7. Duomenų vientisumo užtikrinimas

7.2'. Reikalavimai reikšmėms

Stulpelio apibrėžime apibrėžiamiems reikalavimams galima suteikti vardą. Tai pakankamai svarbu, nes, kuriant lentelę, yra sunku tiksliai numatyti reikalavimus būsimoms reikšmėms. Jei, pavyzdžiui, ateityje papildomai norėtume išskirti ypatingos svarbos projektus, tai susidurtume su sunkumu pakeisti ankstesnįjį reikalavimą stulpelio *Svarba* reikšmėms, nes apibrėždami jį nesuteikėme vardo. Todėl stulpelį *Svarba* geriau yra apibrėžti taip

Svarba `CHAR`(10)
`CONSTRAINT` *Svarbos* `CHECK` (*Svarba* `IN` (‘Maža’, ‘Vidutinė’, ’Didelė’))
`DEFAULT` ‘Vidutinė’

Taip apibrėžtą reikalavimą stulpelio reikšmėms vėliau bus galima pakankamai nesunkiai pakeisti kitu, pvz.,

```
ALTER TABLE DROP CONSTRAINT Svarbos,
ALTER TABLE ADD CONSTRAINT
        Svarbos CHECK (Svarba IN (‘Maža’, ‘Vidutinė’, ’Didelė’, ’Ypatinga’)).
```

7.3'. Lentelės raktų vientisumas

Tiek pirminį raktą, tiek ir kitus raktus (unikaliuosius indeksus), jei tik jie yra nesudėtiniai, t.y. sudaryti tik iš vieno stulpelio, galima apibrėžti reikalavimu stulpeliui. Stulpelis *Nr* taps lentelės *Projektai* pirminiu raktu, o *Pavadinimas* unikaliuoju indeksu, jei šiuos stulpelius sakinyje `CREATE TABLE` apibrėšime taip

```
Nr          INTEGER      NOT NULL PRIMARY KEY CHECK (Nr >= 0),
Pavadinimas VARCHAR(254) NOT NULL CONSTRAINT Raktas2 UNIQUE.
```

Frazę `CONSTRAINT` <vardas> galima pralesti, jei manome, kad vardas raktui (unikaliajam indeksui) nereikalingas. Suteikdami raktui vardą sudarome galimybę atšaukti stulpelio reikšmių unikalumo reikalavimą.

Sudėtinius ir nesudėtinius lentelės raktus, papildančius pirminį raktą, taip pat galima apibrėžti atskiru reikalavimu

```
CONSTRAINT <rakto vardas> UNIQUE(<stulpelių vardai>),
```

kuris nurodomas kaip ir kiti reikalavimai, sukuriant lentelę (`CREATE TABLE`) arba atnaujinant jos sandarą (`ALTER TABLE`).

Dirbtiniems raktams, sudarytiems iš vieno sveikųjų skaičių stulpelio, stulpelio apibrėžime galima nurodyti automatinio reikšmių parinkimo taisyklę

```
GENERATED <ALWAYS|BY DEFAULT> AS IDENTITY
[([START WITH <Skaičius>] [INCREMENT BY <Skaičius>])]
```

Raktiniu žodžiu `ALWAYS` liepiama generuoti reikšmę netgi tada, kai duomenų įvedimo sakinyje yra nurodyta konkreti reikšmė. `BY DEFAULT` atveju sistema reikšmę priskiria tik tuomet, kai duomenų įvedimo sakinyje nėra nurodyta jokia konkreti stulpelio reikšmė. Numatytoji generuojamųjų reikšmių aibė yra natūriniai skaičiai. Bendriau, tokia aibe gali būti sveikųjų skaičių aritmetinė progresija. Pasirinktimi `START WITH` galima nurodyti pradinę aritmetinės progresijos reikšmę, o pasirinktimi `INCREMENT BY` - progresijos skirtumą. Papildykime stulpelio *Nr* apibrėžtį nurodydami jo reikšmių parinkimo taisyklę,

```
Nr INTEGER NOT NULL PRIMARY KEY
GENERATED ALWAYS AS IDENTITY (START WITH 100, INCREMENT BY 1).
```

7.5'. Dalykinės taisyklės ir trigeriai

Apibrėžiant trigerį duomenų atnaujinimo veiksmui (UPDATE) frazę OF <stulpelių sąrašas> galima praleisti. Tuomet trigeris yra aktyvuojamas atnaujinant bet kurio lentelės stulpelio reikšmę. Siekiant efektyvaus sistemos funkcionavimo, svarbu nenurodyti per daug stulpelių.

7.5.1. Reikšmių kitimo protokolavimas

Trigeriai pakankamai dažnai naudojami duomenų keitimams protokoluoti. Kai svarbu labai aukštas duomenų patikimumas, kiekvieną duomenų pakeitimą galima automatiškai įsiminti tam skirtoje lentelėje. Toks duomenų atnaujinimo protokolas, vėliau gali būti labai naudingas, tiriant nepageidautinos situacijos susidarymo priežastis ir ieškant tokios situacijos kaltininko. Sudarykime lentelės *Vykdytojai* stulpelio *Kategorija* pasikeitimų „stebėtoją“. Pasikeitimams registruoti sukurkime lentelę *KategorijųKitimas*, kurioje saugosime darbuotojo, kuriam keičiama kategorija, numerį, kategorijos atnaujinimo momentą (datą ir tikslų laiką), DB vartotojo, kuris atlieka atnaujinimo operaciją, vardą ir senąją bei naująją kategorijos reikšmes:

```
CREATE TABLE KategorijųKitimas (
    Nr              INTEGER     NOT NULL,
    Momentas        TIMESTAMP   NOT NULL DEFAULT (CURRENT TIMESTAMP),
    Atnaujintojas   CHAR(16)    NOT NULL DEFAULT (USER),
    SenaKategorija      SMALLINT,
    NaujaKategorija     SMALLINT).
```

Kad įsiminti visus vykdytojų kategorijų pakeitimus, sukuriame trigerį

```
CREATE TRIGGER KategorijosKeitimas
    AFTER UPDATE OF (Kategorija) ON Vykdytojai
    REFERENCING OLD AS Sena
    REFERENCING NEW AS Nauja
    FOR EACH ROW MODE DB2SQL
    INSERT INTO KategorijųKitimas(Nr, SenaKategorija, NaujaKategorija)
                VALUES (Sena.Nr, Sena.Kategorija, Nauja.Kategorija).
```

Kad įsiminti ne tik atnaujintas vykdytojų kategorijas bet ir pradines, sudarome trigerį duomenų įvedimui

```
CREATE TRIGGER KategorijosĮvedimas
    AFTER INSERT ON Vykdytojai
    REFERENCING NEW AS Nauja
    FOR EACH ROW MODE DB2SQL
    INSERT INTO KategorijųKitimas(Nr, SenaKategorija, NaujaKategorija)
    VALUES (Nauja.Nr, NULL, Nauja.Kategorija).
```

Abiejų šių trigerių kamienuose nenurodomas vartotojo, kuris keičia ar įveda kategorijos reikšmę, vardas. Tam nėra būtinybės, nes vartotojo vardas (stulpelio *Atnaujintojas*) yra apibrėžtas lentelės *KategorijųKitimas* apibrėžimo sakinyje. Ten pat yra apibrėžta, kad atnaujinimo momentas (stulpelio *Momentas* reikšmė) - tai kategorijos pasikeitimo ar jos įvedimo momentas.

7.5.2. Reikalavimų reikšmėms užtikrinimas

Dar viena trigerių taikymo sritis – reikalavimų reikšmėms užtikrinimas. Mes jau aptarėme reikalavimus reikšmėms ir jų užtikrinimą. Reikalavimai, kurie išreiškiami fraze CHECK, turi būti užtikrinami visą lentelės egzistavimo laiką. Kartais svarbu užtikrinti, kad reikšmės

tenkintų konkrečius reikalavimus tik duomenų įvedimo ar jų atnaujinimo metu, bet vėliau nėra būtina ar net neįmanoma užtikrinti tų reikalavimų.

Pakankamai įprasti yra reikalavimai datoms, kuriuose apibrėžiama sąlygos šiandieninės datos atžvilgiu. Pavyzdžiui, jei kurioje nors lentelėje yra asmens gimimo datos stulpelis, tai akivaizdu, kad ta data negali būti ateityje. Panašiai, duomenų bazėje *Darbai* yra prasminga reikalauti, kad, įvedant naujus duomenis į lentelę *Projektai*, projekto pradžia nebūtų praeityje. Tarus, kad kartais registruojami ir jau prasidėję projektai, apibrėžkime reikalavimą stulpelio *Pradžia* reikšmėms, kad įvedant duomenis apie naują projektą nebūtų nurodyta, kad jis prasidėjo anksčiau nei prieš mėnesį. Visų pirma, pastebime, kad šio uždavinio negalima išspręsti taikant jau aptartus reikalavimus reikšmės, kurie apibrėžiami sukuriant lentelę. Stulpelio *Pradžia* negalima apibrėžti taip:

Pradžia DATE CHECK(*Pradžia* > CURRENT DATE - 1 MONTH).

Jei toks reikalavimas būtų įmanomas, tai praėjus nuo duomenų įvedimo ne daugiau kaip vienam mėnesiui sąlyga tikrai taptų neteisinga. Kadangi CHECK sąlygos turi būti užtikrinamos ne tik duomenų įvedimo metu, panašiam reikalavimui užtikrinti, DBVS turėtų pastoviai tikrinti, ar ši sąlyga nepasidaro neteisinga einant laikui. Todėl, DB valdymo sistemos reikalavimuose reikšmėms dažniausiai neleidžiama naudoti vardinių konstantų, kurių reikšmės priklauso anuo konkrečių aplinkybių. Taip išvengiama išorinio poveikio reikšmės teisingumui. Tai leidžia reikalavimų užtikrinimu rūpintis tik duomenų įvedimo ir atnaujinimo momentais, nereikia sąlygų iš naujo tikrinti pasikeitus dienai ar einamojo vartotojo vardui.

Reikalavimams, kurių teisingumas priklauso nuo išorinių sąlygų, užtikrinti galima naudoti trigerius. Kadangi trigeriai visuomet susiejami su konkrečia duomenų keitimo operacija, tai jie užtikrina, kad reikiama sąlygą bus tikrinama tik duomenų keitimo momentu.

Reikalavimą, kad naujai įvedamam projektui nebūtų nurodyta, jog jis prasidėjo anksčiau nei prieš mėnesį, galima užtikrinti tokiu trigeriu:

```
CREATE TRIGGER ProjektųPradžia
    NO CASCADE BEFORE INSERT ON Projektai
    REFERENCING NEW AS NaujasProjektas
    FOR EACH ROW MODE DB2SQL
    WHEN (NaujasProjektas.Pradžia < CURRENT DATE - 1 MONTH)
    SIGNAL SQLSTATE '99998' ('Per sena pradžios data').
```

Panašiai galima užtikrinti reikiamus apribojimus reikšmių atnaujinimui. Pavyzdžiui, gali būti prasminga nekeisti pradžios datos jau prasidėjusiems projektams, ypač, jei jau yra bent vienas projekto vykdytojas

```
CREATE TRIGGER ProjektųPradžiosKeitimas
    NO CASCADE BEFORE UPDATE OF (Pradžia) ON Projektai
    REFERENCING NEW AS Naujas
    FOR EACH ROW MODE DB2SQL
    WHEN (Naujas.Pradžia < CURRENT DATE AND
          EXISTS(SELECT * FROM Vykdymas WHERE Projektas = Naujas.Nr) )
    SIGNAL SQLSTATE '99997' ('Projektas jau vykdomas').
```

7.5.3. Virtualiųjų lentelių atnaujinimas

Jau minėjome, kad virtualiųjų lentelių (rodinių) duomenų atnaujinimo galimybės yra labai ribotos. Pavyzdžiui, jungtinių virtualiųjų lentelių visiškai negalima atnaujinti. Jungtiniai rodiniai labai padeda spręsti loginio duomenų nepriklausomumo problemą. Tai, kad negalima atnaujinti jungtinio rodinio, yra viena iš priežasčių, kodėl nepavyksta visiškai užtikrinti loginio duomenų nepriklausomumo skaidant lenteles. Šiuolaikinėse DBVS šią problemą galima spręsti ypatingos rūšies trigeriais. Trigeriai, kuriais užtikrinamas virtualiųjų lentelių atnaujinimas, yra žymimi raktiniu žodžiu INSTEAD OF.

Pagrindinė mintis, yra duomenų įvedimą, atnaujinimą ir trynimą rodinyje pakeisti trigerio kamienu, kuriame be apribojimų galima vykdyti kelis SQL sakinius. Kitaip tariant, radinio atnaujinimas trigeriu yra keičiamas jo pradinių lentelių keitimu.

Tarkime, duomenų bazėje yra dvi lentelės $L_1(A, B)$ ir $L_2(A, C)$ ir virtualioji lentelė $L(A, B, C)$, jungianti abiejų pradinių lentelių L_1 ir L_2 duomenis, kaip buvo apibrėžta ankstesniame skyriuje. Paprastumo dėlei, tarkime, kad A, B, C yra paprasti, nesudėtiniai atributai. Kad užtikrinti duomenų atnaujinimą virtualiojoje lentelėje L, apibrėžiame trigerį

```
CREATE TRIGGER AtnaujintiL
    INSTEAD OF UPDATE ON L
    REFERENCING OLD AS SenaL NEW AS NaujaL
    FOR EACH ROW MODE DB2SQL
    BEGIN ATOMIC
      VALUES( CASE WHEN NaujaL.A = SenaL.A THEN 0 ELSE
              RAISE_ERROR('99996', 'A negalima keisti') END) ;
      UPDATE L1 SET L1.B = NaujaL.B WHERE L1.A = SenaL.A;
      UPDATE L2 SET L2.C = NaujaL.C WHERE L2.A = SenaL.A;
    END.
```

Šiame trigeryje pavartojome funkciją `RAISE_ERROR`, kurios iškvietimas sukelia tokias pačias pasekmes, kaip anksčiau jau aptartas SQL sakinys `SIGNAL SQLSTATE`. Trigerio kamieno sakiniu `VALUES` sumodeliavome reikalavimą nekeisti reikšmių, siejančių abi primines lenteles, t.y. stulpelio (išorinio rakto) A reikšmių. Sakinio `VALUES` mums prireikė dėl to, kad trigeryje, kuriame naudojama pasirinktis `INSTEAD OF`, neleidžiama naudoti kamieno vykdymo sąlygos `WHEN`. Taip pabrėžiama, kad, atnaujinant lentelės duomenis, veiksmas atliekamas visada ir besąlygiškai arba jis baigiasi pranešimu apie klaidą. Atvejis, kai duomenys neatnaujinami dėl netenkinamos sąlygos, yra negalimas, todėl sąlyga `WHEN` šiuose trigeriuose yra neprasminga ir negalima.

Panašiai trigeriais galima sumodeliuoti ne tik duomenų atnaujinimą (`UPDATE`) virtualiosiose lentelėse, bet ir įterpimą (`INSERT`) bei trynimą (`DELETE`).

Yra keletas apribojimų, taikomų trigeriams, kurių apibrėžime yra nurodyta pasirinktis `INSTEAD OF`. Tokių trigerių, pavyzdžiui, negalima apibrėžti rodiniams, apibrėžtiems su pasirinktimi `WITH CHECK OPTION`.