Kerepes Tamás – Czinkóczki László

Adatbázisok

Adatintegritási szabályok (adatbázis kényszerek)

Kényszerek

- Az adatmodellről mondtuk, hogy az a „logikai struktúra” és az azon értelmezett „kényszerek” és „műveletek” összessége
- Az adatbázis kényszereket még adatintegritási szabályoknak, vagy „deklaratív adatintegritási szabályoknak” is nevezzük („Integrity Constraints”, „Declarative Data Integrity Constraints”)
- Az adatbázisunk tartalmát lehet velük jellemezni illetve korlátozni
- A kényszerek a táblákra/azok oszlopaira vonatkozó szabályok
- Deklarációk, tehát a DDL nyelv részei szintaktikailag

Hogyan „működnek” a kényszerek

- Az adatbáziskezelő rendszer figyeli a felhasználók által megkísérelt adatmódosításokat (INSERT, DELETE, UPDATE, stb), és ha azok megpróbálnák megszegni a „szabályt”, akkor azt azonnal megakadályozza. Az ilyen műveletek nem sikerülnek.
- Többfelhasználós környezetben és tranzakciós működés során persze sok probléma merül fel ekörül.

Hogyan jönnek létre a kényszerek

- A táblák létrehozása során (CREATE TABLE)
- A táblák módosítása során (ALTER TABLE)
- A kényszerek tehát a tábla-definició részei, vagy a tábla oszlopa definiciójának a részei
- Eszerint létezhetnek tábla vagy oszlopszintű kényszerek, de ez már erősen adatbázis-kezelő függő részlet
- A kényszerek nem „mindenhatóak”: lesz olyan érték-korlátozás, amelyet nem tudunk így megadni

A kényszerek típusai

- Ezt az SQL szabvány nem definiálja elég szigorúan, így az egyes adatbáziskezelő rendszerek valamelyest eltérhetnek egymástól ebben:
  - Nem feltétlenül ugyanazokat a lehetőségeket nyújtják
  - Nem feltétlenül azonos az SQL szintaxisa e téren
- Mégis az mondhatjuk, hogy jellemzően a következőféle kényszerek léteznek/létezhetnek:
  - NOT NULL
  - CHECK
  - PRIMARY KEY (elsődleges kulcs)
  - UNIQUE KEY (egyedi kulcs)
  - FOREIGN KEY (külső kulcs)

A NOT NULL kényszer

- A tábla egy oszlopára vonatkozó szabály, amely nem engedélyezi, hogy abban az oszlopban NULL érték tárolódjon
- Az eredeti matematikai modellben ez eleve lehetetlen volt. De később a gyakorlati implementációk ezt megengedték, hacsak nem korlátozzuk a NOT NULL segítségével

Példa a NOT NULL kényszerre

```
CREATE TABLE workers  (  worker_id   NUMBER(6),
  first_name  VARCHAR(25),
  last_name   VARCHAR(25) CONSTRAINT workers_last_name_nn NOT NULL,
  email       VARCHAR(25) NOT NULL,
  . . .
  );
```

A CHECK kényszer

- A tábla egy-egy sorára vonatkozó szabály
- Ez a kényszer egy logikai kifejezés, amelynek minden sorra (külön-külön) igaznak kell lennie
- Tulajdonképpen a NOT NULL tekinthető a CHECK egy speciális esetének. Ha mégis indokolt a NOT NULL, akkor az esetleg az adatbáziskezelő szoftver valamiféle implementációs „tökéletlensége” miatt van.
- Az SQL szintaxis sokszínűsége miatt a következő 3 dián Oracle-kompatibilis példákat mutatunk. Más adatbáziskezelőknél nem lesz lényegi/fogalmi különbség ehhez képest, de formai sajnos igen.

Példa a CHECK kényszerre

```
CREATE TABLE workers  (
  worker_id   NUMBER(6),
  first_name  VARCHAR(25),
  last_name   VARCHAR(25) CONSTRAINT workers_last_name_nn NOT NULL,
  email       VARCHAR(25) NOT NULL,
  . . .
  salary      NUMBER(8,2) CONSTRAINT worker_salary_min
                          CHECK (salary > 0),
  . . .
  );
```

Másik példa a CHECK kényszerre

```
CREATE TABLE workers  (
  worker_id   NUMBER(6),
  first_name  VARCHAR(25),
  last_name   VARCHAR(25) CONSTRAINT workers_last_name_nn NOT NULL,
  email       VARCHAR(25) NOT NULL,
  . . .
  salary      NUMBER(8,2) CHECK (salary > 0),
  . . .
  );
```

Harmadik példa a CHECK kényszerre

```
CREATE TABLE workers  (
  worker_id   NUMBER(6),
  first_name  VARCHAR(25),
  last_name   VARCHAR(25) CONSTRAINT workers_last_name_nn NOT NULL,
  email       VARCHAR(25)NOT NULL,
  . . .
  salary      NUMBER(8,2),
  . . .
  CONSTRAINT worker_salary_min CHECK (salary > 0),
  . . .
  );
```

A PRIMARY KEY kényszer (elsődleges kulcs)

- A relációs adatmodell elméleti oldaláról nézve minden kulcs „egyenrangú”
- A gyakorlatban mégis célszerű, ha a táblának van egy „elsődleges kulcsa”. Ilyenből maximum egy lehet.
- Lényeges kérdés, hogy ez a kulcs csupán egyetlen attribútumból áll (egyoszlopos), vagy többoszlopos (konkatenált)
- A konkatenáltal még összetett kulcsnak (composite key) is nevezik
- A táblában nem lehet két olyan sor, amelynek azonos az elsődleges kulcsa
- Az már implementációs részlet, hogy a PRIMARY KEY kulcshoz tartozó oszlopokban szabad-e NULL értékeknek lenniük. Pl. az Oracle esetében nem.

Példa a PRIMARY KEY kényszerre

```
CREATE TABLE workers  (
  worker_id   NUMBER(6) PRIMARY KEY,
  first_name  VARCHAR(25),
  . . .
  );
```

Második példa a PRIMARY KEY kényszerre

```
CREATE TABLE workers (
  worker_id   NUMBER(6) CONSTRAINT workers_worker_id_pk PRIMARY KEY,
  first_name  VARCHAR(25),
  . . .
  );
```

Harmadik példa a PRIMARY KEY kényszerre

```
CREATE TABLE workers (
  worker_id   NUMBER(6),
  . . .
  CONSTRAINT workers_worker_id_pk PRIMARY KEY(worker_id),
  . . .
  );
```

A UNIQUE kényszer (egyedi kulcs)

- A tábla azon kulcsa, amely nem az elsődleges kulcs
- Ilyenből tetszőleges számú lehet (akár nulla is)
- Implementációs részlet (tehát nem feltétlenül általános), hogy a UNIQUE kulcsok oszlopainak lehet NULL értéke is.
- A NULL értékkel nem szegjük meg az egyediséget, mert a NULL az „ismeretlen” érték
- A szintaxisa szinte ugyanolyan, mint a PRIMARY KEY esetén
- A következő 3 dia az Oracle jelrendszerét követi

Példa a UNIQUE kényszerre

```
CREATE TABLE workers  (
  worker_id   NUMBER(6),
  first_name  VARCHAR(25),
  . . .
  email       VARCHAR(25),
  . . .
  CONSTRAINT workers_email_uk UNIQUE (email),
  . . .
  );
```

Második példa a UNIQUE kényszerre

```
CREATE TABLE workers  (
  worker_id  NUMBER(6),
  first_name VARCHAR(25),
  . . .
  email      VARCHAR(25) CONSTRAINT workers_email_uk UNIQUE,
  . . .
  );
```

Harmadik példa a UNIQUE kényszerre

```
CREATE TABLE workers  (
  worker_id  NUMBER(6),
  first_name VARCHAR(25),
  . . .
  email      VARCHAR(25) UNIQUE,
  . . .
  );
```

A FOREIGN KEY kényszer

- A „külső kulcs” kényszer meggátolja, hogy olyan értéket vigyünk be egy tábla oszlopába, amely érték nem található meg egy táblánk elsődleges kulcsában, vagy egyedi kulcsában
- Legkönnyebben MASTER-DETAIL táblák között érthetjük meg:
  - Tegyük fel, hogy részlegeink vannak (DIVISIONS) és azokban dolgozóink (WORKERS)
  - A részlegeknek elsődleges kulcsa a DIVISION_ID
  - A dolgozóknak elsődleges kulcs a WORKER_ID
  - A WORKERS táblában is van egy DIVISION_ID oszlop, és ez mutatja, hogy melyik dolgozó melyik részlegben dolgozik
  - Nem lehet a dolgozók között olyan, aki „nem létező részlegben” dolgozik

Egy példa a FOREIGN KEY kényszerre

```
CREATE TABLE workers  (
  worker_id   NUMBER(6),
  . . .
  division_id NUMBER(4),
  . . .
  CONSTRAINT workers_division_fk
               FOREIGN KEY (division_id)
                           REFERENCES divisions (division_id),
  . . .
  );
```

Egy másik példa a FOREIGN KEY kényszerre

```
CREATE TABLE workers  (
  worker_id   NUMBER(6),
  . . .
  division_id NUMBER(4),
  . . .
  FOREIGN KEY (division_id)
              REFERENCES divisions (division_id),
  . . .
  );
```

Egy harmadik példa a FOREIGN KEY kényszerre

```
CREATE TABLE workers  (
  worker_id   NUMBER(6),
  . . .
  division_id NUMBER(4) REFERENCES divisions (division_id),
  . . .
  );
```

Egy negyedik példa a FOREIGN KEY kényszerre ☺

```
CREATE TABLE workers  (
  worker_id   NUMBER(6),
  . . .
  division_id NUMBER(4) REFERENCES divisions,
  . . .
  );
```

Egy ötödik példa a FOREIGN KEY amely ugyanazon tábla elsődleges kulcsára hivatkozik

```
CREATE TABLE workers  (
  worker_id   NUMBER(6) PRIMARY KEY,
  . . .
  manager_id  NUMBER(6),
  . . .
  CONSTRAINT workers_manager_fk FOREIGN KEY (manager_id)
    REFERENCES workers (worker_id),
);
```

Egy hatodik példa: FOREIGN KEY amely ugyanazon tábla elsődleges kulcsára hivatkozik

```
CREATE TABLE workers  (
  worker_id   NUMBER(6) PRIMARY KEY,
  . . .
  manager_id  NUMBER(6) REFERENCES workers(worker_id),
);
```

Csupán a vicc kedvéért:

- Majd próbáljanak az eddigi példák alapján, logikusan gondolkodva egy „hetedik” szintaktikai lehetőséget is találni/javasolni

Hierarchikus viszony ábrázolása relációsan

- Ahogy az előbbiekből látszik, ha egy tábla FOREIGN KEY kulcsa ugyanazon tábla PRIMARY KEY kulcsára hivatkozik, ezzel szülő-gyerek kapcsolatot tudunk ábrázolni
- A tábla egyik sora olyan lesz, hogy annak nincs „szülője”
- Ebben a sorban a FOREIGN KEY oszlop értéke NULL

Egy „teljes példa”

```
CREATE TABLE workers  (
  worker_id   NUMBER(6),
  first_name  VARCHAR(25),
  last_name   VARCHAR(25)CONSTRAINT workers_last_name_nn NOT NULL,
  email       VARCHAR(25)CONSTRAINT workers_email_nn NOT NULL,
  start_date  DATE CONSTRAINT workers_start_date_nn NOT NULL,
  position_id VARCHAR(15)CONSTRAINT workers_position_nn NOT NULL,
  salary      NUMBER(8,2),
  commission  NUMBER(8,2),
  manager_id  NUMBER(6,0),
  division_id NUMBER(4,0),
  born        DATE,
  CONSTRAINT worker_salary_min CHECK (salary > 0),
  CONSTRAINT worker_email_uk UNIQUE (email),
  CONSTRAINT workers_worker_id_pk PRIMARY KEY (worker_id),
  CONSTRAINT workers_position_fk FOREIGN KEY (position_id)
    REFERENCES positions (position_id),
  CONSTRAINT workers_manager_fk FOREIGN KEY (manager_id)
    REFERENCES workers (worker_id),
  CONSTRAINT workers_division_fk FOREIGN KEY (division_id)
    REFERENCES divisions (division_id));
```

A kényszerek létrehozása és eldobása

- CREATE TABLE utasítás
- ALTER TABLE ADD CONSTRAINT
- ALTER TABLE DROP CONSTRAINT
- ALTER TABLE ENABLE CONSTRAINT
- ALTER TABLE DISABLE CONSTRAINT
- DROP TABLE
- A PRIMARY KEY vagy UNIQUE megszorítás nem dobható el, ha FOREIGN KEY megszorításban hivatkozunk rá. Az egész „szülő tábla” sem dobható el ilyenkor.

A kényszerek biztosítása

- A NOT NULL és a CHECK kényszerek ellenőrzése viszonylag egyszerű, mert a beszúrni kívánt sor értékeinek az ellenőrzése
- A PRIMARY KEY és a UNIQUE megszorításokat az adatbáziskezelő rendszerek többsége egyedi indexek segítségével biztosítja. Ilyenkor rendszerint az indexek automatikusan létrejönnek a CONSTRAINT létrehozásának eredményeképpen
- Az igazi kihívást a több felhasználó által „egyidőben” elvégzett módosítások ellenőrzése jelenti ilyenkor
- A FOREIGN KEY megszorítást a különböző adatbáziskezelő rendszerek más-más módon biztosítják. Egyes rendszereknél ilyenkor is kötelező az index (a külső kulcson),míg másoknál csak ajánlott.

A kényszerek opcionális lehetőségei

- Egyes adatbáziskezelő rendszerek a kényszereket azonnal a DML utasítás végén ellenőrzik, és ha megpróbálnánk a kényszert megszegni, azt a DML utasítást hibaüzenettel elutasítanák
- Más rendszerek elodázhatják az ellenőrzést a tranzakció végéig. Így pl. bevihetünk először „gyerek-adatot”, és csak utána a „szülő-adatot”.
- Javaslat önálló gyakorlatra:
  - hozzanak létre egy tetszőleges „MASTER” táblát PRIMARY KEY-vel
  - hozzanak létre egy „DETAIL” táblát FOREIGN KEY-vel
  - Próbálják meg először bevinni a szülő-adatot, majd utána a „gyerekét”. Sikerül?
  - Próbálják most fordított sorrendben. Vajon sikerül?
  - Keressék ki az Oracle azon lehetőségét, amellyel ez mégiscsak sikerülne

Amit a kényszerek nem tudnak, de hasznos lehet

- Gyakran lenne szükségünk az eddig tárgyalt 5 szabály mellett azoknál bonyolultabb megszorításokat is alkalmazni. Pl. ilyen a következő: „a dolgozó fizetése nem lehet több, mint a főnökének a fizetése”.
- Ezeket kénytelenek leszünk valami más módon biztosítani. Ha nem lehet deklaratív szabályokkal, akkor programkóddal tesszük majd ezt. Erre igen alkalmasak egyes adatbáziskezelő rendszerekben az adatbázis triggerek.