

# Adatbáziskezelés

## SQL

Katona Gyula Y.

Számítástudományi és Információelméleti Tanszék  
Budapesti Műszaki és Gazdaságtudományi Egyetem

# 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 1:** 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 2:** A legnagyobb mozis 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 3:** 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 4:** 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 5:** 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 lefut 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	H
H	I
UN	UN

$\vee$	I	H	UN
I	I	I	I
H	I	H	UN
UN	I	UN	UN

$\wedge$	I	H	UN
I	I	H	UN
H	H	H	H
UN	UN	H	UN

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 6: 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  $\bowtie\leftarrow$ -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  $\rightarrow\bowtie$ -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 ( $\theta$ -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 7:** Kusturica vetített filmjei és vetítési időpontjai

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

**Példa 8:** Kusturica összes filmje és vetítési időpontjai (ha van)

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

**Példa 9:** Vetített filmek címe, rendezője és vetítési időpontjai

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

**Példa 10:** ugyanez USING használatával

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

**Példa 11:** Összes film címe, rendezője és vetítési időpontja (ha van)

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

**Példa 12:** 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> (<attrib<sub>1</sub>>, ..., <attrib<sub>n</sub>>) **VALUES** (<érték<sub>1</sub>>, ..., <érték<sub>n</sub>>)

**Hatása:** a <reláció> relációba egy új sor kerül, amiben <attrib<sub>1</sub>> attribútum értéke <érték<sub>1</sub>>, stb. A nem meghatározott értékű attribútumok a reláció létrehozásakor az attribútumhoz rendelt alapértelmezett értéket veszik fel.

**Példa 13:** 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)
```



## DML utasítások — UPDATE

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

**Szintakszis:** UPDATE <reláció> SET <attrib<sub>1</sub>>=<érték<sub>1</sub>>, . . . , <attrib<sub>n</sub>>=<érték<sub>n</sub>>  
WHERE <feltétel>

**Hatása:** a <reláció> reláció minden sorában, amelyik illeszkedik a <feltétel> feltételre <attrib<sub>i</sub>> értéke <érték<sub>i</sub>> lesz.

**Példa 14:** 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 15:** 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 beszúrásakor 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 16: 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:

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

## Relációk törlése, módosítása

Relációk törlésére a **DROP TABLE** használandó.

Szintaxis: **DROP TABLE** <relációnév>

Példa 17: 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 18: Vetít relációhoz helyár hozzáadása

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

Példa 19: Mozi relációból a város eltávolítása

**ALTER TABLE** mozi **DROP** város

Példa 20: 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 21: 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 22:** 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 23: 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 24:** 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 25:** 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 } (<attrib<sub>1</sub>>, ..., <attrib<sub>k</sub>> )

**Ilyenkor (<attrib<sub>1</sub>>, ..., <attrib<sub>k</sub>> ) 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? (akadályozzunk 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 FROM Járat
   WHERE R1.hova = R2.honnan)
```

```
SELECT * FROM Eljut
```

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