

Adatbázisok elmélete

SQL

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

Az SQL nyelv

- Relációs nyelv, mint az eddigiek
- oszlopkalkulus jellegű, de némi sorkalkulusos beütéssel

Termékek, (amik szükségszerűen relációs nyelvet is tartalmaztak):

- IBM: System/R
- Relational Software: Oracle
- Relational Systems: Ingres
- Microsoft: SQL server 2000
- Mysql (Oracle, open source)

Szabványok

- SQL89 (SQL1)
- SQL92 (SQL2, mi nagyrészt ezt nézzük most)
- SQL99 (SQL3, ebből is pár dolog, pl. triggererek, rekurzió)
- ...
- SQL:2016

Működő rendszerekben ezek verziói vannak (főleg SQL2).

Fontosabb utasítások

Data Definition Language:

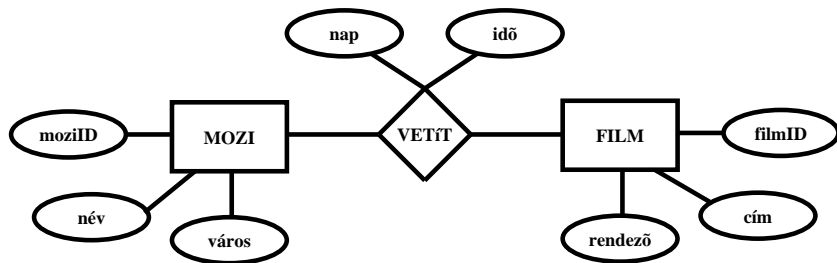
- CREATE - séma létrehozása
- ALTER - séma módosítása
- DROP - séma törlése

Data Modification Language:

- INSERT - adatok beszúrása
- UPDATE - adatok módosítása
- DELETE - adatok törlése
- SELECT - adatok lekérdezése

Természetesen előbb mindig a sémát kell létrehozni, és utána dolgozhatunk vele, de azt lehet egyéb módon is, nem csak SQL-ben. Mi most a lekérdezéseket vesszük először mert eddig úgyis a lekérdező nyelvekről volt szó.

A példákban használt relációs séma



DML utasítások — SELECT

Ezzel valósítható meg a kiválasztás, vetítés és a szorzat.

Szintaxis:

```
SELECT <relációi>.<attrib1>, ..., <relációj>.<attribn>
```

```
FROM <reláció1>, ..., <relációm>
```

```
WHERE <kifejezés>
```

Relációs algebrai megfelelője (de nem pontosan, mert SQL-ben SELECT nem küszöböli ki a többszörös sorokat):

$$\pi_{\langle \text{attrib}_1, \dots, \text{attrib}_n \rangle} \sigma_{\langle \text{kifejezés} \rangle} (\langle \text{reláció}_1 \rangle \times \dots \times \langle \text{reláció}_m \rangle)$$

Példa 1: A budapesti mozik azonosítói és nevei

```
SELECT mozi.mozilID, mozi.név FROM mozi WHERE mozi.város="Budapest"
```

Példa 2: A pénteken hétkor kezdődő filmek azonosítói

```
SELECT vetít.filmID FROM vetít WHERE vetít.nap="péntek" AND vetít.idő="19:00"
```

Megjegyzés:

- **kiértékelés:** minden egyes FROM utáni relációnak megfelel egy-egy sorváltozó, ami az egyes relációk sorain megy végig (egymásba ágyazott ciklusokkal például). Ha találat van, azaz a WHERE feltétel igaz az aktuális értékekre, akkor a SELECT utáni mezők kiíródnak
- úgy gondolhatunk a kiértékelésre, mintha először vennénk a FROM utáni relációk direkt szorzatát és aztán arra csinálnánk a kiválasztást és a vetítést.
- ha többszörös sorokat nem akarunk: SELECT DISTINCT (ennek ára van!!!)
- WHERE el is hagyható
- WHERE-ben mi állhat: erről később
- az eredmény az ORDER BY kulcsszó segítségével rendezhető, megadható hogy mely oszlopok szerint és hogy növeleg vagy csökkenőleg
- A fenti két példa mutatja, hogy a kiválasztás és a vetítés megy, a szorzatra a sorváltozók bevezetése után nézünk példát

SQL Sor- és oszlopváltozók

A FROM után felsorolt relációkhoz **sorváltozókat** rendelhetünk.

Szintaxis (FROM után <reláció_i> helyén): <reláció_i> AS <sorváltozó>

A SELECT után elhelyezett attribútum-hivatkozásokhoz **oszlopváltozókat** rendelhetünk.

Szintaxis (SELECT után <reláció_i>.<attrib_j> helyén):

<reláció_i>.<attrib_j> AS <oszlopváltozó>

Így átnevezés lehetséges az eredmény megjelenítésekor:

Például:

```
SELECT név AS Filmszínház,  
város AS Hely FROM mozi
```

⇒

	Filmszínház	Hely
	:	

Az oszlopváltozók valójában csak az eredményreláció attribútumainak elnevezésére használhatók, a SELECT utasításon belül nem hivatkozhatunk rájuk.

A <reláció_i>. előtag elhagyható, ha egyértelmű, hogy melyik relációról van szó, továbbá a <reláció_i>. előtag helyett <sorváltozó>. előtag is szerepeltethető.

Attribútumhivatkozások

Amikor egy attribútumra akarunk hivatkozni, három lehetőségünk van:

- **<attribútum>** (ha ez egyértelmű)
- **<reláció>.<attribútum>** (ha ez egyértelmű – N.B.: egy reláció többször is szerepelhet a FROM után, lesz példa)
- **<sorváltozó>.<attribútum>** (mindig használható)

Példa 3: A pénteken vetített filmek címei és rendezői (természetes illesztés)
`SELECT cím, rendező FROM film, vetít WHERE vetít.filmID = film.filmID AND nap='péntek'`

Példa 4: Azok a várospárok, ahol vannak azonos nevű mozik
`SELECT m1.város, m2.város FROM mozi AS m1, mozi AS m2 WHERE m1.név = m2.név AND m1.város <> m2.város`

Megjegyzés: a várospárok mindkét sorrendben megjelennek, és több azonos nevű mozi esetén többször is megjelennek.

Az elsőre megoldás: `<>` helyett legyen `<`, amúgy meg **DISTINCT**
`SELECT DISTINCT m1.város, m2.város FROM mozi AS m1, mozi AS m2 WHERE m1.név = m2.név AND m1.város < m2.város`

A WHERE kifejezés

Kifejezés felépítése:

- logikai műveletek: AND, OR, NOT
- összehasonlítás: =, <>, >=, <=, LIKE, BETWEEN
- aritmetikai műveletek: +, -, *, /, MOD, POWER, LN, SIN, COS, ...
- karakterlánc műveletek, összehasonlítás: CONCAT (||), LENGTH, LOWER, SUBSTR, SOUNDEX, ...
- halmazba tartozás: IN (halmaz), ...
- változóhivatkozások: < sorváltozó > . < attribútum >, < reláció > . < attribútum >, < attribútum >
- konstans (szám, karakterlánc): 137, 42e-3, 'füzér', ...
- NULL érték vizsgálata: IS NULL, IS NOT NULL (később lesz)
- alkérdés is lehet itt: (majd erről később)

LIKE és BETWEEN használata

LIKE használata:

- '_' egy tetszőleges karakterre illeszkedik
- '%' tetszőleges karakterláncra illeszkedik

BETWEEN használata: BETWEEN a AND b jelentése $a \leq . \leq b$

Példa 5: A 150 és 200 közötti azonosítójú mozik közül azok, amelyek B-vel kezdődő nevű városban vannak, és a nevük hárombetűs.

```
SELECT név FROM mozi WHERE moziID BETWEEN 150 AND 200 AND város LIKE 'B%' AND név LIKE '___'
```

Műveletek relációkkal

A részeredményül kapott relációkkal **(ha azok sémája lényegében azonos!)** halmazműveleteket (unió, metszet, különbség) végezhetünk.

Unió (valamely eredményrelációban szereplő sorok):

- **Szintaxis:** <eredményreláció1> UNION <eredményreláció2>
- **Példa 6:** A pénteken vagy szombaton játszott filmek :
(SELECT cím FROM film, vetít WHERE vetít.nap = 'péntek' AND film.filmID = vetít.filmID)
UNION
(SELECT cím FROM film, vetít WHERE vetít.nap = 'szombat' AND film.filmID = vetít.filmID)

(nem hatékony!)

Metszet (mindkét eredményrelációban szereplő sorok):

- **Szintaxis:** <eredményreláció1> INTERSECT <eredményreláció2>
- **Példa 7:** A pénteken és szombaton is játszott filmek:
(SELECT cím FROM film, vetít WHERE vetít.nap = 'péntek' AND film.filmID = vetít.filmID)
INTERSECT
(SELECT cím FROM film, vetít WHERE vetít.nap = 'szombat' AND film.filmID = vetít.filmID)

Különbség (az első reláció azon sorai, melyek a másodikban nem szerepelnek):

- **Szintaxis:** <eredményreláció1> MINUS <eredményreláció2>
- **Példa 8:** A pénteken igen, de szombaton nem játszott filmek:
(SELECT cím FROM film, vetít WHERE vetít.nap = 'péntek' AND film.filmID =
vetít.filmID)
MINUS
(SELECT cím FROM film, vetít WHERE vetít.nap = 'szombat' AND film.filmID =
vetít.filmID)

A szabványban **MINUS** helyett **EXCEPT** szerepel, de a gyakorlatban a **MINUS** használatos.

Állítás

Az SQL relációsan teljes.

Bizonyítás.

Most láttuk az **uniót** és **különbséget**, a többi pedig már volt, de újra:

vetítés: $\pi_{A_{i_1}, A_{i_2}, \dots, A_{i_k}}(R)$ -nek megfelelő lekérdezés: `SELECT $A_{i_1}, A_{i_2}, \dots, A_{i_k}$ FROM R`

kiválasztás: $\sigma_F(R)$ -nek megfelel a

`SELECT * FROM R WHERE F'`

ahol F' az, ami F-ből jön átírással (\wedge, \vee, \neg helyett AND, OR, NOT)

szorzat: `SELECT R.A1, R.A2, ..., R.Ak, S.B1, ..., S.Bl FROM R,S` ✓



Multihalmazok-halmazok

- Az SQL alapértelmezésben nem tünteti el a többszörös sorokat, **kivétel:** UNION, INTERSECT, EXCEPT, ennél a háromnál eltűnnek az ismétlődések
- Ha el akarjuk tüntetni az ismétlődéseket:
SELECT DISTINCT
- Ha a halmazműveleteknél mégsem akarom eltüntetni az ismétlődéseket:
UNION ALL, EXCEPT ALL, INTERSECT ALL
- Nem (mindig) éri meg közben is törekedni arra, hogy ne legyen ismétlődés, elég a végén, mert:
Az ismétlődés kiküszöbölése sok munka, mert rendezni kell az egész relációt hozzá.

Aggregátumok

- Aggregátumok számolása: SUM, MIN, MAX, AVG, COUNT,...
- Az, hogy COUNT hogyan kezeli a többszörös sorokat, az rendszerfüggő.
Ha biztosra akarunk menni: COUNT (DISTINCT <attribútum>), COUNT (ALL <attribútum>)
- Lehetőségünk van bizonyos attribútumok értéke szerint csoportosítani az eredményt, és így aggregált sorokat képezni.

Erre az utóbbira példa a következő reláció:

MOZI	mozilD	név	város	székszám
	1	Corvin	Budapest	2500
	2	Elit	Sopron	300
	3	Sopron Plaza Megaflex	Sopron	2000
	4	Szindbád	Budapest	600
	5	Tabán	Budapest	200
	6	Uránia	Pécs	500

Aggregátumok

Csoportosítsunk a város attribútum szerint:

MOZI	mozilD	név	város	székszám
	1	Corvin	Budapest	2500
	4	Szindbád	Budapest	600
	5	Tabán	Budapest	200
	6	Uránia	Pécs	500
	2	Elit	Sopron	300
	3	Sopron Plaza Megaflex	Sopron	2000

Képezzük minden városra a székszámok összegét:

MOZI	város	össz_székszám
	Budapest	3300
	Pécs	500
	Sopron	2300

Példa 9: Mindez SQL-ben

```
SELECT város, SUM(székszám) AS össz_székszám FROM mozi GROUP BY város
```


Aggregátumok

Példa 10: Az egyes városok legkisebb és legnagyobb mozijának mérete
`SELECT város, MIN(székszám), MAX(székszám) FROM mozi GROUP BY város`

Példák, ahol nincs csoportosítás:

Példa 11: A létező legnagyobb és a legkisebb székszám
`SELECT MIN(székszám), MAX(székszám) FROM mozi`

Példa 12: Az összes székszám
`SELECT SUM(székszám) FROM mozi`

Aggregátumok

- **Kiértékelés:** Vesszük a FROM utáni relációk direkt szorzatát (egy reláció szerepelhet többször is a szorzatban, ha sorváltozókat adtunk meg hozzá), a WHERE feltételt teljesítő eseteket a GROUP BY szerint csoportosítjuk, majd kiszámoljuk minden csoportra az aggregátumot és kiírjuk.
- Amennyiben aggregátumokat képzünk a GROUP BY segítségével, akkor csak azokra az attribútumokra hivatkozhatunk közvetlenül a SELECT-ben, ami szerint csoportosítottunk. Ezen attribútumok értékei ugyanis egy aggregátumon belül jól meghatározottak. A többi attribútum az aggregátumon belül többféle értéket is felvehet. Ezért rájuk csak oszlopfüggvényeken (aggregátumokon) keresztül hivatkozhatunk.
- Lehet több oszlop szerint is GROUP BY, ekkor azok a sorok lesznek egy csoportban, ahol mindegyik GROUP BY után felsorolt oszlop értéke megegyezik.
- Lehet GROUP BY aggregátum nélkül is

Példa 13:

SELECT város FROM mozi GROUP BY város

Kiírja az összes várost (pontosan egyszer), ahol van mozi.

Ugyanaz, mint a

SELECT DISTINCT város FROM mozi

Feltétel a csoportokra — HAVING

A csoportosítással együtt tehetünk feltételt a csoportokra. Ebben az esetben csak azokra a csoportokra számolódik ki az aggregátum, amik a feltételnek eleget tesznek.

Példa 14: Azokra a városokra számolunk csak legkisebb és legnagyobb mozit, ahol van legalább 2 mozi

```
SELECT város, MIN(székszám), MAX(székszám) FROM mozi GROUP BY város  
HAVING COUNT(név)>1
```

- a csoportra vonatkozó feltételt a **HAVING** kulcsszó vezeti be
- olyan feltételt írunk ide, ami csoportra vonatkozik (különben **WHERE**-be írnánk)
- csak **GROUP BY**-jal együtt használható
- a kiértékelés során a csoportosítás után minden egyes csoportra megnézzük a feltételt és eldobjuk azokat a csoportokat, amikre a feltétel nem áll és a maradékkal dolgozunk tovább
- **HAVING** megkerülhető, mindent, amit lehet **HAVING**-gel, lehet máshogy is (majd lesz erről szó az alkérdéseknél)

A hat alapkulcsszó

- SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY
- Ebben a sorrendben jönnek
- SELECT és FROM kell, a többi opcionális
- HAVING csak GROUP BY-jal

Alkérdeések

- Az alkérdés eredménye mindig egy reláció, szintaxisa pedig a lekérdezés szintaxisával azonos.
- Tipikusan **WHERE feltételében áll**, ezáltal sokkal összetettebb kiválasztási feltételek jönnek létre, mint a relációs algebrában.

Alkérés FROM záradékban

A kiválasztáshoz használt relációk lehetnek alkérés által származtatott relációk is.

Példa 15: A filmek címe, rendezője és a rendező filmjeinek száma

```
SELECT f1.cím, f1.rendező, f2.filmszám FROM
```

```
film AS f1,
```

```
(SELECT rendező, COUNT(*) AS filmszám FROM film GROUP BY rendező) AS f2
```

```
WHERE f1.rendező = f2.rendező
```

Vigyázat! Itt nem jött létre f2 nevű reláció, csak annyi történik, hogy az f2 nevű sorváltozó befutja az alkérés eredményéül kapott reláció sorait. Egyszer kiszámolódik az alkérés és ennek eredményét használjuk a továbbiakban.

Alkérés WHERE záradékban

Az alkérés eredményét valamely attribútumok értékeivel hasonlítjuk össze a kiválasztáshoz.

Ezeknek az attribútumok számában meg kell egyezniük az alkérés eredményének oszlopszámával.

- **Egyenlőség vizsgálata**

Csak akkor lehetséges, ha az alkérés egysoros relációt ír le (azaz az eredménye egyetlen érték vagy érték-vektor).

Az egyenlőség fennáll, ha az adott attribútumok értékei megegyeznek az alkérés által adott reláció megfelelő attribútumainak értékével.

Szintaxis:

```
SELECT ... FROM ...
```

```
WHERE (<attrib11>, ..., <attrib1n>) = (SELECT <attrib21>, ..., <attrib2n> FROM ...)
```

A nem egyenlőség vizsgálatára a <> használandó.

Példa 16: A legnagyobb mozik nevei

```
SELECT név FROM mozi
```

```
WHERE mozi.székszám = (SELECT MAX(székszám) FROM mozi)
```

- **Tartalmazás vizsgálata**

Több sort adó alkérésre is értelmezett.

A tartalmazás fennáll, ha a vizsgált attribútumok értéke megegyezik az alkérés eredményének valamely sorával.

Szintaxis:

```
SELECT ... FROM ...
```

```
WHERE (<attrib11>, ..., <attrib1n>) IN (SELECT <attrib21>, ..., <attrib2n> FROM ...)
```

A nem tartalmazás vizsgálatára a **NOT IN** használandó.

Példa 17: A nem vetített filmek címe és rendezője

```
SELECT cím, rendező FROM film AS f1
```

```
WHERE f1.filmID NOT IN (SELECT v1.filmID FROM vetít AS v1)
```

- **Alkérdeés valamely vagy minden sorának vizsgálata**

Tipikusan több sort szolgáltató alkérdeés esetén, valamilyen összehasonlító operátorral együtt használatosak az **ANY** és az **ALL** kulcsszavak.

Az **ANY**-t (**ALL**-t) tartalmazó feltétel teljesül, ha a vizsgált attribútumok értéke és az alkérdeés valamely (minden) sorára az összehasonlító operátor igaz értéket ad.

Szintakszis:

```
SELECT ... FROM ...
```

```
WHERE (<attrib11>, ..., <attrib1n>) <op> [ ANY | ALL ] (SELECT <attrib21>, ..., <attrib2n> FROM ...)
```

A „semelyik”, illetve a „nem mind” leírására a **NOT ANY**, illetve a **NOT ALL** használatosak.

Példa 18: Ismét a legnagyobb mozi(k)

```
SELECT m2.város, m2.név, m2.székszám FROM mozi AS m2
```

```
WHERE m2.székszám >= ALL (SELECT m1.székszám FROM mozi AS m1)
```


- **Alkérés ürességének vizsgálata**

Az **EXISTS** kulcsszóval megvizsgálhatjuk, hogy van-e egyáltalán sora az alkérés által leírt relációnak.

Az ezt tartalmazó feltétel teljesül, ha van legalább egy sor.

Szintakszis:

```
SELECT ... FROM ...
```

```
WHERE EXISTS (SELECT <attrib1>, ..., <attribn> FROM ...)
```

A nem létezés vizsgálatára a **NOT EXISTS** használandó.

Példa 19: Azok a városok, ahol van legalább két mozi

```
SELECT m1.város FROM mozi AS m1
```

```
WHERE EXISTS (SELECT * FROM mozi AS m2 WHERE m1.város =m2.város
```

```
AND m1.név<>m2.név)
```

Ez úgy nevezett korrelált alkérés:

ennek kiértékelése során minden egyes lehetséges értékére az m1 sorváltozónak fut az alkérés és kiírás van, ha az alkérés eredménye nem üres.

A korábbi esetekben csak egyszer kellett kiértékelni az alkérést, itt annyiszor, ahány sora a mozi-nak van.

HAVING megkerülése alkérdéssel

Nézzük egy példán, de általában is így megy:

HAVING-gel:

Azokra a városokra számolunk legkisebb és legnagyobb mozit, ahol van legalább 2 mozi

```
SELECT város, MIN(székszám), MAX(székszám)
FROM mozi
GROUP BY város
HAVING COUNT(*)>1
```

HAVING nélkül, alkérdéssel:

```
SELECT város, minszékszám, maxszékszám
FROM (SELECT város, MIN(székszám) AS minszékszám, MAX(székszám)
      AS maxszékszám, COUNT(*) AS darab
      FROM mozi
      GROUP BY város)
WHERE darab >1
```

NULL érték az SQL-ben

Az SQL-ben az ismeretlen vagy nem létező értéket a **NULL** érték jelképezi. A **NULL** használatakor ügyelni kell rá, hogy az aritmetikai és összehasonlító operátorok speciálisan értelmezettek rá.

Például:

NULL * 0 értéke nem 0, hanem **NULL**.

rendező = **NULL** értéke nem IGAZ, nem HAMIS, hanem a logikai

ISMERETLEN érték

⇒ UN.

Háromértékű logika

	\neg
<i>I</i>	<i>H</i>
<i>H</i>	<i>I</i>
<i>UN</i>	<i>UN</i>

\vee	<i>I</i>	<i>H</i>	<i>UN</i>
<i>I</i>	<i>I</i>	<i>I</i>	<i>I</i>
<i>H</i>	<i>I</i>	<i>H</i>	<i>UN</i>
<i>UN</i>	<i>I</i>	<i>UN</i>	<i>UN</i>

\wedge	<i>I</i>	<i>H</i>	<i>UN</i>
<i>I</i>	<i>I</i>	<i>H</i>	<i>UN</i>
<i>H</i>	<i>H</i>	<i>H</i>	<i>H</i>
<i>UN</i>	<i>UN</i>	<i>H</i>	<i>UN</i>

Tehát egy állítás nem csak igaz vagy hamis lehet, hanem „ismeretlen” is és egy WHERE-beli állításnál természetesen csak az számít találatnak, ha igaz az állítás, az „ismeretlen” nem lesz jó.

Furán viselkedik ez a logika:

SELECT moziID, filmID

FROM vetít

WHERE idő > "12:00" OR idő <= "12:00"

Az lenne jó, ha ez minden filmet felsorol, de sajnos aminek nincs ideje, azt nem sorolja fel.

$\implies A \vee \neg A \neq I$ a háromértékű logikában, azaz nem teljesül az, amit megszoktunk, hogy vagy az állítás vagy a tagadása igaz lesz.

Háromértékű logika

Hasonlóan: egy mező értéke nem hasonlítható össze a szokásos módon a **NULL** értékkel (mivel **NULL** nem egy konstans).

Erre az **IS NULL**, illetve az **IS NOT NULL** használatosak.

Példa 20: Azon filmek címe és rendezője, melyeknek ismerjük a rendezőjét.

```
SELECT cím, rendező FROM film WHERE rendező IS NOT NULL
```

Relációk összekapcsolása (join) SQL2-ben

A következőkben ismertetett nyelvi elemek egy része csak szintaktikai édesítőszer, kifejezhető a

`SELECT <attribútumok> FROM R, S WHERE <feltételek>` (*)
segítségével.

Relációk összekapcsolásakor meg kell adni az összekapcsolás módját (belső vagy külső) és a sorok összekapcsolásának feltételét.

Az összekapcsolás módja

- **Belső összekapcsolás** (mint \bowtie -nél): **R INNER JOIN S**
R-nek és S-nek csak azon sorai kerülnek az eredményrelációba, melyekhez van kapcsolódó sor S-ben, illetve R-ben (azaz a másik relációban). (Az, hogy mikor kapcsolódó két sor, az majd a kapcsolódás feltételeinél derül ki.)
- **Bal oldali külső összekapcsolás** (mint $\leftarrow\bowtie$ -nél): **R LEFT [OUTER] JOIN S**
R-nek azon sorai is bekerülnek az eredményrelációba, melyekhez nem kapcsolódik S-beli sor.
Ezekben a csak S-ben szereplő mezők **NULL** értéket kapnak.
- **Jobb oldali külső összekapcsolás** (mint $\bowtie\rightarrow$ -nél): **R RIGHT [OUTER] JOIN S**
Mint a LEFT OUTER JOIN, de R és S szerepe megcserélődik.
- **Teljes külső összekapcsolás** (mint $\bowtie\leftarrow\rightarrow$ -nél): **R FULL [OUTER] JOIN S**
Mind R, mind S azon sorai is bekerülnek az eredményrelációba, melyekhez nem kapcsolódik sor a másik relációból. (MySQL-ben nincs)
Az ezáltal üresen maradó mezők itt is **NULL** értéket kapnak.

A direkt szorzat létrehozására az **R CROSS JOIN S** alak használható, ilyenkor nincs feltétele a sorok összekapcsolásának.

Ez az alapértelmezés, (,) használatakor ilyen illesztés történik.

A sorok összekapcsolásának feltételei

- **Természetes illesztés:** **R NATURAL [INNER | LEFT | RIGHT | FULL] JOIN S**
R és S azon sorai illesztődnek, ahol az azonos nevű attribútumok értéke is megegyezik.
Ez az alapértelmezés.
- **Illesztés azonos nevű attribútumokkal:** **R [INNER | LEFT | RIGHT | FULL] JOIN S USING (<attribútumok>)**
R és S azon sorai illesztődnek, ahol az azonos nevű és <attribútumok>-ban felsorolt attribútumok értéke is megegyezik.
- **Illesztés tetszőleges feltétellel (θ -join):** **R [INNER | LEFT | RIGHT | FULL] JOIN S ON (<feltétel>)**
R és S azon sorai illesztődnek, melyek attribútumai eleget tesznek a megadott feltételnek.

Példák relációk összekapcsolására

Példa 21: Kusturica vetített filmjei és vetítési időpontjaik

```
SELECT cím, nap, idő FROM film INNER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
```

Példa 22: Kusturica összes filmje és vetítési időpontjaik (ha van)

```
SELECT cím, nap, idő FROM film LEFT OUTER JOIN vetít ON rendező='E. Kusturica' AND film.filmID=vetít.filmID
```

Példa 23: Vetített filmek címe, rendezője és vetítési időpontjaik

```
SELECT cím, rendező, nap, idő FROM film NATURAL INNER JOIN vetít
```

Példa 24: ugyanez USING használatával

```
SELECT cím, rendező, nap, idő FROM film INNER JOIN vetít USING (filmID)
```

Példa 25: Összes film címe, rendezője és vetítési időpontja (ha van)

```
SELECT cím, rendező, nap, idő FROM film NATURAL LEFT OUTER JOIN vetít
```

Példa 26: Az összes film-mozi pár

```
SELECT * FROM film CROSS JOIN mozi
```

DML utasítások — INSERT

Sorokat a relációba az **INSERT** utasítással szúrhatunk be.

Szintakszis: **INSERT INTO** <reláció> (<attrib₁>, ..., <attrib_n>) **VALUES** (<érték₁>, ..., <érték_n>)

Hatása: a <reláció> relációba egy új sor kerül, amiben <attrib₁> attribútum értéke <érték₁>, stb. A nem meghatározott értékű attribútumok a reláció létrehozásakor az attribútumhoz rendelt alapértelmezett értéket veszik fel.

Példa 27: Egy új film felvétele

```
INSERT INTO film (cím, rendező) VALUES ('Egy csodálatos elme', 'Ron Howard')
```

Megjegyzés:

- a filmID mező az alapértelmezett értékét kapja, de egy trigger (később lesz) segítségével akár automatikusan növekvő számozást is létrehozhatunk.
- Ha az összes attribútum értékét megadjuk, akkor nem kell őket felsorolni, ebben az esetben a beadott értékek a default attribútumsorrend szerint lesznek hozzárendelve az attribútumokhoz)
- a beszúrt adatokat egy alkérdésből is vehetjük:
Ha van egy filmregi(filmID, cím, rendező) tábla és azokat az adatokat szeretnénk átvinni a film táblába, amik ott nem szerepelnek:

```
INSERT INTO film
```

```
    SELECT filmregi.filmID, filmregi.cím, filmregi.rendező
```

```
    FROM filmregi
```

```
    WHERE filmregi.filmID NOT IN
```

```
        (SELECT filmID FROM film)
```

Sorokat a relációban az **UPDATE** utasítással módosíthatunk.

Szintakszis: UPDATE <reláció> SET <attrib₁>=<érték₁>, . . . , <attrib_n>=<érték_n>
WHERE <feltétel>

Hatása: a <reláció> reláció minden sorában, amelyik illeszkedik a <feltétel> feltételre <attrib_i> értéke <érték_i> lesz.

Példa 28: Az előbb beszúrt rendező nevének átírása rövidített alakba
UPDATE film SET rendező='R. Howard' WHERE rendező='Ron Howard'

DML utasítások — DELETE

Sorokat egy relációból a **DELETE** utasítással törölhetünk.

Szintakszis: **DELETE FROM** <reláció> **WHERE** <feltétel>

Hatása: a <reláció> reláció feltételre illeszkedő sorait törli.

Megjegyzés: a **WHERE** <feltétel> rész elhagyása esetén a reláció összes sorát törli.

Példa 29: Azon filmek törlése, amiknek a rendezője E. K. monogrammú

DELETE FROM film **WHERE** rendező **LIKE** 'E.% K%'

SQL Data Definition Language

- Séma létrehozása
- Séma törlése
- Séma módosítása
- Indexek létrehozása és kezelése (az indexekről később lesz szó részletesen)
- Nézetek létrehozása
- Kényszerek létrehozása
- Triggerek (kicsit)

A triggerek már az SQL3-hoz tartoznak, a triggerek segítségével az adatbázis valamely változásakor egy tárolt eljárást hajthatunk végre. Itt fogunk beszélni a rekurzióról is, mert az is SQL3-as dolog, de az lekérdezés és nem DDL, a segítségével egy reláció tranzitív lezárását lehet kiszámolni rekurzívan.

Relációk létrehozása

Relációk létrehozására a **CREATE TABLE** használandó.

Minden attribútumnak típust és a reláción belül egyedi nevet kell adni.

Szükség esetén megadható alapértelmezett érték is (**DEFAULT** kulcsszóval), melyeket az attribútum azon sorokban vesz fel, ahol a beszúráskor nem adtuk meg a konkrét értékét.

Lehetőség van arra is, hogy kényszereket definiáljunk az attribútumokra (pl. attribútum nem **NULL**, elsődleges kulcs, idegen kulcs, stb.), ezekről később lesz szó.

Szintakszis:

```
CREATE TABLE <relációnév> (  
<attrib1> <adattípus1> [DEFAULT <érték>], ...  
<attribn> <adattípusn> [DEFAULT <érték>]  
)
```

Példa 30: Film reláció létrehozása

```
CREATE TABLE film(  
  filmID number(5),  
  cím varchar(50),  
  rendező char(30),  
  év number(4),  
  hossz number(3) DEFAULT 90)
```

A lehetséges adattípusok függenek az adatbáziskezelőtől.

A főbb típusok:

char(<i>n</i>)	<i>n</i> hosszú karaktersorozat
varchar(<i>n</i>)	maximum <i>n</i> hosszú karaktersorozat
number(<i>n</i>,<i>m</i>)	<i>n</i> hosszú, <i>m</i> tizedesjegyű szám
date	dátum

Relációk törlése, módosítása

Relációk törlésére a **DROP TABLE** használandó.

Szintaxis: **DROP TABLE** <relációnév>

Példa 31: Film reláció törlése :-(

DROP TABLE film

Relációk módosítására az **ALTER TABLE** használandó.

Lehetőség van új attribútum definiálására (**ADD**), attribútum törlésére (**DROP**), attribútum adattípusának módosítására (**MODIFY**), kényszerek módosítására, alapértelmezett érték megadására.

Mindezek csak a jelenlegi adatokkal konzisztensen végezhetők el.

Például nem törölhető olyan attribútum, amire még van hivatkozás.

Szintaxis: **ALTER TABLE** <relációnév> <opciók>

Példa 32: Vetít relációhoz helyár hozzáadása

ALTER TABLE vetít **ADD** helyár **number**(4)

Példa 33: Mozi relációból a város eltávolítása

ALTER TABLE mozi **DROP** város

Példa 34: Ha a helyáratok ezentúl dollárban számoljuk ...

ALTER TABLE vetít **MODIFY** helyár **number**(4,2)

Egy reláció bizonyos attribútumaira indexet készíthetünk: ez egy adatszerkezetet jelent, ami olyan sorok gyors keresését teszi lehetővé amelyek az adott attribútumokon valami adott értékeket vesznek fel.

SQL2-nek sem része, de gyakori, ezért tanuljuk.

Szintaxis: `CREATE INDEX <indexnév> ON <relációnév>(attribútumok listája)`

Példa 35: index a filmcím, rendező párra

```
CREATE INDEX cím-rend ON film(cím, rendező)
```

Ha meguntuk, el lehet dobni: `DROP INDEX cím-rend`

- **előny:** gyors keresés lehetséges az index segítségével
- **hátrány:** az adatszerkezetet karban kell tartani, így lassítja a beszúrást, törlést, módosítást
- az a fontos, hogy miből van több, módosításból vagy lekérdezésből?
- néha a rendszer magától létrehoz indexet (lásd kulcsok)

Nézetek létrehozása

„Permanensen létező”, származtatott relációt hoz létre, amire hivatkozhatunk lekérdezésekkor is.

Szintaxis: `CREATE VIEW <új reláció neve> AS <lekérdezés>`

Nézetek létrehozása

Példa 36: Csináljunk egy nézetet Almodovar filmjeiből

```
CREATE VIEW Almodovarfilm AS
  SELECT filmID, cím
  FROM film
  WHERE rendező='P. Almodovar'
```

- A VIEW-val létrehozott reláció aszerint változik, ahogyan a film tábla változik (a nézettábla nem lesz alapreláció, nem olyan, mintha CREATE TABLE-vel csináltam volna és utána feltöltöttem volna adatokkal)
- de változtathatók az adatok korlátozottan a nézettáblán keresztül is
- Lekérdezésben használható, ugyanúgy ahogy az alaprelációk:
Példa 37: Milyen Almodovar filmeket vetítenek most?
SELECT cím FROM Almodovarfilm NATURAL INNER JOIN vetít
- Egy ilyen kérdés kiértékelésekor az Almodovarfilm nézettábla helyére a lekérdezésfeldolgozó berakja az őt definiáló SELECT-et
- Lehet új attribútumnevet adni a nézettáblában

Megszüntetése: DROP VIEW Almodovarfilm

Ezután már nem lehet olyan lekérdezést írni, amiben ez szerepel.

Nézet helyett új tábla létrehozása

Permanensen létező, relációt hoz létre, amire természetesen hivatkozhatunk lekérdezésekkor is.

Szintaxis: `CREATE TABLE <új reláció neve> AS <lekérdezés>`

Nézet helyett új tábla létrehozása

Példa 38: Csináljunk egy új táblát Almodovar filmjeiből

```
CREATE TABLE Almodovarfilm AS
```

```
  SELECT filmID, cím
```

```
  FROM film
```

```
  WHERE rendező='P. Almodovar'
```

- Az így létrehozott reláció független a film relációtól, teljesen új alapreláció lesz. Ha a film reláció adatai megváltoznak, attól az új reláció adatai nem.
- **Pl. ez jó archiválásnak.**

Ideiglenes táblák létrehozása

Csak az adott session alatt létező, származtatott relációt hoz létre, amire hivatkozhatunk lekérdezésekkor is.

Szintaxis: `CREATE TEMPORARY TABLE <új reláció neve> AS <lekérdezés>`

Ideiglenes táblák létrehozása

Példa 39: Csináljunk egy nézetet Almodovar filmjeiből

```
CREATE TEMPORARY TABLE Almodovarfilm AS  
  SELECT filmID, cím  
  FROM film  
  WHERE rendező='P. Almodovar'
```

- Az így létrehozott reláció független a film relációtól, új reláció lesz. Ha a film reláció adatai megváltoznak, attól az ideiglenes reláció adatai nem.
- A memóriában (vagy cache-ben) tárolódik.
- Lekérdezésben használható, ugyanúgy ahogy az alaprelációk, de csak abban a sessionban.
- Az adatok megváltoztathatóak benne, függetlenül az eredeti táblától.

Kényszerek csoportosítása

Kényszer típusa szerint

- Elsődleges kulcs (**PRIMARY KEY**)
- Egyértékűségi megszorítások (**UNIQUE**)
- Hivatkozási épség, idegen kulcs (**FOREIGN KEY**)
- NULL érték tiltása (**NOT NULL**)
- Értékkészlet (**CHECK**) (MySQL-ben nincs!)
 - ▶ attribútumra vonatkozó feltétel
 - ▶ sorra vonatkozó feltétel
 - ▶ globális feltétel

Elsődleges kulcs, egyediség

Attribútum(ok) elsődleges kulccsá tétele: **PRIMARY KEY**

Attribútum(ok) egyediségének megkövetelése: **UNIQUE**

Mindkét esetben az adott attribútumoknak egyértelműen azonosítaniuk kell a sort.

Különbség: **PRIMARY KEY** csak egy lehet, idegen kulcs csak erre hivatkozhat, sok rendszer automatikusan indexet hoz rá létre.

Szintaxis:

a tábla létrehozásakor, az attribútum definíciójában:

<attribútum> <típus> { PRIMARY KEY | UNIQUE }

relációdefinícióban belül, önállóan, külön sorban:

{ PRIMARY KEY | UNIQUE } (<attrib₁>, ..., <attrib_k>)

Ilyenkor (<attrib₁>, ..., <attrib_k>) együtt a kulcs. Ha egy kulcs több attribútumból áll, akkor csak így lehet megadni.

A kulcsfeltételeket a rendszer minden beszúrás és módosítás előtt ellenőrzi, ezért van automatikusan index rájuk. És persze emiatt óvatosan kell a kulcsok megadásával bánni, mert nagyon lelassíthatják az adatmódosításokat.

Idegen kulcs

A hivatkozási épség fő eszköze az SQL-ben.

Másik reláció elsődleges kulcsára hivatkozás. Kulcsszavak: **FOREIGN KEY, REFERENCES**

Szintaxis:

attribútum definíciójában:

<attribútum> <típus> REFERENCES <hivatkozott reláció>(<hivatkozott attribútum>

relációdefiníció belül, önállóan, külön sorban:

FOREIGN KEY <attribútumok>

REFERENCES <hivatkozott reláció>(<hivatkozott attribútumok>)

A **FOREIGN KEY** kulcsszó után álló attribútumokat nevezzük **idegen kulcsoknak**.

A fenti deklaráció jelentése: ha létezik egy sor a relációban, ahol az idegen kulcsban levő attribútumok valami adott értékeket vesznek fel, akkor léteznie kell a hivatkozott relációban is egy olyan sornak, ahol a hivatkozott attribútumok értékei ugyanezek.

Kell, hogy a hivatkozott attribútumok elsődleges kulcsot alkossanak a hivatkozott relációban.

Az idegen kulcs deklarációja után záradékban megadható, mi történjen, ha a hivatkozott mező megváltozik, törlődik, illetve ha a hivatkozó mező megváltozna. Lehetőség van a változás/törlés megtiltására vagy a hivatkozó mező kijavítására is.

A **NOT NULL** kulcsszóval megtilthatjuk egy attribútum esetében a **NULL** (ismeretlen, nem létező) érték megadását.

Ezt használva mindenképpen valamilyen érték kerül az attribútum valamennyi sorába, ezért csak kötelezően megadandó attribútumok esetén használjuk!

Szintaxis: az attribútum definíciójában:

<attribútum> <típus> NOT NULL

Értékkészlet meghatározása

Attribútum által felvehető értékek halmazát a **CHECK** kulcsszóval korlátozhatjuk.
(MySQL-ben nincs!)

Szintaxis:

attribútumra vonatkozó feltétel: `<attribútum> <típus> CHECK (<feltétel>)`

sorra vonatkozó feltétel, relációdefinícióban: `CHECK (<feltétel>)`

több relációra vonatkozó globális feltétel:

`CREATE ASSERTION <kényszernév> CHECK(<feltétel>)`

Tipikus attribútumra vonatkozó feltételek lehetnek:

értékkészlet felsorolása: `<attribútum> IN (<érték1>, ..., <értékN>)`

intervallum megadása: `<attribútum> BETWEEN <alsó határ> AND <felső határ>`

De bármi állhat itt, ami WHERE után szerepelhet, akár alkérdés is.

Például a vetít tábla létrehozásakor beírhatunk egy ilyen sort:

`CHECK (filmID IN (SELECT film.filmID FROM film))`

Ebben az esetben a vetít tábla minden egyes változásakor leellenőrizzük, hogy létezik-e a megfelelő film a film táblában.

Baj ezzel: csak akkor ellenőrzi, ha a vetít táblával történik valami, azt simán hagyja, hogy a film táblából töröljek, pedig ilyenkor is elromolhat.

Erre megoldás az **ASSERTION**:

```
CREATE ASSERTION vetít-film CHECK (  
    vetít.filmID IN (SELECT film.filmID FROM film) )
```

Ezt a rendszer minden olyan alkalommal ellenőrzi, ha vagy a vetít vagy a film változik.

Megjegyzések:

a kényszerek a **CONSTRAINT** kulcsszó segítségével elnevezhetőek (a PRIMARY KEY, CHECK elé írva)

új kényszer hozzáadására, meglévő törlésére az ALTER TABLE ... {ADD | DROP} CONSTRAINT ad lehetőséget.

Példák kényszerekre

A film és a vetít relációk kényszerekkel kiegészített létrehozása:

```
CREATE TABLE film(  
    filmID number(5) PRIMARY KEY,  
    cím varchar(50) NOT NULL,  
    rendező char(30) NOT NULL,  
    év number(4) CHECK (év >= 1900),  
    hossz number(3) DEFAULT 90 CHECK (hossz BETWEEN 1 AND 300),  
    szinkronizált char(1) DEFAULT 'N' CHECK (szinkronizált IN ('I','N')),  
    UNIQUE(cím, rendező)  
)  
CREATE TABLE vetít(  
    filmID number(5) REFERENCES film(filmID),  
    moziID number(3) REFERENCES mozi(moziID),  
    nap char(9),  
    idő char(5) NOT NULL,  
    CHECK (nap IN ('hétfő', 'kedd', 'szerda', 'csütörtök', 'péntek', 'szombat', 'vasárnap'))  
)
```



Triggerek

SQL2: mindenféle, elég összetett CHECK feltételek, de a rendszerbe bele van építve, hogy mikor kell ellenőriznie valami feltételt

SQL3-as szemlélet: lehetőség van arra, hogy mi mondjuk meg, mikor legyen ellenőrzés

Trigger:

- Mikor legyen ellenőrzés (adott relációba való beszúrásakor, törléskor, módosításkor, tranzakció végén)
- Mi legyen a feltétel, amit ekkor ellenőrzünk?
- Ha a feltétel teljesül, akkor mit csináljunk? (akadályozzuk meg valamit, csináljunk vissza valamit, vagy bármi más)

Paraméternek adható meg, hogy a kiváltó esemény előtt/helyett/után történjen a cselekvés és még sok más is.

Példa triggerre

Séma: Gyártásirányító(név, cím, azonosító, nettóBevétel)

```
CREATE TRIGGER NetBevétTrigger
AFTER UPDATE OF nettóBevétel ON Gyártásirányító
REFERENCING
OLD AS RégiSor
NEW AS Újsor
WHEN (RégiSor.nettóBevétel > Újsor.nettóBevétel)
SET nettóBevétel=Régisor.nettóBevétel
WHERE azonosító=Újsor.azonosító
FOR EACH ROW
```

⇒ Ha valakinek csökkenne a bevétele, nem hagyjuk!

Rekurzió

SQL3-as dolog, ideiglenes elképzelés

Lekérdezés és nem DDL (csak úgy kerül ide, hogy ez is SQL3)

Példa: Van egy **Járat(honnan, hova)** táblánk, amiben azt tároljuk, hogy mely városokból hova mennek közvetlenül gépek. Határozzuk meg ennek a relációnak a tranzitív lezártját, azaz egy olyan **Eljut(honnan, hova)** relációt szeretnénk, amelyben két város akkor szerepel együtt, ha el lehet az egyikből a másikba jutni valahány átszállással.

Ez relációs algebrában nem kifejezhető, de SQL3-ban igen.

```
WITH RECURSIVE Eljut(honnan, hova) AS
  (SELECT honnan, hova FROM Járat)
  UNION
  (SELECT Eljut AS R1, Eljut AS R2
   WHERE R1.hova = R2.honnan)
```

```
SELECT * FROM Eljut
```

Nem lehet bármi a rekurzív definícióban, pl. negációval óvatosan \implies nem biztonságos kifejezés