Adatbázisok elmélete 9. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu
http://www.cs.bme.hu/~kiskat

2005

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

Alkérdések

- 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.

Alkérdés FROM záradékban

A kiválasztáshoz használt relációk lehetnek alkérdés 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 alkérdés eredményéül kapott reláció sorait. Egyszer kiszámolódik az alkérdés és ennek eredményét használjuk a továbbiakban.

Alkérdés WHERE záradékban

Az alkérdés eredményét valamely attribútumok értékeivel hasonlítjuk össze a kiválasztáshoz. Ezeknek az attribútumok számában meg kell egyezniük az alkérdés eredményének oszlopszámával.

- **Egyenlőség vizsgálata**
  Csak akkor lehetséges, ha az alkérdés egysoros relációt ír le (azaz az eredménye egyetlen érték vagy érték-vektor).
  Az egyenlőség fennáll, ha az adott attribútumok értékei megegyeznek az alkérdés által adott reláció megfelelő attribútumainak értékével.
  Szintaxis:
  SELECT . . . FROM . . .
  WHERE (<attrib$_{11}$>, . . . , <attrib$_{1n}$>) = (SELECT <attrib$_{21}$>, . . . , <attrib$_{2n}$> FROM . . .)
  A nem egyenlőség vizsgálatára a <> használandó.
  Példa 2: A legnagyobb mozik nevei

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

- **Tartalmazás vizsgálata**
  Több sort adó alkérdésre is értelmezett.
  A tartalmazás fennáll, ha a vizsgált attribútumok értéke megegyezik az alkérdés eredményének valamely sorával.
  Szintaxis:
  SELECT . . . FROM . . .
  WHERE (<attrib$_{11}$>, . . . , <attrib$_{1n}$>) IN (SELECT <attrib$_{21}$>, . . . , <attrib$_{2n}$> 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)

Alkérdések

- **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.
  Szintakszis:
  SELECT . . . FROM . . .
  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 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)

- **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.
  Szintakszis:
  SELECT . . . FROM . . .
  WHERE EXISTS (SELECT <attrib$_1$>, . . . , <attrib$_n$> 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 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 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

$\Longrightarrow$ UN.

Háromértékű logika

	¬
I	*H*
H	*I*
UN	*UN*

∨	*I*	*H*	*UN*
I	*I*	*I*	*I*
H	*I*	*H*	*UN*
UN	*I*	*UN*	*UN*

∧	*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 számit 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.

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

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 $\ltimes$-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 $\rtimes$-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 7: 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 8: 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 9: Vetített filmek címe, rendezője és vetítési időpontjaik
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$_1$>, . . . , <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 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$_1$>=<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 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 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> (
<attrib$_1$> <adattípus$_1$> [DEFAULT <érték>], . . .
<attrib$_n$> <adattípus$_n$> [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:

char(n)	***n*** hosszú karaktersorozat
varchar(n)	maximum ***n*** hosszú karaktersorozat
number(n,m)	***n*** hosszú, ***m*** tizedesjegyű szám
date	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 helyárakat ezentúl dollárban számoljuk . . .
```
ALTER TABLE vetít MODIFY helyár number(4,2)
```


Indexek

Egy reláció bizonyos attribútumaira indexet készíttethetünk: 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)