

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;
```

- **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);
```

R

A	B
5	20
10	30
20	40
...	...

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 mennyiségű) 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 sémáját, 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 funkcionalitásainak 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

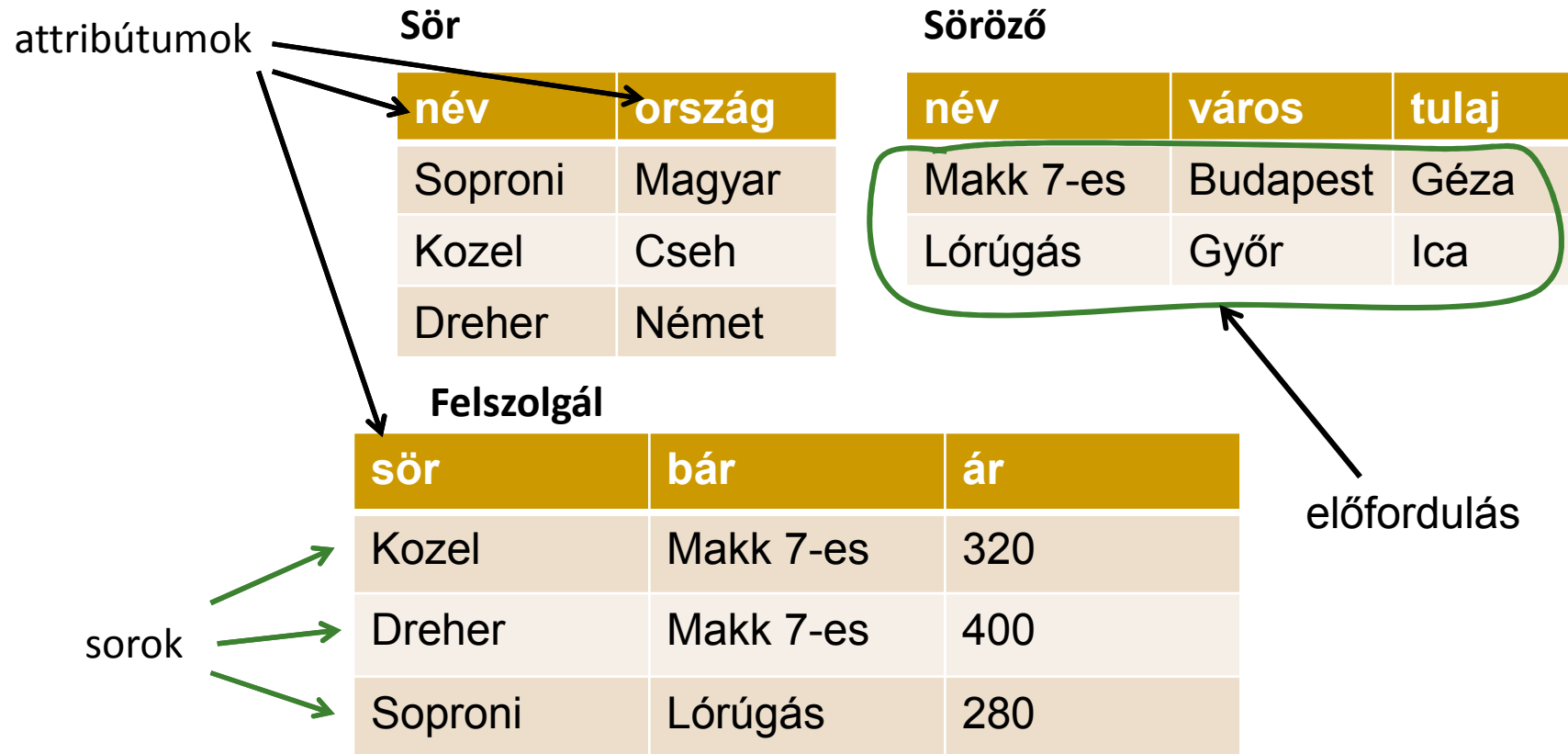
Példa féligstrukturált adatra (XML)

```
<?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, söröző, 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 , SYSDBASE , INGRES, DB2, stb
- Adatbázisok-1 gyakorlaton **ORACLE** adatbázis-kezelő rendszert 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, hatékonysági 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 lekérdezéseknél segítség, például a rekurzív lekérdezéseknél.

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:
<Attribútumnév₁: értéktípus₁, ..., Attrnév_n: értéktípus_n>
röviden **<A₁, ..., A_n>**
- **Relációséma**: Relációnév (sortípus) (itt: kerek zárójelben!)
vagyis **R(Anév₁: értéktípus₁, ..., Anév_n: értéktípus_n)**
röviden **R(A₁, ..., 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: függvénnyel

- A táblázatos szemléltetésből áttérhetünk 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.

Példa 1

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

Példa 2

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

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

Példa 3

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

Példa 4

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

Ebben a modellben Példa 4 nem reláció, de a valóságban megengedünk multihalmazokat 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 **adateleíró 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
 - Nézet táblák 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: sörivők adatbázis

Sörök(név, gyártó)

Sörözők(név, város, tulaj, engedély)

Sörivők(név, város, tel)

Szeret(név, sör)

Felhasználó(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 kapjunk)

➤ **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

- $E := \Pi_{\text{lista}} (E_1)$ 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

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)

Felhasználó(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 kapjunk, azt külön kérni kell
SELECT **DISTINCT** Lista FROM Táblanév

A műveletek szemantikája

név	gyártó
Arany Ászok	Dreher

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

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

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) mindig 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 oszlopnévet 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 oszlopnévek 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 fejezeteit: 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” sörök 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 (értékhalmoz)** egyszerű értékek halmaza

SQL specialitások, nem írhatók át relációs algebraiba:

(--- 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, amelyek 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)) =
MIN(1, MAX(0, (1 - $\frac{1}{2}$))) =
MIN(1, MAX(0, $\frac{1}{2}$)) =
MIN(1, $\frac{1}{2}$) = $\frac{1}{2}$ = 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)

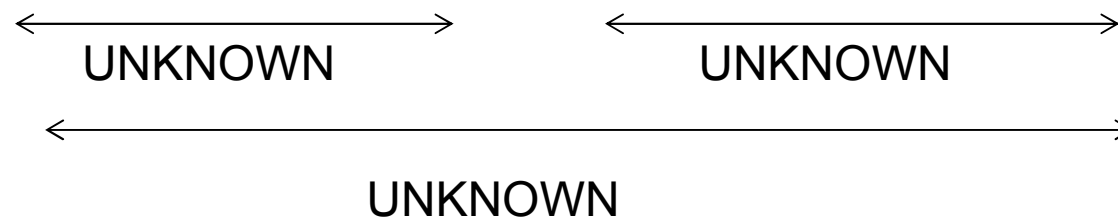
x	y	x AND y	x OR y	NOT x
T	T	T	T	F
T	U	U	T	F
T	F	F	T	F
U	T	U	T	U
U	U	U	U	U
U	F	F	U	U
F	T	F	T	T
F	U	F	U	T
F	F	F	F	T

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>