7.5. Dalykinės taisyklės ir trigeriai
7.5.1. Dalykinių taisyklių užtikrinimas

Duomenų vientisumas yra tampriai susijęs su vidine konkrečios organizacijos darbo tvarka ir galiojančiomis joje taisyklėmis, pvz.:

- darbuotojas negali dalyvauti daugiau negu 3 projektuose;
- darbuotojas negali vadovauti daugiau negu 1 projektui.

DBVS užtikrinamos **ĮSV** (*ECA, Event-Condition-Action*) taisyklės **Įvykis – Sąlyga – Veiksmas**

7.5.2. Trigeriai

Dalykinių taisyklių užtikrinimo reikalingumas pagrįstas 1976 m. straipsnyje (IBM).

Pradėtos diegti po 10 m. **trigerio** pavadinimu:

- su kiekvienu duomenų atnaujinimo įvykiu (**INSERT**, **UPDATE**, **DELETE**) susiejamas veiksmas,
- apibrėžtąjį veiksmą DBVS įvykdo kiekvieną kartą įvykiui atsitikus;
- galima apibrėžti papildomą sąlygą veiksmams atlikti.

Standartizuoti SQL:1999.

Trigeriai kuriami sakiniu **CREATE TRIGGER**, nurodant:

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

Trigeris, realizuojantis dalykinę taisyklę „darbuotojas negali dalyvauti daugiau nei 3 projektuose“:

```
CREATE TRIGGER MaxVykdymųSkaičiusĮterpiant
BEFORE INSERT ON Vykdymas
REFERENCING NEW AS NaujasVykdymas
FOR EACH ROW
WHEN ((SELECT COUNT(*) FROM Vykdymas
        WHERE Vykdymas.Vykdytojas =
                NaujasVykdymas.Vykdytojas) >= 3)
SIGNAL SQLSTATE '99999' ('Viršytas projektų skaičius')
```


Visiškam šios taisyklės užtikrinimui, reikalingas dar vienas trigeris įvykiui **UPDATE**:

```
CREATE TRIGGER MaxVykdymųSkaičiusKeičiant
BEFORE UPDATE OF Vykdytojas ON Vykdymas
REFERENCING NEW AS NaujasVykdymas
FOR EACH ROW
WHEN ((SELECT COUNT(*) FROM Vykdymas
        WHERE Vykdymas.Vykdytojas =
                NaujasVykdymas.Vykdytojas) >= 3)
SIGNAL SQLSTATE '99998' ('Viršytas projektų skaičius')
```


Sakiniu **SET** galima priskirti reikšmę stulpeliui.

Pvz., automatinis numerio parinkimas vykdytojams:

```
CREATE TRIGGER NaujasVykdytojoNr
BEFORE INSERT ON Vykdytojai
REFERENCING NEW AS NaujasVykdytojas
FOR EACH ROW
BEGIN ATOMIC
 SET NaujasVykdytojas.Nr =
          (SELECT MAX(Nr) + 1 FROM Vykdytojai) ;
 SET NaujasVykdytojas.Nr =
          COALESCE(NaujasVykdytojas.Nr, 1) ;
END
```


Trigerio kamieną:

```
BEGIN ATOMIC
 SET NaujasVykdytojas.Nr =
          (SELECT MAX(Nr) + 1 FROM Vykdytojai) ;
 SET NaujasVykdytojas.Nr =
          COALESCE(NaujasVykdytojas.Nr, 1);
END
```

Galima pakeisti vienu sakiniu:

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


PostgreSQL:

```
CREATE TRIGGER <trigerio vardas>
<BEFORE | AFTER>  <INSERT | UPDATE | DELETE>
ON <lentelės vardas>
FOR EACH  < ROW | STATEMENT >
[ WHEN  (<sąlyga>) ]
EXECUTE PROCEDURE
                <funkcijos vardas> (<argumentai )
```

- Galima nurodyti kelis įvykius, tarp kurių rašoma **OR**.
- **OLD** | **NEW** galima naudoti procedūros (funkcijos) apibrėžime ir sąlygoje **WHEN**.
- Sąlygoje **WHEN** negalimos užklausos.

Pvz., trigeris, užtikrinantis, kad joks darbuotojas negali dalyvauti daugiau nei 3 projektuose:

```
CREATE TRIGGER MaxVykdymųSkaičius
BEFORE INSERT OR UPDATE ON Vykdymas
FOR EACH ROW
EXECUTE PROCEDURE MaxVykdymųSkaičius() ;
```


Bendrasis funkcijos apibrėžimas:

```
CREATE FUNCTION <pavadinimas>(<argumentai>)
RETURNS <rezultato tipas>
AS '<apibrėžimas>'
LANGUAGE <kalba> ;
```

Vietoj kabučių galima naudoti $$:

```
CREATE FUNCTION <pavadinimas>(<argumentai>)
RETURNS <rezultato tipas>
AS $$<apibrėžimas>$$
LANGUAGE <kalba> ;
```


Apibrėžimas funkcijos trigeriui:

```
CREATE FUNCTION <pavadinimas>()
RETURNS TRIGGER
AS '<apibrėžimas>'
LANGUAGE plpgsql ;

CREATE FUNCTION <pavadinimas>()
RETURNS TRIGGER
AS $$<apibrėžimas>$$'
LANGUAGE plpgsql ;
```

Apibrėžiant trigerį, funkcija turi būti jau apibrėžta.


```
CREATE FUNCTION MaxVykdymųSkaičius()
RETURNS TRIGGER AS
'BEGIN
IF (SELECT COUNT(*) FROM Vykdymas
    WHERE Vykdymas.Vykdytojas=NEW.Vykdytojas) >= 3
THEN
   RAISE EXCEPTION ''Viršytas projektų skaičius''
END IF;
RETURN NEW;
END; '
LANGUAGE plpgsql ;
```


Tas pat tik naudojant $ vietoje ':

```
CREATE FUNCTION MaxVykdymųSkaičius()
RETURNS TRIGGER AS
$$BEGIN
IF (SELECT COUNT(*) FROM Vykdymas
    WHERE Vykdymas.Vykdytojas=NEW.Vykdytojas) >= 3
THEN
   RAISE EXCEPTION 'Viršytas projektų skaičius'
END IF;
RETURN NEW;
END; $$
LANGUAGE plpgsql ;
```


PostgreSQL trigerio su sąlyga `WHEN` pavyzdys:

```
CREATE TRIGGER MaxVykdymųSkaičius
BEFORE UPDATE ON Vykdymas
FOR EACH ROW
WHEN (OLD.Vykdytojas <> NEW.Vykdytojas)
EXECUTE PROCEDURE MaxVykdymųSkaičius() ;
```

Motyvas: `UPDATE` atveju, reikalavimas „darbuotojas negali dalyvauti daugiau nei 3 projektuose“ gali būti pažeidžiamas tik keičiant stulpelio *Vykdytojas* reikšmę, t. y. tik tuomet tikslinga kviesti procedūrą.

Trigeris nustoja egzistuoti, jį sunaikinus sakiniu

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

Pvz.,

```
DROP TRIGGER NaujasVykdytojoNr
```

PostgreSQL:

```
DROP TRIGGER <trigerio vardas>
             ON <lentelės vardas>;
DROP FUNCTION <funkcijos vardas>;
```


7.5.3. Reikšmių kitimo protokolavimas

Lentelės *Vykdytojai* stulpelio *Kategorija* pasikeitimų „stebėtojas“.

```
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 )
```

Visus vykdytojų kategorijų pakeitimų protokolavimas:

a) kategorijų atnaujinimo įsiminimas:

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


b) pradinių kategorijos reikšmių įsiminimas:

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


7.5.4. Reikalavimų reikšmėms užtikrinimas

`CHECK` reikalavimai turi būti **užtikrinami visada**.

Galimi reikalavimai, kurie turi būti užtikrinami tik duomenų įterpimo ir atnaujinimo momentais.

Pvz., datoms prasmingi reikalavimai šiandieninės datos atžvilgiu: asmens gimimo data negali būti ateityje.

Reikalavimą, kad įvedamas projektas nebūtų prasidėjęs anksčiau nei prieš mėnesį, SQL standarto **negalima užtikrinti reikalavimu** `CHECK`:

```
Pradžia DATE CHECK(Pradžia >
        CURRENT_DATE – INTERVAL '1 MONTH')
```


Apibrėžiant numatytąsias reikšmes, vardines konstantas galime naudoti,

```
Pradžia DATE DEFAULT (CURRENT_DATE)
Pradžia DATE DEFAULT (CURRENT_DATE + 1)
Atnaujintojas CHAR(16) NOT NULL
              DEFAULT(CURRENT_USER)
```

`DEFAULT` reikšmė taikoma tik įvedant duomenis.
`CHECK` sąlyga turi būti tenkinama visą laiką.

Reikalavimus, priklausančius nuo išorinių sąlygų, galima **užtikrinti trigeriais**.

Reikalavimas, kad įvedamas projektas nebūtų prasidėjęs anksčiau nei prieš mėnesį:

```
CREATE TRIGGER ProjektųPradžiaĮvedant
BEFORE INSERT ON Projektai
REFERENCING NEW AS NaujasProjektas
FOR EACH ROW
WHEN (NaujasProjektas.Pradžia <
       CURRENT_DATE – INTERVAL '1 MONTH')
SIGNAL SQLSTATE '99988' ('Per sena pradžios data')
```


Nekeisti pradžios datos jau prasidėjusiems ir vykdomiems projektams:

```
CREATE TRIGGER ProjektųPradžiaKeičiant
BEFORE UPDATE OF (Pradžia) ON Projektai
REFERENCING NEW AS Naujas
FOR EACH ROW
WHEN (Naujas.Pradžia < CURRENT_DATE
    AND EXISTS (SELECT * FROM Vykdymas
                WHERE Projektas=Naujas.Nr))
SIGNAL SQLSTATE '99987' ('Projektas vykdomas')
```


7.5.5. Virtualiųjų lentelių atnaujinimas

VIEW duomenų atnaujinimo galimybės yra labai ribotos.

VIEW atnaujinimo galimybes galima išplėsti

`INSTEAD OF` trigeriais

Šiuos trigerius galima apibrėžti **tik virtualiosioms** lentelėms.

Jie standartizuoti SQL:2008.

Tarkime, $L1(A, B)$ ir $L2(A, C)$ – lentelės ir $L(A, B, C)$ – virtualioji, jungianti L_1 ir L_2.

```
CREATE VIEW L(A, B, C)
AS  SELECT L1.A, L1.B, L2.C
    FROM L1, L2
    WHERE L1.A = L2.A
```

`UPDATE` L – negalima, nes L – jungtinė.

Galima sumodeliuoti sakinį `UPDATE` lentelei L.

```
CREATE TRIGGER AtnaujintiL
INSTEAD OF UPDATE ON L
REFERENCING OLD AS SenaL NEW AS NaujaL
FOR EACH ROW
BEGIN ATOMIC
 VALUES(CASE WHEN NaujaL.A <> SenaL.A
         THEN RAISE_ERROR('99996', 'A nekeisti')
         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
```


- Funkcijos **RAISE_ERROR** rezultatas kaip sakinio **SIGNAL SQLSTATE**.
- **INSTEAD OF** trigeriuose negalima naudoti **WHEN**, nes duomenų **neatnaujinimas** dėl netenkinamos sąlygos - **neprasmingas**.
- Panašiai modeliuojamos **INSERT** ir **DELETE** operacijos.
- **INSTEAD OF** negalima apibrėžti rodiniams su **WITH CHECK OPTION**

Trigerių **privalumai**:

- **pagreitina programavimą** - jie įsimenami duomenų bazėje, ir nereikia kartoti kodo, atitinkančio trigerio kamieną, daugelyje programos vietų;
- **palengvina dalykinių taisyklių užtikrinimą** – apibrėžtas trigeris automatiškai, visuomet ir laiku iškviečiamas;
- **yra globalūs** - pasikeitus dalykinėms taisyklėms, tereikia pakeisti trigerį viename egzemplioriuje.

Trigerių **trūkumai**:

- **DB sudėtingumas**.
- **Paslėpta logika**.
- **Paslėpta įtaka našumui**.

7.6. Transakcijos

Transakcija - loginis darbo su duomenimis vienetas.

Tai - **SQL sakiniai**, kurie dalykiniu požiūriu yra nedalomi.

Kiekvienas atskirai paimtas sakinys sprendžia dalį uždavinio, bet visą uždavinį sprendžia tik visų sakinių įvykdymas.

Tarkime, darbuotojas Nr. 3 (Gražulytė) išeina iš darbo, ir jo vykdomi darbai yra paskiriami darbuotojui Nr. 2:

1)
```
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 ;
```

2)
```
DELETE FROM Vykdytojai WHERE Nr = 3 ;
```


Reikalinga galimybė, kuri nesėkmingai pasibaigus antrojo sakinio vykdymui, leistų atšaukti ir anksčiau atliktus pakeitimus - **transakcijų mechanizmas**.

Transakcija - SQL sakinių seka, kuri užtikrina vienos neprieštaringos DB būsenos pervedimą į kitą neprieštaringą būseną, bet nėra garantuojamas duomenų neprieštaringumas tarp sakinių.

Transakcija arba visa įvykdoma arba visa anuliuojama.

Transakcijos užbaigimo SQL sakiniai:

- **COMMIT** – įteisinti visus, padarytus pakeitimus,
- **ROLLBACK** – atšaukti visus, padarytus pakeitimus.

Pabaigus transakciją, kiti SQL sakiniai yra naujos transakcijos dalis.

Anksčiau pateiktos transakcijos sakinius reiktų vykdyti taip:

- vykdome 1-ąjį sakinį;
- jei sakinys įvykdytas sėkmingai, tai vykdome 2-ąjį sakinį;
- jei iki šiol nebuvo klaidų, tai vykdome sakinį **COMMIT**, priešingu atveju – **ROLLBACK**.

Procedūra SQL sakinių sekai S_1, S_2,…,S_n - transakcijai įvykdyti.

```
for i := 1 to n
   execute Si ;
   if SQL klaida
   then goto error ;
endfor
COMMIT ;
return success;
error:
   ROLLBACK;
return failure;
```


Principas „viskas arba nieko“ reikalauja **transakcijų žurnalo**.

Kiekvienai pakeistai eilutei DBVS žurnale įsimenama pradinė jos būsena.

Įvykdžius **COMMIT**, žurnale pažymima transakcijos pabaiga.

Vykdant **ROLLBACK**, pradinė būsena atstatoma iš žurnalo.

DB administratorius gali atstatyti neprieštaringą DB būseną netgi po avarinio sistemos darbo nutraukimo.

Svarbu prisiminti, kad

transakcijų žurnalui reikia vietos kompiuterio atmintyje.

DB serveryje svarbu rezervuoti pakankamą kiekį atminties transakcijų žurnalui.

Pritrūkus vietos, SQL sakinys pasibaigia SQL klaida.

Pastabos:

- vienu metu vienas vartotojas ar programa gali vykdyti tik vieną transakciją, transakcijos negali būti „įdėtos“ viena į kitą;
- kai darbas baigiamas atsijungiant nuo DB, sistema automatiškai įvykdo sakinį **COMMIT**;
- vienas SQL sakinys negali būti įvykdytas iš dalies, t.y. jis visuomet įvykdomas visiškai arba visiškai neįvykdomas.