

# 2.előadás Információs rendszerek

dr. Hajas Csilla (ELTE IK)

<https://sila.hajas.elte.hu>

## Adatbázis-kezelő rendszerek felépítése SQL gyakorlat (a táblák előkészítése)

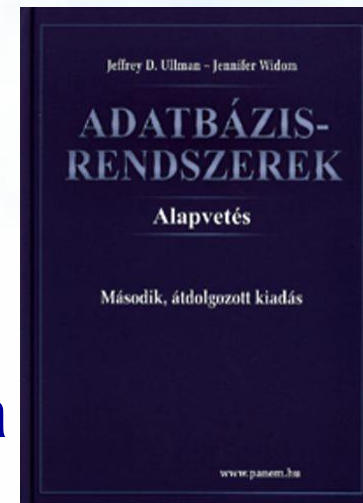
1.fejezet: Az adatbázisrendszerek világa

2.3. Relációsémák definiálása SQL-ben

(Táblák létrehozása és előkészítése)

7.fej. Táblák+megszorítások létrehozása

CREATE TABLE, CONSTRAINTS



# Mit várunk egy ABKR-től?

- Adatbázis: Adatok együttese, amelyet 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.
  - 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)**

# (1) 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), XML, JSON
  - Gráf adatbázisok, NoSQL

## (2) 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

# (3) 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)

## (4) Adatvédelem, adatbiztonság

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

# Adatbázis-kezelők részei

## ➤ Lekérdezés-feldolgozó

- Lekérdezés szintaktikai ellenőrzése
- Adatbázis-objektumok létezésének, és a hozzáférési jogoknak az ellenőrzése (metaadatbázis, rendszertáblák)
- Lekérdezés optimális átfogalmazása
- Végrehajtási tervek készítése
- Az adatstruktúrák, méretek statisztikái alapján várhatóan minimális költségű végrehajtási terv kiválasztása
- Az optimális végrehajtási terv lefuttatása

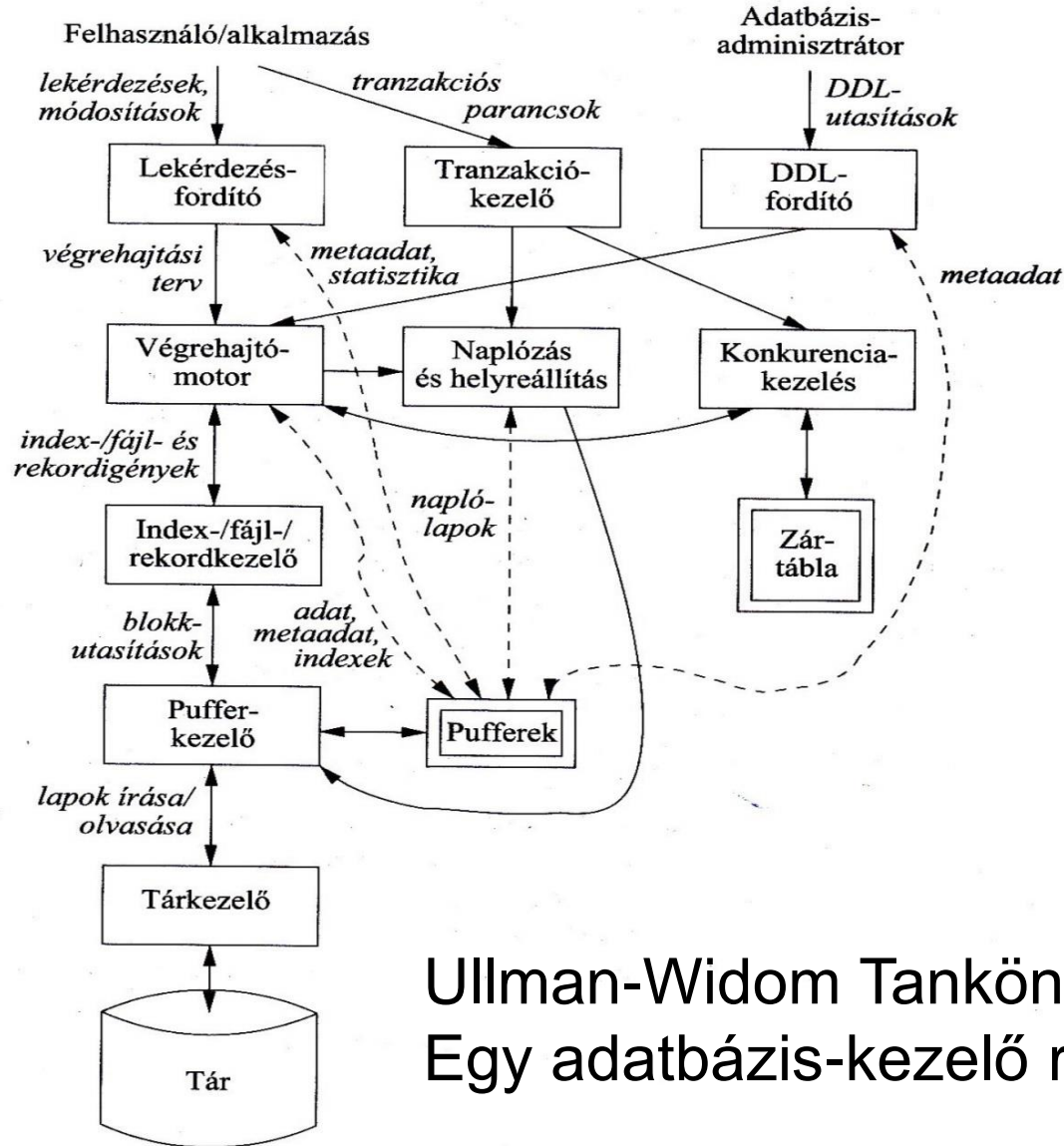
## ➤ Tranzakció-kezelő:

- Tranzakciók párhuzamos és biztonságos végrehajtásának a tranzakciók ACID tulajdonságainak biztosítása

## ➤ Tárkezelő és pufferkezelő

- fizikai adatstruktúrák, táblák, indexek, pufferek kezelése

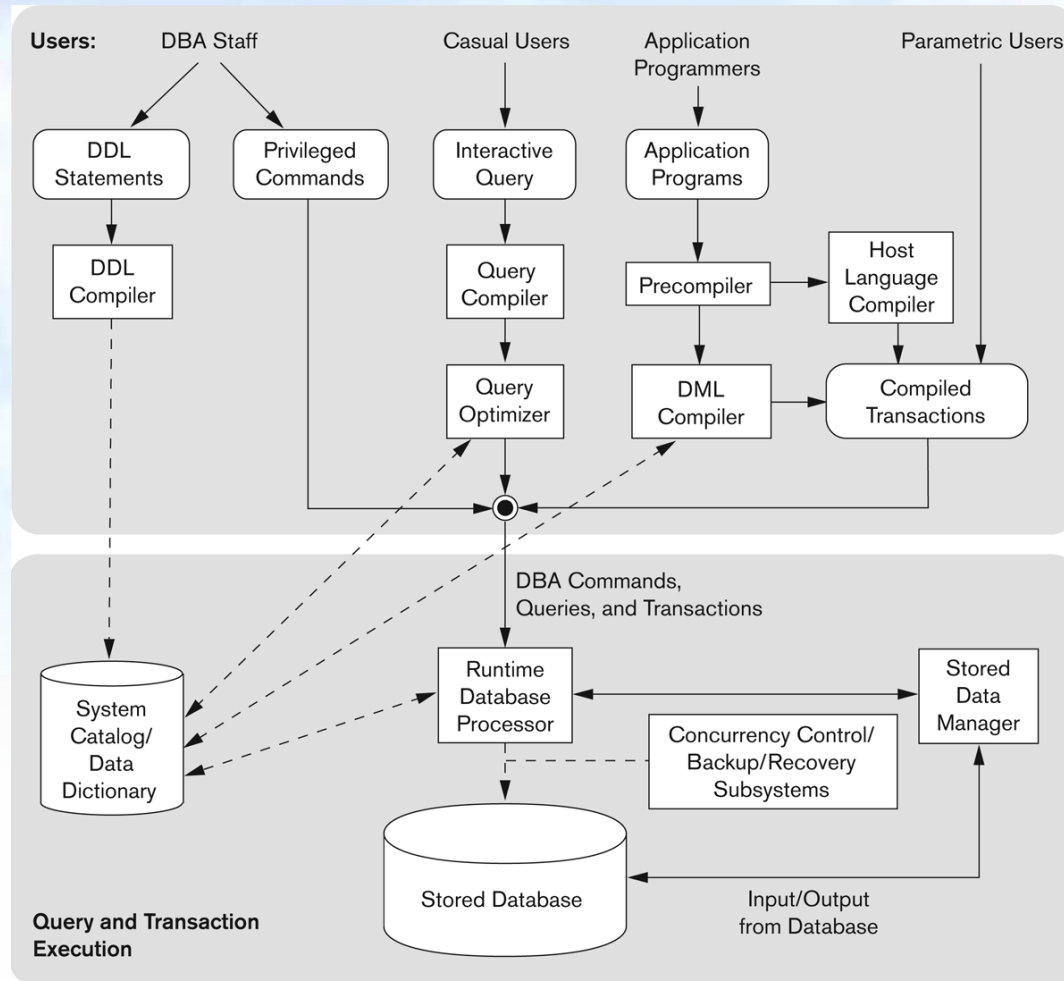
# Adatbázis-kezelő rendszer felépítése



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



# Adatbázis-kezelő rendszer felépítése



Forrás:

Elmasri-Navathe: Fundamentals of Database Systems

Figure 2.3

Component modules of a DBMS and their interactions.

# 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

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

# SQL DDL

## 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,  
    Kiegészítő lehetőségek  
);
```

- 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
- **Kiegészítő lehetőségek** például [DEFAULT], [UNIQUE], [PRIMARY KEY], [FOREIGN KEY, REFERENCES], stb.

# 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)



# SQL DML

## Adatbázis tartalmának módosítása

- **A módosító DML utasítások** az adatbázisban tárolt táblák tartalmát változtatják meg.
- 3-féle módosító utasítás létezik:
  - INSERT** - sorok beszúrása
  - DELETE** – sorok törlése
  - UPDATE** – sorok komponensei értékeinek módosítása
- Most a tábla előkészítésénél csak az elsőt nézzük meg, hogy viszünk fel adatsorokat:

# A táblák feltöltése adatokkal

- **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)  
    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

# Megszorítások (áttekintés)

## (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 (create assertion)

## (6) Triggerek (create trigger)

# 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 lekérdezéseknél 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, ...)

# 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 (attrnév<sub>1</sub>, ... attrnév<sub>k</sub>)**
- 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** egyik sorban sem. Viszont **UNIQUE**-nak deklarált attribútum lehet **NULL**, 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 gazdálkodni, 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) );
```

# Hivatkozási épség, idegen kulcs megszorítások megőrzése

- Példa:  $R = \text{Felszolgál}$ ,  $S = \text{Sörök}$ .
- 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 **Felzolgál** táblából, ahol sör oszlop értéke 'Bud'.
- A 'Bud' nevet 'Budweiser'-re változtatjuk:
  - a **Felzolgál** tábla soraiban is végrehajtjuk ugyanezt a változtatást.

# Példa: Set NULL

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

# A stratégia kiválasztása

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

# 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 (ár <= 5.00)** a beszúrt vagy módosított sor értéke nagyobb 5, a rendszer nem hajtja végre az utasítást.
- **Példa: CHECK (sör IN (SELECT név FROM Sörök))**, ha a Sörök táblából törölünk, ezt a feltételt nem ellenőrzi a rendszer.

# 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 sörök 5 dollárnál:

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

# Megszorítások elnevezése

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

- **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.\%');

# Összefoglalás: Kulcsok, idegen kulcsok

- Kiegészítő lehetőségeket: **Kulcs és idegen kulcs (foreign key) hivatkozási épség megadása**
- **PRIMARY KEY** vagy **UNIQUE**
- Kulcsok megadásának két változata van:
  - Egyszerű kulcs (egy attribútum) vagy
  - Összetett kulcs (attribútumok listája)
- **FOREIGN KEY**, a **hivatkozott attribútumoknak** a másik táblában kulcsnak kell lennie!
- **REFERENCES** kulcsszó használatának két lehetősége: attribútumként vagy sémaelemként lehet megadni.

# Összefoglalás: A stratégia kiválasztása hivatkozási épség megőrzésére

- Ha idegen kulcsot deklarálunk megadhatjuk beszúrásra is és törlésre is a SET NULL és a CASCADE stratégiát:

ON [UPDATE, DELETE][SET NULL CASCADE]

- Ha nem adjuk meg, a default stratégia működik, elutasítja az épséget sértő törlést vagy módosítást.
- Példa: **CREATE TABLE Felszolgál (**  
**söröző CHAR(20), sör CHAR(20), ár REAL,**  
**FOREIGN KEY(sör)REFERENCES Sörök(név)**  
**ON DELETE SET NULL**  
**ON UPDATE CASCADE);**

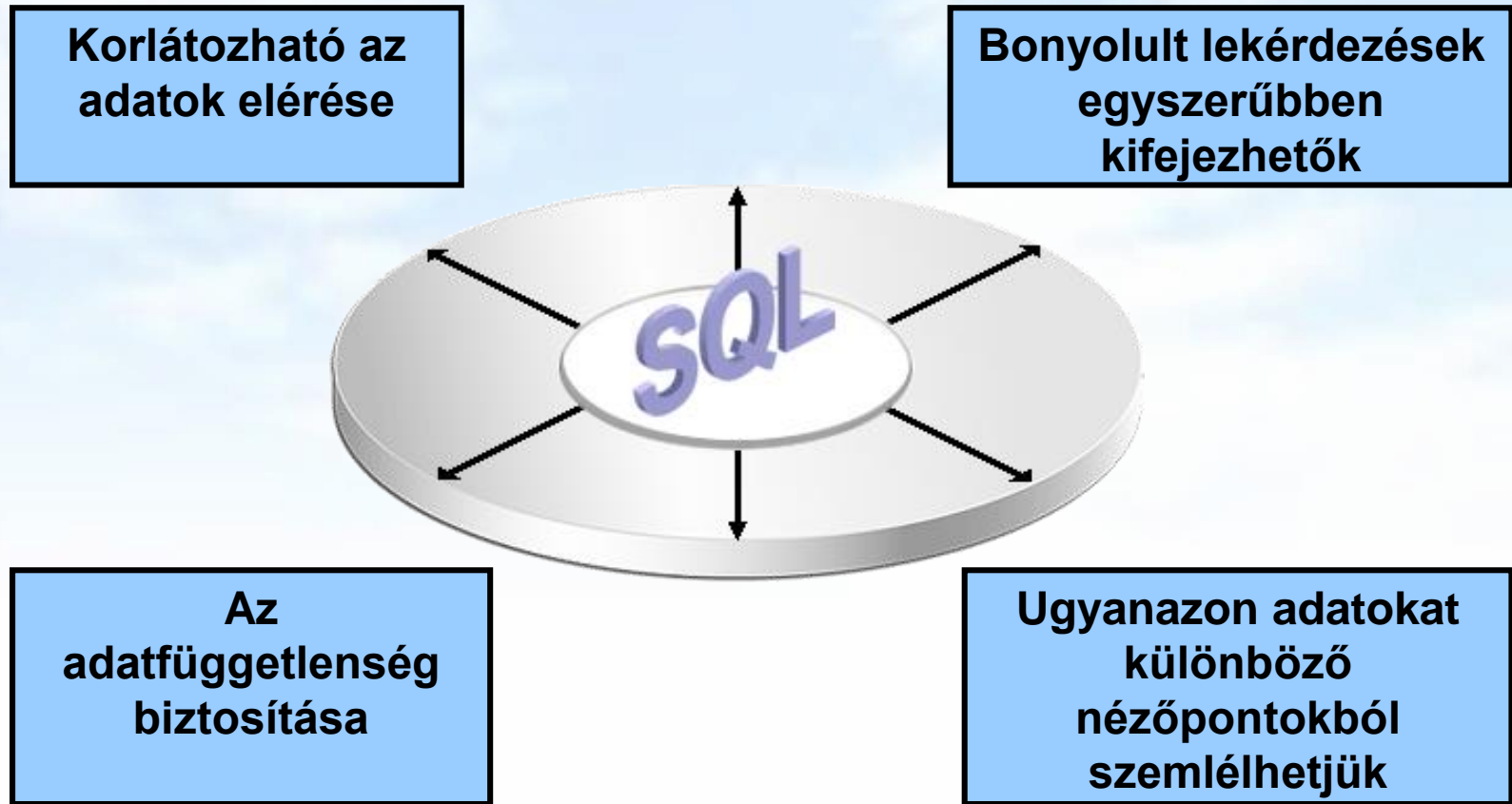
# Mik a nézettáblák?

- **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	SALA
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_FRES	240
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	170
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	170
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	90
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	42
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	58
141	Trenna	Rae	TRAE	650.121.8009	17-OCT-95	ST_CLERK	35
142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	31
143	Randall	Mates	RMATES	650.121.3074	10-MAR-98	ST_CLERK	20
					JUL-95	ST_CLERK	25
					JAN-00	SA_MAN	105
					MAY-96	SA_REP	110
					MAR-98	SA_REP	86
170	Kimberely	Grant	KGRANT	611.44.1044.429205	24-MAY-99	SA_REP	70
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	44
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	130
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	60
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	120
206	William	Gietz	WGIEZT	515.123.8181	07-JUN-94	AC_ACCOUNT	83

20 rows selected.

# A nézettáblák előnyei



# 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>;
```

# Mit jelent az insert utasítások után a COMMIT vagy a 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 **véglegesítődnek**.
- **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 jelennek meg az adatbázisban.

# Kérdés/Válasz

- Köszönöm a figyelmet! Kérdés/Válasz?