting head is a tuple in the result.

Example: Rule Evaluation

Happy(d) \leftarrow Frequent(d,bar) AND
Likes(d,beer) AND Sells(bar,beer,p)

FOR (each d, bar, beer, p)

IF (Frequent(d,bar), Likes(d,beer), and
Sells(bar,beer,p) are all true)

add Happy(d) to the result

- Note: set semantics so add only once.
- Set semantics vice versa bag semantics

Datalog szabályok kiértékelése ---2

- Approach 2: For each subgoal, consider all tuples that make the subgoal true.
- If a selection of tuples define a single value for each variable, then add the head to the result.

Happy(d) \leftarrow Frequent(d,bar) AND
Likes(d,beer) AND Sells(bar,beer,p)
FOR (each f in Frequent, i in Likes,
and s in Sells)
IF (f[1]=i[1] and f[2]=s[1] and i[2]=s[2])
add Happy(f[1]) to the result

Milyen problémák merülnek fel? (erre visszatérünk a biztonságosságnál)

- Relations are finite sets.
- We want rule evaluations to be finite and lead to finite results.
- “Unsafe” rules like $P(x) \leftarrow Q(y)$ have infinite results, even if Q is finite.
- Even $P(x) \leftarrow Q(x)$ requires examining an infinity of x -values.

Atomi formulák-2: Aritmetikai formula

- In addition to relations as predicates, a predicate for a subgoal of the body can be an arithmetic comparison.
- We write arithmetic subgoals in the usual way, e.g., $x < y$.

Példa: Aritmetikai részcélok

- A beer is “cheap” if there are at least two bars that sell it for under \$2.

```
Cheap(beer) ← Sells(bar1,beer,p1) AND  
Sells(bar2,beer,p2) AND p1 < 2.00  
AND p2 < 2.00 AND bar1 <> bar2
```

Negált részcélok

- NOT in front of a subgoal negates its meaning.
- Example: Think of $\text{Arc}(a,b)$ as arcs in a graph.
 - $S(x,y)$ says the graph is not transitive from x to y ; i.e., there is a path of length 2 from x to y , but no arc from x to y .

$S(x,y) \leftarrow \text{Arc}(x,z) \text{ AND } \text{Arc}(z,y)$
AND NOT $\text{Arc}(x,y)$

Biztonságossági elvárás

- A szabályok kiértékelhetőek legyenek, ehhez:
- A szabályban szereplő minden változónak elő kell fordulnia a törlésben nem-negált relációs atomban

Biztonságos szabályok

- A rule is *safe* if:
 1. Each distinguished variable,
 2. Each variable in an arithmetic subgoal, and
 3. Each variable in a negated subgoal,
also appears in a nonnegated,
relational subgoal, amivel az x korlátrozott:
 - $\text{pred}(x, y, \dots)$ argumentuma (értéke a táblából)
 - vagy $x=c$ (konstans)
 - vagy $x=y$ (ahol y korlátrozott)
- Safe rules prevent infinite results.

Példa: Nem biztonságos szabályokra

- Each of the following is unsafe and not allowed:
 1. $S(x) \leftarrow R(y)$
 2. $S(x) \leftarrow R(y) \text{ AND } x < y$
 3. $S(x) \leftarrow R(y) \text{ AND NOT } R(x)$
- In each case, an infinity of x 's can satisfy the rule, even if R is a finite relation.

Datalog programok

- Datalog program = collection of rules.
- In a program, predicates can be either
 1. EDB relációk = Extensional Database = stored table (csak a törzsben szereplő relációk)
 2. IDB relációk = Intensional Database = relation defined by rules (szerepel fej-ben)
- Never both! No EDB in heads.

Datalog programok kiértékelése

- As long as there is no recursion, we can pick an order to evaluate the IDB predicates, so that all the predicates in the body of its rules have already been evaluated.
- If an IDB predicate has more than one rule, each rule contributes tuples to its relation.

Példa: Datalog program

- Using EDB `Sells(bar, beer, price)` and `Beers(name, manf)`, find the manufacturers of beers Joe doesn't sell.

`JoeSells(b) ← Sells('Joe's Bar', b, p)`

`Answer(m) ← Beers(b,m)`

`AND NOT JoeSells(b)`

Példa: Kiértékelése

- Step 1: Examine all **Sells** tuples with first component 'Joe''s Bar'.
 - Add the second component to **JoeSells**.
- Step 2: Examine all **Beers** tuples (b,m).
 - If b is not in **JoeSells**, add m to Answer.

Datalog kifejező ereje

- Without recursion, Datalog can express all and only the queries of core relational algebra.
 - The same as SQL select-from-where, without aggregation and grouping.
- But with recursion, Datalog can express more than these languages.

Relációs algebrai kifejezések átírása

- Mi a leggyakrabban előforduló típus, amiből építkeztek? $\Pi_{\text{Lista}}(\sigma_{\text{Felt}}(R \bowtie S \bowtie \dots))$
Ezt a komponenst támogatja legerősebben az SQL is: **SELECT lista**
FROM táblák összekapcsolása
WHERE felt
Ez felel meg egy Datalog szabálynak.
- **Halmazműveletek:** kezdjük ezzel, hogyan lehet a metszetet és különbséget Datalog szabállyal, és az egyesítést Datalog programmal kifejezni.

Relációs algebra és Datalog ---1

Rel.algebrai műveletek hogyan néznek ki Datalogban?

Halmazműveletek: T.f.h $R(x_1, \dots, x_n)$, $S(x_1, \dots, x_n)$

predikátumokhoz tartozó reláció $R(A_1, \dots, A_n)$, $S(A_1, \dots, A_n)$

➤ $R \cap S$ metszetnek megfelelő szabály:

$\text{Válasz}(x_1, \dots, x_n) \leftarrow R(x_1, \dots, x_n) \text{ AND } S(x_1, \dots, x_n)$

➤ $R - S$ különbségnek megfelelő szabály:

$\text{Válasz}(x_1, \dots, x_n) \leftarrow R(x_1, \dots, x_n) \text{ AND NOT } S(x_1, \dots, x_n)$

➤ $R \cup S$ unió műveletet egyetlen szabállyal nem tudom felírni, mert a törzsben csak AND lehet, OR nem.
Ehhez több szabályból álló Datalog program kell:

$\text{Válasz}(x_1, \dots, x_n) \leftarrow R(x_1, \dots, x_n)$

$\text{Válasz}(x_1, \dots, x_n) \leftarrow S(x_1, \dots, x_n)$

Relációs algebra és Datalog ---2

Kiválasztás:

- $\sigma_{x_i \theta x_j}(R)$ kifejezésnek megfelelő szabály :
Válasz(x_1, \dots, x_n) $\leftarrow R(x_1, \dots, x_n) \text{ AND } x_i \theta x_j$
- $\sigma_{x_i \theta c}(E1)$ kifejezésnek megfelelő szabály:
Válasz(x_1, \dots, x_n) $\leftarrow R(x_1, \dots, x_n) \text{ AND } x_i \theta c$

Vetítés:

- $\Pi_{A_{i1}, \dots, A_{ik}}(R)$ kifejezésnek megfelelő szabály:
Válasz(x_{i1}, \dots, x_{ik}) $\leftarrow R(x_1, \dots, x_n)$

Megjegyzés: név nélküli anonymous változók, amelyek csak egyszer szerepelnek és mindegy a nevük azt aláhúzás helyettesítheti. Például:

HosszúFilm(c,é) $\leftarrow \text{Film}(c, e, h, _, _, _)$ AND $h \geq 100$

Relációs algebra és Datalog ---3

Természetes összekapcsolás: Tegyük fel, hogy

$R(A_1, \dots, A_n, C_1, \dots, C_k)$ és $S(B_1, \dots, B_m, C_1, \dots, C_k)$

- $R \bowtie S$ kifejezésnek megfelelő szabály:

$\text{Válasz}(x_1, \dots, x_n, y_1, \dots, y_m, z_1, \dots, z_k) \leftarrow$

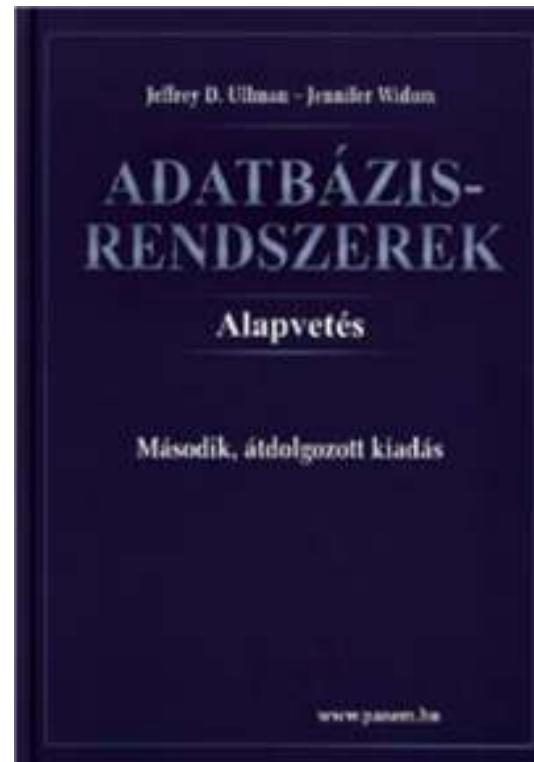
$\leftarrow R(x_1, \dots, x_n, z_1, \dots, z_k) \text{ AND } S(y_1, \dots, y_m, z_1, \dots, z_k)$

- A felírt szabályok biztonságosak.
- minden Q relációs algebrai kifejezéshez van nem rekurzív, biztonságos, negációt is tartalmazó Datalog program, amelyben egy kitüntetett IDB predikátumhoz tartozó kifejezés ekvivalens a Q lekérdezéssel.
- A nem rekurzív, biztonságos, negációt is tartalmazó Datalog kifejezőrő tekintetében EKVIVALENS a relációs algebrával.

Logikai lekérdező nyelv: Datalog Rekurzió a Datalogban és az SQL-ben

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009

10.2. Az Eljut-feladat megoldása:
(monoton és lineáris) rekurzió
a Datalogban és az SQL-ben:
with recursion záradék (select)



Az Eljut-feladat

Tankönyv 10.2. fejezet példája (az ELJUT feladat)

- **Jaratok**(legitarsasag, honnan, hova, koltseg, indulas, erkezes) táblában repülő járatok adatait tároljuk.
- A járatok táblát létrehozó script:
http://people.inf.elte.hu/sila/eduAB/jaratok_tabla.txt
- **Mely (x,y) párokra lehet eljutni x városból y városba?**
- Ezt egy relációs algebrai kifejezésként nem tudjuk megadni zárt alakban, klasszikus SQL SELECT utasítással sem tudjuk kifejezni, csak azt tudjuk, hogy átszállás nélkül, egy, két, stb... átszállással:

Az Eljut-feladatnak nincs algebrai megoldása

select distinct honnan, hova

from jaratok

union

select j1.honnan, j2.hova

from jaratok j1, jaratok j2

where j1.hova=j2.honnan

union

select j1.honnan, j3.hova

from jaratok j1, jaratok j2, jaratok j3

where j1.hova=j2.honnan

and j2.hova=j3.honnan

--- union stb... Ezt így nem lehet felírni...

Az Eljut-feladat Datalogban

Tankönyv 10.2. fejezet példája (az ELJUT feladat)

- Jaratok(legitarsasag, honnan, hova, koltseg, indulas, erkezes) EDB-táblában repülőjáratok adatait tároljuk.
Mely (x,y) párokra lehet eljutni x városból y városba?

- Datalogban felírva (lineáris rekurzió)

$\text{Eljut}(x, y) \leftarrow \text{Jaratok}(l, x, y, k, i, e)$

$\text{Eljut}(x, y) \leftarrow \text{Eljut}(x, z) \text{ AND } \text{Jaratok}(l, z, y, k, i, e)$

- Vagy másképp felírva Datalogban (mi a különbség?)

$\text{Eljut}(x, y) \leftarrow \text{Jaratok}(_, x, y, _, _, _)$ --- anonimus változók

$\text{Eljut}(x, y) \leftarrow \text{Eljut}(x, z) \text{ AND } \text{Eljut}(z, y)$ --- nem lineáris rek.

Az Eljut feladat SQL-99 szabványban

- Datalog **LINEÁRIS, MONOTON** rekurzió átírható:
Eljut(x, y) <- Jaratok(l, x, y, k, i, e)
Eljut(x, y) <- Eljut(x, z) AND Jaratok(l, z, y, k, i, e)
- Hova, mely városokba tudunk eljutni Budapestről?
WITH RECURSIVE Eljut AS
(SELECT honnan, hova FROM Jaratok
UNION
 SELECT Eljut.honnan, Jaratok.hova
 FROM Eljut, Jaratok
 WHERE Eljut.hova = Jaratok.honnan)
SELECT hova FROM Eljut WHERE honnan='Bp';

SQL-99 szabvány: Rekurzív lekérdezés

- A WITH utasítás több ideiglenes relációra vonatkozó definíciója:

```
WITH [RECURSIVE] R1 AS <R1 definíciója>
      [RECURSIVE] R2 AS <R2 definíciója>
      ...
      [RECURSIVE] Rn AS <Rn definíciója>
< R1,R2,...,Rn relációkat tartalmazó lekérdezés>
```

Egy másik példa rekurzióra

- EDB: $\text{Par}(c,p) = p \text{ is a parent of } c.$
- Generalized cousins: people with common ancestors one or more generations back:

$\text{Sib}(x,y) \leftarrow \text{Par}(x,p) \text{ AND } \text{Par}(y,p) \text{ AND } x <> y$

$\text{Cousin}(x,y) \leftarrow \text{Sib}(x,y)$

$\text{Cousin}(x,y) \leftarrow \text{Par}(x,xp) \text{ AND } \text{Par}(y,yp)$
AND $\text{Cousin}(xp,yp)$

- Feladat: Datalog program átírása SQL-99 szabvány SELECT utasításra WITH RECURSIVE záradékkal:

Példa: átírás rekurzív lekérdezésre ---1

- Find Sally's cousins, using SQL like the recursive Datalog example. $\text{Par}(\text{child}, \text{parent})$ is the EDB.

- Megoldás (két oldalon fert el)

WITH Sib(x,y) AS --- ez itt az (1)

SELECT p1.child x, p2.child y
FROM Par p1, Par p2

WHERE p1.parent = p2.parent AND
p1.child <> p2.child;

Like $\text{Sib}(x,y) \leftarrow$
 $\text{Par}(x,p) \text{ AND}$
 $\text{Par}(y,p) \text{ AND}$
 $x <> y$

RECURSIVE Cousin(x,y) AS --- itt jön a (2) ...

--- SELECT ...köv.oldalon ...

SELECT y FROM Cousin WHERE x = 'Sally' ;

Példa: átírás rekurzív lekérdezésre ---2

Required – Cousin
is recursive

WITH Sib(x,y) AS --- előző oldalon (1)...
RECURSIVE Cousin(x,y) AS
(SELECT * FROM Sib) ← Reflects $\text{Cousin}(x,y) \leftarrow \text{Sib}(x,y)$
UNION
(SELECT p1.child x, p2.child y
FROM Par p1, Par p2, Cousin
WHERE p1.parent = Cousin.x AND
p2.parent = Cousin.y) ← Reflects
 $\text{Cousin}(x,y) \leftarrow \text{Par}(x,\text{xp}) \text{ AND }$
 $\text{Par}(y,\text{yp}) \text{ AND }$
 $\text{Cousin}(\text{xp},\text{yp})$

SELECT y FROM Cousin WHERE x = 'Sally' ;

Rekurzív lekérdezések

- Datalog rekurzió segít megérteni az SQL-99 szabványban bevezetett **rekurzív lekérdezések WITH RECURSIVE** záradékát.
- A BSc-n csak **MONOTON** rekurziót vesszük, vagyis nem használjuk nem-monoton különbség műveletet, nincs csoportosítás-aggregálás (ugyanis az olyan lekérdezések, amelyek nem-monotonok, megengedik a negációt és aggregálást az olyan különös hatással van a rekurzióra, ezt csak MSc kurzusokon vesszük).
- Gyakorlaton a rekurzív **Eljut-feladatnak** az Oracle gépes-megoldásait is megnézzük, ami nem lesz majd vizsgán, csak a gyakorlaton próbáljuk ki.

Oracle megoldások: with utasítással

- Az Oracle SQL a WITH RECURSIVE utasítást (UNION) nem támogatja, ott másképpen oldották meg WITH utasítással (Oracle 11gR2 verziótól használható)

WITH eljut (honnan, hova) as

(select honnan, hova from jaratok

UNION ALL

select jaratok.honnan, eljut.hova

from jaratok, eljut

where jaratok.hova=eljut.honnan

)

SEARCH DEPTH FIRST BY honnan SET SORTING

CYCLE honnan SET is_cycle TO 1 DEFAULT 0

select distinct honnan, hova from eljut order by honnan;

Oracle megoldások: connect by

- Oracle sokkal korábban vezette be a hierarchikus lekérdezéseket, és ezt bővítette ki a rekurzióra is:

```
SELECT DISTINCT hova FROM jaratok
WHERE HOVA <> 'DAL'
  START WITH honnan = 'DAL'
  CONNECT BY NOCYCLE PRIOR hova = honnan;
```
- Oracle-ben további hasznos függvények is használhatók:

```
SELECT LPAD(' ', 4*level) || honnan, hova,
       level-1 Atszallasok,
       sys_connect_by_path(honnan||'->'||hova, '/'),
       connect_by_isleaf, connect_by_iscycle
  FROM jaratok
 WHERE START WITH honnan = 'SF'
   CONNECT BY NOCYCLE PRIOR hova = honnan;
```

Kérdés/Válasz

- Köszönöm a figyelmet! Kérdés/Válasz?
- Vizsgára csak az Eljut feladatot kell ismerni és az SQL-99 szabvány WITH RECURSION utasítását, de csak papíron. Feladat: Datalog program átírása SELECT-re WITH RECURSION záradékkal (SQL-99)
- Házi feladat: Gyakorlás Oracle Példatár 3.fejezete tartalmaz hierarchikus lekérdezésekre feladatokat:
<http://people.inf.elte.hu/sila/eduAB/Feladatok.pdf>
- Oracle gépes-megoldások nem lesznek a vizsgán, csak a gyakorlaton próbáljuk ki: az Eljut-feladathoz a Jaratok táblát létrehozó script: create_jaratok_tabla.txt

SQL DDL: Táblák, megszorítások (constraints), triggerek, nézettáblák

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009



- 7.1.-7.4. Megszorítások
- 7.5.-7.6. Triggerek
- 8.1.-8-2. Nézettáblák
- 8.5.-8.6. Tárolt nézettáblák
- Megj.: 8.3.-8.4. Indexek (Adatbázisok-2 kurzuson lesznek)

Relációs lekérdező nyelvek

Adatbázisok-1 kurzuson háromféle nyelvet tanulmányozunk:

- **Relációs algebra:** procedurális, algebrai megközelítés, megadjuk a kiértékelési tervet, többféle lehetőség összevetése, hatékonysági vizsgálatok.
- **Datalog:** deklaratív, logika alapú megközelítés, amely az összetett lekérdezéseknel, például rekurzióval segítség.
- **SQL szabvány relációs lekérdező nyelv:** gyakorlatban, SQL története, szabványok, az SQL fő komponensei: SQL DDL (sémaleíró nyelv) **milyen objektumok lehetnek?** DML (adatkezelő és lekérdező nyelv), tranzakció-kezelés, DCL (vezérlő nyelv) **milyen jogosultságok, privilegiumok?** SQL-2003/PSM, ezt a gyakorlatban: PL/SQL (Oracle)

--- Ma az SQL DDL sémaleíró nyelv utasításait nézzük át!

Ismétlés: Relációsémák definiálása

- Az SQL tartalmaz **adatleíró részt (DDL)**, az adatbázis **objektumainak** a leírására és megváltoztatására.
Objektumok leíró parancsa a **CREATE** utasítás.
 - CREATE – létrehozni, az objektumok leíró parancsa
 - DROP – eldobni, a teljes leírást és minden az ehhez kapcsolódott hozzáférhetetlennek válik
 - ALTER – módosítani a leírást
- A relációt az SQL-ben táblának (TABLE) nevezik, az SQL alapvetően háromféle táblát kezel:
 - Alaptáblák [CREATE | ALTER | DROP] TABLE
 - Nézetttáblák [CREATE [OR REPLACE] | DROP] VIEW
 - Átmeneti munkatáblák (WITH záradéka a SELECT-nek)
- **Alaptáblák** megadása: **CREATE TABLE**

Megszorítások (áttekintés) => (1)

(1) Kulcsok és idegen kulcsok megadása

- A hivatkozási épség fenntartása
- Megszorítások ellenőrzésének késleltetése

(2) Értékekre vonatkozó feltételek

- NOT NULL feltételek
- Attribútumra vonatkozó CHECK feltételek

(3) Sorokra vonatkozó megszorítások

- Sorra vonatkozó CHECK feltételek

(4) Megszorítások módosítása (constraints)

(5) Önálló megszorítások (assertions)

(6) Triggerek (triggers)

Idegen kulcsok megadása

- Az első előadáson a táblák létrehozásánál vettünk kiegészítő lehetőségeket **Kulcs és idegen kulcs (foreign key) hivatkozási épség megadása**
- Az egyik tábla egyik oszlopában szereplő értékeknek szerepelnie kell egy másik tábla bizonyos attribútumának az értékei között.
- A **hivatkozott attribútumoknak** a másik táblában kulcsnak kell lennie! (PRIMARY KEY vagy UNIQUE)
- Példa: **Felszolgál(söröző, sör, ár)** táblára megszorítás, hogy a sör oszlopában szereplő értékek szerepeljenek a **Sörök(név, gyártó)** táblában a név oszlop értékei között.

Idegen kulcs megadása: attribútumként

REFERENCES kulcsszó használatának két lehetősége:
attribútumként vagy sémaelemként lehet megadni.

1.) Attribútumonként (egy attribútumból álló kulcsra)

Példa:

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

```
CREATE TABLE Felszolgál (
    söröző CHAR(20),
    sör   CHAR(20) REFERENCES Sörök(név),
    ár     REAL );
```

Idegen kulcs megadása: sémaelemként

2.) Sémaelemként (egy vagy több attr.-ból álló kulcsra)

FOREIGN KEY (attribútum lista)

REFERENCES relációnév (attribútum lista)

Példa:

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

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

Hivatkozási épség, idegen kulcs megszorítások megőrzése

- Példa: R = Felszolgál, S = Sörök.
- Egy idegen kulcs megszorítás R relációról S relációra kétféleképpen sérülhet:
 1. Egy R -be történő beszúrásnál vagy R -ben történő módosításnál S -ben nem szereplő értéket adunk meg.
 2. Egy S -beli törlés vagy módosítás „lógó” sorokat eredményez R -ben.

Hogyan védekezzünk? --- (1)

- Példa: $R = \text{Felszolgál}$, $S = \text{Sörök}$.
- Nem engedjük, hogy **Felszolgál** táblába a **Sörök** táblában nem szereplő sört szúrjanak be vagy **Sörök** táblában nem szereplő sörre módosítsák (nincs választási lehetőségünk, a rendszer visszautasítja a megszorítást sértő utasítást)
- A **Sörök** táblából való törlés vagy módosítás, ami a **Felszolgál** tábla sorait is érintheti (mert sérül az idegen kulcs megszorítás) 3-féle módon kezelhető (lásd köv.oldal)

Hogyan védekezzünk? --- (2)

1. **Alapértelmezés (Default)** : a rendszer nem hajtja végre a törlést.
2. **Továbbgyűrűzés (Cascade)**: 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 sört = '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 (
    söröző CHAR(20) ,
    sör          CHAR(20) ,
    ár           REAL ,
    FOREIGN KEY(sör)
        REFERENCES Sörök(név)
        ON DELETE SET NULL
        ON UPDATE CASCADE
) ;
```

Megszorítások ellenőrzésének késleltetése

- Körkörös megszorítások miatt szükség lehet arra, hogy a megszorításokat ne ellenőrizze, amíg az egész tranzakció be nem fejeződött.
- Bármelyik megszorítás deklarálható **DEFERRABLE** (késleltethető) vagy **NOT DEFERRABLE**-ként (vagyis minden adatbázis módosításkor a megszorítás közvetlenül utána ellenőrzésre kerül). DEFERRABLE-ként deklaráljuk, akkor lehetőségünk van arra, hogy a megszorítás ellenőrzésével várjon a rendszer a tranzakció végéig.
- Ha egy megszorítás késleltethető, akkor lehet
 - **INITIALLY DEFERRED** (az ellenőrzés a tranzakció jóváhagyásáig késleltetve lesz) vagy
 - **INITIALLY IMMEDIATE** (minden utasítás után ellenőrzi)

Megszorítások (áttekintés) => (2)

(1) Kulcsok és idegen kulcsok

- A hivatkozási épség fenntartása
- Megszorítások ellenőrzésének késleltetése

(2) Értékekre vonatkozó feltételek

- NOT NULL feltételek
- Attribútumra vonatkozó CHECK feltételek

(3) Sorokra vonatkozó megszorítások

- Sorra vonatkozó CHECK feltételek

(4) Megszorítások módosítása (constraints)

(5) Önálló megszorítások (assertions)

(6) Triggerek (triggers)

Értékekre vonatkozó feltételek

- Egy adott oszlop értékeire vonatkozóan adhatunk meg megszorításokat.
- A CREATE TABLE utasításban az attribútum deklarációban **NOT NULL** kulcsszóval
- az attribútum deklarációban **CHECK(<feltétel>)**
A **feltétel**, mint a WHERE feltétel, alkérdés is használható. A feltételben csak az adott attribútum neve szerepelhet, más attribútumok (más relációk attribútumai is) csak alkérdésben szerepelhetnek.

Példa: értékekre vonatkozó feltétel

```
CREATE TABLE Felszolgál (
    söröző CHAR(20) NOT NULL,
    sör      CHAR(20) CHECK ( sör IN
        (SELECT név FROM Sörök)) ,
    ár       REAL CHECK ( ár <= 5.00 )
) ;
```

Mikor ellenőrzi?

- Érték-alapú ellenőrzést csak **beszúrásnál** és **módosításnál** hajt végre a rendszer.
- **Példa:** CHECK ($\text{ár} \leq 5.00$) a beszúrt vagy módosított sor értéke nagyobb 5, a rendszer nem hajtja végre az utasítást.
- **Példa:** CHECK ($\text{sör} \in (\text{SELECT } \text{név}$ FROM Sörök) , ha a Sörök táblából törlünk, ezt a feltételt nem ellenőrzi a rendszer.

Megszorítások (áttekintés) => (3)

(1) Kulcsok és idegen kulcsok

- A hivatkozási épség fenntartása
- Megszorítások ellenőrzésének késleltetése

(2) Értékekre vonatkozó feltételek

- NOT NULL feltételek
- Attribútumra vonatkozó CHECK feltételek

(3) Sorokra vonatkozó megszorítások

- Sorra vonatkozó CHECK feltételek

(4) Megszorítások módosítása (constraints)

(5) Önálló megszorítások (assertions)

(6) Triggerek (triggers)

Sorokra vonatkozó 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: sor-alapú megszorítások

- Csak Joe bárja nevű sörözőben lehetnek drágábbak a söröök 5 dollárnál:

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

Megszorítások (áttekintés) => (4)

(1) Kulcsok és idegen kulcsok

- A hivatkozási épség fenntartása
- Megszorítások ellenőrzésének késleltetése

(2) Értékekre vonatkozó feltételek

- NOT NULL feltételek
- Attribútumra vonatkozó CHECK feltételek

(3) Sorokra vonatkozó megszorítások

- Sorra vonatkozó CHECK feltételek

(4) Megszorítások módosítása (constraints)

(5) Önálló megszorítások (assertions)

(6) Triggerek (triggers)

Megszorítások elnevezése

- Nevet tudunk adni a megszorításoknak, amire később tudunk hivatkozni (könnyebben lehet később majd törlni, módosítani)

Tankönyv példái:

- név CHAR(30) **CONSTRAINT** NévKulcs
PRIMARY KEY,
- nem CHAR(1) **CONSTRAINT** FérfiVagyNő
CHECK (nem IN ('F', 'N')),
- **CONSTRAINT** Titulus
CHECK (nem = 'N' OR név NOT LIKE 'Ms.\%')

Megszorítások módosítása

Tankönyv példái:

- **ALTER TABLE FilmSzínész ADD CONSTRAINT NévKulcs PRIMARY KEY (név);**
- **ALTER TABLE FilmSzínész ADD CONSTRAINT FérfiVagyNő CHECK (nem IN ('F', 'N'));**
- **ALTER TABLE FilmSzínész ADD CONSTRAINT Titulus CHECK (nem = 'N' OR név NOT LIKE 'Ms.\%');**

Megszorítások (áttekintés) => (5)

(1) Kulcsok és idegen kulcsok

- A hivatkozási épség fenntartása
- Megszorítások ellenőrzésének késleltetése

(2) Értékekre vonatkozó feltételek

- NOT NULL feltételek
- Attribútumra vonatkozó CHECK feltételek

(3) Sorokra vonatkozó megszorítások

- Sorra vonatkozó CHECK feltételek

(4) Megszorítások módosítása (constraints)

(5) Önálló megszorítások (assertions)

(6) Triggerek (triggers)

Önálló megszorítások: Assertions

- SQL aktív elemek közül a leghatékonyabbak nincs hozzárendelve sem sorokhoz, sem azok komponenseihez, hanem **táblákhoz kötődnek.**
- Ezek is **az adatbázissémához tartoznak** a relációsémához és nézetekhez hasonlóan.
- **CREATE ASSERTION <név>**
CHECK (<feltétel>);
- A feltétel tetszőleges táblára és oszlopra hivatkozhat az adatbázissémából.

Példa: önálló megszorítások

- A **Felszolgál(söröző, sörtípus, ár)** táblában nem lehet olyan söröző, ahol a sörtípusok átlagára 5 dollárnál több

```
CREATE ASSERTION CsakOlcsó CHECK
(
    NOT EXISTS (
        SELECT söröző
        FROM Felszolgál
        GROUP BY söröző
        HAVING 5.00 < AVG(ár)
    )
);
```

(SELECT ..
olyan sörözők,
ahol a sörtípusok
átlagosan
drágábbak
5 dollárnál)

Példa: önálló megszorítások

- Az Sörvivő(név, cím, telefon) és Söröző(név, cím, engedély) táblákban nem lehet több bár, mint amennyi sörvívó van.

```
CREATE ASSERTION KevésBár CHECK (
    (SELECT COUNT(*) FROM Söröző) <=
    (SELECT COUNT(*) FROM Sörvívó)
) ;
```

Önálló 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 KevésBár megszorításra. Ugyanez igaz a **Sörivők** táblába történő beszúrásokra is.

Megszorítások (áttekintés) => (6)

(1) Kulcsok és idegen kulcsok

- A hivatkozási épség fenntartása
- Megszorítások ellenőrzésének késleltetése

(2) Értékekre vonatkozó feltételek

- NOT NULL feltételek
- Attribútumra vonatkozó CHECK feltételek

(3) Sorokra vonatkozó megszorítások

- Sorra vonatkozó CHECK feltételek

(4) Megszorítások módosítása (constraints)

(5) Önálló megszorítások (assertions)

(6) Triggerek (triggers)

Megszorítások v.s. triggerek

- **Aktív elemek** – olyan kifejezés vagy utasítás, amit egyszer eltároltunk az adatbázisban és azt várjuk tőle, hogy a megfelelő pillanatban lefusson (pl. adatok helyességének ellenőrzése)
- **A megszorítás** adatelemek közötti kapcsolat, amelyet az adatbázis-kezelő rendszernek fent kell tartania.
- **Triggerek** olyankor hajtódnak végre, amikor valamilyen megadott esemény történik, mint például sorok beszúrása egy táblába.

Miért hasznosak a triggerek?

- Az önálló megszorításokkal (assertions) sok minden le tudunk írni, az ellenőrzésük azonban gondot jelenthet.
- Az attribútumokra és sorokra vonatkozó 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-Tevékenység szabályok

- A triggereket esetenként *ECA* szabályoknak (*event-condition-action*) **esemény-feltétel-tevékenység** szabályoknak is nevezik.
- **Esemény:** általában valamilyen módosítás a adatbázisban, INSERT, DELETE, UPDATE.
- **Mikor?:** BEFORE, AFTER, INSTEAD
- **Mit?:** OLD ROW, NEW ROW FOR EACH ROW
OLD/NEW TABLE FOR EACH STATEMENT
- **Feltétel :** SQL igaz-hamis-ismeretlen feltétel.
- **Tevékenység :** SQL utasítás, BEGIN..END,
SQL/PSM tárolt eljárás

Példa triggerre

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

Példa: trigger definíció

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

Triggerek --- 1

- A *triggerek*, amelyeket szokás **esemény-feltételek** szabályoknak is nevezni, az eddigi megszorításuktól három dologban térnek el:
- A triggereket a rendszer csak akkor ellenőrzi, ha bizonyos események bekövetkeznek. A megengedett események általában egy adott relációra vonatkozó beszúrás, törlés, módosítás, vagy a tranzakció befejeződése.

Triggerek --- 2

- A kiváltó esemény azonnali megakadályozása helyett a trigger először egy *feltételt* vizsgál meg
- Ha a trigger feltétele teljesül, akkor a rendszer végrehajtja a triggerhez tartozó *tevékenységet*. Ez a művelet ezután megakadályozhatja a kiváltó esemény megtörténtét, vagy meg nem történtté teheti azt.

Tankönyv példája (7.5. ábra)

-- Nem engedi csökkenteni a gyártásirányítók nettó bevételét:

```
CREATE TRIGGER NetBevétTrigger  
AFTER UPDATE OF nettóBevétel ON GyártásIrányító  
REFERENCING  
    OLD ROW AS RégiSor,  
    NEW ROW AS ÚjSor  
FOR EACH ROW  
WHEN (RégiSor.nettóBevétel > ÚjSor.nettóBevétel)  
    UPDATE GyártásIrányító  
    SET nettóBevétel = RégiSor.nettóBevétel  
WHERE azonosító = ÚjSor.azonosító;
```

Tankönyv példája (7.6. ábra)

-- Az átlagos nettó bevétel megszorítása:

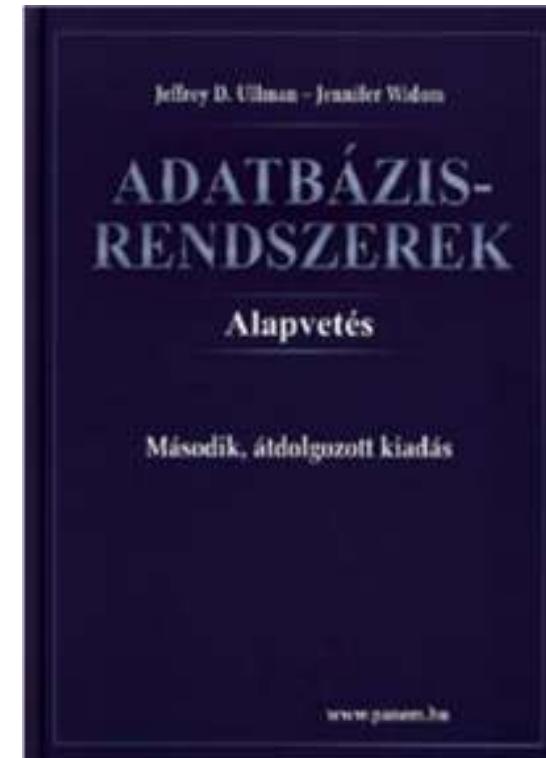
```
CREATE TRIGGER ÁtlagNetBevétTrigger  
AFTER UPDATE OF nettóBevétel ON GyártásIrányító  
REFERENCING  
    OLD TABLE AS RégiAdat,  
    NEW TABLE AS ÚjAdat  
FOR EACH STATEMENT  
WHEN (500000 > (SELECT AVG(nettóBevétel)  
                    FROM GyártásIrányító)  
    DELETE FROM GyártásIrányító  
    WHERE (név, cím, azonosító) IN ÚjAdat;  
INSERT INTO gyártásIrányító (SELECT...) ;
```

SQL DDL: nézettáblák(VIEW)

SQL DML: inline nézetek (SELECT)

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009

-
- 8.1.-8-2. Nézettáblák (View)
 - 8.5.-8.6. Tárolt nézettáblák
- Megj.: A kimaradó 8.3.-8.4. Indexek az Adatbázisok-2 kurzuson lesznek!



Nézettáblák

- Ez volt a Tankönyv 7.fejezete az integritási megszorításokról és a triggerekről.
- Ezután következik **a Tankönyv 8.fejezete** a nézettáblákról, és az adatok módosításáról a nézettáblákon keresztül.

Mik a nézettáblák?

- A nézettábla olyan reláció, amit tárolt táblák (vagyis alaptáblák) és más nézettáblák felhasználásával definiálunk.
- EMPLOYEES table

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_FRES	2400
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	1700
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	1700
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	900
104	Eduardo	Elisofon	EDRNST	900.421.4568	21-MAY-91	IT_PROG	600
105	Diana	Lorentz	DLORENTZ	900.421.4567	07-FEB-98	IT_PROG	420
106	Irene	Moussa	IMOURGEO	650.123.5234	18-NOV-99	ST_MAN	580
107	Trenor	Pax	TRAJS	650.121.8007	17-OCT-95	ST_CLERK	350
108	Curtis	Dienow	CDAVIES	860.121.2074	21-JAN-97	ST_CLERK	310
109	Randall	Matos	RMATCG	860.121.2074	15-MAR-90	ST_CLERK	290
EMPLOYEE_ID	LAST_NAME		SALARY	JUL-95	ST_CLERK		
149	Zlotkey		10500	JAN-00	SA_MAN	1050	
174	Abel		11000	MAY-96	SA REP	1100	
170	Taylor		0000	MAR-98	SA REP	860	
171	Parmeroy	Ogert	PARMERO	011.44.1044.420200	24-MAY-99	SA REP	700
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	440
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MKT_MAN	1300
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MKT REP	600
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	1200
206	William	Gietz	WGIETZ	515.123.0101	07-JUN-94	AC_ACCOUNT	830

20 rows selected.

A nézettáblák előnyei

Korlátozható az
adatok elérése

Bonyolult lekérdezések
egyszerűbben
kifejezhetők

Az
adatfüggetlenség
biztosítása

Ugyanazon adatokat
különböző
nézőpontokból
szemlélhetjük



Virtuális vagy materializált?

- Kétféle nézettábla létezik:
 - **Virtuális** = nem tárolódik az adatbázisban, csak a relációt megadó lekérdezés.
 - **Materializált** = kiszámítódik, majd tárolásra kerül.

Nézettáblák létrehozása és törlése

- Létrehozása:

```
CREATE [OR REPLACE] [FORCE | NOFORCE]  
[MATERIALIZED] VIEW <név>  
AS <lekérdezés>  
  
[WITH CHECK OPTION [CONSTRAINT constraint] ]  
[WITH READ ONLY [CONSTRAINT constraint] ] ;
```

- Alapesetben virtuális nézettábla jön létre.

- Nézettábla megszüntetése:

```
DROP VIEW <név>;
```

Példa: nézettábla létrehozása

- **Mit_ihat(név, sör)** nézettáblában a sörvík mellett azon söröket tároljuk, amelyeket legalább egy olyan sörozőben felszolgálnak, amelyet látogat:

```
CREATE VIEW Mit_ihat AS
    SELECT név, sör
    FROM Látogat, Felszolgál
    WHERE L.söröző = F.söröző;
```

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 (lásd módosítások, SQL DML)
- Példa lekérdezés:

```
SELECT sör FROM Mit_ihat  
WHERE név = 'Sally';
```

Módosítható nézettáblák

- Az SQL szabvány formálisan leírja, hogy mikor lehet egy nézettáblát módosítani és mikor nem, ezek a szabályok meglehetősen bonyolultak.
 - Ha a nézettábla definíciójában a SELECT után nem szerepel DISTINCT, további kikötések:
 - A WHERE záradékban R nem szerepelhethez egy alkérdezésben sem
 - A FROM záradékban csak R szerepelhet, az is csak egyszer és más reláció nem
 - A SELECT záradék listája olyan attribútumokat kell, hogy tartalmazzon, hogy az alaptáblát fel lehessen tölteni (vagyis kötelező a kulcsként vagy not null-nak deklarált oszlopok megadása)

Tankönyv példája: nézettáblára

Tk.8.1. Példa: Egy olyan nézettáblát szeretnénk, mely a Film(cím, év, hossz, színes, stúdióNév, producerAzon) reláció egy részét jelképezi, pontosabban a Paramount stúdió által gyártott filmek címét és gyártási évét

```
CREATE VIEW ParamountFilm AS  
    SELECT cím, év  
    FROM Film  
    WHERE stúdióNév = 'Paramount' ;
```

Nézeteken instead-of-triggerek

Tk. 8.8. Példa: Az előző nézettábla módosítható, és hogy az alaptáblába való beszúráskor a stúdióNév attribútum helyes értéke , 'Paramount' legyen, ezt biztosítja ez az **INSTEAD OF (helyette)** típusú trigger:

```
CREATE TRIGGER ParamountBeszúrás  
    INSTEAD OF INSERT ON ParamountFilm  
    REFERENCING NEW ROW AS ÚjSor  
    FOR EACH ROW  
    INSERT INTO Film(cím, év, stúdióNév)  
    VALUES(Újsor.cím, ÚjSor.év, 'Paramount');
```

Materializált (tárolt) nézettáblák

- Adattárházaknál használják (MSc kurzusok)
- **Probléma:** minden alkalommal, amikor az alaptáblák valamelyike változik, a materializált nézettábla frissítése is szükséges válhat.
 - Ez viszont néha túl költséges.
- **Megoldás:** Periodikus frissítése a materializált nézettábláknak, amelyek egyébként „nem aktuálisak”.

Példa nézetek használatára

- Képezzük osztályonként az összfizetést, vegyük ezen számok átlagát, és adjuk meg, hogy mely osztályokon nagyobb ennél az átlagnál az összfizetés.

```
CREATE OR REPLACE VIEW osztaly_osszfiz AS
SELECT onev, SUM(fizetes) ossz_fiz
FROM dolgozo d, osztaly o
WHERE d.oazon = o.oazon
GROUP BY onev;
```

```
CREATE OR REPLACE VIEW atlag_koltseg AS
SELECT SUM(ossz_fiz)/COUNT(*) atlag
FROM osztaly_osszfiz;
```

```
SELECT * FROM osztaly_osszfiz
WHERE ossz_fiz > (SELECT atlag FROM atlag_koltseg) ;
```

Példa nézetek helyett munkatáblák

- Ugyanez WITH átmeneti munkatáblával megadva:

```
WITH osztaly_osszfiz AS  
( SELECT onev, SUM(fizetes) ossz_fiz  
  FROM dolgozo d, osztaly o  
 WHERE d.oazon = o.oazon  
 GROUP BY onev),
```

```
      atlag_koltseg AS  
( SELECT SUM(ossz_fiz)/COUNT(*) atlag  
  FROM osztaly_osszfiz)
```

```
SELECT * FROM osztaly_osszfiz  
WHERE ossz_fiz > (SELECT atlag FROM atlag_koltseg);
```

Kérdés/Válasz

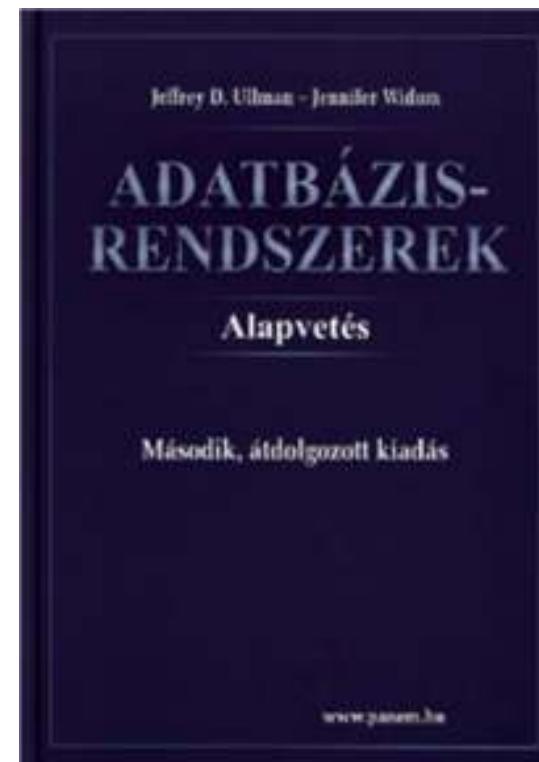
- Köszönöm a figyelmet! Kérdés/Válasz?
- Házi feladat: Gyakorlás az Oracle Példatár feladatai:
- DML-utasítások, tranzakciók (lásd 5EA)
 - DML-utasítások: insert, update, delete (Példatár 5.fej.)
 - Adatbázis-tranzakciók: commit, rollback, savepoint
- DDL-utasítások (lásd 6EA)
 - DDL-utasítások: adattáblák létrehozása, módosítása, integritási megszorítások (Példatár 5.fejezet folyt.) és
 - Nézettábla létrehozása és törlése, táblák tartalmának módosítása nézettáblákon keresztül (Példatár 6.fej.)

<http://people.inf.elte.hu/sila/eduAB/Feladatok.pdf>

Tankönyvben: SQL/PSM Gyakorlaton: Oracle PL/SQL

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiad, 2009

Motiváció: 10.2. Rekurzió SQL-ben,
az „Eljut”-feladat Oracle-ben
9.3. Az SQL és a befogadó nyelv
közötti felület (sormutatók)
9.4. SQL/PSM Sémában
tárolt függvények és eljárások



Az „Eljut feladat” SQL-99 szabványban

Tankönyv 10.2. fejezet példája (az ELJUT feladat)

- Jaratok(legitarsasag, honnan, hova, koltseg, indulas, erkezes) táblában repülőjáratok adatait tároljuk. Mely városokba tudunk eljutni Bp-ről?
- **WITH RECURSIVE** eljut(honnan, hova) AS
(SELECT honnan, hova FROM jaratok
UNION
 SELECT eljut.honnan, jaratok.hova
 FROM eljut, jaratok
 WHERE eljut.hova = jaratok.honnan)
SELECT hova **FROM** eljut **WHERE** honnan='Bp';

Oracle megoldások: WITH utasítással

- Az Oracle SQL a WITH RECURSIVE utasítást nem támogatja, ott más képpen oldották meg WITH utasítással (Oracle 11gR2 verziótól)
- `with eljut(honnan, hova) as
(select honnan, hova from jaratok
union all
select jaratok.honnan, eljut.hova
from jaratok, eljut
where jaratok.hova=eljut.honnan
)
SEARCH DEPTH FIRST BY honnan SET SORTING
CYCLE honnan SET is_cycle TO 1 DEFAULT 0
select distinct honnan, hova from eljut order by honnan;`

Oracle megoldások: connect by

- SELECT DISTINCT hova FROM jaratok
WHERE HOVA <> 'DAL'
START WITH honnan = 'DAL'
CONNECT BY NOCYCLE PRIOR hova = honnan;

- SELECT LPAD(' ', 4*level) || honnan, hova,
level-1 Atszallasok,
sys_connect_by_path(honnany||'->'||hova, '/'),
connect_by_isleaf, connect_by_iscycle
FROM jaratok
START WITH honnan = 'SF'
CONNECT BY NOCYCLE PRIOR hova = honnan;

Eljut feladat PL/SQL-ben ---1

- **Rek1.feladat:** Mely (x, y) várospárokra lehet egy vagy több átszállással eljutni x városból y városba?
- Ehhez hozzuk létre eljut(honnan,hova) táblát,

```
DROP TABLE eljut;
CREATE TABLE eljut(
    honnan VARCHAR2(10),
    hova VARCHAR2(10));
```
- Írunk egy olyan PL/SQL programot, ami feltölti az ELJUT táblát a sorait a járatok tábla alapján (ehhez ciklust szervezni, az insert több sor felvitele 2.alakja alkérdéssel járatok és eljut táblák alapján)

Eljut feladat PL/SQL-ben ---2

- Az ELJUT feladat megoldása Oracle PL/SQL-ben
- A ciklus során ellenőrizni kell, hogy addig hajtsuk végre a ciklust, amíg növekszik az eredmény (Számláló)
- **DECLARE** RegiSzamlalo Integer;
UjSzamlalo Integer;
- Deklarációs rész után **BEGIN ... END;** között az utasítások, először az eljut táblának kezdeti értéket adunk (a megvalósításnál az INSERT-nél figyelni, hogy ne legyenek ismétlődő sorok: select distinct)
delete from eljut;
insert into eljut (SELECT distinct honnan, hova
FROM jaratok);

Eljut feladat PL/SQL-ben ---3

- Szamlalo változóknak adunk kiindulási értéket:

RegiSzamlalo := 0;

`select count(*) into UjSzamlalo from eljut;`

- A ciklust addig kell végrehajtani, amíg növekszik az eredmény (Szamlalo) duplikátumokra figyelni!

LOOP

`insert into eljut (lásd a köv.oldalon...)`

`select count(*) into UjSzamlalo from eljut;`

`EXIT WHEN UjSzamlalo = RegiSzamlalo;`

`RegiSzamlalo := UjSzamlalo;`

`END LOOP;`

`commit;`

Eljut feladat PL/SQL-ben ---4

- Az eljut tábla növelése a ciklusban, figyelni kell a duplikátumokra, csak olyan várospárokat vegyük az eredményhez, ami még nem volt!

insert into eljut

```
(select distinct eljut.honnan, jaratok.hova  
from eljut, jaratok --- *from (lineáris rekurzió)  
where eljut.hova = jaratok.honnan  
and (eljut.honnan,jaratok.hova)  
      NOT IN (select * from eljut));
```

- Megjegyzés: PSM-ben a **nem-lineáris rekurzió** is megengedett: **from eljut e1, eljut e2 ---*from**-ban

Eljut feladat PL/SQL-ben ----5

- **Rek2.feladat:** Mely (x,y) város párokra hány darab átszállással és milyen költségekkel lehetséges egy vagy több átszállással eljutni x városból y városba?
- Ehhez készítsünk Eljut2(honnan, hova, atszallas, koltseg) táblát. Írunk egy olyan PL/SQL programot, ami feltölti az ELJUT táblát.
- **Rek3.feladat:** Tegyük fel, hogy nemcsak az érdekel, hogy el tudunk-e jutni az egyik városból a másikba, hanem az is, hogy utazásunk során az átszállások is ésszerűek legyenek, ami azt jelenti, hogy ha több járattal utazunk, akkor nézni kell átszálláskor az érkező járatnak legalább egy órával a rákövetkező indulás előtt meg kell érkeznie, és 6 óránál ne kelljen többet várnia.

Tankönyvben: SQL/PSM Gyakorlaton: Oracle PL/SQL

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiad, 2009

9.3. Az SQL és a befogadó nyelv
közötti felület (sormutatók)

9.4. SQL/PSM Sémában tárolt
függvények és eljárások



PL/SQL programozás (Gábor A.-Juhász I.)

Tankönyvtárban ingyenesen elérhető:

http://www.tankonyvtar.hu/hu/tartalom/tamop425/0046_plsql_programozas/adatok.html

SQL programnyelvi környezetben

- Milyen problémák merülnek fel, amikor egy alkalmazás részeként, programban használjuk az SQL utasításokat?
- 1.) **Osztott változók használata:** közös változók a nyelv és az SQL utasítás között (ott használható SQL utasításban, ahol kifejezés használható).
 - 2.) **A típuseltérés problémája:** Az SQL magját a relációs adatmodell képezi. Tábla – gyűjtemény, sorok multihalmaza, mint adattípus nem fordul elő a magasszintű nyelvekben. A lekérdezés eredménye hogyan használható fel? Megoldás: kurzorral, az eredmény soronkénti bejárása.

Háromféle programozási megközelítés

- 1.) **SQL kiterjesztése procedurális eszközökkel**, az adatbázis séma részeként tárolt kódrészekkel, tárolt modulokkal (pl. **PSM** = Persistent Stored Modules, **Oracle PL/SQL**).
- 2.) **Beágyazott SQL** (sajátos előzetes beágyazás EXEC SQL. - Előfordító alakítja át a befogadó gázdanyelvre/host language, pl. C)
- 3.) **Hívásszintű felület**: hagyományos nyelvben programozunk, függvénykönyvtárat használunk az adatbázishoz való hozzáféréshez (pl. **CLI** = call-level interface, JDBC, PHP/DB)

Legfontosabb PSM utasítások --- 1

1. Eljáráshívás: The call statement

```
CALL <procedure name>(<argument  
list>);
```

Use SQL/PSM statement CALL, with the name of the desired procedure and arguments.

➤ Example:

```
CALL JoeMenu('Moosedrool', 5.00);
```

Legfontosabb PSM utasítások --- 2

2. The return statement

Függvényhívás: Functions used in SQL expressions wherever a value of their return type is appropriate.

RETURN <expression> sets the return value of a function.

- Unlike C, etc., RETURN *does not* terminate function execution.

3. Változók deklarálása

DECLARE <name> <type>
used to declare local variables.

Legfontosabb PSM utasítások --- 3

4. Értékadás - Assignment statements

SET <variable> = <expression>;

- Example: SET b = 'Bud' ;

5. Statement group

BEGIN . . . END for groups of statements.

- Separate statements by semicolons.

6. Statement labels

- give a statement a label by prefixing a name and a colon.

7. SQL utasítások

- DELETE, UPDATE, INSERT, MERGE
- (de SELECT nem, azt később nézzük)

Lekérdezések használata a PSM-ben

- A típuseltérés problémája: Az SQL magját a relációs adatmodell képezi. Tábla – gyűjtemény, sorok multihalmaza, mint adattípus nem fordul elő a magasszintű nyelvekben. A lekérdezés eredménye hogyan használható fel?
- Hárrom esetet különböztetünk meg attól függően, hogy a SELECT FROM [WHERE stb] lekérdezés eredménye skalárértékkel, egyetlen sorral vagy egy listával (multihalmazzal) tér-e vissza.

Lekérdezések használata a PSM-ben

- SELECT eredményének használata:
 1. SELECT eredménye egy **skalárértékkel** tér vissza, **elemi kifejezésként** használhatjuk.
 2. SELECT **egyetlen sorral** tér vissza
SELECT e₁, ..., e_n INTO vált₁, ... vált_n
 - A végrehajtásnál visszatérő üzenethez az
 - SQL STATE változóban férhetünk hozzá.
 3. SELECT eredménye **több sorból álló tábla**, akkor az eredményt soronként bejárhatóvá tesszük, **kurzor** használatával.

Oracle PL/SQL

- ELTE Adatbázisok gyakorlaton: Oracle PL/SQL
- Oracle® Database PL/SQL Language Reference
- PL/SQL
- Procedurális nyelv
- Az SQL DML-t
kombinálja a
procedurális
nyelvi feldolgozással
(adatbázis + programozás)

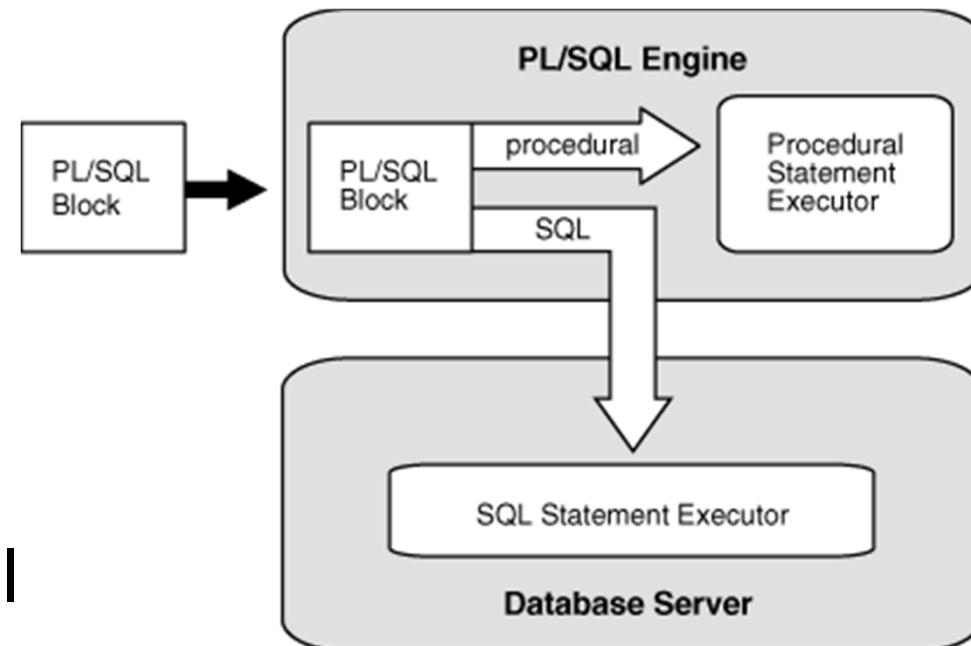


Figure 1-1 PL/SQL Engine

PL/SQL

- Blokkos szerkezet
- Kiegészítés az SQL-hez képest:
 - Változók
 - Típusok
 - Vezérlési szerkezetek
 - Alprogramok
 - Csomagok
 - Kurzorok, kurzorváltozók
 - Kivételkezelés
 - Objektumorientált eszközök

PL/SQL

- Egy PL/SQL blokk szerkezete:

[címke]

[DECLARE

 deklarációs utasítások]

BEGIN

 végrehajtandó utasítások

[EXCEPTION

 kivételkezelés]

END [név];

PL/SQL

- PL/SQL Ref.: Example 1-1 PL/SQL Block Structure

<< label >> (optional)

DECLARE -- Declarative part (optional)

-- Declarations of local types, variables, & subprograms

BEGIN -- Executable part (required)

-- Statements (which can use items declared in declarative part)

[EXCEPTION -- Exception-handling part (optional)

-- Exception handlers for exceptions (errors) raised in executable part]

END;

/

PL/SQL

- Példa: nem csinál semmit

```
BEGIN  
    null;  
END;
```

```
/
```

- Példa: töri a Dolgozo tábla tartalmát

```
BEGIN  
    delete from Dolgozo;  
END;
```

```
/
```

PL/SQL – Deklarációs rész

➢ Tartalma lehet

➢ Típus definíció

➢ Változó deklaráció

Név típus [[NOT NULL] {:= | DEFAULT} kifejezés];

Példák: belépési idő változó, illetve dolgozók száma változó és az alapértelmezett értéke 0.

belepesi_ido DATE;

dolg_szama NUMBER NOT NULL DEFAULT 0;

dolg_fizetes NUMBER NOT NULL := 1000;

PL/SQL – Deklarációs rész

- Tartalma lehet
 - Nevesített konstans deklaráció
 - Név CONSTANT típus [NOT NULL] {:= | DEFAULT} kifejezés;

Példa: fizetés konstans, melynek értéke 1000.

fizetes CONSTANT NUMBER := 1000;

- Kivétel deklaráció
- Kurzor definíció
- Alprogram definíció

PL/SQL - adattípusok

- Numerikus
 - NUMBER – ez így lebegőpontos
 - NUMBER(3) – ez így fixpontos
 - FLOAT – nem korlátozott lebegőpontos
 - INT, INTEGER, SMALLINT – korlátozott fixpontos
 - stb...

PL/SQL - adattípusok

- Karakteres
 - CHAR
 - VARCHAR2
 - NVARCHAR2
 - stb...

PL/SQL - adattípusok

- Logikai
 - BOOLEAN --- 3-értékű logika

PL/SQL - adattípusok

- Dátum
 - DATE
 - TIMESTAMP(p)
 - INTERVAL

PL/SQL - adattípusok

- A deklarációban a típus lehet
 - Skalár adattípus
 - Hivatkozási típus: %TYPE, %ROWTYPE
 - Változónév / rekordnév /
adatbázis_tábla_név.oszlopnév /
kurzorváltozó_név / kollekcionév /
objektumnév%TYPE
 - Adatbázis_táblanév /
kurzornév%ROWTYPE

PL/SQL - adattípusok

- PL/SQL Ref.: Example 2-24 Assigning Values

DECLARE -- You can assign initial values here

counter NUMBER := 0;

done BOOLEAN;

emp_rec employees%ROWTYPE;

BEGIN -- You can assign values here too

done := (counter > 100);

emp_rec.first_name := 'Antonio';

emp_rec.last_name := 'Ortiz';

END;

PL/SQL - adattípusok

- Rekord típus deklaráció
 - TYPE név IS RECORD (
mezőnév típus [[NOT NULL] {:= |
DEFAULT} kifejezés],
...);
 - Példa: telefonkönyv rekord

```
DECLARE
  TYPE telkonyv IS RECORD (
    szam NUMBER,
    nev VARCHAR2(20));
BEGIN
  NULL;
END;
```

PL/SQL - adattípusok

- Rekord deklaráció

telefonkonyv telkonyv;

- Rekord mezőjének elérése

telefonkonyv.nev;

PL/SQL - adattípusok

- Altípusok
 - Alaptípusok megszorítása
 - SUBTYPE név IS
alaptípus_név[(korlát)] [NOT
NULL];
 - Példa: beépített altípus az INTEGER

```
SUBTYPE INTEGER IS NUMBER(38,0);
```

PL/SQL - adattípusok

- Tömbök

- `TYPE típusnév IS VARRAY(n) OF <elemek típusa>;`

```
TYPE szamtomb IS VARRAY(10) OF  
NUMBER;
```

PL/SQL - adattípusok

- Típuskonverziók
 - Implicit a skalártípusok között
 - Explicit a beépített függvények használatával
 - TO_DATE
 - TO_NUMBER
 - TO_CHAR

PL/SQL – Kiírás a konzolra

- A PL/SQL nem tartalmaz I/O utasításokat.
- A DBMS_OUTPUT csomag segítségével üzenetet helyezhetünk el egy belső pufferbe.
- PUT_LINE eljárás üzenetet ír a pufferbe
- A puffer tartalmát a SET SERVEROUTPUT ON utasítással jeleníthetjük meg a képernyőn
- Példa: Hello World!

```
SET SERVEROUTPUT ON
BEGIN
    DBMS_OUTPUT.PUT_LINE('Hello World!');
END;
/
```

PL/SQL – Utasítások

- Üres
 - NULL;
- Értékkadó
 - X := 0;
- Ugró
 - GOTO címke;

PL/SQL – Utasítások

- Elágazás
 - IF
 - CASE
- Ciklusok
 - Végtelen
 - WHILE
 - FOR
 - Kurzor FOR (később)
- SQL utasítások

PL/SQL – IF utasítás

- Szintaxis:

IF (feltétel)

 THEN utasítás [utasítás] ...

 [ELSIF (feltétel)]

 THEN utasítás [utasítás] ...] ...

 [ELSE utasítás [utasítás] ...]

END IF;

PL/SQL – IF utasítás

```
SET SERVEROUTPUT ON
DECLARE
    a number(3) := 100;
BEGIN
    IF ( a = 10 ) THEN
        dbms_output.put_line('Value of a is 10' );
    ELSIF ( a = 20 ) THEN
        dbms_output.put_line('Value of a is 20' );
    ELSIF ( a = 30 ) THEN
        dbms_output.put_line('Value of a is 30' );
    ELSE
        dbms_output.put_line('None of the values is matching');
    END IF;
    dbms_output.put_line('Exact value of a is: '|| a );
END;
/
```

PL/SQL – CASE utasítás

- Szintaxis:

CASE kifejezés

WHEN érték1 THEN utasítás1;

...

ELSE utasítás

END CASE;

PL/SQL – CASE utasítás

```
SET SERVEROUTPUT ON
DECLARE
    grade char(1) := 'A';
BEGIN
    CASE grade
        when 'A' then dbms_output.put_line('Excellent');
        when 'B' then dbms_output.put_line('Very good');
        when 'C' then dbms_output.put_line('Well done');
        when 'D' then dbms_output.put_line('You passed');
        when 'F' then dbms_output.put_line('Better try
again');
        else dbms_output.put_line('No such grade');
    END CASE;
END;
```

PL/SQL – LOOP utasítás

- Végtelen ciklus
- Szintaxis:

```
LOOP  
utasítás(ok);  
END LOOP;
```

- EXIT-re lép ki
 - Ehelyett használható EXIT WHEN (feltétel) is

PL/SQL – LOOP utasítás

➤ SET SERVEROUTPUT ON

DECLARE

 x number := 10;

BEGIN

 LOOP

 dbms_output.put_line(x);

 x := x + 10;

 IF x > 50 THEN

 exit; -- itt lep majd ki

 END IF;

 END LOOP;

 dbms_output.put_line('After Exit x is: ' || x);

END;

/

PL/SQL – WHILE utasítás

- Előtesztelős ciklus
- Szintaxis:

```
WHILE feltétel LOOP  
utasítás(ok);  
END LOOP;
```

PL/SQL – WHILE utasítás

```
SET SERVEROUTPUT ON
DECLARE
    a number(2) := 10;
BEGIN
    WHILE a < 20 LOOP
        dbms_output.put_line('value of a: ' || a);
        a := a + 1;
    END LOOP;
END;
```

PL/SQL – FOR utasítás

- Számlálós ciklus
- Szintaxis:

FOR számláló IN [REVERSE] kezdőérték ..
Végérték LOOP
utasítás(ok);
END LOOP;

PL/SQL – FOR utasítás

```
SET SERVEROUTPUT ON
DECLARE
    a number(2);
BEGIN
    FOR a in 10 .. 20 LOOP
        dbms_output.put_line('value of a: ' || a);
    END LOOP;
END;
```

SQL utasítások PL/SQL-ben

- Nem használható SELECT, csak spec.esetben
 - amikor egy sort ad vissza kiegészül egy INTO (ill. ált. BULK COLLECT INTO) utasításrésszel
- DML utasítások: INSERT, DELETE, UPDATE
 - kiegészülnek egy RETURNING utasításrésszel, segítségével az érintett sorok alapján számított értéket kaphatunk meg
- MERGE
 - „UPsert” funkcionalitás
- COMMIT, ROLLBACK, SAVEPOINT

SQL utasítások PL/SQL-ben

- SELECT értékének kiválasztása egy változóba
 - SELECT select_kifejezés INTO változónév
FROM táblanév;
- Példa: King adatainak tárolása a dolg változóban:
 - DECLARE
dolg dolgozo%ROWTYPE;
BEGIN
SELECT * INTO dolg
FROM dolgozo
WHERE dnev='KING';
END;

SQL utasítások PL/SQL-ben

- PL/SQL Ref: Example 2-25 SELECT INTO
DECLARE

```
bonus NUMBER(8,2);
BEGIN
  SELECT salary * 0.10 INTO bonus
  FROM employees
  WHERE employee_id = 100;
  DBMS_OUTPUT.PUT_LINE('bonus = ' || TO_CHAR(bonus));
END;
/
```

SQL utasítások PL/SQL-ben

- Törlés egy táblából

```
DELETE [FROM] tábla hivatkozás  
[WHERE feltétel]  
[returning utasításrész];
```

- A RETURNING formája

```
RETURNING  
egysoros select kifejezés[, ...]  
INTO {változó[, ...] | rekord};
```

SQL utasítások PL/SQL-ben

- Beszúrás egy táblába

```
INSERT INTO tábla [hivatkozás  
[(oszlop, ...)]  
VALUES  
{ (sql_kifejezés, [...]) | rekord }  
[returning utasításrész];
```

SQL utasítások PL/SQL-ben

- Táblában érték módosítása

```
UPDATE tábla hivatkozás  
SET oszlop = sql_kifejezés [, ...]  
[WHERE feltétel]  
[RETURNING utasításrész];
```

SQL utasítások PL/SQL-ben

```
DECLARE      -- PL/SQL Ref.: Example 6-1 Static SQL Statements
    emp_id    employees.employee_id%TYPE := 299;
    emp_first_name employees.first_name%TYPE := 'Bob';
    emp_last_name  employees.last_name%TYPE  := 'Henry';
BEGIN
    INSERT INTO employees (employee_id, first_name, last_name)
        VALUES (emp_id, emp_first_name, emp_last_name);
    UPDATE employees
        SET first_name = 'Robert'
        WHERE employee_id = emp_id;
    DELETE FROM employees
        WHERE employee_id = emp_id
        RETURNING first_name, last_name
        INTO emp_first_name, emp_last_name;
    COMMIT;
    DBMS_OUTPUT.PUT_LINE (emp_first_name || ' ' || emp_last_name);
END;
/
```

SQL utasítások PL/SQL-ben

- PL/SQL Ref.: Example 6-4 SQL%ROWCOUNT

```
DROP TABLE emp_temp;
CREATE TABLE emp_temp AS
    SELECT * FROM employees;
DECLARE
    mno NUMBER(6) := 122;
BEGIN
    DELETE FROM emp_temp WHERE manager_id = mno;
    DBMS_OUTPUT.PUT_LINE ('Number of employees
        deleted: ' || TO_CHAR(SQL%ROWCOUNT));
END;
/
```

PL/SQL - Kurzorok

- ~Iterátorok ahhoz, hogy adatbázisok sorait tudjuk kezelni PL/SQL-ben
- Két típus:
 - Implicit
 - Explicit

PL/SQL - Kurzorok

- Implicit kurzort az Oracle hoz létre, amennyiben SQL utasítást futtatunk és nincs hozzá explicit kurzor. Ilyen például a következő dián lévő FOR-ban SELECT, de lehet bármelyik DML utasítás is.
- Explicit kurzort mi tudunk létrehozni

PL/SQL - Kurzorok

- Implicit kurzor FOR ciklushoz

```
FOR ciklusváltozó_név IN (SELECT  
utasítás)
```

```
    LOOP  
        utasítások;  
    END LOOP;
```

- A ciklusváltozó kurzornév%ROWTYPE típusú lesz
- Megnyitja, betölti az aktív halmaz összes sorát, majd bezárja a kurzort

PL/SQL - Kurzorok

- Példa: az alábbi program kiírja minden dolgozó kódját és nevét PL/SQL-ből implicit kurzort használva.

```
SET SERVEROUTPUT ON
BEGIN
    FOR cikl IN (SELECT * FROM Dolgozo)
    LOOP
        dbms_output.put_line('Kod: ' ||
cikl.dkod || ', nev: ' || cikl.dnev);
    END LOOP;
END;
```

PL/SQL - Kurzorok

- Explicit kurzor létrehozás (a deklarációs részben):

```
CURSOR név [(paraméterlista)]
[RETURN sortípus]
IS
select utasítás;
```

- Ha nem adunk meg sortípust, akkor az Oracle kikövetkezteti a legtöbb esetben.

PL/SQL - Kurzorok

- Használathoz a kurzort meg kell nyitni. Erre az OPEN utasítás szolgál:

OPEN kurzornév [aktuális paraméterlista];

PL/SQL - Kurzorok

- A kurzorból a sorokat változókba kell betölteni, erre a **FETCH** utasítást használjuk:

```
FETCH {kurzornév | kurzorváltozó név}
      { INTO {rekordnév | változónév lista}
      |
      BULK COLLECT INTO kollekciótípus lista
      LIMIT sorok};
```

PL/SQL - Kurzorok

- Használat után a kurzort be kell zárni a CLOSE utasítással:

```
CLOSE {kurzornév | kurzorváltozó  
név};
```

PL/SQL - Kurzorok

- Példa: az alábbi program kiírja minden dolgozó kódját és nevét PL/SQL-ből explicit kurzort használva.

```
SET SERVEROUTPUT ON
DECLARE
    CURSOR curs IS SELECT * FROM Dolgozo;
    dolg Dolgozo%ROWTYPE;
BEGIN
    OPEN curs;
    LOOP
        FETCH curs into dolg;
        EXIT WHEN curs%NOTFOUND;
        dbms_output.put_line('Kod: ' ||
dolg.dkod || ', nev: ' || dolg.dnev);
    END LOOP;
END;
```

PL/SQL - Kurzorok

- Kurzorattribútumok
 - %FOUND
 - Megnyitás után, de az első betöltés előtt értéke NULL
 - Sikeres betöltés esetén értéke TRUE
 - Sikertelen betöltés esetén értéke FALSE
 - %NOTFOUND
 - A fentebbi negáltja

PL/SQL - Kurzorok

- Kurzorattribútumok
 - %ISOPEN
 - Amennyiben a kurzor meg van nyitva, értéke TRUE
 - Ellenkező esetben FALSE
 - %ROWCOUNT
 - Megnyitás után, de az első betöltés előtt értéke 0
 - minden sikeres betöltés esetén egyel nő az értéke

PL/SQL - Kurzorok

```
DECLARE    -- PL/SQL Ref.: Example 6-14 %ROWCOUNT Attribute
CURSOR c1 IS
    SELECT last_name FROM employees;
    name employees.last_name%TYPE;
BEGIN
    OPEN c1;
    LOOP
        FETCH c1 INTO name;
        EXIT WHEN c1%NOTFOUND OR c1%NOTFOUND IS NULL;
        DBMS_OUTPUT.PUT_LINE(c1%ROWCOUNT || '.' || name);
        IF c1%ROWCOUNT = 5 THEN
            DBMS_OUTPUT.PUT_LINE('--- Fetched 5th record ---');
        END IF;
    END LOOP;
    CLOSE c1;
END;
```

PL/SQL - Kurzorok

- Amennyiben UPDATE vagy DELETE utasítást szeretnénk használni explicit kurzorral hasznos lehet a WHERE CURRENT OF kurzornév utasítás, mellyel a kurzorba a legutóbbi FETCH által betöltött sor módosítható / törölhető, explicit zárolást eredményez.

PL/SQL - Kurzorok

- Példa: ha valakinek a foglalkozása manager és a fizetése még nem éri el az 5000-et, akkor állítsuk 5000-re. Csak a ciklust leírva:

LOOP

```
    FETCH curs INTO v_curs;
    EXIT WHEN curs%NOTFOUND;
    IF v_curs.foglalkozas='MANAGER'
AND v_curs.fizetes<5000 THEN
        UPDATE Dolgozo SET fizetes=5000
            WHERE CURRENT OF curs;
    END IF;
END LOOP;
```

PL/SQL - Kurzorok

```
DECLARE -- PL/SQL Ref.: Example 6-43 FOR UPDATE Cursor
    my_emp_id employees.employee_id%type;
    my_job_id employees.job_id%type;
    my_sal   employees.salary%type;
    CURSOR c1 IS
        SELECT employee_id, job_id, salary
        FROM employees FOR UPDATE;
BEGIN
    OPEN c1;
    LOOP
        FETCH c1 INTO my_emp_id, my_job_id, my_sal;
        IF my_job_id = 'SA_REP' THEN
            UPDATE employees
            SET salary = salary * 1.02
            WHERE CURRENT OF c1;
        END IF;
        EXIT WHEN c1%NOTFOUND;
    END LOOP;
    CLOSE c1;
END;
```

PL/SQL - Kurzorok

```
DECLARE --PL/SQL REF: Example 6-17 Parameters to Explicit Cursors
    emp_job    employees.job_id%TYPE := 'ST_CLERK';
    emp_salary employees.salary%TYPE := 3000;
    my_record  employees%ROWTYPE;
    CURSOR c1 (job VARCHAR2, max_wage NUMBER) IS
        SELECT * FROM employees
        WHERE job_id = job AND salary > max_wage;
    BEGIN
        OPEN c1(emp_job, emp_salary);
        LOOP
            FETCH c1 INTO my_record;
            EXIT WHEN c1%NOTFOUND;
            DBMS_OUTPUT.PUT_LINE
                ('Name = ' || my_record.last_name || ', salary = ' ||
                 my_record.salary || ', Job Id = ' || my_record.job_id );
        END LOOP;
    END;
/
```

PL/SQL - Kurzorok

- Kurzorváltozók
 - Nem kell fordítási időben ismerni a SELECT utasítást
 - Referencia típusú változó
 - Két lépéses létrehozás

1. REF CURSOR típus létrehozása

```
TYPE név IS REF CURSOR [RETURN
{{táblanév|kurzornév|kurzorváltozónév}
%ROWTYPE | rekordnév%TYPE |
rekordtípusnév|
kurzorreferenciatípus_név}];
```

PL/SQL - Kurzorok

1. Kurzorváltozó deklarálása

```
kurzorváltozó_neve  
ref_cursor_típus_neve;
```

PL/SQL - Kurzorok

- Kurzorreferencia típus lehet
 - Erős, amennyiben szerepel RETURN rész, ekkor a fordító majd ellenőrzi a később kapcsolt SELECT típuskompatibilitását.
 - Gyenge, melyhez bármilyen lekérdezés hozzákapcsolható.
- Megnyitására az OPEN . . . FOR utasítás használandó
OPEN kurzorváltozó_név FOR select utasítás;

PL/SQL - Alprogramok

- Deklarálhatóak
 - Blokkba ágyazva
 - Séma szinten
 - Csomagban

PL/SQL - Alprogramok

- A különbség az eljárás és a függvény között
 - Eljárás: direkt módon nem adnak vissza értéket, általában utasítások lefuttatása a cél (illetve logikailag egy egységbe tartozó utasítások egy helyen kezelése)
 - Függvény: visszaad egy értéket, általában arra használják, hogy kiszámítsanak valamit és azt visszaadják.

PL/SQL - Alprogramok

- Miért használjuk?
 - Átláthatóbbá teszi a kódot
 - Támogatja az újrafelhasználást
 - OOP-szerű

PL/SQL - Alprogramok

- Eljárás deklaráció

```
PROCEDURE eljárás_neve [(formális  
paraméterlista)]  
IS  
[deklarációs utasítások]  
BEGIN  
    végrehajtandó utasítások  
    [EXCEPTION kivételkezelő]  
END [név] ;
```

PL/SQL - Alprogramok

- Függvény deklaráció

```
FUNCTION függvény_neve [(formális  
paraméterlista)]  
RETURN típus IS  
[deklarációs utasítások]  
BEGIN  
    vérehajtandó utasítások  
    [EXCEPTION kivételkezelő]  
END [név] ;
```

PL/SQL - Alprogramok

- Példa: PL/SQL blokkban deklarált eljárás (koszon) és függvény(fix_szam), melyeket meghívunk a PL/SQL programból.

```
SET SERVEROUTPUT ON
DECLARE
    szam NUMBER(2);
    PROCEDURE koszon IS
        BEGIN
            dbms_output.put_line('Hello!');
        END koszon;
    function fix_szam RETURN NUMBER is
        BEGIN
            RETURN 10;
        END fix_szam;
    BEGIN
        koszon;
        szam := fix_szam;
        dbms_output.put_line(szam);
    END;
```

PL/SQL - Alprogramok

- Formális paraméterlista
név [{IN|OUT|IN OUT} [NO COPY]] típus
[{::=|DEFAULT} kifejezés];
 - IN: érték szerinti paraméterátadás
 - OUT: eredmény szerinti paraméterátadás
 - IN OUT: érték-eredmény szerinti paraméterátadás
 - NOCOPY: hint a fordítónak, hogy IN OUT esetben se másoljon értéket

PL/SQL - Alprogramok

- A paraméterösszerendelés történhet pozíció, és/vagy név alapján
 - Keverhetjük a kettő módszert, ekkor először a pozíció, utána a név szerintiek jönnek
- A lokális és csomagbeli nevek túlterhelhetőek
- Példa: különféle formális paraméterek használata. Az inp paramétert csak beolvassuk és értékét használjuk, az outp paraméterbe csak eredményt írunk, az inout paraméterből olvasunk is és írunk is bele. A példában pozíció szerinti paraméter-összerendelés történik.

PL/SQL - Alprogramok

```
➢ SET SERVEROUTPUT ON
DECLARE
    szam1 NUMBER(2) := 1;
    szam2 NUMBER(2);
    szam3 NUMBER(2) := 3;
    PROCEDURE muvelet (inp IN NUMBER, outp OUT
NUMBER, inout IN OUT NUMBER) IS
        BEGIN
            dbms_output.put_line('in parameter: '
|| inp || ', in out parameter: ' || inout);
            outp := inp + inout;
            inout := outp + inp;
        END muvelet;
    BEGIN
        muvelet(szam1, szam2, szam3);
        dbms_output.put_line('out parameter: ' ||
szam2 || ', in out parameter: ' || szam3);
    END;
```

PL/SQL - Alprogramok

- Hatáskör-, és élettartamkezelés
 - Statikus (egy név csak a deklarációjától kezdve él)
 - Dinamikus (alprogramok és blokkok esetén)

PL/SQL - Alprogramok

- Tárolt alprogramok
 - Van lehetőség arra, hogy létrehozzunk tárolt eljárást/függvényt
 - Ekkor azt az sqldeveloper eltárolja, később hívható lesz
 - Ez jó az újrafelhasználhatóság szempontjából

PL/SQL - Alprogramok

- Tárolt eljárás létrehozása

```
CREATE [OR REPLACE] PROCEDURE név
[formális paraméterlista]
IS
[deklarációs utasítások]
BEGIN
    végrehajtandó utasítások
    [EXCEPTION kivételek kezelő]
END [név];
```

PL/SQL - Alprogramok

- Tárolt függvény létrehozása

```
CREATE [OR REPLACE] FUNCTION név
[formális paraméterlista]
RETURN típus IS
[deklarációs utasítások]
BEGIN
    végrehajtandó utasítások
    [EXCEPTION kivételek kezelő]
END [név];
```

PL/SQL - Alprogramok

- Tárolt alprogram újrafordítása

```
ALTER {PROCEDURE | FUNCTION} név  
      COMPILE [DEBUG];
```

- Tárolt alprogram törlése

```
DROP {PROCEDURE | FUNCTION} név;
```

PL/SQL - Alprogramok

- Tárolt alprogram meghívása

```
CALL alprogram_név([aktuális  
paraméterlista])  
    [INTO változó];
```

PL/SQL - Kivételkezelés

- Futás közbeni hibák kezelésére
- Két fajta kivétel
 - Beépített
 - Felhasználó által definiált

PL/SQL - Kivételkezelés

- Kivételkezelés szintaxisa
 - [DECLARE deklarációs utasítások]
 - BEGIN végrehajtandó utasítások
 - EXCEPTION
 - WHEN exception1 THEN végrehajtandó utasítások exception1 esetén
 - WHEN exception2 THEN végrehajtandó utasítások exception1 esetén
 - WHEN exception3 THEN végrehajtandó utasítások exception1 esetén
 - ...
 - WHEN others THEN végrehajtandó utasítások egyéb esetben
 - END;

PL/SQL - Kivételkezelés

- Példa: Lekérdezzük a dolgozó nevét, amennyiben nincs ilyen kódú: 'Nincs ilyen dolgozo', egyéb hiba esetén a 'Hiba' hibaüzenetet adjuk.

```
SET SERVEROUTOUT ON
DECLARE
    kod Dolgozo.dkod%TYPE;
    nev Dolgozo.dnev%TYPE;
BEGIN
    SELECT dkod, dnev
    INTO kod, nev
    FROM Dolgozo
    WHERE dkod=kod;
    dbms_output.put_line(kod);
    dbms_output.put_line(nev);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        dbms_output.put_line('Nincs ilyen kodu
dolgozo');
    WHEN OTHERS THEN
        dbms_output.put_line('Hiba');
END;
```

PL/SQL - Kivételkezelés

- Saját kivétel definiálása

```
DECLARE  
    sajat_kivetel EXCEPTION;
```

- Kivétel hívás

```
RAISE kivétele_neve;
```

PL/SQL - Kivételkezelés

- Példa: amennyiben a bekért változó értéke negatív, dobunk egy negativ_ertek kivételt, majd kezeljük azt egy üzenettel. Ha nem történt hiba, kiírjuk a számot.

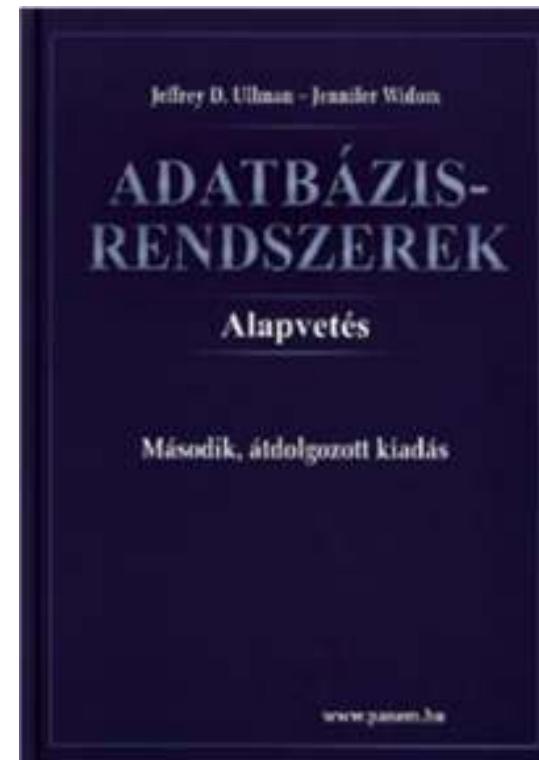
```
SET SERVEROUTOUT ON
DECLARE
    negativ_ertek EXCEPTION;
    szam NUMBER := &szam;
BEGIN
    IF (szam < 0) THEN
        RAISE negativ_ertek;
    END IF;
    dbms_output.put_line(szam);
EXCEPTION
    WHEN negativ_ertek THEN
        dbms_output.put_line('A szam nem lehet
negativ!');
    WHEN OTHERS THEN
        dbms_output.put_line('Hiba');
END;
```

Kiegészítés: SQL PSM

- Akit érdekel az Ullman.Widom Tankönyv és Ullman-Widom Stanford University előadások:
- [AB1 08ea SQL PSM.pdf](#)

Magas szintű adatbázismodellek

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009



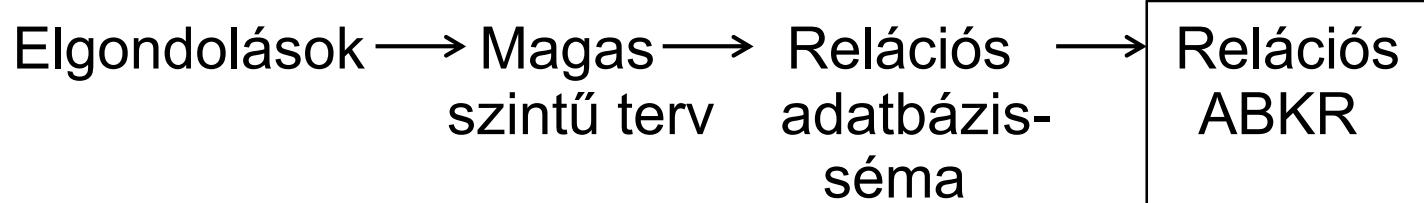
4.1.- 4.4. E/K-modell elemei

4.5.- 4.6. E/K-diagram átírása
relációs modellé

- Kiegészítő tananyag:
- 4.7.- 4.8. UML-diagram átírása relációs modellé
- (nincs a vizsgán, de a szakdolgozatban hasznos)

Magas szintű adatbázismodellek

- Vizsgáljuk meg azt a folyamatot, amikor egy új adatbázist létrehozunk, vegyük példaként a sörivós adatbázist.
- Az adatbázis-modellezés és implementálás eljárása



- Modellezés
 - komplex valós világ leképezése, absztrakció
- **Tervezési fázis:**
 - Milyen információkat kell tárolni?
 - Mely információelemek kapcsolódnak egymáshoz?
 - Milyen megszorításokat kell figyelembe venni? stb...

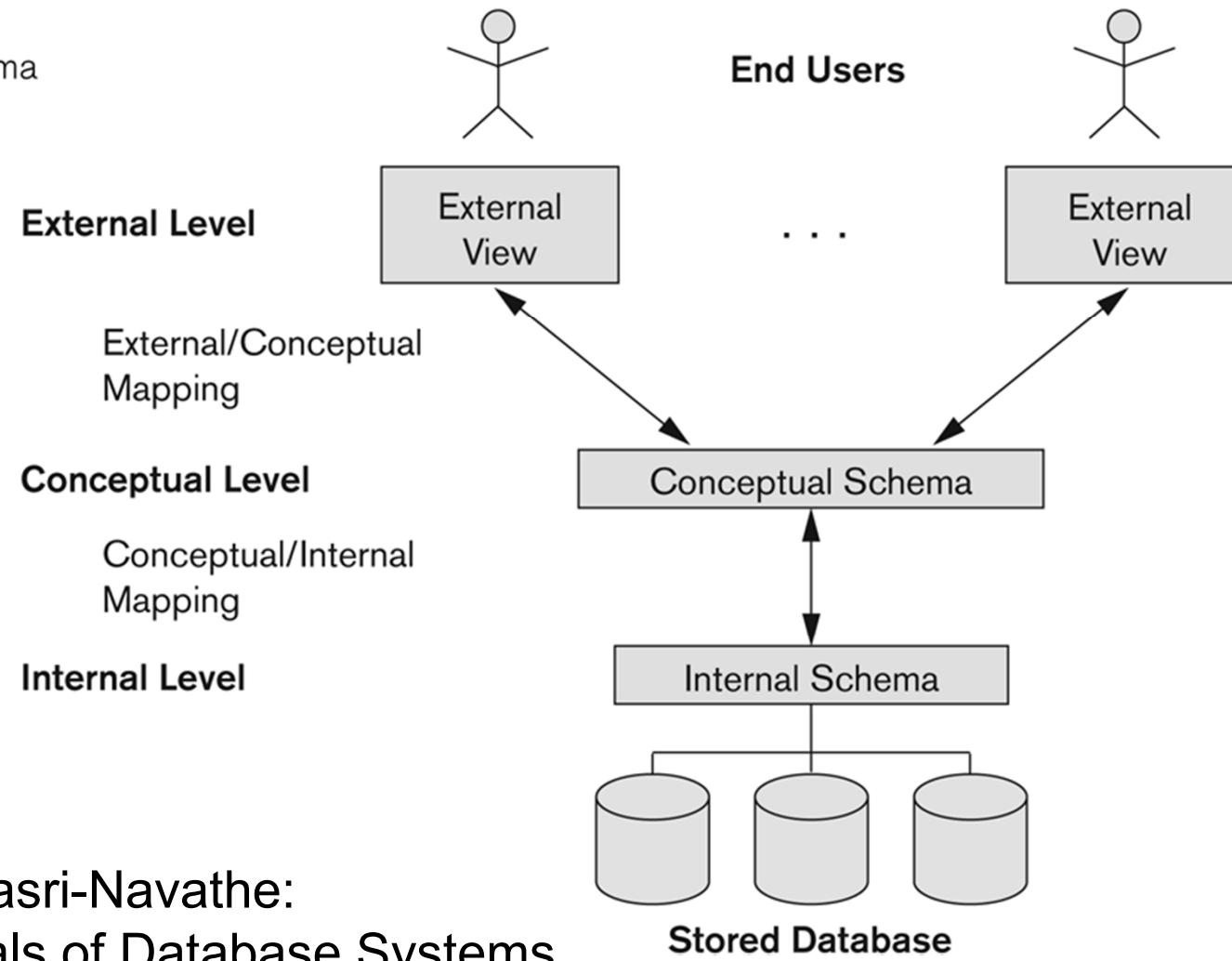
Az adatmodellek 3 szintje

- Hogyan látjuk az adatbázist?
- A 3 szintű ANSI/SPARC architektúra
 - Logikai (külső, a felhasználói szemléletnek megfelelő szinten, nézetek)
 - Fogalmi (conceptual) (absztrakt, szintetizálja az összes felhasználói szemléletet)
 - Fizikai (belső, az adatbázis valamelyen fizikai adatstruktúrában letárolva a háttértárolón)

Az adatmodellek 3 szintje

Figure 2.2

The three-schema architecture.



Forrás: Elmasri-Navathe:
Fundamentals of Database Systems

Az adatbázis-sématervezés lépései

(Ullman-Widom Tankönyv 4.fejezete)

- **Top-down:** a feldolgozandó információ elemzése,
- az információk közti kapcsolatok meghatározása
- az eredmény ábrázolása (**E/K diagram**),
- **adatbázisterv készítése** (transzformációs lépés),
- adatbázisterv finomítása (**összevonások**),

(Ullman-Widom Tankönyv 3.fejezete)

- **Bottom-up:** megszorítások modellezése,
(funkcionális, többértékű) **függőségek** meghatározása
- relációs adatbázissémák tervezése, **dekomponálás:**
(VM, FŐ felbontás), **normalizálás** (3NF, BCNF, 4NF),
- **Összevetés:** optimális adatbázisterv készítése,
- **Implementálás:** az **adatbázisterv megvalósítása.**

Egyed-kapcsolat modell elemei

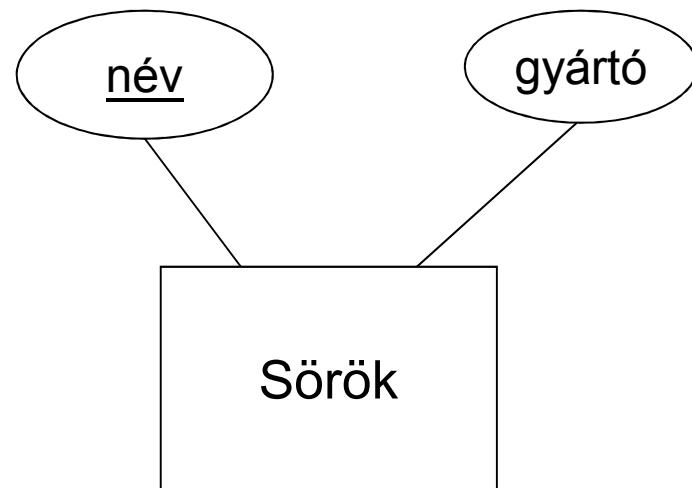
- Egyed-kapcsolat modell: E/K modell
(Entity-relationship ER) alapfogalmak:
- Egyedhalmazok (absztrakt objektumok osztálya)
 - Miről gyűjtünk adatokat?
 - Mit tegyünk egy gyűjteménybe? - hasonlóság
 - Hasonló egyedek összessége
- Attribútumok
 - Megfigyelhető tulajdonságok, megfigyelt értékek
 - Az egyedek tulajdonságait írják le
- Kapcsolatok
 - Más egyedhalmazokkal való kapcsolatuk

E/K modell elemei: Egyedhalmazok

- $E(A_1, \dots, A_n)$ egyedhalmaz **séma**:
 - E az egyedhalmaz neve,
 - A_1, \dots, A_n tulajdonságok,
 - $\text{DOM}(A_i)$ – lehetséges értékek halmaza.
 - például: tanár(név, tanszék).
- $E(A_1, \dots, A_n)$ sémájú egyedhalmaz **előfordulása**:
 - A konkrét egyedekből áll
 - $E = \{e_1, \dots, e_m\}$ egyedek (entitások) halmaza, ahol
 - $e_i(k) \in \text{DOM}(A_k)$,
 - semelyik két egyed sem egyezik meg minden attribútumban (léteznek és megkülönböztethetők)

E/K-diagram: Egyedhalmazok

- E/K diagram: séma-szinten grafikusan ábrázoljuk
- Egyedhalmazok: téglalap
- Tulajdonságok: ovális
 - az elsődleges kulcschoz tartozó tulajdonságokat aláhúzzuk.

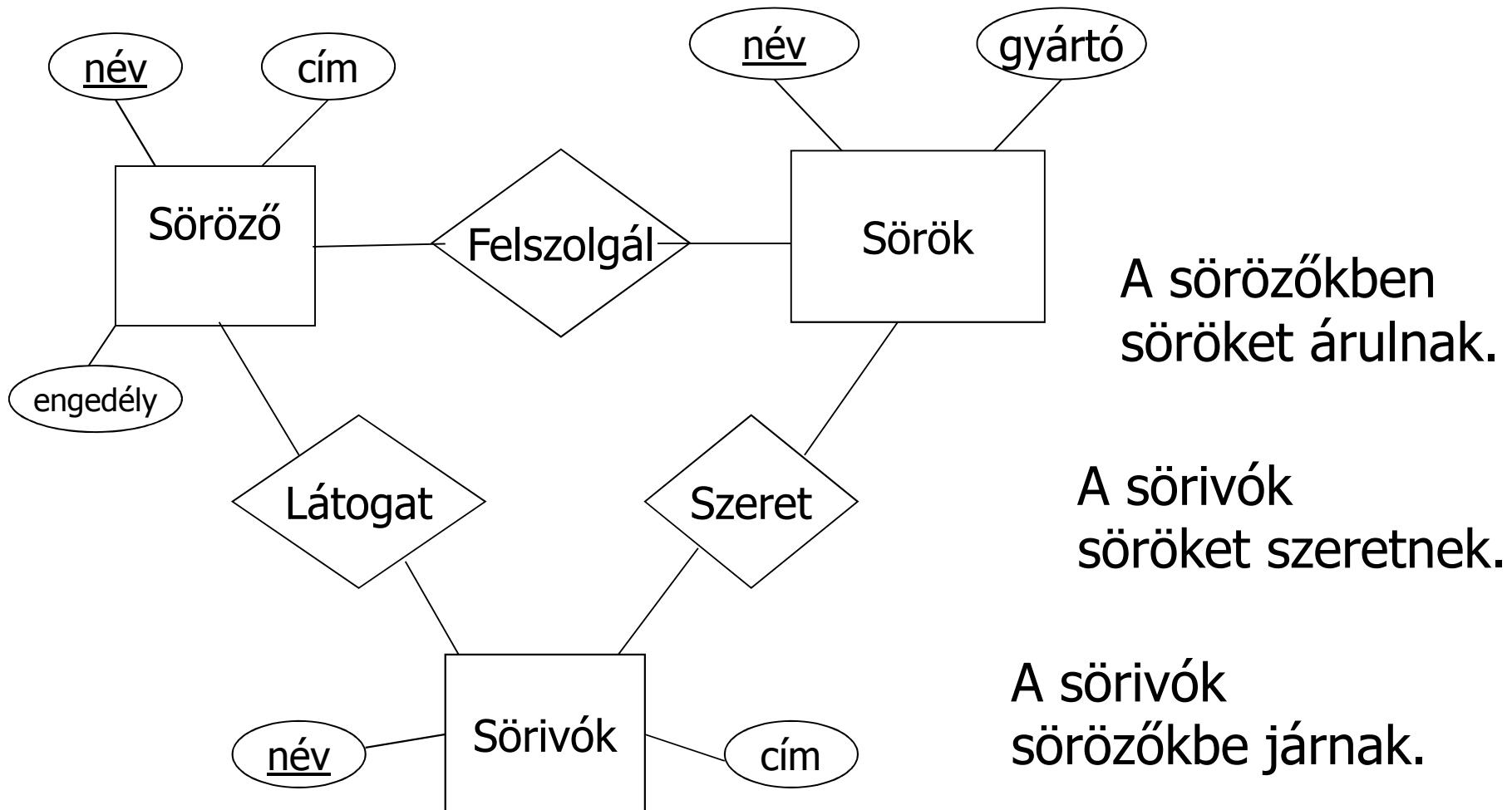


E/K modell elemei: Kapcsolatok

- K(E₁,...,E_p) a kapcsolat sémája,
 - K a kapcsolat neve,
 - E₁,...,E_p egyedhalmazok sémái,
 - p=2 bináris kapcsolat, p>2 többágú kapcsolat,
 - például: tanít(tanár,tárgy).
- K(E₁,...,E_p) sémájú kapcsolat előfordulása:
 - K = {(e₁,...,e_p)} egyed p-esek halmaza, ahol
 - e_i ∈ E_i,
 - a kapcsolat előfordulásaira tett megszorítások határozzák meg a kapcsolat típusát.

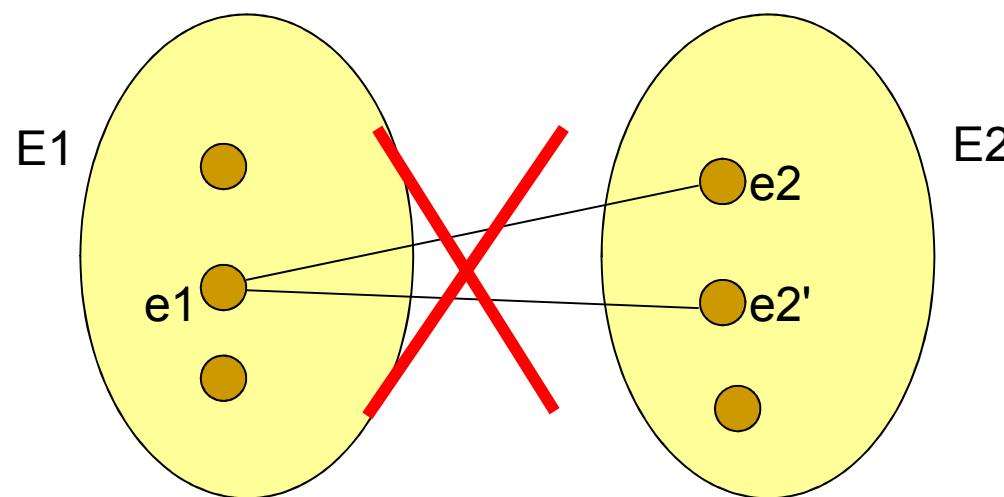
E/K-diagram: Kapcsolatok

- A kapcsolatok jele: **rombusz**



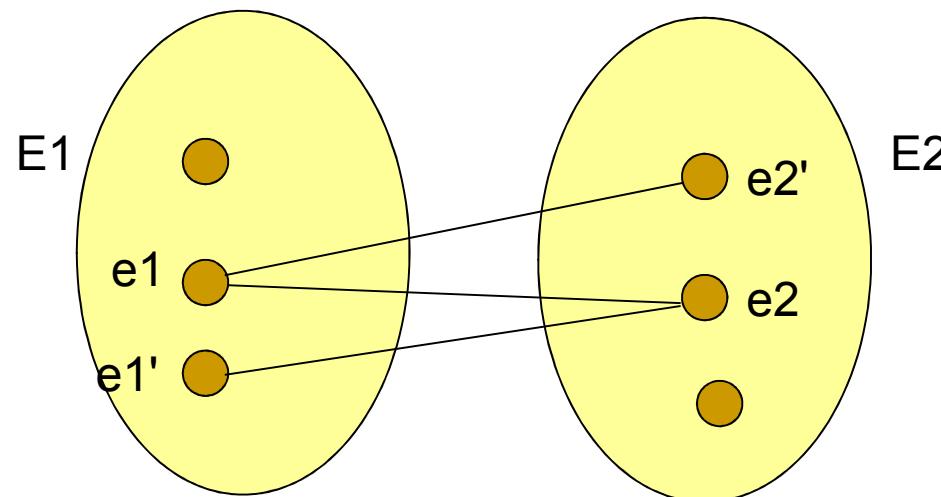
Kapcsolatok típusai

- $K(E1, E2)$ bináris kapcsolat, **sok-egy** (n:1)
 - $K \{(ei, ej)\}$ alakú előfordulásaiban nem szerepelhet egyszerre $(e1, e2)$ és $(e1, e2')$, ha $e2$ és $e2'$ különböznek,
 - másképpen: **K előfordulásaiban minden $E1$ -beli egyedhez legfeljebb 1 $E2$ -beli egyed tartozhat,**
 - például: született(név, ország).



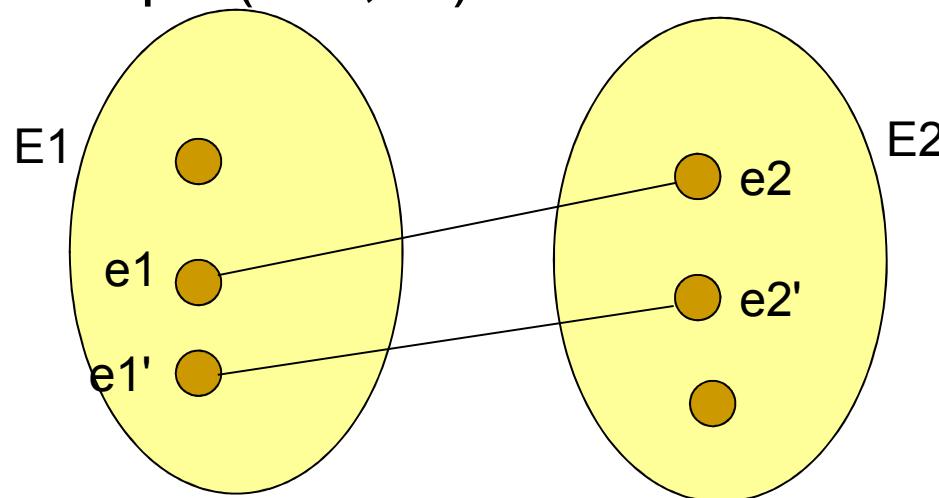
Kapcsolatok típusai

- $K(E_1, E_2)$ bináris kapcsolat, **sok-sok** ($n:m$),
 - $K \{(e_i, e_j)\}$ alakú előfordulásai nincsenek korlátozva,
 - előfordulhat (de nem kötelező, hogy előforduljon) az ábrán látható helyzet, vagyis minden **E1-beli egyedhez több E2-beli egyed tartozhat, és fordítva, minden E2-beli egyedhez több E1-beli egyed tartozhat,**
 - például: tanul(diák,nyelv).

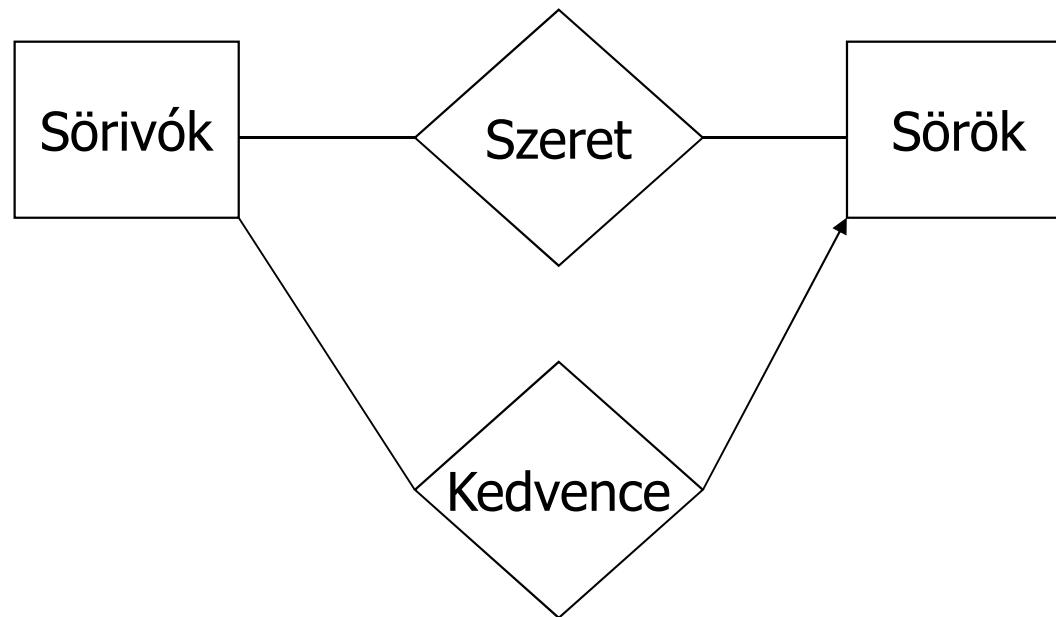


Kapcsolatok típusai

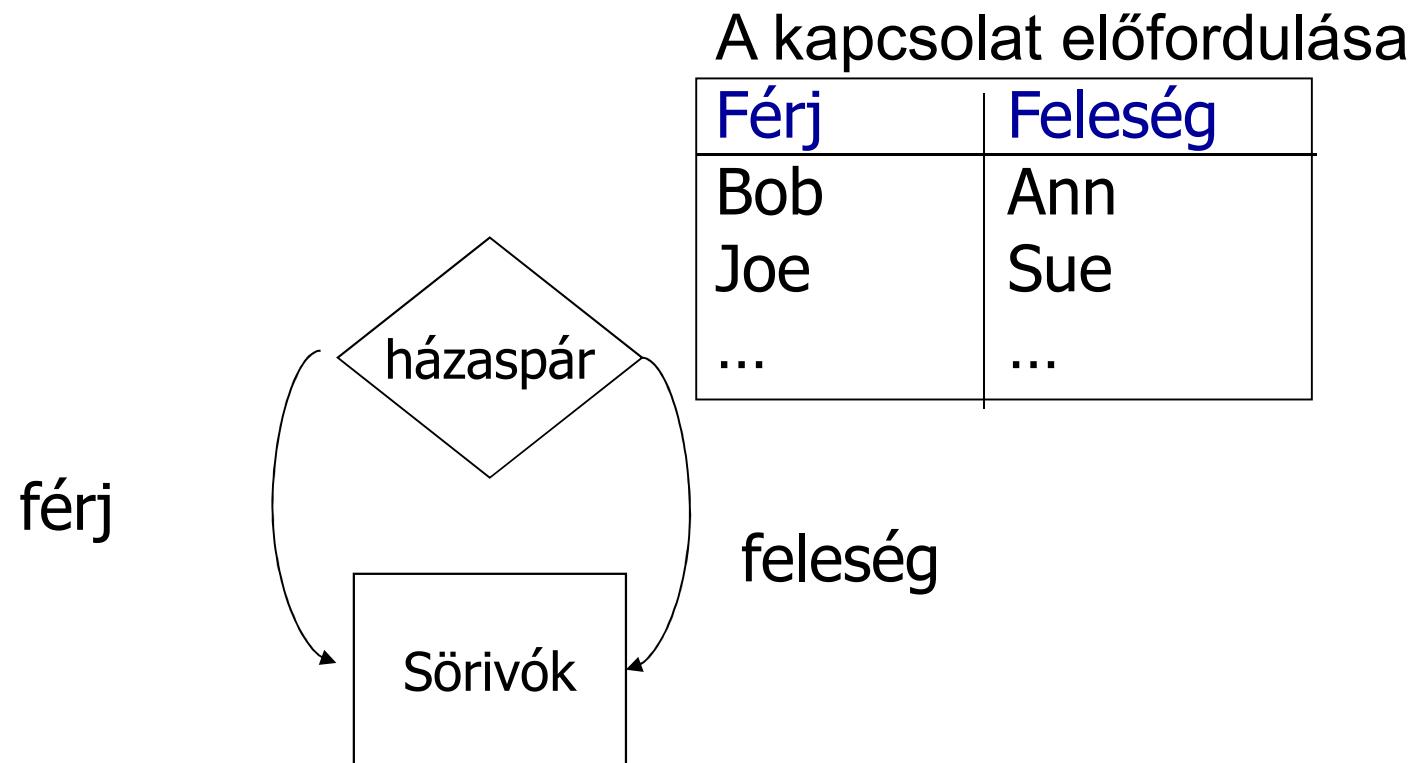
- $K(E1, E2)$ bináris kapcsolat, **egy-egy** (1:1),
 - $K \{(ei, ej)\}$ alakú előfordulásai egyszerre sok-egy és egy sok típusúak, vagyis **minden $E1$ -beli egyedhez legfeljebb egy $E2$ -beli egyed tartozhat, és fordítva, minden $E2$ -beli egyedhez legfeljebb egy $E1$ -beli egyed tartozhat,**
 - nem kötelezően szerepel minden egyed a kapcsolatban,
 - például: házaspár(férfi,nő).



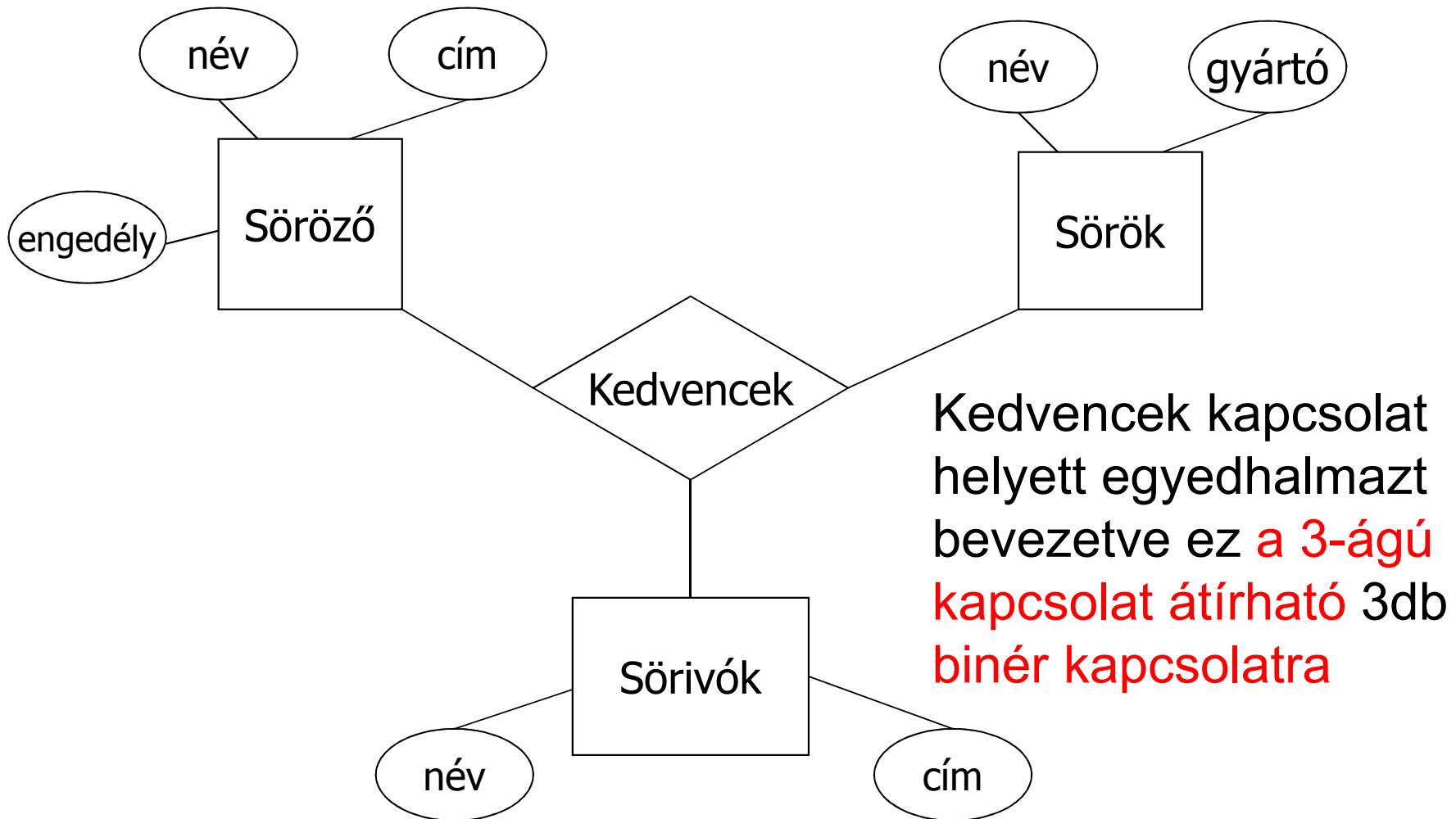
Két egyedhalmaz között több kapcsolat is lehet



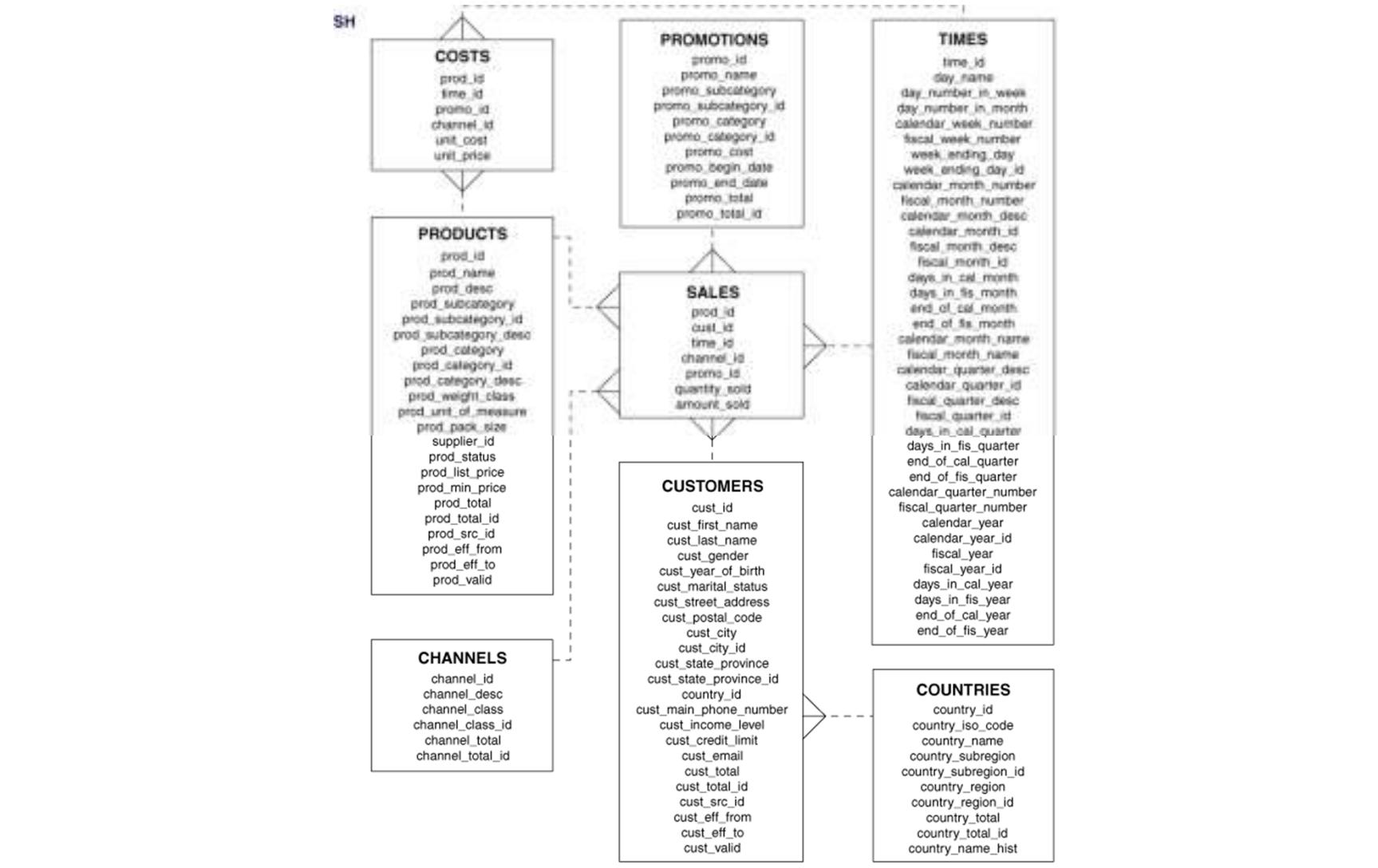
Egy egyedhalmaz önmagával is kapcsolódhat: Szerepek (Roles)



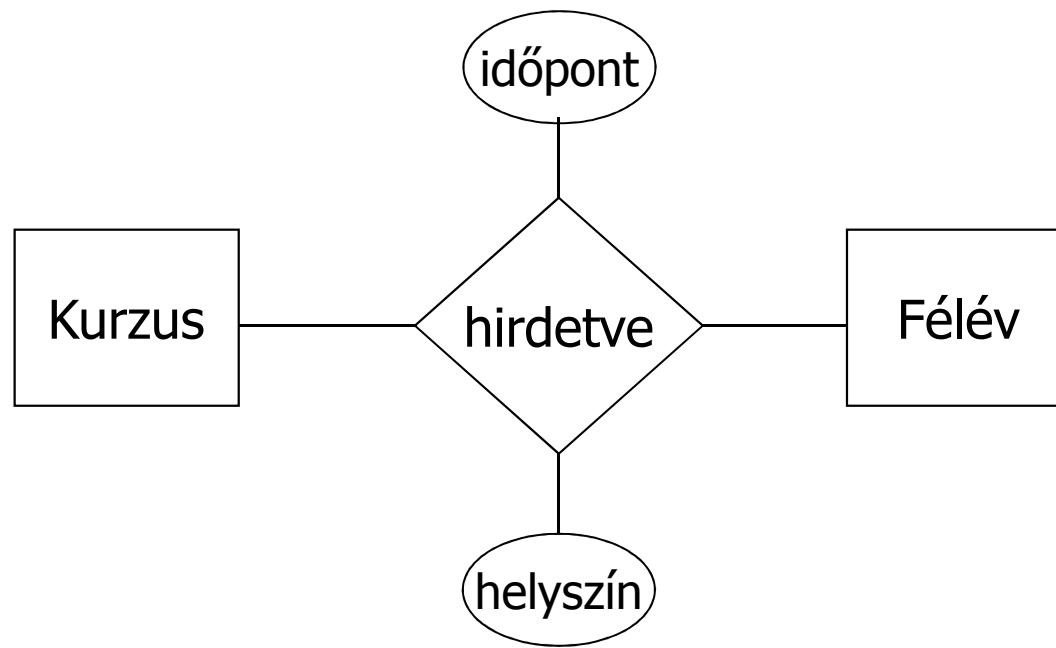
Példa: Többágú (3-ágú) kapcsolatra



Példa: Oracle SH (Sales History) séma

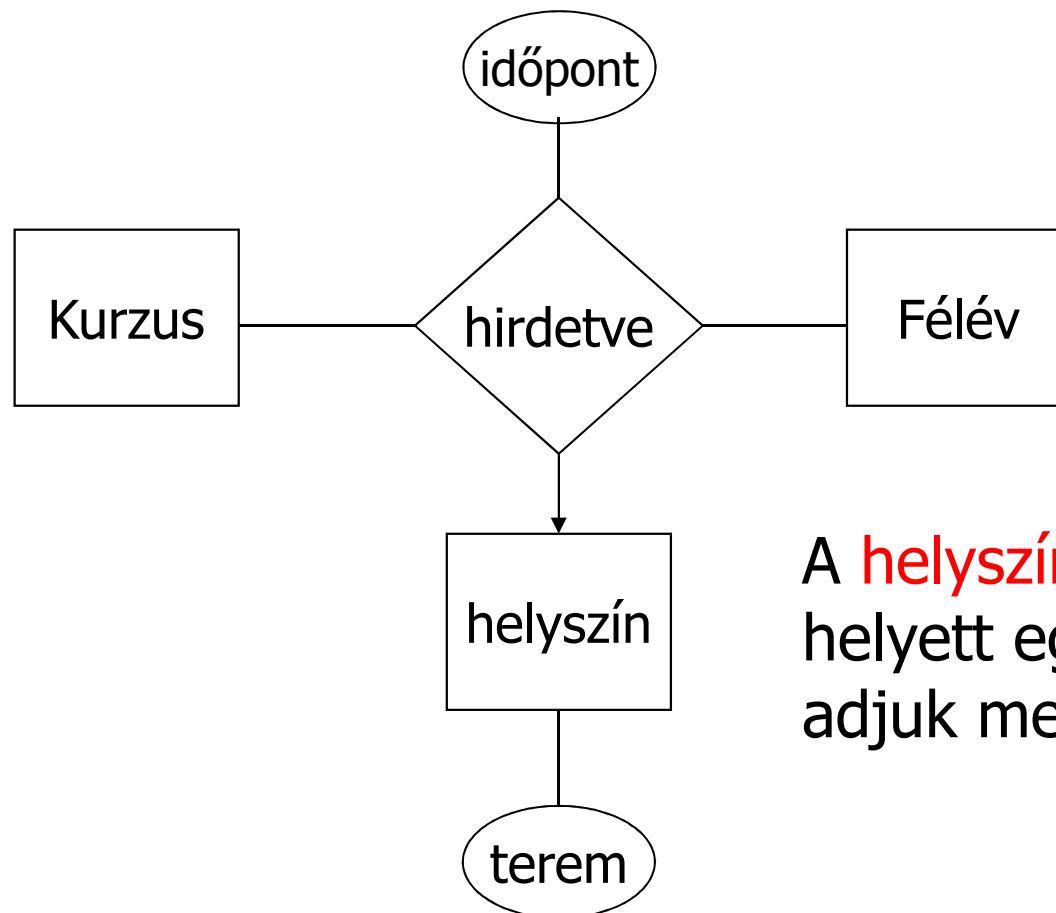


Kapcsolatnak is lehet attribútuma



Az **időpont** és **helyszín** a Kurzus és Félév együttes függvénye, de egyiké sem külön.

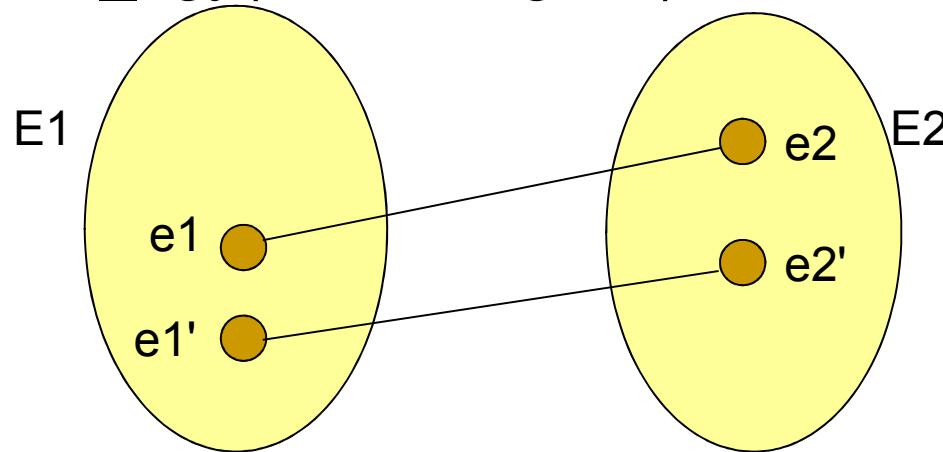
Tervezési kérdés: Attribútum vagy egyedhalmaz?



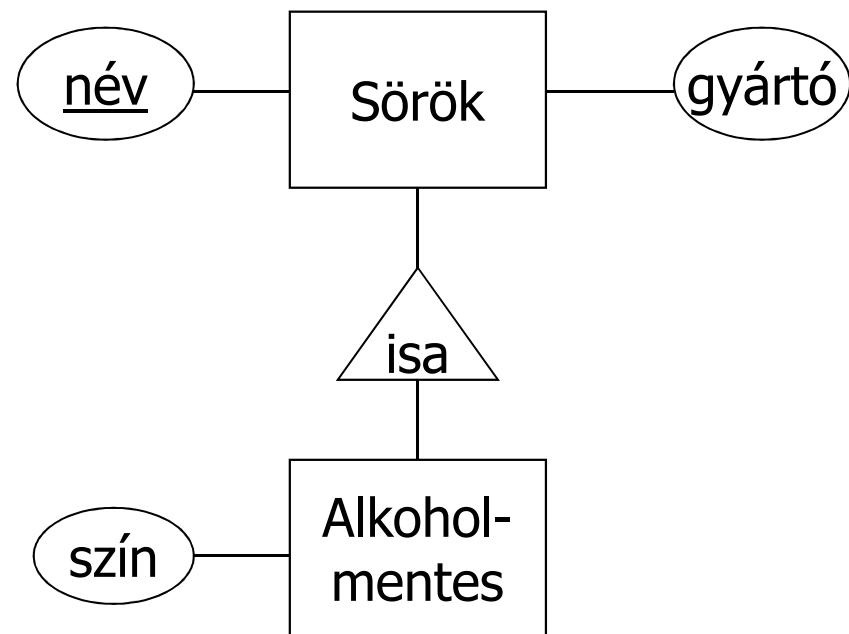
A **helyszínt** itt attribútum
helyett egyedhalmazként
adjuk meg

Speciális „is-a” (az-egy) kapcsolat

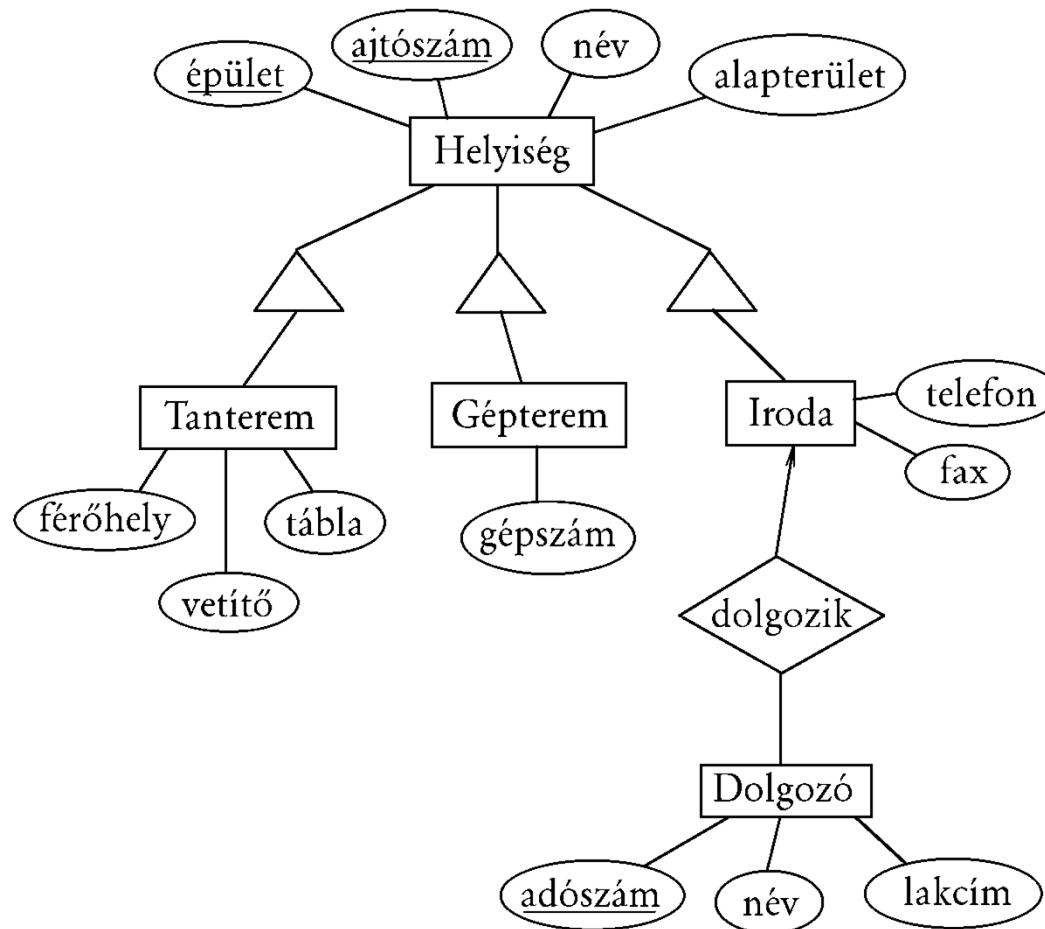
- K(E1,E2) bináris kapcsolat,
 - öröklődési kapcsolat ("az egy", ISA),
 - "a PC is a computer" = "a PC az egy számítógép",
 - speciális egy-egy kapcsolat,
 - K {(ei,ej)} alakú előfordulásaiban **az összes E1-beli egyed szerepel**,
 - például: az_egy(főnök,dolgozó).



Alosztályok és öröklődés



Példa: „is-a” (az-egy) kapcsolatra

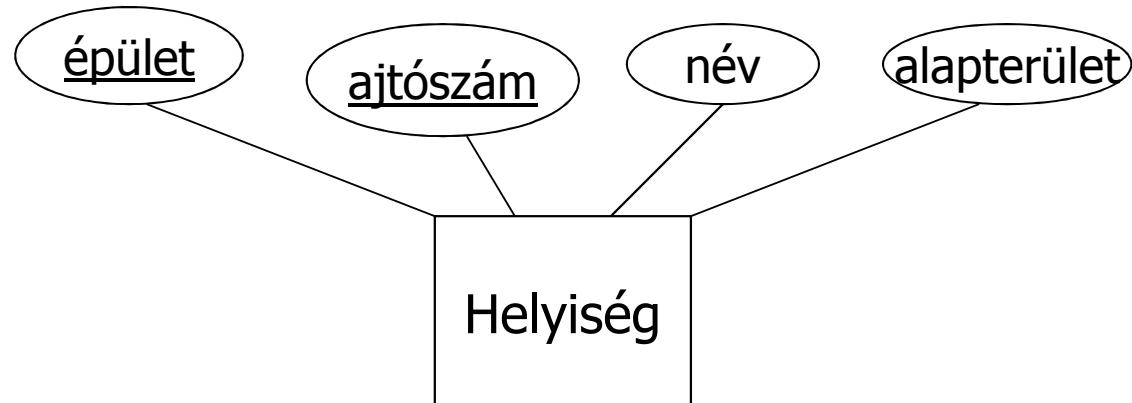


Kulcs megszorítás jele: aláhúzás

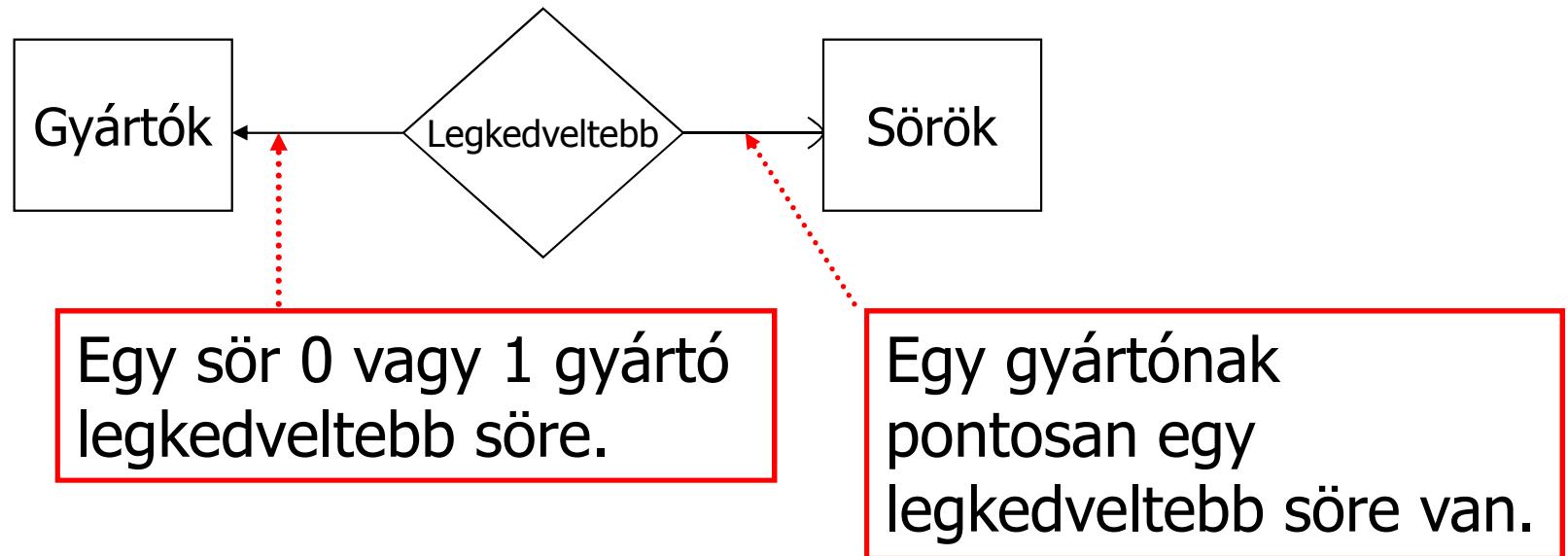
Példa egyszerű kulcsra: név a Sörök elsődleges kulcsa:



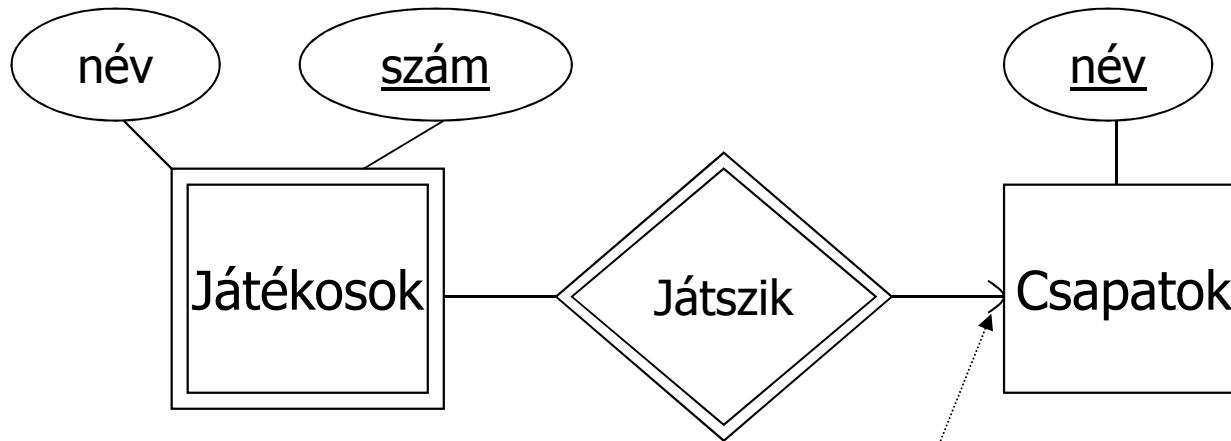
Példa összetett kulcsra: épület, ajtószám két-attribútumos
elsődleges kulcsa a Helyiség-nek:



Hivatkozási épség megszorítás jele a kerek végződés →)



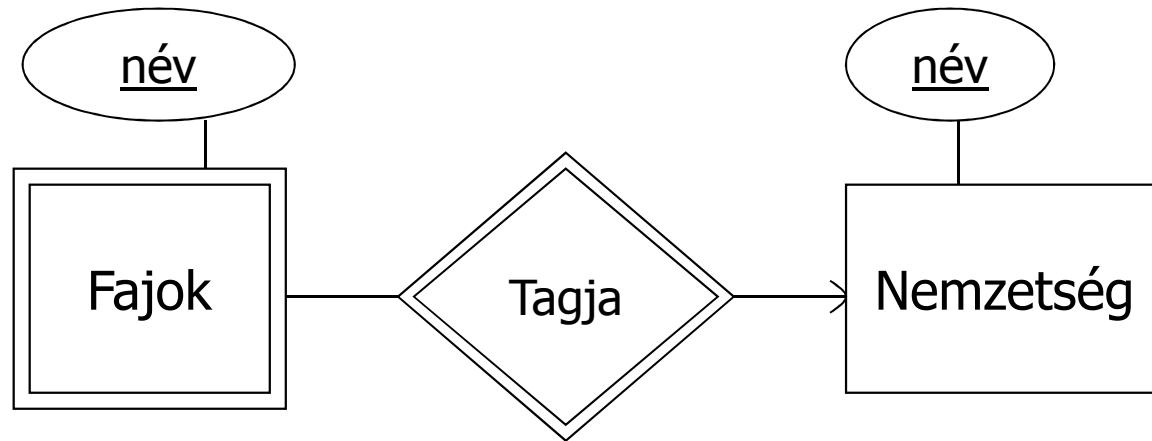
Erős és gyenge egyedhalmaz



A kerek végződés jelzi, hogy minden játékoshoz kötelezően tartozik egy csapat, amely az azonosításhoz használható.

- Dupla rombusz: sok-egy gyenge kapcsolat.
- Dupla téglalap: gyenge egyedhalmaz.

Erős és gyenge egyedhalmaz



Tankönyv 4.21. példája:
például az emberek a **Homo sapiens** fajhoz tartoznak,
ahol **Homo** a nemzetseg neve, a **sapiens** a faj neve
(sajnos maguk a fajok nevei nem egyértelműek, két
vagy több nemzettségen is lehet ugyanolyan fajnév).

Tervezési alapelvek

- valósághű modellezés:
 - megfelelő tulajdonságok tartozzanak az egyedosztályokhoz, például a tanár neve ne a diák tulajdonságai közé tartozzon
- redundancia elkerülése:
 - az **index(etr_kód,lakcím,tárgy,dátum,jegy)** rossz séma, mert a lakcím annyiszor ismétlődik, ahány vizsgajegye van a diáknak, helyette 2 sémát érdemes felvenni:
hallgató(etr_kód,lakcím), vizsga(etr-kód,tárgy,dátum,jegy).
- egyszerűség:
 - fölöslegesen ne vegyük fel egyedosztályokat
 - például a **naptár(év,hónap,nap)** helyett a megfelelő helyen inkább **dátum** tulajdonságot használunk
- tulajdonság vagy egyedosztály:
 - például a **vizsgajegy** osztály helyett **jegy** tulajdonságot használunk.

Modellezési feladatok (Tankönyv)

- **4.1.1. feladat.** Tervezzünk egy bank részére adatbázist, amely tartalmazza az ügyfeleket és azok számláit. Az ügyfelekről tartsuk nyilván a nevüket, címüket, telefonszámukat és TAJ-számukat. A számláknak legyen számlaszámuk, típusuk (pl. takarékbetét-számla, folyószámla stb.) és egyenlegük. Továbbá, meg kell jelölni azokat az ügyfeleket, akiknek van számlájuk. Adjuk meg az E/K diagramját ennek az adatbázisnak. Alkalmazzunk nyilakat a kapcsolatokban a multiplicitások jelölésére.

Modellezési feladatok (Tankönyv)

- **4.1.3. feladat.** Adjuk meg az E/K modelljét egy olyan adatbázisnak, amely csapatokat, játékosokat és azok szurkolóit tartja nyilván:
 - minden csapatról tároljuk a nevét, játékosait, csapatkapitányát (ő is egy játékos), mezük színét.
 - minden játéknak legyen neve.
 - minden rajongóról tartsuk nyilván a nevét, kedvenc csapatát, kedvenc játékosát és kedvenc színét.
- Vigyázzunk, a színek halmaza nem lehet a csapatok egy attribútumának típusa. Hogyan lehet ezzel a megszorítással együtt megfelelő modellt készíteni?

Modellezési feladatok (Tankönyv)

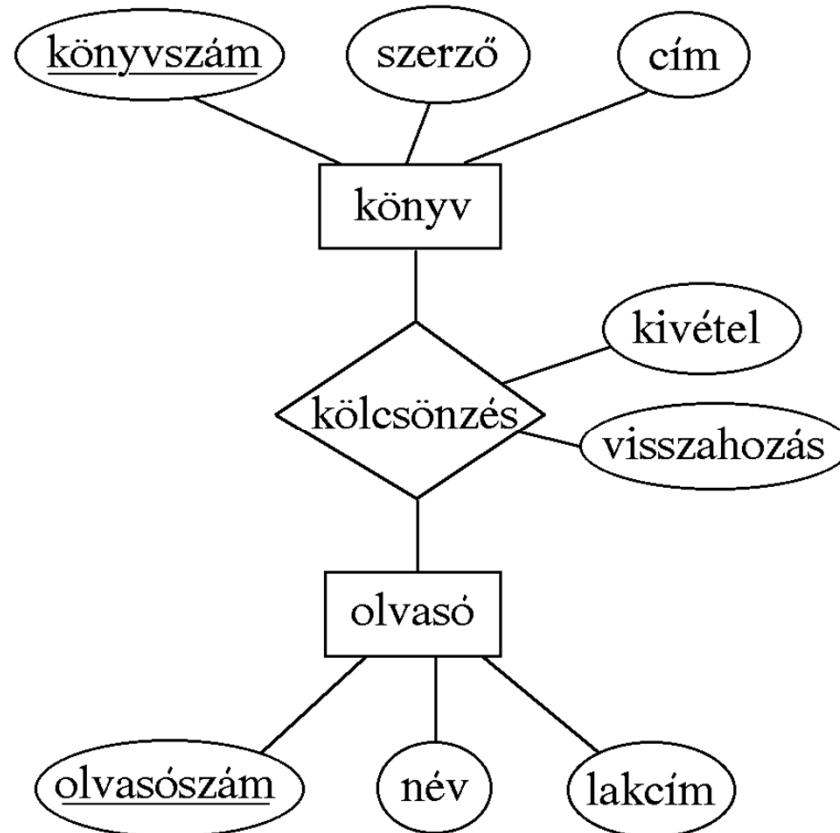
- **4.1.9. feladat.** Tervezzünk adatbázist egy tanulmányi osztály számára. Ez az adatbázis tartalmazza a hallgatókat, oktatókat, tanszékeket és kurzusokat. Ezenkívül tartsuk nyilván, hogy a hallgatók milyen kurzusokat vettek fel, az adott kurzust mely oktató oktatja, a hallgatók jegyeit, a kurzusoknál az oktató munkáját segítő hallgatókat, egy adott kurzust mely tanszék ajánlotta, és minden olyan információt, ami a fentiek megvalósításához szükséges. Megjegyezzük, hogy ez a feladat nagy szabadságot enged a korábbiakhoz képest. Dönten kell a kapcsolatok típusáról (sok-sok, sok-egy vagy egy-egy), az alkalmas típus megválasztásáról, illetve arról, hogy milyen segédinformációkat használunk.

E/K-diagram átírása

Tankönyv 4.5.-4.6. E/K-diagram átírása relációkká

- Egyedhalmazok átírása relációkká
- E/K-kapcsolatok átírása relációkká
- Egyszerűsítés, összevonások
- Gyenge egyedhalmazok kezelése
- Osztályhierarchia átalakítása relációkká

Példa: Egy könyvtár adatmodellje

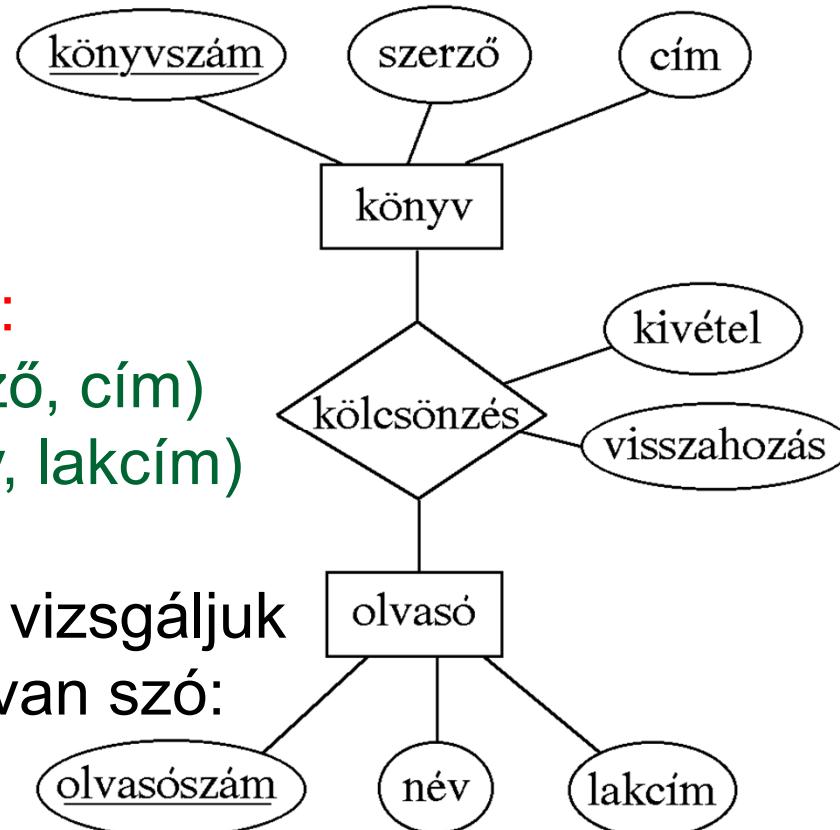


E/K diagram átírásra példa

1.) Egyedhalmazok átírása:

KÖNYV (könyvszám, szerző, cím)

OLVASÓ (olvasószám, név, lakcím)



Egyedhalmazok átírásánál vizsgáljuk meg, ha összetett típusról van szó:
Például a lakcím (rekord)
vagy a szerző (többértékű,
halmaz vagy rendezett lista) milyen megoldások lehetnek?

2.) Ezután jön majd a kapcsolatok átírása:

KÖLCSÖN (könyvszám, olvasószám, kivétel, visszahozás)

Egyedhalmazok átírásánál az összetett attribútumok leképezése

- Tegyük fel, hogy az OLVASÓ táblában a **lakcím** attribútumot (helység, utca, házszám) struktúraként, vagyis rekordként szeretnénk kezelní.
- Relációs adatmodellben erre egy lehetőség van: az OLVASÓ (olvasószám, név, lakcím) séma helyett a OLVASÓ (olvasószám, név, helység, utca, házszám) sémára térünk át.

Egyedhalmazok relációkká való átírásánál Többértékű attribútumok leképezése ---1

- Kérdés, hogy többszerzős könyveket hogyan tartsunk nyilván az adatbázisban.
- **1.megoldás: Megadás egyértékű attribútumként.**
A szerző megadására szolgáló szövegmezőben felsoroljuk a szerzőket.
- Hátrányok:
 - a szerzőket külön-külön nem tudjuk kezelní
 - sok szerző esetleg nem fér el a megadott mezőben

Egyedhalmazok relációkká való átírásánál Többértékű attribútumok leképezése ---2

- 2.megoldás: Megadás többértékű attribútumként.

a.) Sorok többszörözése. A KÖNYV táblában egy könyvhöz annyi sort veszünk fel, ahány szerzője van:

Könyvszám	Szerző	Cím
1121	Ullman	Adatbázisok
1121	Widom	Adatbázisok
3655	Radó	Világatlasz
2276	Karinthy	Így írtok ti
1782	Jókai	Aranyember

A megfelelő relációséma:

KÖNYV (könyvszám, szerző, cím)

- A fenti megoldás **hátránya**, hogy a többszerzős könyvek címét több példányban kell megadni, ami redundanciát jelent.

Egyedhalmazok relációkká való átirásánál Többértékű attribútumok leképezése ---3

b.) új tábla felvétele: KÖNYV (könyvszám, szerző, cím)
sémát az alábbi két sémával helyettesítjük:

KÖNYV (könyvszám, cím)

SZERZŐ (könyvszám, szerző)

c) Sorszámozás. Ha a szerzők sorrendje nem közömbös, akkor a SZERZŐ táblát egy sorszám mezővel kell bővíteni (emlékeztetünk rá, hogy a relációs adatmodell nem definiálja a rekordok sorrendjét):

KÖNYV (könyvszám, cím)

SZERZŐ (könyvszám, sorszám, szerző)

E/K diagram átírása relációs adatbázistervre

Mi minek felel meg:

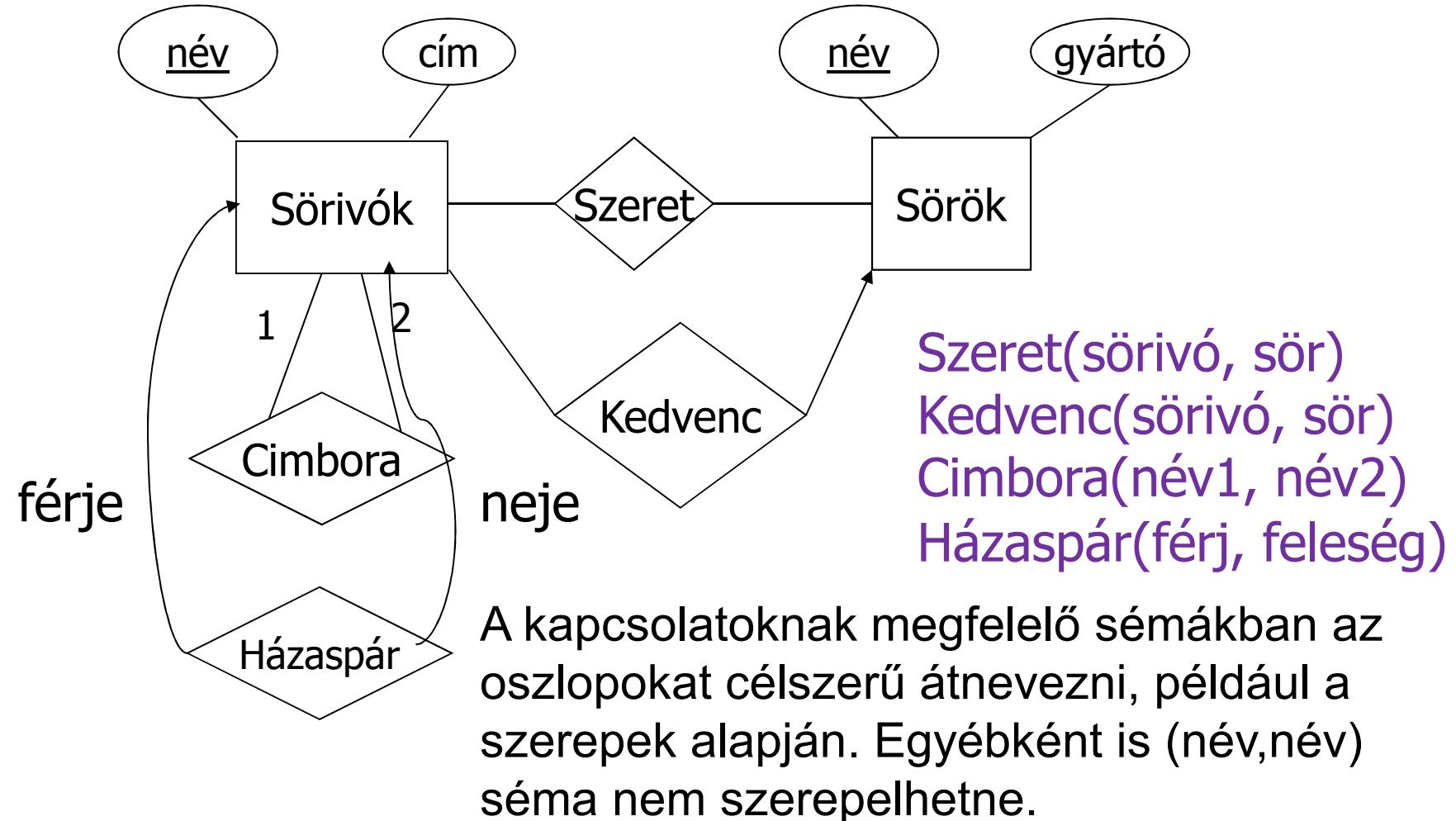
- | | |
|---|---|
| ➤ egyedhalmaz séma
$E(A_1, \dots, A_n)$ | ↔ relációséma
$E(A_1, \dots, A_n)$ |
| ➤ tulajdonságok | ↔ attribútumok |
| ➤ (szuper)kulcs | ↔ (szuper)kulcs |
| ➤ egyedhalmaz
előfordulása | ↔ reláció |
| ➤ e egyed | ↔ $(e(A_1), \dots, e(A_n))$ sor |
| ➤ $R(E_1, \dots, E_p, A_1, \dots, A_q)$
kapcsolati séma, ahol
Ei egyedhalmaz,
Aj saját tulajdonság | ↔ $R(K_1, \dots, K_p, A_1, \dots, A_q)$
relációséma, ahol
Ki az Ei (szuper)kulcsa |
| ➤ E/K modell | ↔ Relációs adatmodell |

E/K diagram átírása relációs adatbázistervre

- A transzformálás előtt a tulajdonságokat átnevezhetjük, hogy a **relációsémában ne szerepeljen kétszer ugyanaz az attribútum**.
- Az **az_egy kapcsolat** esetén a speciális osztály saját attribútumaihoz hozzávesszük az általános osztály (szuper)kulcsát.
- Ha $R(E1, E2)$ sok-egy kapcsolat, akkor $R(K1, K2)$ relációsémának a $K1$ szuperkulcsa lesz.
- A **gyenge entitás** relációsémáját bővíteni kell a meghatározó kapcsolat(ok)ban szereplő egyed(ek) kulcsával.

Példa: E/K diagram átírása relációkká

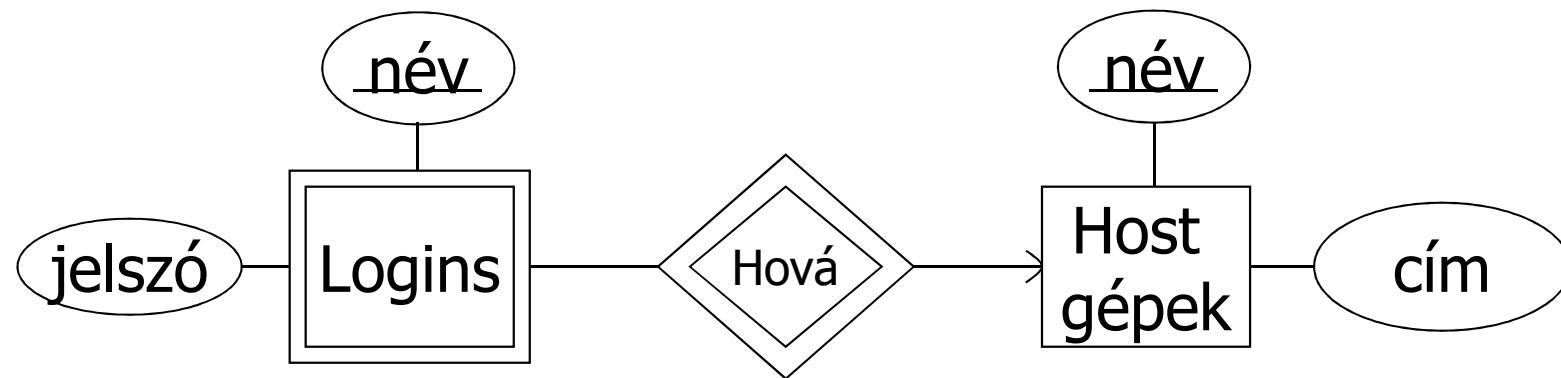
Az egyedek átírása után a kapcsolatok átírása:



Relációk összevonása

- Összevonhatunk 2 relációt, ha az egyik egy **sok-egy** kapcsolatnak megfelelő reláció, a másik pedig a sok oldalon álló egyedhalmaznak megfelelő reláció.
- Példa:
Sörivók(név, cím) és **Kedvenc(ivó,sör)** összevonható, és kapjuk az **Sörivó1(név,cím,kedvencSöre)** sémát.

Gyenge egyedhalmaz átírása



Hostgépek(hostNév, cím)

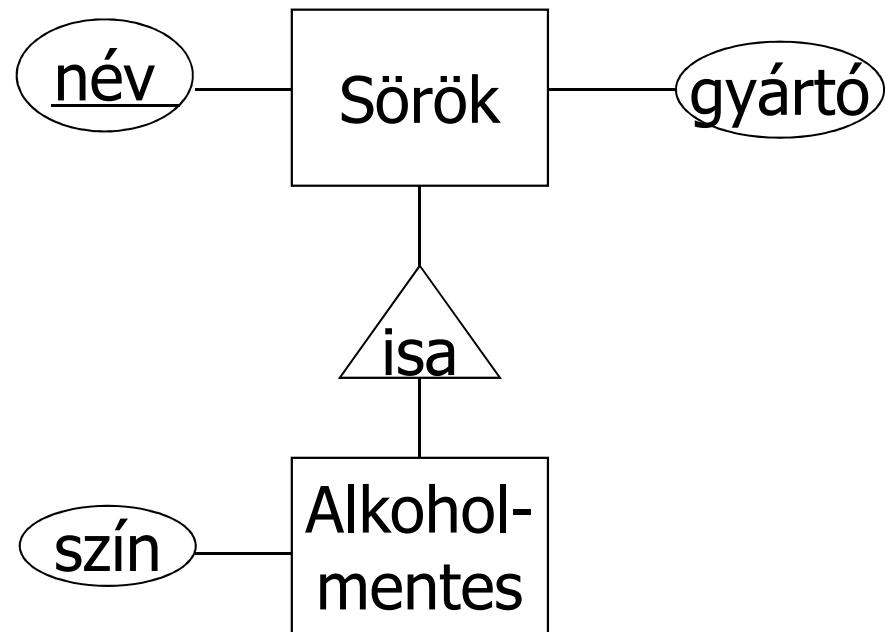
Logins(loginNév, hostNév, jelszó)

Hová(loginNév, hostNév, hostNév2)

Beolvastjuk a
Logins relációba

A logins kulcsa összetett: loginNév, hostNév
Kétszer szerepelne az azonos értékű
hostNév a Hová sémában

Alosztály átírására relációkká



Alosztályok átírása: három megközelítés

- **E/R stílusban:** Egy reláció minden alosztályra, de az általános osztályból csak a kulcsokat vesszük hozzá a saját attribútumokhoz.
- **Objektumorientált stílusban:** Egy reláció minden alosztályra, felsorolva az összes tulajdonságot, beleértve az örökölteket is.
- **Nullértékek használatával:** Egyetlen reláció az öröklődésben résztvevő összes osztályra. Ha egy egyed nem rendelkezik egy alosztály speciális tulajdonságával, akkor ezt az attribútumot NULL értékkel töltjük majd ki.

E/K tipusú átalakítás ---1

név	gyártó
Bud	Anheuser-Busch
Summerbrew	Pete's

Sörök

név	szín
Summerbrew	világos

Alkoholmentes

Az olyan lekérdezésekre jó, hogy egy adott gyártó milyen söröket gyárt, beleértve az alkoholmenteseket is.

Objektumorientált megközelítés ---2

név	gyártó
Bud	Anheuser-Busch
Sörök	

név	gyártó	szín
Summerbrew	Pete's	világos
Alkoholmentes		

Az olyan lekérdezésekre jó, hogy egy adott gyártó milyen színű alkoholmentes söröket gyárt.

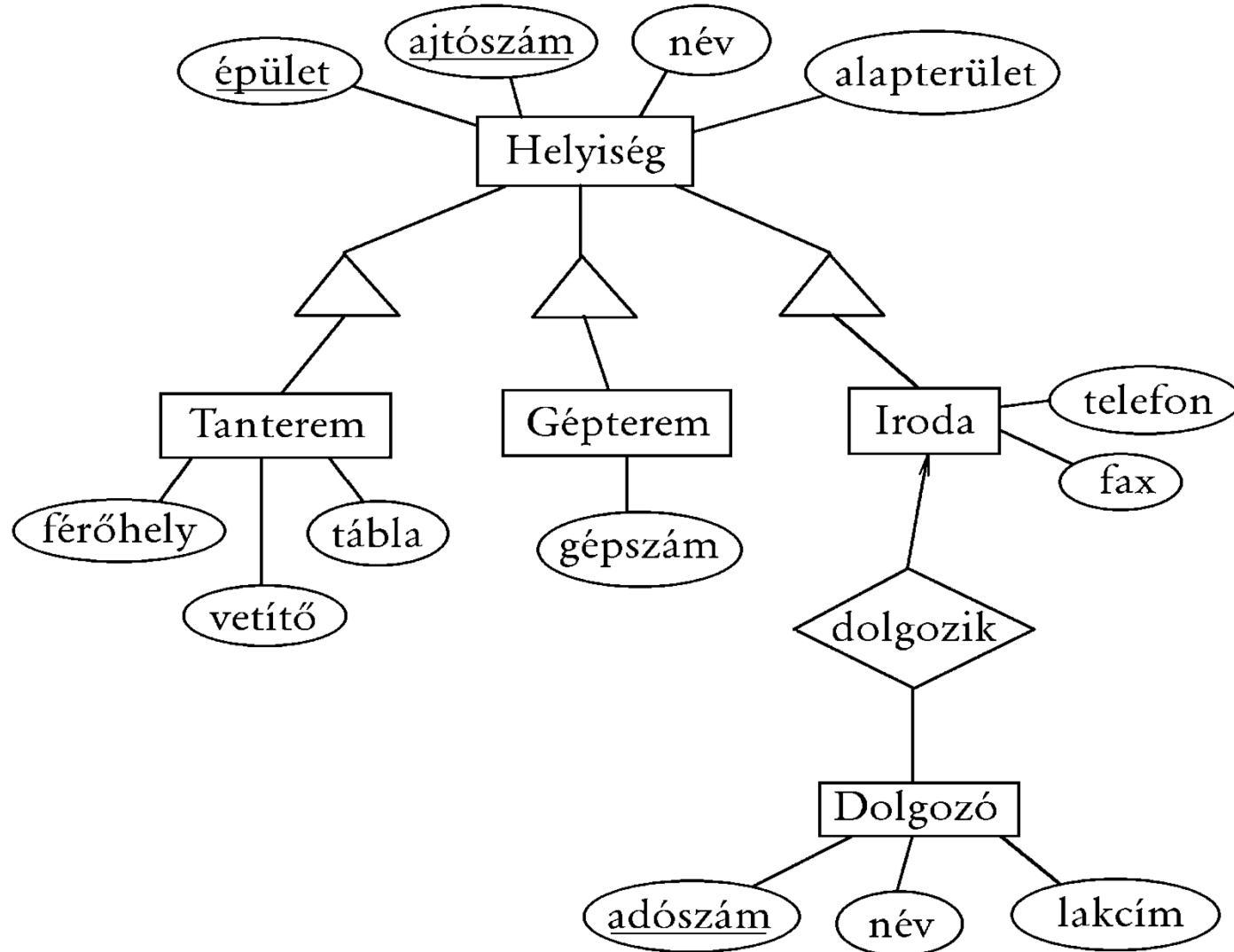
Nullértékek használatával ---3

név	gyártó	szín
Bud Summerbrew	Anheuser-Busch Pete's	NULL világos

Sörök

Általában kevesebb hely elég a tárolásra, kivéve ha nagyon sok attribútum marad nullértékű.

Példa: Alosztály átírása relációkká



E/K típusú átalakítás --- 1

2. **Minden altípushoz külön tábla felvétele**, egy egyed több táblában is szerepelhet. A főtípus táblájában minden egyed szerepel, és annyi altípuséban ahánynak megfelel. Az altípusok a főtípustól csak a kulcs-attribútumokat öröklik.
(E/K stílusú reprezentálás.)

HELYISÉG (épület, ajtószám, név, alapterület)

TANTEREM (épület, ajtószám, férőhely, tábla, vetítő)

GÉPTEREM (épület, ajtószám, gépszám)

IRODA (épület, ajtószám, telefon, fax)

DOLGOZÓ (adószám, név, lakcím, épület, ajtószám)

Hátrány: Előfordulhat, hogy több táblában kell keresni (például ha a tantermek nevére és férőhelyére vagyunk kíváncsiak, akkor össze kell kapcsolni a táblákat).

Objektumorientált megközelítés --- 2

1. **Minden altípushoz külön tábla felvétele**, egy egyed csak egy táblában szerepel. Az altípusok öröklik a főtípus attribútumait.
(Objektumorientált stílusú reprezentálás)

HELYISÉG (épület, ajtószám, név, alapterület)

TANTEREM (épület, ajtószám, név, alapterület, férőhely, tábla, vetítő)

GÉPTEREM (épület, ajtószám, név, alapterület, gépszám)

IRODA (épület, ajtószám, név, alapterület, telefon, fax)

DOLGOZÓ (adószám, név, lakcím, épület, ajtószám)

Hátrányok:

- Kereséskor gyakran több táblát kell vizsgálni (ha például a D épület 803. számú terem alapterületét keressük).
- Kombinált altípus (például számítógépes tanterem) csak új altípus felvételével kezelhető.

Nullértékek használata relációk egyesítéséhez --- 3

3. Egy közös tábla felvétele, az attribútumok uniójával.

Az aktuálisan értékkel nem rendelkező attribútumok
NULL értékűek.

(Reprezentálás nullértékekkel)

HELYISÉG (épület, ajtószám, név, alapterület, férőhely, tábla,
vétítő, gépszám, telefon, fax)

DOLGOZÓ (adószám, név, lakcím, épület, ajtószám)

Hátrányok:

- Az ilyen egyesített táblában általában sok NULL attribútumérték szerepel.
- Elveszíthetjük a típusinformációt (például ha a gépteremnél a gépszám nem ismert és ezért NULL, akkor a gépterem lényegében az egyéb helyiségek kategóriájába kerül).

Kérdés/Válasz

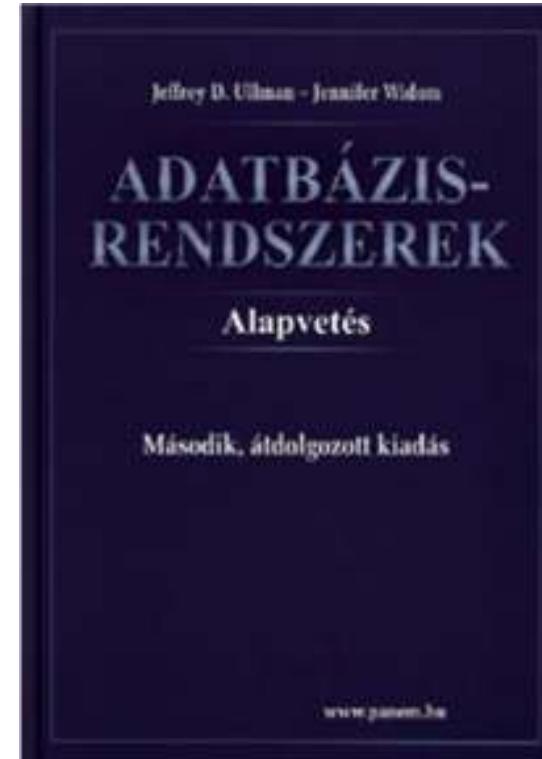
- Köszönöm a figyelmet! Kérdés/Válasz?

Relációs adatbázisok tervezése ---1

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás, 2009

-
- 3.3.1. Bevezetés: anomáliák
 - 3.3.2. Relációk felbontása
 - 3.1. Funkcionális függőségek
 - 3.2. Funk.függőségek szabályai,
 X^+ attribútumhalmaz lezártja,
és X^+ kiszámítására algoritmus,
Funkc.függőségi halmazok lezárása,
Funkcionális függőségek vetítése

Folyt. 2.részben: 3.3.3-3.3.4. Boyce-Codd normálforma és
3.4. A felbontások tulajdonságai, 3.részben 3NF, stb.



Relációs adatmodell története

- E.F. Codd 1970-ban publikált egy cikket
A Relational Model of Data for Large Shared Data Banks
Link: <http://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf>
amelyben azt javasolta, hogy az adatokat táblázatokban,
relációkban tárolják. Az elméletére alapozva jött létre a
relációs adatmodell, és erre épülve jöttek létre a relációs
adatmodellen alapuló (piaci) relációs adatbázis-kezelők.
- **Relációs sématervezés:** függőségeken alapuló felbontás,
normalizálás: Ezen a kurzuson a funkcionális függőségen
alapuló Boyce_Codd normálformát (BCNF) és a 3NF-t,
illetve a többértékű függőségen alapuló 4normálformát
tanuljuk, és megvizsgáljuk a felbontások tulajdonságait
(veszteségmentesség, függőségőrzés).

Tankönyv 3.fejezet: Bevezető példa

- Több tábla helyett -> vegyük egy táblában lenne
Melyik séma jobb?

Sörivó(név, cím, kedveltSörök, gyártó, aKedvencSör)

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

- Redundáns adat, a ??? helyén mi szerepel, ha mindenkinnek csak egy lakcíme és a KedvencSöre lehet, vagyis a név meghatározza a címet és a KedvencSör-t és a Söröknek is egy gyártója
lehet

Hibás tervezés problémái

A rosszul tervezettség anomáliákat is eredményez
Sörivó(név, cím, kedveltSörök, gyártó, aKedvencSör)

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

- **Módosítási anomália:** ha Janeway-t Jane-re módosítjuk, megtesszük-e ezt minden sornál?
- **Törlési anomália:** Ha senki sem szereti a Bud sört, azzal töröljük azt az infót is, hogy ki gyártotta.
- **Beillesztési anomália:** és felvinni ilyen gyártót?

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.
- Beillesztési anomália : megszorítás, trigger kell, hogy ellenőrizni tudjuk (pl. a kulcsfüggőséget)
- Redundancia (többszörös tárolás feleslegesen)

Dekomponálás (felbontás)

A fenti problémáktól dekomponálással (felbontással) tudunk megszabadulni:

- Definíció:
 $d=\{R_1, \dots, R_k\}$ az (R, F) **dekompozíciója**, ha nem marad ki attribútum, azaz $R_1 \cup \dots \cup R_k = R$. Az adattábla felbontását projekcióval végezzük.
- Példa:
 $R=ABCDE$, $d=\{AD, BCE, ABE\}$
3 tagú dekompozíció, ahol
 $R_1=AD$, $R_2=BCE$, $R_3=ABE$,

Felbontásra vonatkozó elvárások

➤ Elvárások

- (1) A **vetületek** legyenek jó tulajdonságúak, és a vetületi függőségi rendszere egyszerű legyen (normálformák: BCNF, 3NF, 4NF, később)
- (2) A **felbontás** is jó tulajdonságú legyen, vagyis ne legyen információvesztés:
Veszteségmentes legyen a felbontás (VM)
- (3) **Függőségek megőrzése** a vetületekben (FŐ)
 - Tételek (ezekre nézünk majd algoritmusokat)
 - Mindig van VM BCNF-ra való felbontás
 - Mindig van VM FŐ 3NF-ra való felbontás

Relációs sématervezés (vázlat)

- **I. Függőségek:** a sématervezésnél használjuk
 - Funkcionális függőség
 - Többértékű függőség
- **II. Normalizálás:** „ jó” sémákra való felbontás
 - Funkcionális függőség -> BCNF
 - Funkcionális függőség -> 3NF
 - Többértékű függőség -> 4NF
- **III. Felbontás tulajdonságai:** „ jó” tulajdonságok
 - Veszeségmentesség
 - Függőségőrző felbontás

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.

Funkcionális függőségek (definíció)

Definíció. Legyen $R(U)$ egy relációséma, továbbá X és Y az U attribútum-halmaz részhalmazai.

X -től *funkcionálisan függ* Y (jelölésben $X \rightarrow Y$), ha bármely R feletti T tábla esetén valahányszor két sor megegyezik X -en, akkor megegyezik Y -on is $\forall t1,t2 \in T$ esetén $(t1[X]=t2[X] \Rightarrow t1[Y]=t2[Y])$.

Ez lényegében azt jelenti, hogy az X -beli attribútumok értéke egyértelműen meghatározza az Y -beli attribútumok értékét.

- **Jelölés:** $R \models X \rightarrow Y$ vagyis
 R kielégíti $X \rightarrow Y$ függőséget

Példa: Funkcionális függőség

Sörivók(név, cím, kedveltSörök, gyártó, aKedvencSör)

Feltehetjük például, hogy az alábbi FF-ek teljesülnek:

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

Mert név -> cím

Mert név -> aKedvencSör

Mert kedveltSörök -> gyártó

Jobboldalak szétvágása (FF)

- $X \rightarrow A_1 A_2 \dots A_n$ akkor és csak akkor teljesül R relációra, ha $X \rightarrow A_1$, $X \rightarrow A_2, \dots$, $X \rightarrow A_n$ is teljesül R -en.
- Példa: $A \rightarrow BC$ ekvivalens $A \rightarrow B$ és $A \rightarrow C$ függőségek kettősével.
- Baloldalak szétvágására nincs szabály!!!
- Ezért elég nézni, ha a FF-k jobboldalán egyetlen attribútum szerepel

Kulcs, szuperkulcs

- Funktionális függőség $X \rightarrow Y$ speciális esetben, ha $Y = U$, ez a kulcsfüggőség.
- $R(U)$ relációséma esetén az U attribútum-halmaz egy K részhalmaza akkor és csak akkor **szuperkulcs**, ha a $K \rightarrow U$ FF teljesül.
- A kulcsot tehát a függőség fogalma alapján is lehet definiálni: olyan K attribútum-halmazt nevezünk kulcsnak, amelytől az összes többi attribútum függ (vagyis szuperkulcs), de K -ból bármely attribútumot elhagyva ez már nem teljesül (vagyis minimális szuperkulcs)

Példa: szuperkulcs, kulcs

Sörivók(név, cím, kedveltSörök, gyártó, aKedvencSör)

- {név, kedveltSörök} **szuperkulcs**, ez a két attr. meghatározza funkcionálisan a maradék attr-kat.
 - név -> cím aKedvencSör
 - kedveltSörök -> gyártó
- {név, kedveltSörök} **kulcs**, hiszen sem {név}, sem {kedveltSörök} nem szuperkulcs.
 - név -> gyártól; kedveltSörök -> cím nem telj.
- Az előbbi kívül nincs több kulcs, de számos szuperkulcs megadható (ami ezt tartalmazza)

Másik példa (több kulcs is lehet)

- Legyen ABC sémán def.FF-ek: $AB \rightarrow C$ és $C \rightarrow B$.
 - Példa: A = utca, B = város, C = irányítószám.
- Itt két kulcs is van: $\{A,B\}$ és $\{A,C\}$.

Az implikációs probléma

- Legyenek $X_1 \rightarrow A_1, X_2 \rightarrow A_2, \dots, X_n \rightarrow A_n$ adott FF-k, szeretnénk tudni, hogy $Y \rightarrow B$ teljesül-e olyan relációkra, amire az előbbi FF-k teljesülnek.
 - Példa: $A \rightarrow B$ és $B \rightarrow C$ teljesülése esetén $A \rightarrow C$ biztosan teljesül.
- Implikációs probléma eldöntése definíció alapján (minden előfordulásra ellenőrizni) túl nehéz, de van egyszerűbb lehetőség: levezetési szabályok segítségével, lásd Armstrong-axiómák.

Armstrong-axiómák

Legyen $R(U)$ relációséma és $X, Y \subseteq U$, és jelölje XY az X és Y attribútum-halmazok egyesítését. F legyen funkcionális függőségek tetsz. halmaza.

Armstrong axiómák:

- A1 (reflexivitás): $Y \subseteq X$ esetén $X \rightarrow Y$.
- A2 (bővíthetőség): $X \rightarrow Y$ és tetszőleges Z esetén $XZ \rightarrow YZ$.
- A3 (tranzitivitás): $X \rightarrow Y$ és $Y \rightarrow Z$ esetén $X \rightarrow Z$.

Levezetés fogalma

- F implikálja $X \rightarrow Y$ -t (F -nek következménye $X \rightarrow Y$), ha minden olyan táblában, amelyben F összes függősége teljesül, azokra $X \rightarrow Y$ is teljesül.
Jelölés: $F \models X \rightarrow Y$, ha F implikálja $X \rightarrow Y$ -et.
- $X \rightarrow Y$ levezethető F -ből, ha van olyan $X_1 \rightarrow Y_1, \dots, X_k \rightarrow Y_k, \dots, X \rightarrow Y$ véges levezetés, hogy $\forall k$ -ra $X_k \rightarrow Y_k \in F$ vagy $X_k \rightarrow Y_k$ az FD1, FD2, FD3 axiómák alapján kapható a levezetésben előtte szereplő függőségekből.
Jelölés: $F \vdash X \rightarrow Y$, ha $X \rightarrow Y$ levezethető F -ből

További levezethető szabályok:

4. Szétvághatósági (vagy felbontási) szabály
 $F \vdash X \rightarrow Y$ és $Z \subseteq Y$ esetén $F \vdash X \rightarrow Z$.
5. Összevonhatósági (vagy unió) szabály
 $F \vdash X \rightarrow Y$ és $F \vdash X \rightarrow Z$ esetén $F \vdash X \rightarrow YZ$.
6. Pszeudotranzitivitás
 $F \vdash X \rightarrow Y$ és $F \vdash WY \rightarrow Z$ esetén $F \vdash XW \rightarrow Z$.

Bizonyítás (4): Reflexivitási axióma miatt $F \vdash Y \rightarrow Z$, és tranzitivitási axióma miatt $F \vdash X \rightarrow Z$.

Bizonyítás (5): Bővítési axióma miatt $F \vdash XX \rightarrow YX$ és $F \vdash YX \rightarrow YZ$, és $XX = X$, valamint a tranzitivitási axióma miatt $F \vdash X \rightarrow YZ$.

Bizonyítás (6): Bővítési axióma miatt $F \vdash XW \rightarrow YW$, és $YW = WY$, és a tranzitivitási axióma miatt $F \vdash XW \rightarrow Z$.

Szétvághatóság/összevonhatóság

- A szétvághatósági és összevonhatósági szabályok következménye:

$$F \vdash X \rightarrow Y \Leftrightarrow \forall A_i \in Y \text{ esetén } F \vdash X \rightarrow A_i$$

- A következmény miatt feltehető, hogy a függőségek jobb oldalai 1 attribútumból állnak.
- **Fontos!** A függőségeknek csak a jobboldalát lehet szétbontani, a baloldalra ez természetesen nem igaz (például $\{\text{filmcím}, \text{év}\} \rightarrow \text{stúdió}$)

Armstrong-axiómák (téTEL)

- TÉTEL: Az Armstrong-axiómarendszer helyes és teljes, azaz minden levezethető függőség implikálódik is, illetve azok a függőségek, amelyeket F implikál azok levezethetők F-ből.

$$F \vdash X \rightarrow Y \Leftrightarrow F \models X \rightarrow Y$$

Implikáció eldöntése --- Lezárással

- Implikációs probléma:
Legyenek $X_1 \rightarrow A_1, X_2 \rightarrow A_2, \dots, X_n \rightarrow A_n$ adott FF-k, szeretnénk tudni, hogy $Y \rightarrow B$ teljesül-e minden olyan relációkra, amire az előbbi FF-k teljesülnek. Hogyan tudjuk ellenőrizni, hogy egy előfordulás nem teljesíti $Y \rightarrow B$?
- Mivel az Armstrong axiómarendszer helyes és teljes, elegendő a levezetési szabályokkal levezeti. Még a levezetési szabályoknál is van egyszerűbb út: **kiszámítjuk Y lezártját: Y^+ -t**
- Attribútum-halmaz lezárására teszt:

Attribútumhalmaz lezártja (definíció)

- Adott R séma és F funkcionális függőségek halmaza mellett, Y^+ az összes olyan A attribútum halmaza, amire $Y \rightarrow A$ következik F -ből.
- (R,F) séma esetén legyen $Y \subseteq R$.
- Definíció: $Y^{+(F)} := \{A \mid F \vdash Y \rightarrow A\}$
az Y attribútum-halmaz lezárása F -re nézve.

Attribútumhalmaz lezártja (lemma)

- LEMMA: $F \vdash Y \rightarrow Z \Leftrightarrow Z \subseteq Y^+$.

Bizonyítás:

(\Rightarrow) $\forall A \in Z$ esetén a reflexivitás és tranzitivitás miatt $F \vdash Y \rightarrow A$, azaz $Z \subseteq Y^+$.

(\Leftarrow) $\forall A \in Z \subseteq Y^+$ esetén $F \vdash Y \rightarrow A$, és az egyesítési szabály miatt $F \vdash Y \rightarrow Z$.

- Lemma következménye: az implikációs probléma megoldásához elég az Y^+ -t hatékonyan kiszámolni.

Algoritmus Y^+ attr.halmaz lezártja

- Input: Y attribútumhz., F funk.függőségek hz.
- Output: Y^+ (zárás, típusa: attribútumhalmaz)
- Algoritmus Y^+ kiszámítására:
/* Iteráció, amíg $Y(n)$ változik */

$Y(0) := Y$

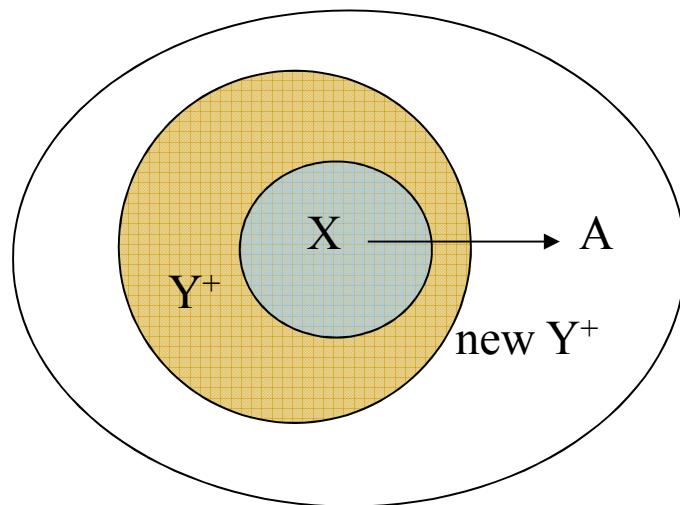
$Y(n+1) := Y(n) \cup \{A \mid \underline{X \rightarrow Z \in F}, A \in Z, X \subseteq Y(n)\}$

Ha $Y(v+1) = Y(v)$, akkor Output: $Y(v) = Y^+$.

- Miért működik az Y^+ lezárási algoritmus?
(Tankönyv 3.2.5. szakasz, 81-83.oldal)

Lezárás (teszt)

- **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.



A lezárást kiszámító algoritmus „helyes”

- Az algoritmus „tényleg” Y^+ -t számítja ki. Vagyis:
 1. Ha az A attribútum valamely i -re belekerül $Y(i)$ -be, akkor A valóban eleme Y^+ -nak.
 2. Másfelől, ha $A \in Y^+$, akkor létezik olyan i , amire A belekerül $Y(i)$ -be.
- Az első állítás: Miért csak az igaz funkcionális függőségeket fogadja el a lezárási algoritmus? Könnyen bizonyítható indukcióval [Tk.3.2.5.]

A lezárást kiszámító algoritmus „teljes”

- A második állítás: Miért talál meg minden igaz függőséget a lezárási algoritmus? [Tk.3.2.5.]
- Konstrukciós bizonyítás: Tegyük fel, hogy $A \in Y^+$, és nem olyan i , amire A belekerül $Y(i)$ -be

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

- Ekkor I-re minden F^+ -beli függőség teljesül
- I-re viszont nem teljesül $X \rightarrow A$

Példa: Attribútumhalmaz lezárása

$R = ABCDEFG, \{AB \rightarrow C, B \rightarrow G, CD \rightarrow EG, BG \rightarrow E\}$

$X = ABF, X^+ = ?$

$$X(0) := ABF$$

$$X(1) := ABF \cup \{C, G\} = ABCFG$$

$$X(2) := ABCFG \cup \{C, G, E\} = ABCEFG$$

$$X(3) := ABCEFG$$

$$X^+ = ABCEFG$$

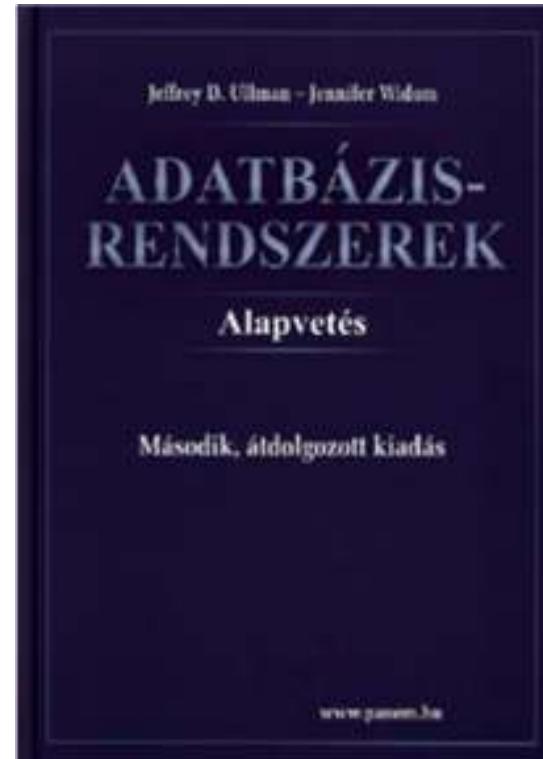
Tankönyv 3.5.2. feladata (111.o.)

- **Órarend adatbázis:** Kurzus(**K**), Oktató(**O**), Időpont(**I**), Terem(**T**), Diák(**D**), Jegy(**J**)
- **Feltételek:**
 - Egy kurzust csak egy oktató tarthat: **K**→**O**.
 - Egy helyen egy időben egy kurzus lehet: **IT**→**K**.
 - Egy időben egy tanár csak egy helyen lehet: **IO**→**T**.
 - Egy időben egy diák csak egy helyen lehet: **ID**→**T**.
 - Egy diák egy kurzust egy végső jeggyel zár: **KD**→**J**.
- **R=KOITDJ F=** {**K**→**O**, **IT**→**K**, **IO**→**T**, **ID**→**T**, **KD**→**J**}
- **Feladat:** Határozzuk meg a (**R**, **F**) kulcsait
az **X⁺ kiszámítási algoritmusa** segítségével.

Relációs adatbázisok tervezése ---2

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009

-
- 3.2.8. Funkcionális függ-ek vetítése
 - 3.3.3. Boyce-Codd normálforma
 - 3.3.4. BCNF-ra való felbontás
 - Folyt. 3.4. Információ visszanyerése
a komponensekből. Chase-teszt
a veszteségmentesség ellenőrzésére



FF-i halmaz vetülete (definíció)

- Tegyük fel, hogy adott az R reláció egy F funkcionális függőségi halmazzal.
- Vegyük R egy vetítését L-re: $R_1 = \Pi_L(R)$, ahol L az R reláció sémájának néhány attribútuma.
- Mely függőségek állnak fenn a vetületben?
- Erre a választ az **F funkcionális függőségek L-re való vetülete** adja, azok a függőségek, amelyek
 - (1) az F-ből levezethetők és
 - (2) csak az L attribútumait tartalmazzák.

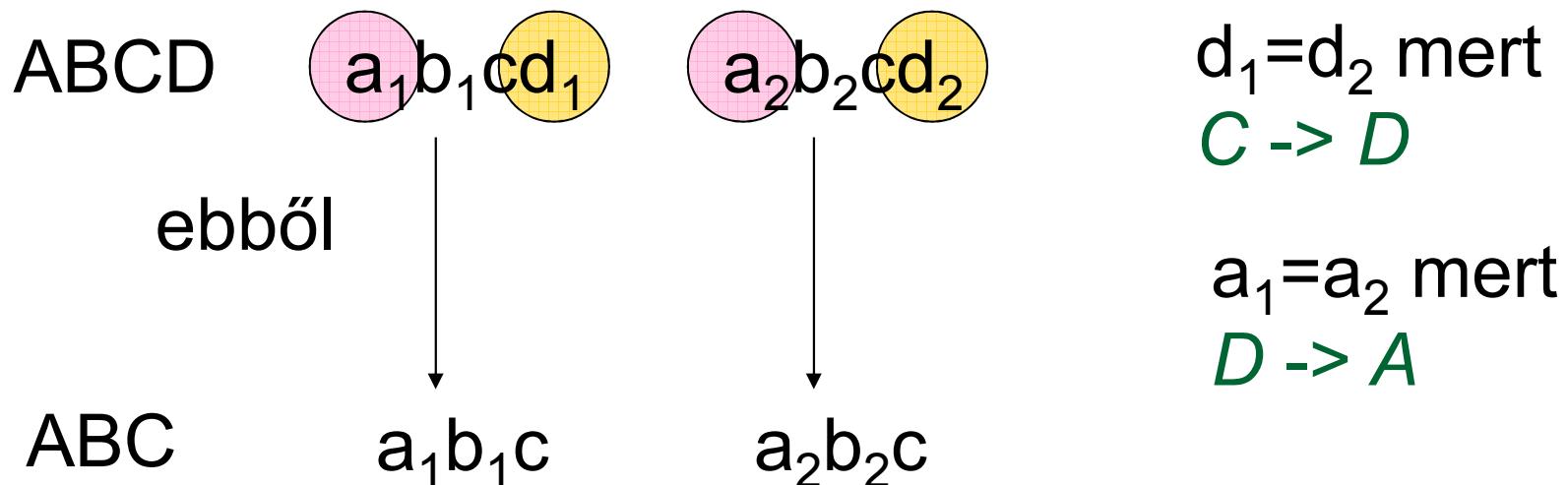
FF-ek vetítése

- Motiváció: „normalizálás”, melynek során egy reláció sémát több sémára bonthatunk szét.
- Definíció: Függőségek vetülete
 - Adott (R, F) , és $R_i \subseteq R$ esetén:
 $\Pi_{R_i}(F) := \{ X \rightarrow Y \mid F \vdash X \rightarrow Y, XY \subseteq R_i \}$
- Példa: $R = ABCD$ $F = \{ AB \rightarrow C, C \rightarrow D, D \rightarrow A \}$
 - Bontsuk fel R -et $R_1 = ABC$ és $R_2 = AD$ -re.
 - Milyen FF-k teljesülnek $R_1 = ABC$ -n?
 - ABC -n nemcsak az $AB \rightarrow C$, de a $C \rightarrow A$ is teljesül!

Miért igaz az előző példa?

Példa: $R=ABCD$ $F=\{AB \rightarrow C, C \rightarrow D, D \rightarrow A\}$

$d=\{ABC, AD\}$. Milyen FF-k teljesülnek ABC -n?



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

Ezért ABC -n az $AB \rightarrow C$ és a $C \rightarrow A$ is teljesül!

Boyce-Codd normálforma

- Definíció: R reláció Boyce-Codd normálformában, BCNF-ban (BCNF) van, ha
 - minden $X \rightarrow Y$ nemtriviális FF-re R -ben (nemtriviális, vagyis Y nem része X -nek)
 - az X szuperkulcs (szuperkulcs, vagyis tartalmaz kulcsot).

Példa BCNF-ra

Sörivók(név, cím, kedveltSörök, gyártó,
aKedvencSör)

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

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

egy másik példa BCNF-ra

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.

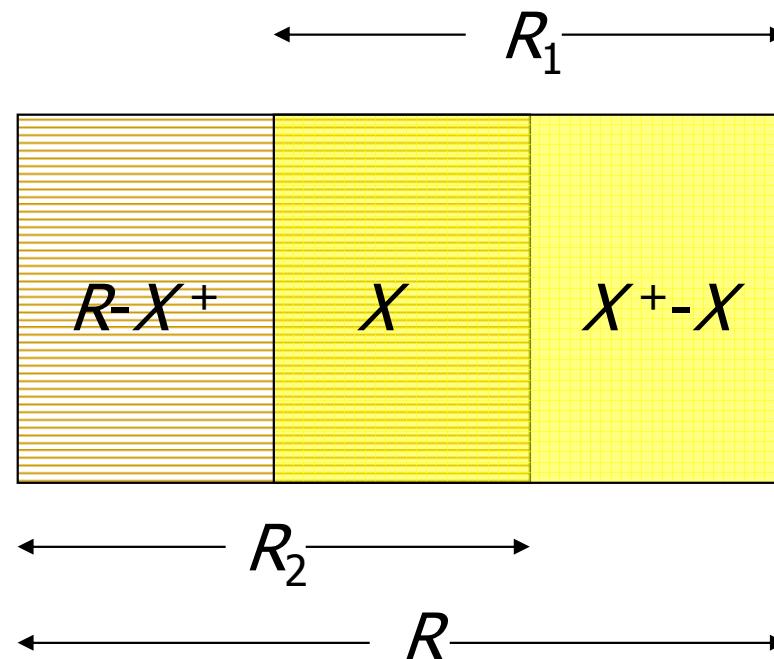
BCNF-re való felbontás ---1

- 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.

BCNF-re való felbontás ---2

- 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 ---1

Sörivók(név, cím, kedveltSörök, gyártó,
aKedvencSör)

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

- Végyük $\text{név} \rightarrow \text{cím}$ FF-t:
- $\{\text{név}\}^+ = \{\text{név}, \text{cím}, \text{aKedvencSör}\}$.
- A dekomponált relációsémák:
 1. Sörivók1(név, cím, aKedvencSör)
 2. Sörivók2(név, kedveltSörök, gyártó)

Példa: BCNF dekompozíció ---2

- Meg kell néznünk, hogy az Sörivók1 és Sörivók2 táblák BCNF-ben vannak-e.
- Az FF-ek projektálása könnyű.
- A Sörivók1(név, cím, aKedvencSör), az FF-ek név->cím és név->aKedvencSör.
- Tehát az egyetlen kulcs: {név}, azaz Sörivók1 relációséma BCNF-ben van.

Példa: BCNF dekompozíció ---3

- Az Sörivók2(név, kedveltSörök, gyártó) esetén az egyetlen FF: kedveltSörök->gyártó, az egyetlen kulcs: {név, kedveltSörök}.
 - Sérül a BCNF.
- $\text{kedveltSörök}^+ = \{\text{kedveltSörök}, \text{gyártó}\}$, a Sörivók2 felbontása:
 1. Sörivók3(kedveltSörök, gyártó)
 2. Sörivók4(név, kedveltSörök)

Példa: BCNF dekompozíció ---4

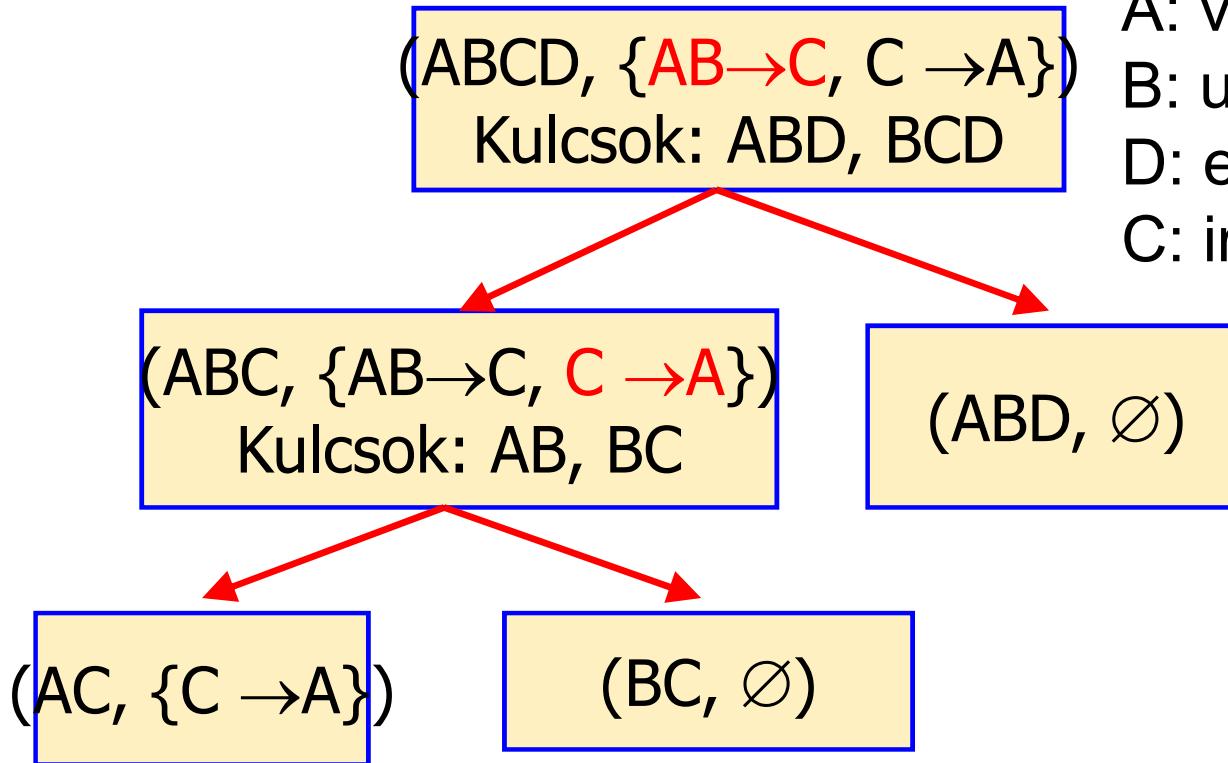
- Az Sörivók dekompozíciója tehát:
 1. Sörivók1(név, cím, aKedvencSör)
 2. Sörivók 3(kedveltSörök, gyártó)
 3. Sörivók 4(név, kedveltSörök)
- A Sörivók1 a sörivókról, a Sörivók3 a sörökről, az Sörivók4 a sörivók és kedvelt söreikről tartalmaz információt.

Miért működik a BCNF?

- **Feladat-1:** Az algoritmus befejeződik, mert legrosszabb esetben két attribútumból álló sémaig jutunk. Bebizonyítandó, hogy minden két attribútumú séma BCNF-ban van! (mert nincs olyan FF, ami sértené a BCNF definíciót)
- **Feladat-2:** A felbontás jó tulajdonágú, vagyis veszteségmentes felbontást ad, visszatérünk erre: Bizonyítsuk be, hogy ha $R(A, B, C)$ reláció esetén $B \rightarrow C$ teljesül, akkor $R_1(A, B)$, $R_2(B, C)$ felbontás minden veszteségmentes

Példa: BCNF-ra való felbontás

$R=ABCD$, $F=\{AB \rightarrow C, C \rightarrow A\}$



Például R : lakcím
A: város, kerület
B: utca, házszám
D: emelet, ajtó
C: irányítószám

Tehát $d=(AC, BC, ABD)$ veszteségmentes BCNF dekompozíció.
(\emptyset azt jelenti, hogy csak a triviális függőségek teljesülnek a sémában.)

Tankönyv 3.5.2. feladata (111.o.)

- **Órarend adatbázis:** Kurzus(K), Oktató(O), Időpont(I), Terem(T), Diák(D), Jegy(J)

- **Feltételek:**

Egy kurzust csak egy oktató tarthat: $K \rightarrow O$.

Egy helyen egy időben egy kurzus lehet: $IT \rightarrow K$.

Egy időben egy tanár csak egy helyen lehet: $IO \rightarrow T$.

Egy időben egy diák csak egy helyen lehet: $ID \rightarrow T$.

Egy diák egy kurzust egy végső jeggyel zár: $KD \rightarrow J$.

- **R=KOITDJ F=** $\{K \rightarrow O, IT \rightarrow K, IO \rightarrow T, ID \rightarrow T, KD \rightarrow J\}$
- **Feladat:** Adjuk meg az algoritmussal egy BCNF dekompozícióját!

Relációs adatbázisok tervezése ---2b

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009

3.4. Információ visszanyerése
a komponensekből. Chase-teszt
a veszteségmentesség ellenőrzésére



Felbontásra vonatkozó elvárások

➤ Elvárások

- (1) **A vetületek** legyenek jó tulajdonságúak, és a vetületi függőségi rendszere egyszerű legyen (normálformák: BCNF, 3NF, 4NF)
 - (2) **Veszteségmentes** legyen a felbontás, vagyis ne legyen információvesztés
 - (3) **Függőségek megőrzése** a vetületekben (FŐ)
- BCNF-ra való felbontás algoritmusa
- minden veszteségmentes felbontást ad
 - De nem feltétlen függőségőrző a felbontás

Veszteségmentes szétvágás ---1

- A fenti jelölésekkel: ha $r = \Pi_{R_1}(r) \bowtie \dots \bowtie \Pi_{R_k}(r)$ teljesül, akkor az előbbi összekapcsolásra azt mondjuk, hogy **veszteségmentes**. Itt r egy R sémájú reláció-előfordulást jelöl.

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	

Veszteségmentes szétvágás ---2

- Megjegyzés: Könnyen belátható, hogy $r \subseteq \Pi_{R_1}(r) \bowtie \dots \bowtie \Pi_{R_k}(r)$ mindenkorban teljesül.
- Példa: itt 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 VM ellenőrzése ---1

- 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?
- Vagyuk $R_1 \bowtie R_2 \bowtie R_3$ egy $t = (a, b, c, d)$ sorát. Bizonyítani kell, hogy $t R$ egy sora. A következő tablót készítjük:

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

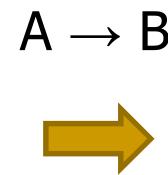
Itt pl. az (a, b_1, c_1, d) sor azt jelzi, hogy R -nek van olyan sora, aminek R_1 -re való levetítése (a, d) , ám ennek a B és C attribútumokhoz tartozó értéke ismeretlen, így egyáltalán nem biztos, hogy a t sorról van szó.

Chase-teszt VM ellenőrzése ---2

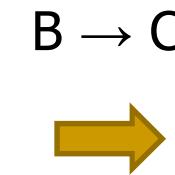
- Az F-beli függőségeket használva egyenlővé tesszük azokat a szimbólumokat, amelyeknek ugyanazoknak kell lennie, hogy valamelyik függőség ne sérüljön.
- Ha a két egyenlővé teendő szimbólum közül az egyik index nélküli, akkor a másik is ezt az értéket kapja.
- Két indexes szimbólum esetén a kisebbik 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 VM ellenőrzése ---3

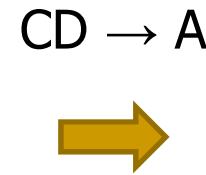
A	B	C	D
a	b ₁	c ₁	d
a	b ₂	c	d ₂
a ₃	b	c	d



A	B	C	D
a	b ₁	c ₁	d
a	b ₁	c	d ₂
a ₃	b	c	d



A	B	C	D
a	b ₁	c	d
a	b ₁	c	d ₂
a ₃	b	c	d



A	B	C	D
a	b ₁	c	d
a	b ₁	c	d ₂
a	b	c	d

Chase-teszt VM ellenőrzése ---5

- 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.

Relációs adatbázisok tervezése ---3

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009

3.2.7. Funkcionális függőségi
halmazok lezárása (min.bázis)

3.4.4. Függőségek megőrzése

3.5. Harmadik normálforma és
3NF-szintetizáló algoritmus



Relációs sématervezés (vázlat)

- **I. Függőségek:** a sématervezésnél használjuk
 - Funkcionális függőség
 - Többértékű függőség
- **II. Normalizálás:** „ jó” sémákra való felbontás
 - Funkcionális függőség -> BCNF
 - Funkcionális függőség -> 3NF
 - Többértékű függőség -> 4NF
- **III. Felbontás tulajdonságai:** „ jó” tulajdonságok
 - Veszeségmentesség
 - Függőségőrző felbontás

Függőségek megőrzése

- Függőségőrző felbontás: a dekompozíciókban érvényes függőségekből következzen az eredeti sémára kirótt összes függőség.
- Milyen függőségek lesznek érvényesek a dekompozíció sémáiban?
- Példa: $R=ABC$, $F= \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$ vajon a $d= (AB, BC)$ felbontás megőrzi-e a $C \rightarrow A$ függőséget?

Függőségek megőrzése (definíció)

- Definíció: Függőségek vetülete

Adott (R, F) , és $R_i \subseteq R$ esetén:

$$\Pi_{R_i}(F) := \{ X \rightarrow Y \mid F \vdash X \rightarrow Y, XY \subseteq R_i \}$$

- Definíció: Adott (R, F) esetén $d = (R_1, \dots, R_k)$ függőségőrző dekompozíció akkor és csak akkor, ha minden F -beli függőség levezethető a vetületi függőségekből:

minden $X \rightarrow Y \in F$ esetén

$$\Pi_{R_1}(F) \cup \dots \cup \Pi_{R_k}(F) \vdash X \rightarrow Y$$

Példa: függőségek vetülete

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

Nézzük meg az AC -re való vetületet:

- $A^+ = ABC$; 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.
- A kapott FF-ek: $A \rightarrow B$, $A \rightarrow C$ és $B \rightarrow C$.
- AC -re projekció: $A \rightarrow C$.

Függőségek megőrzése (tételek)

- A függőségőrzésből nem következik a veszteségmentesség:

$R=ABCD$, $F= \{A \rightarrow B, C \rightarrow D\}$, $d=\{AB, CD\}$
függőségőrző, de nem veszteségmentes.

- A veszteségmentességből nem következik a függőségőrzés

$R=ABC$, $F= \{AB \rightarrow C, C \rightarrow A\}$, $d=\{AC, BC\}$
veszteségmentes, de nem függőségőrző.

A 3normálforma -- motiváció

- Bizonyos FF halmazok esetén a felbontáskor elveszíthetünk függőségeket.
- $AB \rightarrow C$ és $C \rightarrow B$.
 - Példa1: $A = \text{utca}$, $B = \text{város}$, $C = \text{irányítószám}$.
 - Példa2: $A = \text{oktató}$, $B = \text{időpont}$, $C = \text{kurzus}$.
- Két kulcs van: $\{A,B\}$ és $\{A,C\}$.
- $C \rightarrow B$ megsérte a BCNF-t, tehát AC , BC -re dekomponálunk.
- A probléma az, hogy AC és BC sémákkal nem tudjuk kikényszeríteni $AB \rightarrow C$ függőséget.

A probléma megoldása: 3NF

- 3. normálformában (3NF) úgy módosul a BCNF feltétel, hogy az előbbi esetben nem kell dekomponálnunk.
- Egy attribútum **elsődleges attribútum (prím)**, ha legalább egy kulcsnak eleme.
- $X \rightarrow A$ megsérte 3NF-t akkor és csak akkor, ha X nem szuperkulcs és A nem prím.

Példa: 3NF

- Az előző példában $AB \rightarrow C$ 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, de a 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.
- Az (1) tulajdonság teljesül a BCNF esetében.
- A 3NF (1) és (2)-t is teljesíti.
- A BCNF esetén (2) sérülhet (*utca-város-irszám*)

Tk.3.2.7. Minimális bázis (definíció)

Egy relációhoz F minimális bázis, amikor az olyan függőségekből áll, amelyre három feltétel igaz:

1. F összes függőségének jobb oldalán egy attribútum van.
2. Ha bármelyik F-beli függőséget elhagyjuk, a fennmaradó halmaz már nem bázis.
3. Ha bármelyik F-beli funkcionális függőség bal oldaláról elhagyunk egy vagy több attribútumot, akkor az eredmény már nem marad bázis.

Minimális bázist kiszámító algoritmus

1. Kezdetben G az üreshalmaz.
2. minden $X \rightarrow Y \in F$ helyett vegyük az $X \rightarrow A$ függőségeket, ahol $A \in Y - X$).
Megj.: Ekkor minden G -beli függőség $X \rightarrow A$ alakú.
3. minden $X \rightarrow A \in G$ -re, amíg van olyan $B \in X$ -re $A \in (X - B)^+$ a G -szerint, vagyis $(X - B) \rightarrow A$ teljesül, akkor $X := X - B$.
Megjegyzés: E lépés után minden baloldal minimális lesz.
4. 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

Mohó algoritmus minimális bázis előállítására

1. Jobb oldalak minimalizálása:

$X \rightarrow A_1, \dots, A_k$ függőséget cseréljük le az
 $X \rightarrow A_1, \dots, X \rightarrow A_k$ k darab függőségre.

2. A halmaz minimalizálása:

Hagyjuk el az olyan $X \rightarrow A$ függőségeket, amelyek a bázist nem befolyásolják, azaz
while F változik

if $(F - \{X \rightarrow A\})^* = F^*$ then $F := F - \{X \rightarrow A\}$;

3. Bal oldalak minimalizálása:

Hagyjuk el a bal oldalaktól azokat az attribútumokat, amelyek a bázist nem befolyásolják, azaz
while F változik

for all $X \rightarrow A \in F$

for all $B \in X$

if $((F - \{X \rightarrow A\}) \cup \{(X - \{B\}) \rightarrow A\})^* = F^*$ then $F := (F - \{X \rightarrow A\}) \cup \{(X - \{B\}) \rightarrow A\}$

Normálformák (3NF)

- Az algoritmusban különböző sorrendben választva a függőségeket, illetve attribútumokat, különböző minimális bázist kaphatunk.
- $F = \{A \rightarrow B, B \rightarrow A, B \rightarrow C, A \rightarrow C, C \rightarrow A\}$
 $(F - \{B \rightarrow A\})^* = F^*$, mivel $F - \{B \rightarrow A\} \vdash B \rightarrow A$
 $F := F - \{B \rightarrow A\}$
 $(F - \{A \rightarrow C\})^* = F^*$, mivel $F - \{A \rightarrow C\} \vdash A \rightarrow C$
 $F := F - \{A \rightarrow C\} = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$ **minimális bázis**, mert több függőség és attribútum már nem hagyható el.
- $F = \{A \rightarrow B, B \rightarrow A, B \rightarrow C, A \rightarrow C, C \rightarrow A\}$
 $(F - \{B \rightarrow C\})^* = F^*$, mivel $F - \{B \rightarrow C\} \vdash B \rightarrow C$
 $F := F - \{B \rightarrow C\} = \{A \rightarrow B, B \rightarrow A, A \rightarrow C, C \rightarrow A\}$ **is minimális bázis**, mert több függőség és attribútum már nem hagyható el.

Normálformák (3NF)

- Az algoritmusban különböző sorrendben választva a függőségeket, illetve attribútumokat, különböző minimális bázist kaphatunk.
- $F = \{AB \rightarrow C, A \rightarrow B, B \rightarrow A\}$
 $(F - \{AB \rightarrow C\} \cup \{A \rightarrow C\})^* = F^*$, mivel
 $(F - \{AB \rightarrow C\} \cup \{A \rightarrow C\}) \vdash AB \rightarrow C$ és $F \vdash A \rightarrow C$.
 $F := (F - \{AB \rightarrow C\} \cup \{A \rightarrow C\}) = \{A \rightarrow C, A \rightarrow B, B \rightarrow A\}$ **minimális bázis**, mert több függőség és attribútum már nem hagyható el.
- $F = \{AB \rightarrow C, A \rightarrow B, B \rightarrow A\}$
 $(F - \{AB \rightarrow C\} \cup \{B \rightarrow C\})^* = F^*$, mivel
 $(F - \{AB \rightarrow C\} \cup \{B \rightarrow C\}) \vdash AB \rightarrow C$ és $F \vdash B \rightarrow C$.
 $F := (F - \{AB \rightarrow C\} \cup \{B \rightarrow C\}) = \{B \rightarrow C, A \rightarrow B, B \rightarrow A\}$ **is minimális bázis**, mert több függőség és attribútum már nem hagyható el.

Normálformák (3NF)

- Algoritmus függőségőrző 3NF dekompozíció előállítására:
- Input: (R, F)
 - Legyen $G := \{X \rightarrow A, X \rightarrow B, \dots, Y \rightarrow C, Y \rightarrow D, \dots\}$ az F minimális bázisa.
 - Legyen S az R sémának G -ben nem szereplő attribútumai.
 - Ha van olyan függőség G -ben, amely R összes attribútumát tartalmazza, akkor legyen $d := \{R\}$, különben legyen $d := \{S, XA, XB, \dots, YC, YD, \dots\}.$

Normálformák (3NF)

- Algoritmus függőségőrző és **veszteségmentes** 3NF dekompozíció előállítására:
- Input: (R, F)
 - Legyen $G := \{X \rightarrow A, X \rightarrow B, \dots, Y \rightarrow C, Y \rightarrow D, \dots\}$ az F minimális bázisa.
 - Legyen S az R sémának G -ben nem szereplő attribútumai.
 - Ha van olyan függőség G -ben, amely R összes attribútumát tartalmazza, akkor legyen $d := \{R\}$, különben legyen K az R egy kulcsa, és legyen $d := \{K, S, XA, XB, \dots, YC, YD, \dots\}$.

Normálformák (3NF)

- Algoritmus függőségőrző és veszteségmentes 3NF redukált (kevesebb tagból álló) dekompozíció előállítására:
- Input: (R, F)
 - Legyen $G := \{X \rightarrow A, X \rightarrow B, \dots, Y \rightarrow C, Y \rightarrow D, \dots\}$ az F minimális bázisa.
 - Legyen S az R sémának G -ben nem szereplő attribútumai.
 - Ha van olyan függőség G -ben, amely R összes attribútumát tartalmazza, akkor legyen $d := \{R\}$, különben legyen K az R egy kulcsa, és legyen $d := \{K, S, XAB, \dots, YCD, \dots\}$.
 - Ha K része valamelyik sémának, akkor K -t elhagyhatjuk.

Miért működik?

3NF-szintetizáló algoritmus:

- 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 tulajdonságainak következik.

Ellenőrző kérdések

- Adott X-attr.hz, F-ff.hz. Attr.hz.lezártjának kiszámítása.
- Adott R-rel.séma, F-ff.hz. Kulcsok meghatározása.
- Adott R-rel.séma, F-ff.hz. BCNF-e? (def. alapján)
- Adott R-rel.séma, F-ff.hz. 3NF-e? (def. alapján)

típusú kérdésekhez lásd 1.) gyakorló feladatok :

Ellenőrző kérdések

1.) Adott R relációs séma és F funkcionális függőségek halmaza. Attribútum halmaz lezártjának kiszámolására tanult algoritmus felhasználásával határozza meg az adott séma kulcsait, és azt, hogy BCNF-ben vagy 3NF-ben van-e?

a.) Cím(Város, Utcahsz, Irányítószám)
röviden $R(V, U, I)$, és
 $F = \{I \rightarrow V, VU \rightarrow I\}$.

Ellenőrző kérdések

b.) Tankönyv 3.5.2. feladata: Órarend adatbázis

Jelölje röviden:

K - Kurzus

O - Oktató

I - Időpont

T - Terem

D - Diák (hallgató)

J - Jegy

Feltételek (funkcionális függőséggel megadva)

$K \rightarrow O$ vagyis egy kurzust csak egy oktató tarthat

$IT \rightarrow K$ nincs óraütközés, egy helyen egy időben egy kurzus lehet

$IO \rightarrow T$ az oktatónak nincs óraütközése

$ID \rightarrow T$ a diákoknak sincs óraütközése

$KD \rightarrow J$ egy diák egy kurzust egy végső jeggyel zár

$R = KOITDJ$ és $F = \{K \rightarrow O, IT \rightarrow K, IO \rightarrow T, ID \rightarrow T, KD \rightarrow J\}$

Ellenőrző kérdések

c.) Adott SzallításInfo (SzallAzon, SzallNev, SzallCim, AruKod, TermekNev, MeEgys, Ar) reláció séma,

amit így is rövidíthetünk $R(S, N, C, K, T, M, A)$, és

a séma feletti funkcionális függőségek:

$SzallAzon \rightarrow \{SzallNev, SzallCim\}$,

$AruKod \rightarrow \{TermekNev, MeEgys\}$,

$\{SzallAzon, AruKod\} \rightarrow Ar$,

vagyis a röviden $F = \{S \rightarrow NC, K \rightarrow TM, SK \rightarrow A\}$.

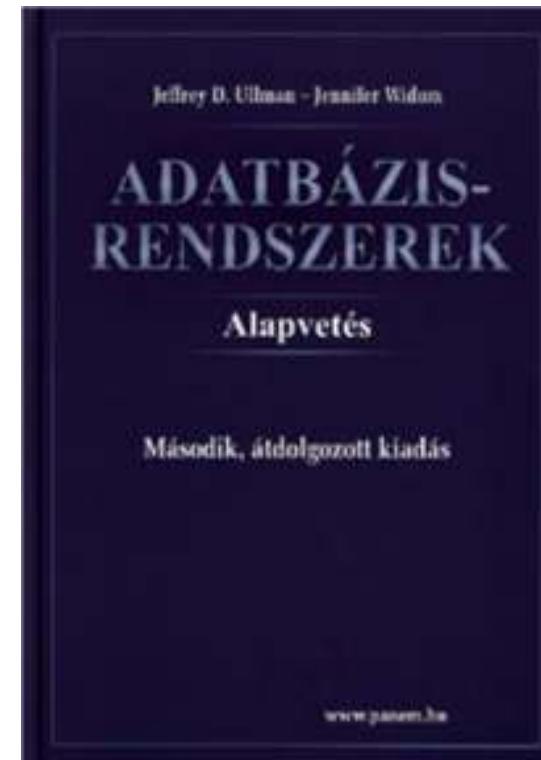
Ellenőrző kérdések

- Adott R, F. Bontsuk fel VM BCNF-ra
- Adott R, F. Bontsuk fel VM FŐ 3NF-ra
- Adott R, F és d dekompozíció. Chase algoritmussal döntsük el, hogy veszteségmentes-e a dekompozíció.
 - a.) Az 1a. feladat R sémáját szétvágjuk IU, VU sémákra.
 - b.) Az 1b. feladat R sémáját szétvágjuk KOIT, IDT, KDJ sémákra.
 - c.) Az 1c. feladat R sémáját szétvágjuk SNC, KTMA sémákra.

Relációs adatbázisok tervezése

4.rész Többértékű függőségek

Tankönyv: Ullman-Widom:
Adatbázisrendszerek Alapvetés
Második, átdolgozott kiadás,
Panem, 2009



3.6. Többértékű függőségek,
- Negyedik normálforma
- Funkcionális és többértékű
függőségek következtetése

Többértékű függőségek és 4NF

- Hasonló utat járunk be, mint a funkcionális függőségek esetén:
 - Definiáljuk a többértékű függőséget
 - implikációs probléma
 - axiomatizálás
 - levezethető függőségek hatékony meghatározása
(lezárás helyett a séma particiója függőségi bázisa)
 - veszteségmentes dekompozíció
 - 4. normálforma
 - veszteségmentes **4NF** dekompozíció előállítása

A TÉF definíciója

- A *többértékű függőség* (TÉF): az R reláció fölött $X \rightarrow\!\!> Y$ teljesül: ha bármely két sorra, amelyek megegyeznek az X minden attribútumán, az Y attribútumaihoz tartozó értékek felcserélhetőek, 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

Sörivók(név, cím, tel, kedveltSörök)

- A sörivók telefonszámai függetlenek az általuk kedvelt söröktől.
 - név->->tel és név ->->kedveltSörök.
- Így egy-egy sörivó minden telefonszáma minden általa kedvelt sörrrel kombinációban áll.
- Ez a jelenség független a funkcionális függőségektől.
 - itt a név->cím az egyetlen FF.

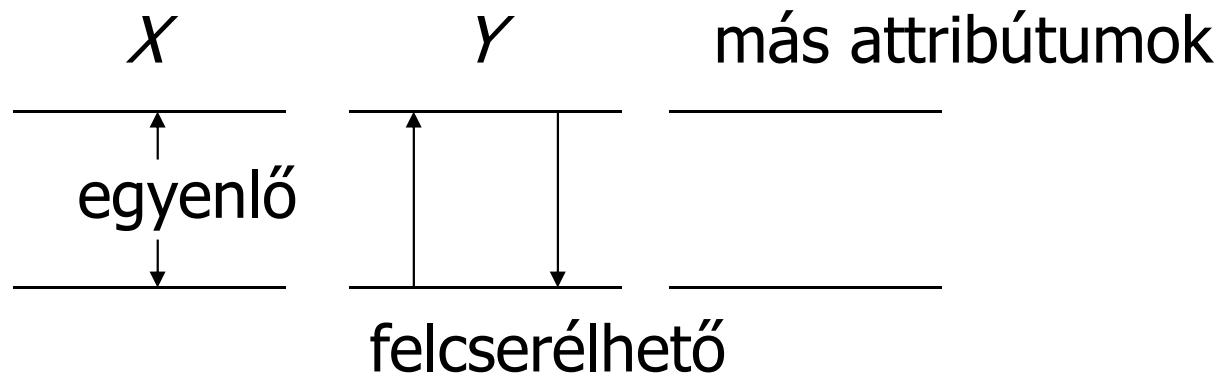
A név->->tel által implikált sorok

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



Többértékű függőségek

- Definíció: $X, Y \subseteq R$, $Z := R - XY$ esetén $X \rightarrow \rightarrow Y$ többértékű függőség. (tf)
- A függőség akkor teljesül egy táblában, ha bizonyos mintázú sorok létezése garantálja más sorok létezését.
- A formális definiciót az alábbi ábra szemlélteti.
- Ha létezik t és s sor, akkor u és v soroknak is létezniük kell, ahol az azonos szimbólumok azonos értékeket jelölnek.

	X	Y	Z
t	x	y1	
s	x	y2	
$\exists u$	x	y1	
$\exists v$	x	y2	

Többértékű függőségek

Definíció (Formálisan): Egy R sémájú r reláció kielégíti az $X \rightarrow\rightarrow Y$ függőséget, ha $t, s \in r$ és $t[X] = s[X]$ esetén létezik olyan $u, v \in r$, amelyre $u[X] = v[X] = t[X] = s[X]$, $u[Y] = t[Y]$, $u[Z] = s[Z]$, $v[Y] = s[Y]$, $v[Z] = t[Z]$.

Állítás: Elég az u, v közül csak az egyik létezését megkövetelni.

	X	Y	Z
t	x	y1	z1
s	x	y2	z2
$\exists u$	x	y1	z2

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 jobboldalakt felbonthattuk, míg ebben az esetben ez sem tehető meg.

Példa: többattribútumos jobboldal

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

- Egy sörivónak több telefonja lehet, minden számot két részre otsztunk: tTársaság (pl. Vodafone) és a maradék hét számjegy.
- Egy sörivő 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 FÉK-ek teljesülnek:

név ->-> tTársaság tel

név ->-> kedveltSörök gyártó

Példa adat

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

név	tTársaság	tel	kedveltS	gyártó
Sue	30	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.

Többértékű függőségek

➤ Axiomatizálás

Funkcionális függőségek	Többértékű függőségek	Vegyes függőségek
A1 (reflexivitás): $Y \subseteq X$ esetén $X \rightarrow Y$.	A4 (komplementer): $X \rightarrow \rightarrow Y$ és $Z = R - XY$ esetén $X \rightarrow \rightarrow Z$.	A7 (funkcionálisból többértékű): $X \rightarrow Y$ esetén $X \rightarrow \rightarrow Y$.
A2 (tranzitivitás): $X \rightarrow Y$ és $Y \rightarrow Z$ esetén $X \rightarrow Z$.	A5 (tranzitivitás): $X \rightarrow \rightarrow Y$ és $Y \rightarrow \rightarrow S$ esetén $X \rightarrow \rightarrow S - Y$.	A8 (többértékűből és funkcionálisból funkcionális): $X \rightarrow \rightarrow Y$ és $W \rightarrow S$, ahol $S \subseteq Y$, $W \cap Y = \emptyset$ esetén $X \rightarrow S$.
A3 (bővíthetőség): $X \rightarrow Y$ és tetszőleges Z esetén $XZ \rightarrow YZ$.	A6 (bővíthetőség): $X \rightarrow \rightarrow Y$ és tetszőleges $V \subseteq W$ esetén $XW \rightarrow \rightarrow YV$.	

Többértékű függőségek

- Jelölés a továbbiakban:
 - F funkcionális függőségek halmaza
 - M többértékű függőségek halmaza
 - D vegyes függőségek (funkcionális és többértékű függőségek) halmaza
- Tétel (helyes és teljes axiómarendszerek):
 - A1,A2,A3 helyes és teljes a funkcionális függőségekre,
 - A4,A5,A6 helyes és teljes a többértékű függőségekre,
 - A1,A2,A3,A4,A5,A6,A7,A8 helyes és teljes a vegyes függőségekre.

Többértékű függőségek

- Állítás: $X \rightarrow\rightarrow Y$ -ből nem következik, hogy $X \rightarrow\rightarrow A$, ha $A \in Y$. (A jobb oldalak nem szedhetők szét!)
- Bizonyítás: A következő r tábla kielégíti az $X \rightarrow\rightarrow AB$ -t, de nem elégíti ki az $X \rightarrow\rightarrow A$ -t. q.e.d.

$X \rightarrow\rightarrow A$ esetén
ennek a
sornak is
benne kellene
lenni a
táblában.

X	A	B	C
x	a	b	c
x	e	f	g
x	a	b	g
x	e	f	c

x	a	f	g
---	---	---	---

Többértékű függőségek

- Állítás: $X \rightarrow \rightarrow Y$ és $Y \rightarrow \rightarrow V$ -ből nem következik, hogy $X \rightarrow \rightarrow V$.
(A szokásos tranzitivitás nem igaz általában!)
- Bizonyítás: A következő r tábla kielégíti az $X \rightarrow \rightarrow AB$ -t, $AB \rightarrow \rightarrow BC$ -t, de nem elégíti ki az $X \rightarrow \rightarrow BC$ -t. q.e.d.

X	A	B	C
x	a	b	c
x	e	f	g
x	a	b	g
x	e	f	c

$X \rightarrow \rightarrow BC$ esetén ennek a sornak is benne kellene lenni a táblában.

x	e	b	c
---	---	---	---

Többértékű függőségek

- A veszteségmentesség, függőségőrzés definíciójában most F funkcionális függőségi halmaz helyett D függőségi halmaz többértékű függőségeket is tartalmazhat.
- Így például $d=(R_1, \dots, R_k)$ veszteségmentes dekompozíciója R -nek D -re nézve, akkor és csak akkor, ha minden D -t kielégítő r tábla esetén $r = \Pi_{R_1}(r) |><| \dots |><| \Pi_{R_k}(r)$
- A következő téTEL miatt a veszteségmentesség implikációs problémára vezethető vissza, így hatékonyan eldönthető.
- TÉTEL: A $d=(R_1, R_2)$ akkor és csak akkor veszteségmentes dekompozíciója R -nek, ha
 $D \vdash R_1 \cap R_2 \rightarrow\rightarrow R_1 - R_2$.

Többértékű függőségek

- A 4.normálforma definiálása előtt foglaljuk össze, hogy melyek a triviális többértékű függőségek, vagyis amelyek minden relációban teljesülnek.
- Mivel minden funkcionális függőség többértékű függőség is, így a triviális funkcionális egyben triviális többértékű függőség is.
 1. $Y \subseteq X$ esetén $X \rightarrow \rightarrow Y$ triviális többértékű függőség.
 - Speciálisan $Y = \emptyset$ választással $X \rightarrow \rightarrow \emptyset$ függőséget kapjuk, és alkalmazzuk a komplementer szabályt, azaz $Z = R - X \emptyset$, így az $X \rightarrow \rightarrow R - X$ függőség is minden teljesül, azaz:
 2. $XY = R$ esetén $X \rightarrow \rightarrow Y$ triviális többértékű függőség.
 - A szuperkulcs, kulcs definíciója változatlan, azaz X szuperkulcsa R -nek D -re nézve, ha $D \vdash X \rightarrow R$.
 - A minimális szuperkulcsot **kulcsnak** hívjuk.

Többértékű függőségek

- A 4.normálforma hasonlít a BCNF-re, azaz minden nem triviális többértékű függőség bal oldala szuperkulcs.
- Definíció: R **4NF**-ben van D-re nézve, ha $XY \neq R$, $Y \not\subset X$, és

$$D \vdash X \rightarrow Y \text{ esetén } D \vdash X \rightarrow R.$$

- Definíció: $d = \{R_1, \dots, R_k\}$ dekompozíció **4NF**-ben van D-re nézve, ha minden R_i **4NF**-ben van $\Pi_{R_i}(D)$ -re nézve.
- Állítás: Ha R **4NF**-ben van, akkor **BCNF**-ben is van.
- Bizonyítás. Vegyük egy nem triviális $D \vdash X \rightarrow A$ **funkcionális** függőséget. Ha $XA = R$, akkor $D \vdash X \rightarrow R$, ha $XA \neq R$, akkor a $D \vdash X \rightarrow A$ nem triviális többértékű függőség és a **4NF** miatt $D \vdash X \rightarrow R$. q.e.d.
- Következmény: Nincs minden **függőségőrző** és **veszteségmentes** **4NF** dekompozíció.

Többértékű függőségek

- Veszteségmentes 4NF dekompozíciót minden tudunk készíteni a naiv BCNF dekomponáló algoritmushoz hasonlóan.
- Naiv algoritmus veszteségmentes 4NF dekompozíció előállítására:
 - Ha R 4NF-ben van, akkor megállunk, egyébként
 - van olyan nem triviális $X \rightarrow\!\!\rightarrow Y$, amely R-ben teljesül, de megsérti a 4NF-et, azaz X nem szuperkulcs.
 - Ekkor R helyett vegyük az $(XY, R-Y)$ dekompozíciót.
 - A kettévágásokat addig hajtjuk végre, amíg minden tag 4NF-ben nem lesz.

ALGORITMUS VÉGE.

Többértékű függőségek

- Az is feltehető, hogy X és Y diszjunkt, mert különben Y helyett az Y-X-et vehettük volna jobb oldalnak.
- $XY \neq R$, így *mindkét tagban csökken az attribútumok száma*.
- $XY \cap (R-Y) = X \rightarrow Y = XY - (R-Y)$, azaz a kéttagú dekompozícióknál bizonyított állítás miatt *veszteségmentes kettévágást kaptunk*.
- Legrosszabb esetben a 2 oszlopos sémákig kell szétbontani, amelyek minden 4NF-ben vannak, mivel nem lehet bennük nem triviális többértékű függőség.

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 $X \rightarrow\!\!\!> Y$ nemtriviális FÉK 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 versus 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 $X \rightarrow\!\!\!> Y$ megsérte a 4NF-t, akkor R -t 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

Sörivók(név, cím, tel, kedveltSörök)

FF: név -> cím

FÉK-ek: név ->-> tel
 név ->-> kedveltSörök

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

Példa folytatás

- Dekompozíció név -> cím szerint:
 1. Sörivók1(név, cím)
 - u Ez 4NF-beli; az egyetlen függőség név-> cím.
 2. Sörivók2(név, tel, kedveltSörök)
 - u Nincs 4NF-ben. A név ->-> tel és név ->-> kedveltSörök függőségek teljesülnek. A három attribútum együtt kulcs (mivel nincs nemtriviális FF).

Példa: Sörivók2 dekompozíciója

- Mind a név ->-> tel, mind a név ->-> kedveltSörök szerinti dekompozíció ugyanazt eredményezi:
 - Sörivók3(név, tel)
 - Sörivók4(név, kedveltSörök)