7. Duomenų vientisumo užtikrinimas

7.1. Reikalavimai duomenų vientisumui

Duomenų vientisumo sąvoka siejama su informacijos, esančios duomenų bazėje, teisingumu ir pilnumu. Duomenų vientisumo užtikrinimas yra viena iš pagrindinių DBVS funkcijų. Kadangi tik atsakingas DB vartotojas gerai žino duomenų savybes, tai duomenų vientisumui užtikrinti DBVS sudaro vartotojui galimybę apibrėžti **reikalavimus duomenų vientisumui**. DBVS užtikrina, kad būtų griežtai prisilaikoma tų reikalavimų. Trečiame skyriuje jau aptarėme paprasčiausius reikalavimus. Dabar juos ir kitus reikalavimus aptarsime detaliau, o paskui sužinosime, kaip duomenų vientisumo reikalavimai apibrėžiami SQL kalba.

- **Reikšmės būtinumas**. Dalis stulpelių privalo turėti konkrečią reikšmę. Tokie stulpeliai eilutėse negali įgyti `NULL` reikšmę. Pavyzdžiui, kategorijų vientisumas reikalauja, kad visų raktą sudarančių stulpelių reikšmės būtų konkrečios. DB valdymo sistemai galima nurodyti, kad `NULL` reikšmė stulpelyje yra neleistina.
- **Apribojimai reikšmėms**. Kiekvienam stulpeliui, apibrėžiant lentelę, nurodomas jo duomenų tipas. DBVS užtikrina, kad įvedamos stulpelio reikšmės atitiktų konkretų tipą. Tačiau šito dažnai nepakanka. Pavyzdžiui, sunku įsivaizduoti, ką galėtų reikšti neigiama darbuotojo kategorija ar projekto vykdymo trukmė. DBVS leidžia nurodyti stulpelių reikšmėms konkrečius reikalavimus.
- **Lentelės raktų (kategorijų) vientisumas**. Lentelės raktas kiekvienoje lentelės eilutėje turi turėti unikalią reikšmę. Vienodos rakto reikšmės lentelėje yra neleistinos, kitaip nebus užtikrinama vienareikšmiškos identifikacijos savybė. DBVS leidžia apibrėžti stulpelių rinkinius, kurių reikšmės turi būti unikalios lentelėje.
- **Nuorodų vientisumas**. Nuorodų vientisumas reikalauja, kad kiekvieno išorinio rakto reikšmė duomenų bazėje būtų arba tuščia, arba sutaptų su pirminio rakto reikšme lentelėje, į kurią išorinis raktas nurodo. Apibrėžus lentelei išorinį raktą, DBVS užtikrina, kad nuorodų vientisumo reikalavimas nebūtų pažeistas.
- **Dalykinės taisyklės**. Kartais reikalavimus eilutės reikšmėms galima apibrėžti tik atsižvelgiant į reikšmes kitose eilutėse ar net kitose lentelėse. Pavyzdžiui, aviakompanija negali parduoti į reisą bilietų daugiau negu lėktuve yra vietų. Įvedant į DB duomenis apie naują keleivį, reikia patikrinti, ar keleivį galima užregistruoti, ar, įvedus duomenis, nebus viršytas vietų skaičius lėktuve. SQL kalboje yra trigerio sąvoka, kuria galima modeliuoti tokias konkretaus dalyko taisykles.
- **Duomenų neprieštaringumas**. Daugelis realaus pasaulio veiksmų duomenų bazėje modeliuojamos keletu SQL sakinių. Tačiau pati operacija logiškai gali būti nedaloma, pavyzdžiui, pinigų pervedimas iš vienos sąskaitos į kitą. Jei dvi sąskaitos duomenų bazėje vaizduojamos dviem eilutėmis, tai operacijai realizuoti, reikės atlikti du duomenų atnaujinimo sakinius – vienoje eilutėje sumažinti reikšmę, o kitoje – padidinti. Tokia logiškai nedaloma operacija vadinama transakcija. Atlikus tik vieną iš šių veiksmų, duomenys taptų prieštaringais. DBVS užtikrina loginį keleto sakinių vientisumą.

Jau žinome, kad konkretus stulpelis jokioje lentelės eilutėje neturės `NULL` reikšmės, jei sukurdami lentelę stulpelio apibėžimą papildysime fraze `NOT NULL`. Aptarsime kaip apibrėžiami kiti duomenų vientisumo reikalavimai.

7.2. Apribojimai reikšmėms

Apribojimai reikšmėms - tai sąlygos stulpelių reikšmėms, kurios turi būti tenkinamos visoms lentelės eilutėms. Sąlygą galima apibrėžti konkrečiam stulpeliui ar jų grupei. Kiekvieną kartą, kai į lentelę įvedama nauja eilutė ar atnaujinama esama, DBVS tikrina apibrėžtus reikalavimus. SQL sakinys nevykdomas, jei bent viena sąlyga nepatenkinama.

Reikalavimus reikšmėms galima nurodomi apibrėžiant lentelę (SQL sakiniu `CREATE TABLE`) arba keičiant jos struktūrą (sakiniu `ALTER TABLE`), pavyzdžiui,

```
ALTER TABLE Vykdytojai
ADD CONSTRAINT TeisingosKategorijos CHECK Kategorija BETWEEN 1 AND 6 .
```

Šiuo sakiniu apibrėžiamas reikalavimas lentelės *Vykdytojai* stulpelio *Kategorija* reikšmėms, pagal kurį šio stulpelio reikšmės turi būti ne mažesnės už 1 ir ne didesnės už 6. Jeigu apibrėžiant sąlygą esamai lentelei, bent vienoje eilutėje sąlyga yra netenkinama, tai DBVS neįvykdo sakinį `ALTER TABLE`. Apribojimui suteikiamas vardas *TeisingosKategorijos*, kurį galima panaudoti panaikinant šį apribojimą:

```
ALTER TABLE Vykdytojai DROP CONSTRAINT TeisingosKategorijos.
```

Apribojimas *TeisingosKategorijos* išreikštas gana paprastai, tačiau apribojimą galima išreikšti ir sudėtinga paieškos sąlyga. Paprastą sąlygą stulpelio reikšmėms galima nurodyti tiesiog stulpelio apibrėžime. Sudėtingesnės sąlygos, pavyzdžiui, kelių stulpelių tarpusavio priklausomybė, apibrėžiamos atskirai. Sukurkime lentelę *Projektai* su apribojimais reikšmėms:

```
CREATE TABLE Projektai (
    Nr           INTEGER      NOT NULL  CHECK (Nr >= 0),
    Pavadinimas  VARCHAR(254) NOT NULL,
    Svarba       CHAR(10)
                 CHECK  (Svarba IN ('Maža', 'Vidutinė', 'Didelė'))
                 DEFAULT  'Vidutinė',
    Pradžia      DATE,
    Trukmė       SMALLINT     CHECK (Trukmė > 0),
    CONSTRAINT TrumpųjųSvarba CHECK (Trukmė > 6 OR Svarba = 'Didelė') ) .
```

Šiame sakinyje yra apibrėžti keli apribojimai lentelės stulpelių reikšmėms: stulpelio *Nr* reikšmės yra neneigiami skaičiai, stulpelio *Svarba* galimų reikšmių aibę sudaro keturios išvardytos reikšmės, o stulpelio *Trukmė* reikšmių aibė yra visi teigiami sveiki skaičiai, kurių dvejetainis kodas telpa dviejuose baituose. Papildoma sąlyga, kuriai suteiktas vardas *TrumpųjųSvarba*, reikalauja, kad projektams, kurių trukmė ne didesnė už 6 mėn., būtų priskiriamas didelės svarbos požymis. Ši sąlyga nusako dviejų stulpelių reikšmių tarpusavio priklausomybę.

7.3. Lentelės raktų vientisumas

Kiekviena lentelės eilutė turi unikalų pirminio rakto reikšmių rinkinį. Sukuriant lentelę, galima nurodyti stulpelius, sudarančius lentelės pirminį raktą. Tuomet, DBVS automatiškai tikrins pirminio rakto reikšmių unikalumą. Unikalumas tikrinamas kiekvieną kartą, kai tik vykdomas `INSERT` ar `UPDATE` sakinys. Naujos eilutės įterpimas baigiasi pranešimu apie klaidą, jei jos pirminio rakto reikšmė sutampa su kitos eilutės pirminio rakto reikšme. Pirminį raktą galima apibrėžti ir jau egzistuojančiai lentelei. Apibrėžkime pirminius raktus visoms trims DB *Darbai* lentelėms:

```
ALTER TABLE Vykdytojai ADD PRIMARY KEY (Nr),
ALTER TABLE Projektai  ADD PRIMARY KEY (Nr),
ALTER TABLE Vykdymai ADD PRIMARY KEY (Projektas, Vykdytojas).
```

Dažnai pirminis raktas yra vienintelis lentelės raktas. Tačiau lentelė gali turėti ir daugiau raktų, kurių reikšmių unikalumą taip pat reikia užtikrinti. Visi kiti lentelės raktai arba, tiksliau, stulpelių rinkiniai, kurių reikšmės turi būti unikaliomis lentelėje, yra skelbiami **unikaliais indeksais**. Unikalus indeksas nebūtinai yra lentelės raktas, tai gali būti ir viršraktis. Unikaliam indeksui apibėžti daugelyje komercinių DBVS yra sakinys `CREATE INDEX` su parinktimi `UNIQUE`. Tarkime lentelėje *Projektai* negali būti dviejų projektų su tokiu pačiu pavadinimu, t.y. lentelė turi antrą raktą *Pavadinimas*. Šį stulpelį paskelbsime raktu įvykdydami sakinį:

`CREATE UNIQUE INDEX` *Raktas_Pavadinimas* `ON` *Projektai*(*Pavadinimas*).

Pirminis raktas lentelėje visada yra tik vienas (jei jis iš viso yra), todėl jam nereikia kito vardo. Visi kiti raktai turi konkrečius vardus. Pirmą kartą rakto (unikalaus indekso) vardas panaudojamas sukuriant (apibrėžiant) jį. Vėliau šį vardą galima pavartoti tik vieninteliu atveju – sunaikinant jį, pavyzdžiui,

`DROP INDEX` *Raktas_Pavadinimas*.

Tiek pirminis raktas, tiek ir raktas (unikalus indeksas) yra indekso atskiri atvejai. Indeksas bendriausia prasme yra fizinė sąvoka, todėl sakinys `CREATE INDEX` nėra standartinis. Indeksus plačiau aptarsime kitame skyriuje. Pagal standartą SQL2 stulpelius, kurių reikšmės turi būti unikalios lentelėje, galima nurodyti apibrėžiant lentelę sakiniu `CREATE TABLE`.

Kadangi kategorijų vientisumas reikalauja, kad joks lentelės rakto atributas nei vienoje eilutėje neįgytų `NULL` reikšmės, tai daugelyje komercinių DBVS yra reikalaujama, kad visi stulpeliai, sudarantys pirminį raktą arba unikalų indeksą būtų apibrėžiami su sąlyga `NOT NULL`.

Išskirtinė pirminio rakto savybė, lyginant jį su kitais raktais (unikaliais indeksais), yra galimybė dalyvauti sąryšiuose tarp lentelių. Sąryšiai tarp lentelių išreiškiami išoriniais raktais.

7.4. Išoriniai raktai

Neformaliai nuorodų vientisumas reiškia reikšmių atitinkamybę tarp logiškai susijusių lentelių. Nuorodų vientisumu užtikrinama pastovi atitinkamybė tarp pirminio rakto ir su juo susijusių išorinių raktų. Kodas (E.F. Codd) taip apibrėžė nuorodų vientisumą: "Reliacinėje duomenų bazėje kiekvienai išorinio rakto reikšmei, jei ji nėra `NULL` reikšmė, turi egzistuoti konkreti pirminio rakto reikšmė".

Pirminių ir išorinių raktų tarpusavio ryšys yra nustatomas sudarant DB reliacinę schemą. Dažnai šis ryšys vadinamas ryšiu tarp **pagrindinės lentelės** ir **priklausomos lentelės** arba **tarp** "**tėvo" ir "vaiko"**. Kiekvienai pagrindinės lentelės eilutei gali atitikti kelios priklausomos lentelės eilutės. Pagrindinės lentelės eilutę vadinsime **pagrindine eilute**, o ją atitinkančias priklausomos lentelės eilutes – **priklausomomis eilutėmis**. Kadangi ne tuščiai išorinio rakto reikšmei visada atitinka tik viena pirminio rakto reikšmė, tai priklausomai eilutei visada galima rasti ne daugiau kaip vieną pagrindinę eilutę. Kadangi lentelė gali turėti kelis išorinius raktus, tai kiekvieną priklausomą eilutę gali atitikti ir kelios pagrindinės eilutės, tačiau po vieną kiekvienam išoriniam raktui, t.y. skirtingose pagrindinėse lentelėse. Lentelės *Vykdytojai* ir *Projektai* yra pagrindinės, *Vykdymas* yra priklausoma lentelė. Kadangi lentelė *Vykdymas* turi du išorinius raktus, tai kiekvieną jos eilutę atitinka dvi pagrindinės eilutės: viena lentelėje *Vykdytojai*, o kita - *Projektai*. Priklausomos lentelės eilutę, kuriai pagrindinėje lentelėje nėra ją atitinkančios pagrindinės eilutės, vadinsime **našlaite**. Jei priklausomoje lentelėje yra eilutė – našlaitė, tai tokioje duomenų bazėje yra pažeista nuorodų vientisumo taisyklė. Todėl eilučių – našlaičių duomenų bazėje turi nebūti.

Tam, kad duomenų bazėje būtų užtikrinamas nuorodų vientisumas, DBVS turi pastoviai stebėti naujų duomenų įvedimo ir esamų duomenų atnaujinimo operacijas. DBVS stebi, kad neatsirastų eilučių – našlaičių. Išskirkime keturis būdingus duomenų keitimus, kurių metu gali būti pažeidžiamas nuorodų vientisumas (gali atsirasti eilutės-našlaitės), ir sudarykime reikalavimus, kurių prisilaikant taip neatsitiks.

1) **Naujos priklausomos eilutės įterpimas**. Įterpiant naują eilutę į priklausomą lentelę, kiekvieno išorinio rakto reikšmė turi atitikti konkrečią pirminio rakto reikšmę pagrindinėje lentelėje, kitaip naujoji eilutė bus našlaite. Pastebėsime, kad naujos eilutės įvedimas pagrindinėje lentelėje nesudaro sunkumų.
2) **Išorinio rakto atnaujinimas priklausomoje eilutėje**. Išorinio rakto reikšmę galima keisti tik tuomet, jei naujoji reikšmė atitinka konkrečią pirminio rakto reikšmę, kitaip priklausoma eilutė taps našlaite.
3) **Pagrindinės eilutės šalinimas**. Jeigu pašalinsime pagrindinę eilutę, kuriai konkrečioje priklausomoje lentelėje yra nuo jos priklausomų eilučių, tai tos priklausomos eilutės taps našlaitėmis. Kadangi pagrindinė eilutė nebūtinai turi priklausomų eilučių, tai priklausomos eilutės šalinimas nesudaro sunkumų.
4) **Pirminio rakto atnaujinimas pagrindinėje eilutėje**. Jei pagrindinei eilutei pakeisime jos tapatumo požymį (pirminio rakto reikšmę), tai nuo jos priklausomos eilutės taps našlaitėmis, kadangi jų išorinio rakto reikšmės rodys į jau neegzituojančią pirminio rakto reikšmę.

Pirmuosius du uždavinius DBVS sprendžia be vartotojo. Pirmasis, kuris susidaro įterpiant naują eilutę į priklausomą lentelę, sprendžiamas tikrinant ar naujosios eilutės išorinio rakto reikšmė kartu yra ir pagrindinės lentelės pirminio rakto reikšmė. DBVS neleidžia įterpti (įvykdyti `INSERT` sakinio) naujos eilutės, jei joje nurodytai išorinio rakto reikšmei nerandamas atitikmuo pagrindinėje lentelėje. Antrasis uždavinys (išorinio rakto atnaujinimas priklausomoje eilutėje) sprendžiamas panašiai. Sistema tikrina, ar vykdant `UPDATE` sakinį išorinio rakto reikšmė nėra keičiama į neegzistuojančią pirminio rakto reikšmę. Priklausomai eilutei neleidžiama tapti našlaite. DBVS leidžia keisti išorinio rakto reikšmę, jei naujoji reikšmė atitinka konkrečią pirminio rakto reikšmę.

Paskutiniai du uždaviniai yra sudėtingesni. Panagrinėkime, kaip galima pasielgti aplinkybėmis, kurios susidaro šalinant iš lentelės *Vykdytojai* eilutę, atitinkančią darbuotoją Nr. 1. Tiksliau, kas galėtų atsitikti su lentelės *Vykdymas* eilutėmis, kuriose minimas šis vykdytojas? Priklausomai nuo konkrečių aplinkybių galima:

- uždrausti šalinti iš pagrindinės lentelės *Vykdytojai* eilutę, jei bent vienoje priklausomos lentelės *Vykdymas* eilutėje yra minimas šalinamasis vykdytojas. Šalinti bus galima, pavyzdžiui, tik tuomet kai visiems projektams, kuriuose dalyvavo vykdytojas Nr. 1, bus surasti kiti vykdytojai, pakeisiantys šalinamąjį;
- kartu pašalinti ir visas priklausomos lentelės *Vykdymas* eilutes, kuriose minimas šalinamasis vykdytojas Nr. 1. Taip pašalinami duomenys ne tik apie patį vykdytoją, bet ir apie jo dalyvavimą projektuose;
- visose priklausomose eilutėse stulpeliui *Vykdymas.Vykdytojas* priskirti reikšmę `NULL`, t.y. "pamiršti" vykdytoją, kuris dalyvavo projektuose ir "prisiminti" tik tai, kad buvo vykdytojas, kuris dirbdamas konkrečiose pareigose skyrė projektui konkretų kiekį valandų, bet jo numeris ir juo labiau pavardė – nežinomi, nes jis jau nedirba;
- visose priklausomose eilutėse stulpeliui *Vykdymas.Vykdytojas* priskirti reikšmę pagal nutylėjimą, pavyzdžiui, numerį darbuotojo, kuris laikinai pavaduoja visus išeinančius iš darbo.

Šalinant įstaigos darbuotoją, kai kurie sprendimo būdai yra labiau pagrįsti negu kiti. Tačiau, kitomis aplinkybėmis, teisingesni gali būti kiti būdai. Konkrečiose aplinkybėse, visi šie uždavinio sprendimai yra prasmingi.

Paskutinįjį, pirminio rakto atnaujinimo pagrindinėje eilutėje uždavinį, galima spręsti panašiai kaip ir trečiąjį, pagrindinės eilutės šalinimo uždavinį.

Apibrėžiant išorinį raktą galima nurodyti sistemai, kaip spręsti paskutiniuoius du uždavinius. Kiekvienam išoriniam raktui galima nurodyti konkrečią pagrindinės eilutės šalinimo taisyklę ir konkrečią jos tapatumo požymio atnaujinimo taisyklę.

Pagrindinė eilutė gali būti šalinama pagal vieną iš keturių taisyklių:

- `RESTRICT` – uždrausti pagrindinės eilutės šalinimą, jei ši turi priklausomų eilučių;
- `CASCADE` – šalinant pagrindinę eilutę, pašalinti ir visas priklausomas eilutes;

- SET NULL – šalinant pagrindinę eilutę, visose priklausomose eilutėse išorinio rakto stulpeliams priskirti NULL reikšmę;
- DEFAULT – šalinant pagrindinę eilutę, visose priklausomose eilutėse išorinio rakto stulpeliams priskirti reikšmę pagal nutylėjimą.

Ne visos šiuolaikinės komercinės DBVS leidžia apibrėžti visas keturias taisykles. DB2, pavyzdžiui, neleidžia nurodyti išorinio rakto reikšmės priskyrimo pagal nutylėjimą.

SQL2 standartas leidžia nurodyti vieną iš keturių taisyklių, apibrėžiančių DBVS veiksmus atnaujinant pagrindinės eilutės pirminio rakto reikšmes:

- RESTRICT – uždrausti pagrindinės eilutės pirminio rakto reikšmių atnaujinimą, jei yra bent viena nuo jos priklausoma eilutė;
- CASCADE – atnaujinant pagrindinės eilutės pirminio rakto reikšmes, atnaujinti ir visų priklausomų nuo jos eilučių išorinių raktų reikšmes;
- SET NULL – atnaujinant pagrindinės eilutės pirminio rakto reikšmes, visų priklausomų nuo jos eilučių išorinių raktų stulpeliams priskirti NULL reikšmę;
- DEFAULT – atnaujinant pagrindinės eilutės pirminio rakto reikšmes, visų priklausomų nuo jos eilučių išorinių raktų stulpeliams priskirti reikšmę pagal nutylėjimą.

Pagrindinės lentelės eilučių šalinimo ir atnaujinimo taisyklės nurodomos apibrėžiant išorinį raktą. Paprastai lentelės išoriniai raktai apibrėžiami sukuriant lentelę. Tačiau juos galima apibrėžti sakiniu ALTER TABLE ir vėliau. Duomenų bazėje *Darbai* tik viena lentelė *Vykdymas* turi du išorinius raktus. Apibrėžkime išorinį raktą į lentelę *Vykdytojai* su pagrindinės eilutės šalinimo taisykle CASCADE ir atnaujinimo taisykle RESTRICT:

```
ALTER TABLE Vykdymas ADD FOREIGN KEY Į_Vykdytojus (Vykdytojas)
REFERENCES Vykdytojai ON DELETE CASCADE ON UPDATE RESTRICT.
```

Apibrėžiant išorinį raktą, pagrindinė lentelė (*Vykdytojai*) turi būti jau sukurta ir jos pirminis raktas privalo būti apibrėžtas. Išorinį raktą į lentelę *Projektai* su pagrindinės eilutės šalinimo taisykle RESTRICT ir su ta pačia atnaujinimo taisykle galima apibrėžti taip:

```
ALTER TABLE Vykdymas ADD FOREIGN KEY Į_Projektus (Projektas)
REFERENCES Projektai ON DELETE RESTRICT ON UPDATE RESTRICT.
```

Šiuose apibrėžimuose pavartoti vardai *Į_Vykdytojus* ir *Į_Projektus* yra išorinių raktų vardai. Kitą kartą išorinio rakto vardą galima panaudoti tik šalinant jį. Norint pakeisti pagrindinės eilutės šalinimo ar atnaujinimo taisyklę, iš pradžių reikia pašalinti senąjį išorinį raktą, o po to apibrėžti naują. Išorinis raktas pašalinamas tuo pačiu sakiniu ALTER TABLE, pavyzdžiui,

```
ALTER TABLE Vykdymas DROP FOREIGN KEY Į_Projektus .
```

7.5. Dalykinės taisyklės ir trigeriai

Duomenų vientisumas yra susijęs su konkrečiomis aplinkybėmis, pavyzdžiui, konkrečios įstaigos vidine darbo tvarka ir galiojančiomis joje taisyklėmis. Įstaigoje, kurią atitinka duomenų bazė *Darbai*, gali galioti tokie vidiniai reikalavimai-taisyklės:

- kiekvienas darbuotojas negali dalyvauti daugiau negu trijuose projektuose;
- kiekvienas darbuotojas negali vadovauti daugiau negu vienam projektui.

Standarte SQL1 tokios taisyklės nenagrinėjamos. Už dalykinių taisyklių silaikymąsi atsako DB vartotojai ir jų sudarytos programos.

Reikalavimų tikrinimas taikomosiomis programomis turi kelis svarbius trūkumus:

- **Veiksmų dubliavimas**. Jei kelios programos atlieka panašias konkrečios lentelės duomenų atnaujinimo operacijas, tai kiekviena jų turi paprogramius, užtikrinančius dalykinių taisyklių laikymąsi. Net jei tokie paprogramiai yra bendro naudojimo, tai kartojasi programos dalis, kurioje prie konkrečių sąlygų kviečiamas konkretus paprogramis.

- **Suderinamumo sunkumai**. Jeigu keletas programų, kurias sudarė skirtingi programuotojai atnaujina konkrečios lentelės duomenis, tai tikimybė, kad dalykinės taisyklės bus užtikrinamos skirtingai, yra didelė.
- **Sunkumai keičiant taisykles.** Pasikeitus taisyklėms, reikia peržiūrėti visas programas, kuriose užtikrinamos šios taisyklės, ir teisingai sutvarkyti jas.
- **Sudėtingumas.** Dažnai įstaigose galioja daug taisyklių. Todėl, netgi nedidelė programa, kuri atlieka duomenų atnaujinimą vienoje lentelėje, gali tapti gana sudėtinga dėl reikalavimo prisilaikyti nustatytų taisyklių.

1986 m. DB valdymo sistemoje Sybase pirmą kartą buvo pavartota **trigerio** sąvoka. Panaudojant trigerius dalykinės taisyklės tapo DB dalimi. 90-jų pradžioje trigeriai buvo realizuoti jau daugumoje komercinių DBVS.

Apibrėždamas trigerį, vartotojas su konkrečiu lentelės duomenų keitimo įvykiu, susieja konkrečius veiksmus, kurios DBVS įvykdo kiekvieną kartą, kai tas įvykis atsitinka. Yra trys duomenų keitimo įvykiai, kuriuos atitinka trys duomenų atnaujinimo sakiniai: `INSERT`, `UPDATE` ir `DELETE`. Veiksmai, kuriuos reikia atlikti įvykus konkrečiam įvykiui, apibrėžiami SQL sakiniais.

Nors trigerius gana greitai įdiegė daugelis komercinių RDBVS, tačiau šios sąvokos nėra ir SQL2 standarte. Todėl skirtingose DBVS trigeriai apibrėžiami gana skirtingai. Apibrėždami trigerius prisilaikysime DB2 SQL sintaksės.

Trigeriai kuriami sakiniu `CREATE TRIGGER`, kuriame nurodoma:

- duomenų atnaujinimo įvykis (angl. *triggered action*) (`INSERT`, `UPDATE` ar `DELETE`), su kuriuo susiejamas trigeris;
- trigerio kvietimo momentas: prieš vykdant duomenų atnaujinimą ar po jo;
- trigerį kvietimo dažnis: kviesti jį tik vieną kartą vykdant SQL duomenų atnaujinimo sakinį, ar daryti tai kiekvienai atnaujinamai eilutei;
- trigerio kūnas - vienas ar keli SQL sakiniai, kuriuos reikia įvykdyti, kai trigeris iškviečiamas;
- trigerio kūno vykdymo sąlyga: iškviesto kūno sakinius vykdyti besąlygiškai ar tik tuomet, kai patenkinta konkreti sąlyga.

Sukurkime trigerį, kuris atliktų lentelės *Vykdymas* išorinio rakto *Vykdytojas* vaidmenį pagal pirminio rakto atnaujinimo taisyklę `CASCADE`:

```
CREATE TRIGGER ModifikuotiNr
    AFTER UPDATE OF Nr ON Vykdytojai
    REFERENCING OLD AS SeniVykdytojai
                NEW AS NaujiVykdytojai
    FOR EACH ROW MODE DB2SQL
    UPDATE Vykdymas SET Vykdymas.Vykdytojas = NaujiVykdytojai.Nr
                    WHERE Vykdymas.Vykdytojas = SeniVykdytojai.Nr.
```

Šis trigeris užtikrins, kad kai tik lentelės *Vykdytojai* konkrečioje eilutėje pasikeis stulpelio *Nr* reikšmė, tai tas pasikeitimas atsispindės ir lentelėje *Vykdymas*, t.y. pakeitus kurio nors darbuotojo numerį lentelėje *Vykdytojai* jis bus pakeistas ir lentelėje *Vykdymas.* Fraze `REFERENCING` priskiriamas laikinas vardas *SeniVykdytojai* lentelės *Vykdytojai* būsenai prieš duomenų keitimą, ir vardas *NaujiVykdytojai* tos lentelės būsenai po keitimo. Taip sudaroma galimybė trigerio kūne vartoti tiek senąją vykdytojo numerio reikšmę, tiek ir naująją. Suprantama, šis trigeris yra beprasmis, jeigu DBVS leidžia apibrėžti išorinį raktą. Išorinis raktas paprastesnis už trigerį, todėl tokiam veiksmui realizuoti reikia vartoti išorinį raktą, o ne trigerį. Tačiau yra nemažai DB vartotojų, kurie tvirtina, kad trigeriai yra daug patogesni ir atiduoda pirmenybę jiems.

Sukurkime trigerį, užtikrinantį, kad darbuotojas nedalyvautų daugiau negu trijuose projektuose:

```
CREATE TRIGGER MaxVykdymųKiekis
    NO CASCADE BEFORE INSERT ON Vykdymas
    REFERENCING NEW AS NaujasVykdymas
    FOR EACH ROW MODE DB2SQL
    WHEN ((SELECT COUNT(*) FROM Vykdymas
            WHERE Vykdymas.Vykdytojas = NaujasVykdymas.Vykdytojas) >= 3 )
    SIGNAL SQLSTATE '99999' ('Viršytas projektų kiekis') .
```

Frazėje `WHEN` nurodyta papildoma sąlyga trigerio kūno vykdymui. Ši sąlyga tikrinama, kai į lentelę *Vykdymas* įterpiama nauja eilutė. Trigerio kūnas vykdomas tik tada, kai lentelėje *Vykdymas* jau yra pažymėta, kad vykdytojas, kurio duomenys įvedami, jau dalyvauja bent trijuose projektuose.

Trigerio kūno sakinyje `SIGNAL SQLSTATE` yra nurodytas klaidos kodas ir prasmingas pranešimas apie klaidą, kurie perduodami vartotojui, jei stulpelio *Vykdymas*.*Vykdytojas* reikšmės atnaujinimas pažeistų nustatytą taisyklę. Paprastai šis SQL sakinys yra vartojamas tik trigerio kūne. Juo nutraukiamas pagrindinis duomenų atnaujinimo sakinys ir pranešama apie klaidą.

Tam, kad visiškai užtikrinti minėtą dalykinę taisyklę, reikia dar vieno panašaus trigerio susieto su vykdytojo numerio keitimu lentelėje *Vykdymas*.

Kitas dažnai trigeriuose vartojamas yra `SET` sakinys. Juo priskiriama reikšmė stulpeliui konkrečioje eilutėje. Šis sakinys vartojamas tik kartu su frazėmis `BEFORE` (trigerį kviesti prieš atnaujinant duomenis) ir `FOR EACH ROW` (trigerį kviesti kiekvienai atnaujinamai eilutei). Paprastai juo priskiriamos reikšmės naujos eilutės stulpeliams. Prisiminkime, kad stulpeliai *Vykdytojai*.*Nr* ir *Projektai*.*Nr* yra tapatumo požymiai. Įterpiant naują eilutę, atitinkančią naują darbuotoją, jam suteikiamas iki šiol nenaudotas tapatumo numeris. Tokio numerio (naujos eilutės stulpelio *Vykdytojai*.*Nr* reikšmės) parinkimą galima pavesti tokiam trigeriui:

```
CREATE TRIGGER NaujasVykdytojoNr
    NO CASCADE BEFORE INSERT ON Vykdytojai
    REFERENCING NEW AS NaujasVykdytojas
    FOR EACH ROW MODE DB2SQL
    BEGIN ATOMIC
      SET NaujasVykdytojas.Nr = (SELECT MAX(Nr)+1 FROM Vykdytojai);
      SET NaujasVykdytojas.Nr = COALESCE(NaujasVykdytojas.Nr, 1);
    END.
```

Šio trigerio kūną sudaro du SQL sakiniai, todėl jie užrašyti tarp bazinių žodžių `BEGIN ATOMIC` ir `END`. Trigerio kūno sakiniai atskiriami kabliataškiais. Pirmuoju kūno sakiniu apibrėžiama, kad kiekvienos naujos eilutės stulpeliui *Nr* priskiriama didžiausia iki šiol buvusi lentelėje šio stulpelio reikšmė pridėjus prie jos vienetą. Antrasis sakinys yra skirtas atvejui, kai įvedama pirmoji lentelės *Vykdytojai* eilutė (tuomet `MAX`(*Nr*) yra `NULL`). Du trigerio kūno sakinius galima pakeisti ir vienu sakiniu su sąlyginiu reiškiniu:

```
SET NaujasVykdytojas.Nr = (SELECT COALESCE(MAX(Nr),0)+1 FROM Vykdytojai).
```

Su kiekvienu įvykiu galima susieti kelis trigerius. Jei įvykis iššaukia kelis trigerius, tai jie vykdomi jų sukūrimo tvarka.

Vykdant trigerio kūno sakinius galimas duomenų atnaujinimas, kuris gali sukelti įvykius, reikalaujančius iškviesti kitus trigerius ar net jį patį. Vieno trigerio vykdymas gali iššaukti daugelio trigerių vykdymą. Kad tokia įvykių seka netaptų begaline, sukuriant trigerį, reikia nurodyti frazę `NO CASCADE`. Šia parametru nurodoma, kad apibrėžiamo trigerio vykdymas neiššauks kitų trigerių vykdymo. Svarbu prisiminti, kad trigeris gali pakeisti ir pagrindinės duomenų atnaujinimo operacijos rezultatą, t.y. pakeitus konkrečią reikšmę, ją dar kartą gali pakeisti trigeris, pavyzdžiui, atkurti reikšmę, buvusią prieš keitimą!

Pagrindinis trigerių privalumas - jais apibrėžtos dalykinės taisyklės saugomos duomenų bazėje. Trigeriai:

- **pagreitina programavimą** - jie įsimenami duomenų bazėje ir veiksmų nereikia kartoti kiekvienoje programoje;
- **palengvina dalykinių taisyklių užtikrinimą** – apibrėžtas trigeris visuomet iškviečiamas reikiamu momentu;
- **yra globalūs** - pasikeitus dalykinėms taisyklėms, tereikia pakeisti trigerį viename egzemplioriuje, o ne visas programas, kuriose yra taisyklių užtikrinimo sakiniai.

Tačiau trigeriai turi ir trūkumų:

- **DB sudėtingumas**. Trigeriams tapus DB dalimi, ši tampa daug sudėtingesne. Kai duomenų bazėje yra daug trigerių, tai netgi paprastos taikomosios programos sukūrimas gali būti sudėtingu uždaviniu dėl sudėtingos trigerių veiksmų logikos.
- **Paslėpta logika**. Kai dalykinės taisyklės yra "paslėptos" duomenų bazėje, paprasti duomenų atnaujinimo veiksmai, gali iššaukti sudėtingą duomenų analizę. Taikomosios programos kūrėjas neturi galimybės tvarkyti visus vykstančius duomenų bazėje procesus, kadangi veiksmai gali iššaukti kitus, paslėptus veiksmus.
- **Paslėpta įtaka našumui**. Kai bazėje yra trigerių, duomenų atnaujinimas nėra skaidrus. Paprastas reikšmės atnaujinimas gali iššaukti daugelio duomenų apdorojimą, kuriam reikės daug laiko. Gali būti sunku suvokti, kodėl duomenų atnaujinimas vyksta taip lėtai.

Trigeris nustoja egzistavęs, kai jis sunaikinamas sakiniu

```
DROP TRIGGER <trigerio vardas> .
```

7.6. Transakcijos

Transakcijos sąvoka yra viena iš pagrindinių DBVS. Transakcija jos bendriausia prasme - tai loginis darbo su duomenimis vienetas. Kitaip tariant, tai - SQL sakiniai, kurie loginiu požiūriu yra nedalomi. Kiekvienas atskirai paimtas sakinys sprendžia uždavinio dalį, bet visą uždavinį išsprendžia tik visų sakinių įvykdymas.

Tarkime, darbuotojas Nr. 3 (Gražulytė) išeina iš darbo, ir jo darbai yra pavedamas darbuotojui Nr. 2. Išeinančiojo darbai paskiriami naujam vykdytojui, priskiriant jam visas darbo valandas, kurias kiekvienam projektui turėjo skirti išeinantysis darbuotojas. Šiuo atveju tai padaryti galima, nes visuose projektuose, kuriose dalyvauja vykdytojas Nr. 3, dalyvauja ir Nr. 2. Šiam uždaviniui spręsti reikia keisti duomenis, esančius dviejose lentelėse: *Vykdytojai* ir *Vykdymas*. Uždavinį sprendžia tokie du SQL sakiniai:

```
UPDATE Vykdymas AS A
SET A.Valandos = A.Valandos +
                 (SELECT SUM(B.Valandos) FROM Vykdymas AS B
                  WHERE B.Vykdytojas = 3 AND B.Projektas = A.Projektas)
WHERE Vykdytojas = 2 ;
DELETE  FROM Vykdytojai WHERE Nr = 3 .
```

Pirmuoju sakiniu vykdytojui Nr. 2 yra pridedamos visos vykdytojo Nr. 3 darbo valandos pagal kiekvieną projeką. Antruoju sakiniu vykdytojo Nr. 3 duomenys yra pašalinami iš lentelės *Vykdytojai*. Tarkime, kad lentelė *Vykdytojai* turi jau apibrėžtą išorinį raktą *Į_Vykdytojus*, todėl reikiamos eilutės iš šios lentelės bus pašalintos automatiškai, vykdant antrąjį transakcijos sakinį. Mūsų transakcija yra gana paprasta – ją sudaro tik du SQL sakiniai. Dažnai transakcijas sudaro daug daugiau sakinių.

Reliacinėje DBVS transakcijos paklūsta reikalavimui: "transakcijos sakiniai sudaro nedalomą vienetą: arba visi jie yra sėkmingai įvykdomi arba nei vienas iš jų nėra įvykdomas".

DBVS reikia užtikrinti šį reikalavimą netgi tuomet, kai sėkmingai įvykdžius dalį sakinių, nėra galimybės sėkmingai pratęsti darbą. DBVS privalo užtikrinti, kad dalinis transakcijos įvykdymas neatsispindėtų duomenų bazėje.

Tarkime, dėl konkrečių priežasčių (pvz., sakiniui įvykdyti vartotojas neturi atitinkamos teisės-privilegijos) nepavyko sėkmingai įvykdyti pirmąjį sakinį. Tuomet, norint išsaugoti duomenų neprieštaringumą, antrąjį sakinį negalima vykdyti. Antrąjį sakinį galima vykdyti tik sėkmingai įvykdžius pirmąjį. Tačiau sėkmingai įvykdžius pirmąjį SQL sakinį, nėra garantijos, kad pavyks tai padaryti ir su antruoju. Sėkmingai įvykdžius pirmąjį ir nepavykus sėkmingai užbaigti antrojo sakinio (jei pvz., antrajam sakiniui įvykdyti vartotojas neturi reikiamos teisės ar jo įvykdymui pritrūksta sistemos resursų) duomenys tampa prieštaringais: papildomi darbai vykdytojui Nr. 2 jau paskirti, bet juos vis dar atlieka ir vykdytojas Nr. 3.

Kad apsaugoti nuo tokių blogų situacijų reikia priemonės, kuri nesėkmingai pasibaigus antrajam sakiniui, leistų atšaukti ir jau atliktus pakeitimus, t.y. anuliuojanti pirmojo sakinio rezultatą. Tokia priemonė, kuri leidžia užfiksuoti (įtvirtinti) visus iki tol padarytus pakeitimus arba atšaukti juos yra **transakcija**. Transakcija leidžia keletą SQL sakinių naudoti kaip loginį vienetą, t.y. kaip vieną nedalomą sakinį.

Transakcija - tai SQL sakinių seka, kuri vieną neprieštaringą DB būseną perveda į kitą neprieštaringą būseną, bet duomenų neprieštaringumas nėra užtikrinamas tarp sakinių. Transakcija yra arba visa įvykdoma, arba visa anuliuojama (visi atlikti DB pakeitimai atšaukiami). Logiškai visa transakcijos sakinių seka yra vienas veiksmas. Vartotojas, nusprendęs, kaip jam užbaigti transakciją, iškviečią vieną iš SQL dviejų sakinių:

- `COMMIT` – užbaigti transakciją sėkmingai, įteisinant (užfiksuojant) visus, padarytus duomenų bazėje pakeitimus,
- `ROLLBACK` - užbaigti transakciją nesėkmingai, anuliuojant (atšaukiant) visus, padarytus duomenų bazėje pakeitimus nuo pat transakcijos pradžios.

Kiekvienu iš šių dviejų sakinių transakcija yra užbaigiama - kiti SQL sakiniai bus jau kitos transakcijos dalis.

Panaudojant transakcijos užbaigimo sakinius (`COMMIT` ir `ROLLBACK`) jau pateiktą dviejų SQL sakinių transakciją reikia vykdyti taip:

- vykdome pirmąjį sakinį;
- jei sakinys buvo įvykdytas sėkmingai, tai vykdome antrąjį sakinį;
- jei iki šiol nebuvo klaidų, tai vykdome sakinį `COMMIT`, priešingu atveju – `ROLLBACK`.

Transakcijos sakiniai vykdomi nuosekliai. Įvykus pirmai klaidai, t.y. nepavykus įvykdyti konkretaus sakinio, vykdomas sakinys `ROLLBACK`, o visi kiti iki tol neįvykdyti transakcijos ir lieka nevykdyti. Sėkmingai įvykdžius visus transakcijos sakinius, įvykdomas sakinys `COMMIT`. Pavaizduokime šią procedūrą schema.

Procedūra SQL sakinių sekai S_1, S_2,…,S_n, kuri sudaro transakciją, įvykdyti.
for i := 1 to n
  execute S_i ;
  if SQL error then
    goto error ;
endfor
`COMMIT` ;
return success;
error:
`ROLLBACK`;
return failure;

Jei transakcijos nepavyko sėkmingai užbaigti, pašalinus nesėkmės priežastį, transakciją galima pakartoti iš pradžių. Transakcija yra ne tik darbo su duomenimis vienetas, bet ir duomenų atstatymo vienetas - įvykus klaidai atstatoma būsena, kuri buvo prieš pradedant atnaujinti duomenis.

Principas “viskas arba nieko”, kurį DBVS taiko SQL sakinių, sudarančių transakciją, atžvilgiu, reikalauja didelių DBVS pastangų ir kompiuterio resursų. Skirtingose DBVS šiam principui realizuoti yra naudojami skirtingi metodai, bet visi jie remiasi **transakcijų žurnalu**.

Kiekvieną kartą, kai duomenų bazėje atliekamas duomenų atnaujinimas, DBVS kiekvienai atnaujintai eilutei papildo transakcijų žurnalą įrašu. Įraše įsimenama du eilutės egzemplioriai: eilutės būsena prieš atnaujinimą ir po atnaujinimo. Tik po to, kai žurnale padaromas įrašas, DBVS atnaujina duomenis bazėje. Įvykdžius sakinį `COMMIT`, žurnale pažymima transakcijos pabaiga. Vartotojui vykdant `ROLLBACK`, DBVS kreipiasi į žurnalą ir iš jo atkuria DB būseną, buvusią prieš prasidedant transakcijai. Transakcijai pasibaigus, įrašus apie apie jos vykdymą galima pašalinti iš žurnalo.

DB administratorius gali atkurti neprieštaringą DB būseną netgi po avarinio sistemos darbo nutraukimo, pavyzdžiui, dingus elektros srovei. Tai padaroma iškvietus specialią DBVS programą, kuri pagal transakcijų žurnalą suranda visas nepabaigtas transakcijas ir visų jų atliktus duomenų pakeitimus anuliuoja.

Transakcijų žurnalui reikia papildomos vietos kompiuterio atmintyje. Didelis kiekis atnaujinamų vienoje transakcijoje duomenų gali būti didelia kliūtimi sėkmingai užbaigti transakciją. Gali atrodyti, kad atnaujinant duomenis praktiškai nereikalinga papildoma atmintis, nes vieni duomenys keičiami kitais, užimančiais kompiuterio atmintyje panašaus dydžio vietą. Tačiau, keičiant didelį kiekį duomenų, transakcijų žurnalui gali prireikti tiek papildomos atminties, kad ši savo dydžiu viršys visų bazėje esančių duomenų apimtį. Taip atsitinka dėl dviejų kiekvienos keičiamos eilutės egzempliorių saugojimo. Kita neigiama transakcijų žurnalo savybė – jo atnaujinimui reikia sistemos resursų, o tai mažina duomenų apdorojimo našumą.

Transakcijų užtikrinimo uždavinį labai pasunkina tai, kad duomenų baze vienu metu gali naudotis daug vartotojų, t.y. vienu metu asinchroniškai gali būti vykdoma daug transakcijų. DB valdymo sistemai dėl to kylančias problemas ir jų sprendimo metodus aptarsime kitame skyriuje.

Pabaigai, keletas papildomų pastabų:

- daugelyje komercinių DBVS vienu metu konkretus vartotojas ar programa gali vykdyti tik vieną transakciją, transakcijos negali būti "įdėtos" viena į kitą;
- jei vartotojo ar programos darbas baigiamas atsijungiant nuo DB sakiniu `CONNECT RESET`, prieš tai neįvykdžius jokio transakcijos užbaigimo sakinio, sistema automatiškai įvykdo sakinį `COMMIT`;
- vienas SQL sakinys negali būti įvykdytas iš dalies, t.y. jis visuomet visiškai įvykdomas arba visiškai neįvykdomas.