

Több táblára vonatkozó lekérdezések

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



7.1.-7.2. Megszorítások, kulcsok és idegen kulcsok, hivatkozási épség

6.2.1.-6.2.4. Több táblára vonatkozó lekérdezések az SQL-ben, lekérdezések értelmezése

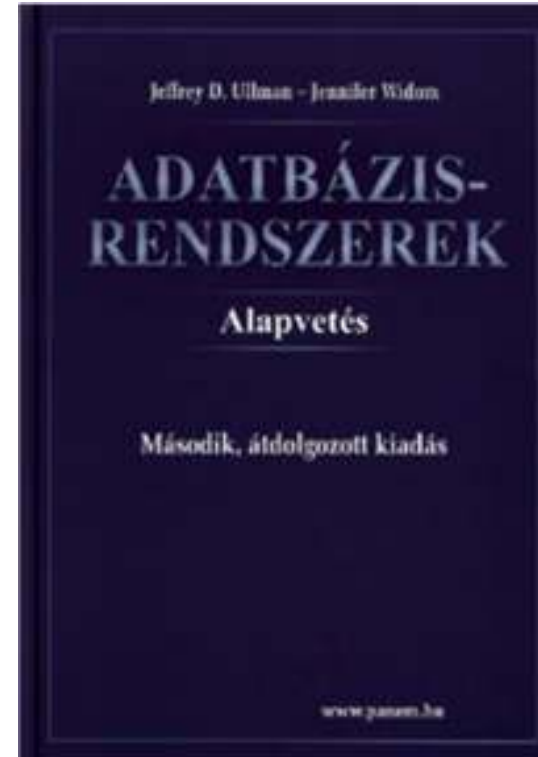
6.2.5.-6.4.2. Halmazműveletek az SQL-ben

6.3.1-6.3.5. Alkérdeések (where, having, from záradékban)

6.3.6.-6.3.8. Összekapcsolások az SQL-ben

Megszorítások, hivatkozási épség

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



7.1.-7.2. Megszorítások, kulcsok és idegen kulcsok, hivatkozási épség

- A SQL gyakorlatok felépítése miatt eddig az egytáblás lekérdezésekkel foglalkoztunk: Oracle Példatár 1-2-fej.
 - A többtáblás lekérdezésekhez szükségünk van a táblák közötti összefüggések további megszorítások megadására
-

Ismétlés: relációsémák definiálása

- Az SQL tartalmaz **adateleíró 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 mindazt, ami ehhez 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
 - Nézet táblák [CREATE [OR REPLACE] | DROP] VIEW
 - Átmeneti munkatáblák (WITH záradéka a SELECT-nek)
- **Alaptáblák** megadása: **CREATE TABLE**

Ismétlés: 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 gazdálkodni, hogyan lehet **NULL-t** kifejezésekben és hogyan lehet feltételekben használni.

Idegen kulcsok megadása

- Még egy kiegészítő lehetőség Mi köthet össze két táblát? **Idegen kulcs (foreign key) 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.) Attribútumként (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));

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 szűrjanak 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:

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 **Felzolgál** 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 Felszolgal (
    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 deklarálnak **DEFERRABLE** (késleltethető) vagy **NOT DEFERRABLE**-ként (vagyis minden adatbázis módosításkor a megszorítás közvetlenül utána ellenőrzésre kerül). **DEFERRABLE**-ként deklarálnak, 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)

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 Felszolgal (
  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)
);
```

Tankönyv példája sor alapú megszorításra

Attribútumokra és sorokra vonatkozó megszorítások

Példa: Ha egy színész neme férfi, akkor
a neve nem kezdődhet 'Ms.'-el

```
CREATE TABLE FilmSzínész (  
    név CHAR(30) PRIMARY KEY,  
    cím VARCHAR(255) NOT NULL,  
    nem CHAR(1),  
    születésiDátum DATE,  
    CHECK (nem = 'N' OR név NOT LIKE 'Ms. %')  
);
```


Megszorítások elnevezése

- Nevet tudunk adni a megszorításoknak, amire később tudunk hivatkozni (könnyebben lehet később majd törölni, 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.\%');

Több táblára vonatkozó lekérdezések

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

6.2.1.-6.2.4. 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 $R_1 [t_1], \dots, R_n [t_n]$
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 sorváltozónak 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 kapjunk, 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], ..., kifn [[AS] onévn]  
FROM R1 [t1], ..., Rn [tn]  
WHERE feltétel (vagyis logikai kifejezés)
```

Alapértelmezés (a műveletek szemantikája -- általában)

- A FROM záradékban levő relációkhoz tekintünk egy-egy **sorváltozót**, 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 sorváltozóiból indulunk ki, és tekintjük a hozzájuk tartozó relációk Descartes-szorzatát. Átnevezéssel valamint R.A jelöléssel elérjük, hogy minden attribútumnak egyedi neve legyen.
- 2.) A WHERE záradékot átalakítjuk egy kiválasztási feltétellé, melyet alkalmazunk az elkészített szorzatra.
- 3.) Végül a SELECT záradék alapján létrehozuk a kifejezések listáját, a (kiterjesztett) vetítési művelethez.

$$\Pi_{\text{onév}_1, \dots, \text{onév}_n} (\sigma_{\text{feltétel}} (R_1 \times \dots \times R_n))$$

Példa: Két tábla összekapcsolása ---1

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

```
SELECT sör
```

```
FROM Szeret, Látogat
```

```
WHERE bár = 'Joe' 's Bar'
```

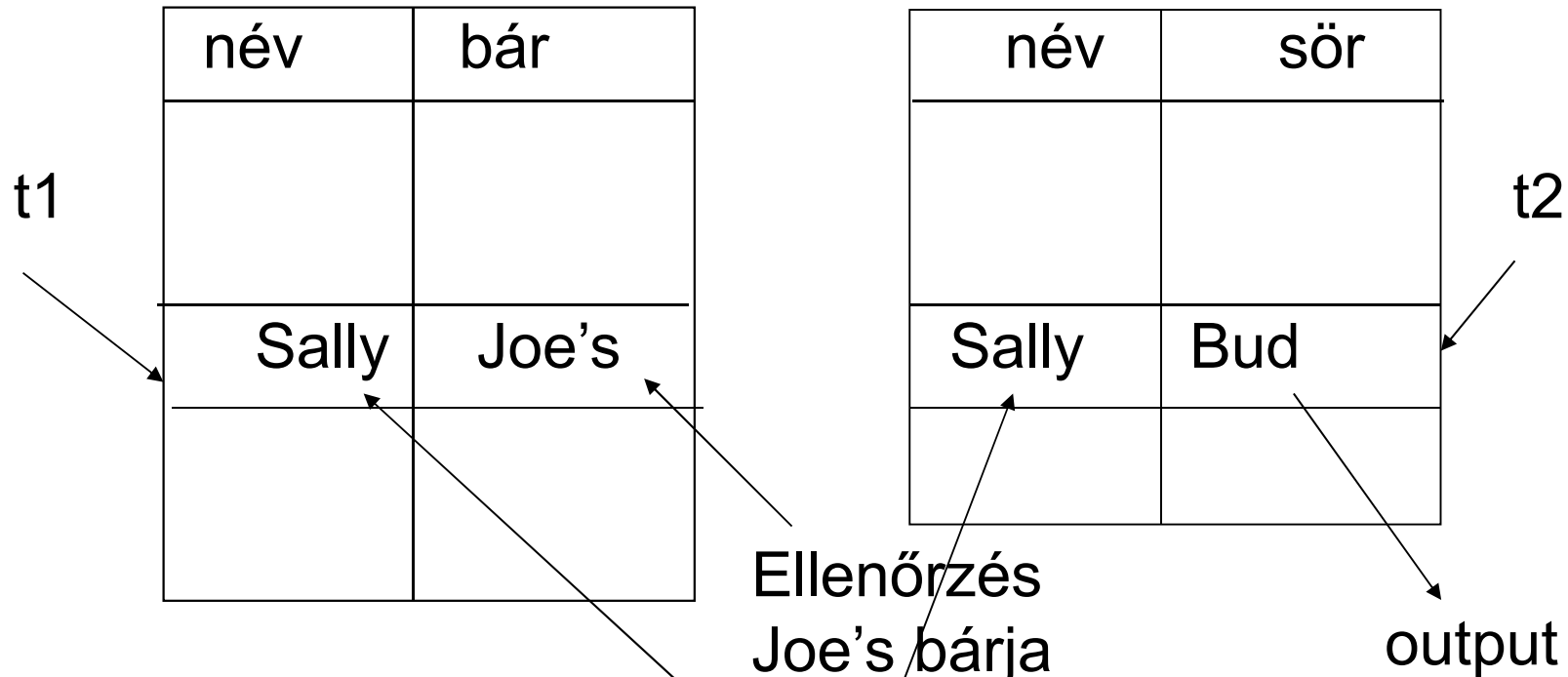
```
AND Látogat.név = Szeret.név;
```

- Kiválasztási feltétel: **bár = 'Joe' 's Bar'**
- Összekapcsolási feltétel: **Látogat.név = Szeret.név**
- Alapértelmezését lásd a következő oldalon
- Összekapcsolások SQL:1999-es szintaxisát is nézzük majd

Példa: Két tábla összekapcsolása ---2

Látogat

Szeret



Ellenőrizzük, hogy
megegyeznek-e

Tábla önmagával való szorzata ---1

- Bizonyos lekérdezéseknél arra van szükségünk, hogy ugyanannak a relációnak több példányát vegyük.
- Ahhoz, hogy meg tudjuk különböztetni a példányokat a relációkat átnevezzük, másodnevet adunk, vagyis **sorváltozókat** írunk mellé a FROM záradékban.
- A relációkat mindig 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

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

6.2.5.-6.4.2. Egyesítés, metszet
és különbség az SQL-ben



Halmazműveletek az SQL-ben

- Mi hiányzik még, hogy a relációs algebra alpműveleteit mindet 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 alkérdés valójában az adatbázisban tárolt tábla azokat a sörivókat és söröket, amelyekre a sörivó szereti az adott sört **és** a sörivó 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 sörivó látogat olyan bárt, ahol ezt a sört felszolgálják

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

Példa: ALL (multihalmaz szemantika)

- Látogat(név, bár) és Szeret(név, sör) táblák felhasználásával kilistázzuk azokat a sörivókat, akik több bárt látogatnak, mint amennyi sört szeretnek, és annyival többet, mint ahányszor megjelennek majd az eredményben

(SELECT név FROM Látogat)

EXCEPT ALL

(SELECT név FROM Szeret) ;

Több táblára vonatkozó lekérdezések

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

6.3.1-6.3.5. Alkérdezések
(where, having, from záradékban)



Alkérdések

- A FROM listán és a WHERE záradékban (valamint a GROUP BY HAVING záradékában) zárójellezett 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.

Alkérdeések 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)
 - (kif₁, ... 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 konstanstábla, akkor a kiértékelés mindig a legbelsőből halad kifelé.
- **Korrelált alkérdés**, amely többször kerül kiértékelésre, minden egyes kiértékelés megfelel egy olyan értékadásnak, 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.
 - Futásidejű 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: Felszolgal(bár, sör, ár) táblában keressük meg azokat a bárokat, ahol a Miller ugyanannyiba kerül, mint Joe bárjában a Bud.

```
SELECT bár
```

```
FROM Felszolgal
```

```
WHERE sör = 'Miller' AND
```

```
    ár = (SELECT ár
```

```
        FROM Felszolgal
```

```
        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

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

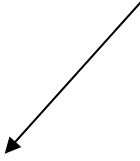
Skalár értékekből álló multihalmazt visszaadó alkérdések: ALL művelet

- $x \Leftrightarrow \text{ALL}(\text{alkérdés})$ akkor és csak akkor igaz, ha x az alkérdés egyetlen sorával sem egyezik meg.
- \Leftrightarrow helyett tetszőleges összehasonlítás szerepelhet.
- **Példa:** $x \geq \text{ALL}(\text{alkérdés})$ x az alkérdés eredményének maximum értékével azonos.

Példa: ALL

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

A külső lekérdezés
Felszolgáljának söre
egyetlen alkérdésbeli
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
1	2
3	4

R

b	c
2	5
2	6

S

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

Itt R és S sorait 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
   WHERE gyártó = b1.gyártó
    AND név <> b1.név);
```

Azon b1
sörtől
különböző
sörök,
melyeknek
ugyanaz
a gyártója.

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

➤ A FROM listán és a WHERE záradékban valamint a HAVING záradékban zárójelezett SELECT-FROM-WHERE utasításokat (alkérdéseket) is használhatunk.

(1) A gyakorlaton az I.ZH-ban csak a WHERE záradékban valamint a HAVING záradékban lehet használni majd alkérdéseket, FROM listán nem! A nézettáblákkal és az ún.inline nézetekkel való megoldások csak II.ZH-ban.

(2) **FROM listában**: a tényleges relációk helyett alkérdések, másodnevet adunk, ezzel tudunk a soraira hivatkozni.
FROM új listaelem: (lekérdezés) [AS] sorváltozó

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

Alkérdeések használata FROM listán

- **FROM záradékban** alkérdeéssel 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 alkérdeés, 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)

Alkérdeések használata FROM listán

- **FROM záradékban** alkérdeéssel 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

FROM Szeret, (SELECT név

FROM Látogat

WHERE bár = 'Joe' 's bar') JD

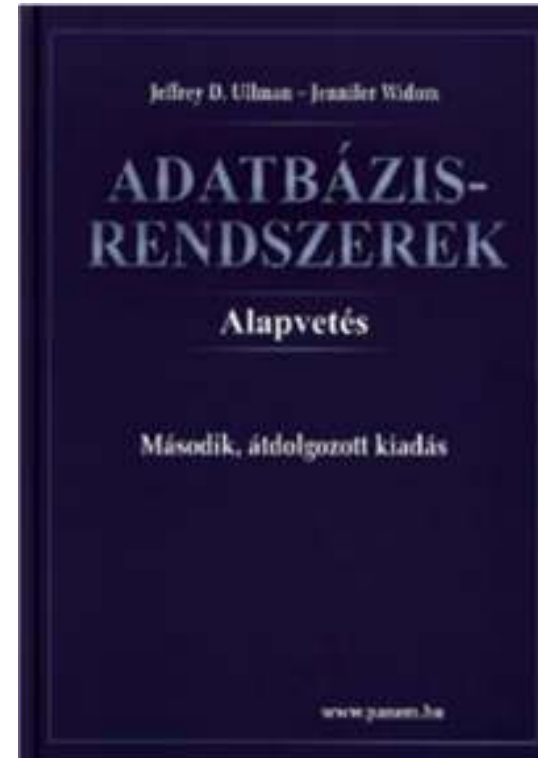
WHERE Szeret.név = JD.név;

Sörivők, akik látogatják
Joe's bárját.

Több táblára vonatkozó lekérdezések

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

6.3.6.-6.3.8. Összekapcsolások
az SQL-ben



Összekapcsolások az SQL:1999-ben

- 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.
- Használjunk 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

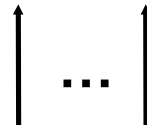
EMPLOYEES

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
50	Shipping
50	Shipping
50	Shipping
50	Shipping
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 nevű társaiktól.
- Ne használjunk alias nevet azon oszlopokra, amelyeket a USING kulcsszó után adtunk meg és az SQL utasításban még más helyen is szerepelnek.

Sorváltók használata tábláknál

- A lekérdezések átláthatósága miatt használhatunk sorváltót (tábla alias neveket).
- A sorváltók használata javítja a lekérdezés teljesítményét.
- A sorváltók maximum 30 karakter hosszúak lehetnek (minél rövidebb annál jobb)
- A sorváltók 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 kulcsszóval 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	80	2500
176	Taylor	80	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 nevezzük 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 x 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 kereszt-szorzatát (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?**
- **Gyakorlás a 4EA-hoz:** Több táblára (DEPT és EMP tábla) vonatkozó lekérdezésekre példák, összekapcsolások.
- **Házi feladat:** Oracle Példatár 3.fejezet feladatai:
<http://people.inf.elte.hu/sila/eduAB/Feladatok.pdf>