Adatbázisok elmélete 11. 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

Nézetek létrehozása

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

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

Példa 1: 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 2: 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.

Kényszerek

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**)
  - ⋆ 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 }

relációdefiníción belül, önállóan, külön sorban:
{ PRIMARY KEY | UNIQUE } (<attrib$_1$>,. . . ,<attrib$_k$>)

Ilyenkor (<attrib$_1$>, . . . ,<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>)
relációdefiníción 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 deklarálása 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.

NULLitás

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.

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 töröljék, 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

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 beszúráskor, 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
```

$\Longrightarrow$ 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 $\Longrightarrow$ nem biztonságos kifejezés