Adatbázisok elmélete 12. előadás

Csima Judit
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 136/b
csima@cs.bme.hu

2003. Március 25.

- **Alkérdés valamely vagy minden sorának vizsgálata**
  Tipikusan több sort szolgáltató alkérdés 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 alkérdés valamely (minden) sorára az összehasonlító operátor igaz értéket ad.

  Szintaxis: SELECT . . . WHERE (<attrib$_{11}$>, . . . , <attrib$_{1n}$>) <op> [ANY | ALL] (SELECT <attrib$_{21}$>, . . . , <attrib$_{2n}$> FROM . . .)
  A "semelyik", illetve a "nem mind" leírására a **NOT ANY**, illetve a **NOT ALL** használatosak.
  Példa 1: Ismét a legnagyobb mozi(k)
  SELECT m2.varos, m2.nev, m2.szekszam FROM mozi AS m2 WHERE m2.szekszam >= ALL (SELECT m1.szekszam FROM mozi AS m1)

- **Alkérdés ürességének vizsgálata**
  Az **EXISTS** kulcsszóval megvizsgálhatjuk, hogy van-e egyáltalán sora az alkérdés által leírt relációnak.
  Az ezt tartalmazó feltétel teljesül, ha van legalább egy sor.
  Szintaxis: SELECT . . . WHERE EXISTS (SELECT <attrib$_1$>, . . . , <attrib$_n$> FROM . . .)
  A nem létezés vizsgálatára a **NOT EXISTS** használandó.
  Példa 2: Azok a városok, ahol van legalább két mozi
  SELECT m1.varos FROM mozi AS m1 WHERE EXISTS (SELECT * FROM mozi AS m2 WHERE m1.varos =m2.varos AND m1.nev<>m2.nev)
  Ez úgy nevezett korrelált alkérdés: ennek kiértékelése során minden egyes lehetséges értékére az m1 sorváltozónak lefut az alkérdés és kiírás van, ha az alkérdés eredménye nem üres.
  A korábbi esetekben csak egyszer kellett kiértékelni az alkérdést, 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 varos, MIN(szekszam), MAX(szekszam) FROM mozi GROUP BY varos HAVING COUNT(nev)>1

HAVING nélkül, alkérdéssel:

```
SELECT varos, minszekszam, maxszekszam
FROM  (SELECT varos, MIN(szekszam) AS minszekszam, MAX(szekszam)
       AS maxszekszam, COUNT(*) AS darab
       FROM mozi GROUP BY varos)
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**.
rendezo = NULL értéke nem IGAZ, nem HAMIS, hanem a logikai ISMERETLEN érték $\Longrightarrow$ 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 vetit
WHERE ido>”12:00” OR ido <=”12:00”

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

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

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 3: Azon filmek címe és rendezője, melyeknek ismerjük a rendezőjét.
SELECT cim, rendezo FROM film WHERE rendezo 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 ⟕-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 ⟖-nél): **R RIGHT [OUTER] JOIN S**
  Mint a LEFT OUTER JOIN, de R és S szerepe megcserélődik.
- **Teljes külső összekapcsolás** (mint ⟗-nél): **R FULL [OUTER] JOIN S**
  Mind R, mind S azon sorai is bekerülnek az eredményrelációba, melyekhez nem kapcsolódik sor a másik relációból.
  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 4: Kusturica vetített filmjei és vetítési időpontjaik
SELECT cim, nap, ido FROM film INNER JOIN vetit ON rendezo="E. Kusturica" AND film.filmID=vetit.filmID

Példa 5: Kusturica összes filmje és vetítési időpontjaik (ha van)
SELECT cim, nap, ido FROM film LEFT OUTER JOIN vetit ON rendezo="E. Kusturica" AND film.filmID=vetit.filmID

Példa 6: Vetített filmek címe, rendezője és vetítési időpontjaik
SELECT cim, rendezo, nap, ido FROM film NATURAL INNER JOIN vetit

Példa 7: ugyanez USING használatával
SELECT cim, rendezo, nap, ido FROM film INNER JOIN vetit USING (filmID)

Példa 8: Összes film címe, rendezője és vetítési időpontja (ha van)
SELECT cim, rendezo, nap, ido FROM film NATURAL LEFT OUTER JOIN vetit

Példa 9: Az összes film-mozi pár
SELECT * FROM film CROSS JOIN mozi

DML utasítások — INSERT

Sorokat a relációba az INSERT utasítással szúrhatunk be.

Szintaxis: INSERT INTO <reláció> (<attrib$_1$>, ..., <attrib$_n$>) VALUES (<érték$_1$>, ..., <érték$_n$>)

Hatása: a <reláció> relációba egy új sor kerül, amiben <attrib$_1$> attribútum értéke <érték$_1$>, stb. A nem meghatározott értékű attribútumok a reláció létrehozásakor az attribútumhoz rendelt alapértelmezett értéket veszik fel.

Példa 10: Egy új film felvétele
INSERT INTO film (cim, rendezo) 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, cim, rendezo) tábla és azokat az adatokat szeretnénk átvinni a film táblába, amik ott nem szerepelnek:

```
INSERT INTO film
    SELECT filmregi.filmID, filmregi.cim, filmregi.rendezo
    FROM filmregi
    WHERE filmregi.filmID NOT IN
          (SELECT filmID FROM film)
```


DML utasítások — UPDATE

Sorokat a relációban az UPDATE utasítással módosíthatunk.

Szintaxis: UPDATE <reláció> SET <attrib$_1$>=<érték$_1$>, ..., <attrib$_n$>=<érték$_n$> WHERE <feltétel>

Hatása: a <reláció> reláció minden sorában, amelyik illeszkedik a <feltétel> feltételre <attrib$_i$> értéke <érték$_i$> lesz.

Példa 11: Az előbb beszúrt rendező nevének átírása rövidített alakba
UPDATE film SET rendezo="R. Howard" WHERE rendező="Ron Howard"

DML utasítások — DELETE

Sorokat egy relációból a DELETE utasítással törölhetünk.

Szintaxis: 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 12: Azon filmek törlése, amiknek a rendezője E. K. monogrammú
DELETE FROM film WHERE rendezo 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.

Szintaxis:
CREATE TABLE <relációnév> (
<attrib$_1$> <adattípus$_1$> [DEFAULT <érték>], . . .
<attrib$_n$> <adattípus$_n$> [DEFAULT <érték>]
)

Példa 13: Film reláció létrehozása

```
CREATE TABLE film(
filmID number(5),
cim varchar(50),
rendezo char(30),
ev number(4),
hossz number(3) DEFAULT 90)
```

A lehetséges adattípusok függenek az adatbáziskezelőtől.
A főbb típusok:

char(n)	n hosszú karaktersorozat
varchar(n)	maximum n hosszú karaktersorozat
number(n,m)	n hosszú, m tizedesjegyű szám
date	dátum