2.7a. Bendrumo ir egzistavimo kvantoriai užklausose

Pateiksime ankstesnio uždavinio „vykdytojų, dalyvaujančių projekte Nr. 1, pavardės“ sprendimo dar vieną formuluotę, pavartojant predikatą – egzistavimo kvantorių:

```
SELECT Pavardė FROM Vykdytojai
WHERE EXISTS ( SELECT * FROM Vykdymas
               WHERE Vykdytojas = Nr AND Projektas = 1 ).
```

Šiame sakinyje išorinės užklausos paieškos sąlygoje yra pavartotas predikatas, kurio sintaksė yra EXISTS (SELECT * FROM…). Šio predikato reikšmė yra „tiesa“ tuomet ir tik tuomet, kai vidinės užklausos rezultatas yra netuščioji aibė.

Pastaroji užklausa yra priklausomoji. Vidinės, predikate pavartotos užklausos rezultatas priklauso nuo parametro *Nr*, kuris įgyja reikšmę išorinėje užklausoje. Šioje užklausoje projekto Nr. 1 vykdytoju yra laikomas kiekvienas vykdytojas, kuriam egzistuoja bent vienas vykdymas, kuriame jis dalyvauja vykdant tą projektą.

Paieškos sąlyga

```
EXISTS (SELECT * FROM…)
```

yra ekvivalentiška sąlygai

```
(SELECT COUNT(*) FROM…) > 0.
```

Šiame reiškinyje palyginimo operacijos vienas iš operandų yra užklausa. Kadangi užklausos, kurioje nėra duomenų grupavimo ir SELECT frazėje yra tik jungtinė funkcija, rezultatas visuomet yra vienintelė reikšmė, tai tokią užklausą galima naudoti palyginimo operacijoje.

Predikatų logikos kvantorių atitikmenys naudojami ir tokiuose predikatuose

```
<reiškinys> <palyginimo operacija> <ALL | ANY | SOME> ( <užklausa> )
```

Apskaičiuojant predikato ALL reikšmę lyginamojo reiškinio reikšmė yra lyginama su visomis užklausos rezultatą sudarančiomis reikšmėmis. Predikato ALL reikšmė yra „tiesa“ tuomet ir tik tuomet, kai užklausos rezultatas yra tuščioji aibė arba palyginimo reikšmė yra „tiesa“ visoms užklausos rezultato reikšmėms.

Predikato ANY reikšmė yra „tiesa“ tuomet ir tik tuomet, kai palyginimo reikšmė yra „tiesa“ bent vienai užklausos rezultato reikšmei. Predikatas SOME yra predikato ANY sinonimas.

Darbuotojų kvalifikacijas, kuriose visi darbuotojai yra ne mažesnės negu 2-os kategorijos, galima sužinoti užklausa

```
SELECT DISTINCT A.Kvalifikacija FROM Vykdytojai A
WHERE 2 <= ALL (SELECT B.Kategorija FROM Vykdytojai B
                WHERE A.Kvalifikacija = B.Kvalifikacija).
```

Predikatas ALL labai palengvina tokio uždavinio sprendimą. Kad užklausos rezultatą sudarytų tik skirtingos kvalifikacijos, panaudojome frazę DISTINCT. Pagrindinės (išorinės) užklausos sąlyga yra tikrina kiekvienai lentelės *Vykdytojai* eilutei. Kai šioje lentelėje yra labai daug eilučių užklausos vykdymui gali prireikti nemažai laiko, nes vidinė užklausa yra priklausoma nuo parametro *A.Kvalifikacija*. Tarus, kad skirtingų kvalifikacijų yra žymiai mažiau negu darbuotojų, galima sudaryti efektyvesnį uždavinio sprendinį. Užklausoje galima pasinaudoti laikinąja lentele, kad vidinė užklausa būtų vykdoma tik skirtingoms kvalifikacijoms,

```
SELECT A.Kvalifikacija
FROM (SELECT DISTINCT Kvalifikacija FROM Vykdytojai) AS A
WHERE 2 <= ALL (SELECT B.Kategorija FROM Vykdytojai B
                WHERE A.Kvalifikacija = B.Kvalifikacija).
```

Jau žinomas vykdytojų, dalyvaujančius projekte Nr. 1, pavardes galima sužinoti dar ir taip:

```
SELECT Pavardė FROM Vykdytojai
WHERE Nr = ANY (SELECT Vykdytojas FROM Vykdymas WHERE Projektas = 1).
```

Nesunku pastebėti, kad ši užklausa struktūriškai yra labai panaši į užklausą, pateiktą ankstesniajame skyrelyje, panaudojant predikatą `IN`. Predikatą `IN` galima išreikšti per predikatą `ANY`, o predikatą `NOT IN` per predikatą `ALL`. Užrašas

<reiškinys> `IN` (<užklausa>)

yra ekvivalentiškas užrašui

<reiškinys> `= ANY` (<užklausa>),

o užrašas

<reiškinys> `NOT IN` (<užklausa>)

yra ekvivalentiškas užrašui

<reiškinys> `<> ALL` (<užklausa>).

Darbuotojų kvalifikacijas, kuriose visi darbuotojai yra ne mažesnės negu 2-os kategorijos, galima išreikšti be predikato `ALL`

```
SELECT DISTINCT A.Kvalifikacija FROM Vykdytojai A
WHERE (SELECT COUNT(*) FROM Vykdytojai B
        WHERE A.Kvalifikacija = B.Kvalifikacija AND A.Kategorija < 2) = 0.
```

Šioje užklausoje kvalifikacijos, kuriose visi darbuotojai yra ne mažesnės negu 2-os kategorijos, yra apibrėžtos kaip kvalifikacijos, kuriose nėra mažesnės negu antros kategorijos darbuotojų.

2.9a. Konstantinės lentelės

Standarte SQL2 yra numatyta galimybė apibrėžti lentelę betarpiškai SQL sakinyje. Tai - **konstantinės lentelės**. Tokioje lentelėje visos eilutės išvardijamos betarpiškai atskiriant jas tarpusavyje kableliais. Kiekvienos eilutės stulpelių reikšmės yra rašomos tarp riestinių skliaustų, atskiriant reikšmes kableliais. Konkretaus stulpelio reikšmė eilutėje pateikiama reiškiniu, paprasčiausiu atveju, konstanta. Konstantinės lentelės apibrėžiamos sakiniu `VALUES`.

Kadangi sakiniu `VALUES` iš reiškinių galima sudaryti lentelę, tai jis vadinamas lentelės konstruktoriumi. Šis sakinys taip pat yra ypatinga užklausa. Sakinio rezultatas yra lentelė, kurios turinys yra nurodytas sakinyje. Pavyzdžiui, sistemos datą galima sužinoti tokia užklausa

```
VALUES (CURRENT DATE).
```

Lentelė (užklausa), sudaryta iš vienos eilutės, kurioje yra 3 datos: vakar, šiandien ir rytoj, sudaroma sakiniu:

```
VALUES (CURRENT DATE – 1 DAY, CURRENT DATE, CURRENT DATE + 1 DAY).
```

Lentelę su tais pačiais duomenimis, tačiau išdėstytais vieno stulpelio trijose eilutėse, galima išreikšti labai panašiai

```
VALUES (CURRENT DATE – 1 DAY), (CURRENT DATE), (CURRENT DATE + 1 DAY).
```

Jei sakinyje `VALUES` yra išvardyta n eilučių: e_1, e_2, ..., e_n, $n > 1$, t.y.

```
VALUES (e1), (e2),..., (en),
```

tai šis sakinys yra ekvivalentiškas sakiniui

```
VALUES(e1) UNION ALL VALUES(e2) ... UNION ALL VALUES(en).
```

Todėl visos sakinyje `VALUES` išvardytos eilutės turi būti tarpusavyje suderintos – sudarytos iš vienodo skaičiaus reikšmių (stulpelių) ir atitinkamų reikšmių tipai neprieštaringi.

Sakiniu `VALUES` sudarytą lentelę, kaip kiekvienos užklausos rezultatą, galima rūšiuoti ir grupuoti, pvz.,

```
VALUES (CURRENT DATE – 1 DAY), (CURRENT DATE), (CURRENT DATE + 1 DAY)
ORDER BY 1 DESC.
```

Sakinyje `VALUES` negalima suteikti stulpeliams pavadinimų. Tačiau, tai galima padaryti apibrėžiant papildomą laikinąją lentelę, pvz.,

```
SELECT Vakar, Šiandien, Rytoj FROM
(VALUES (CURRENT DATE – 1 DAY), (CURRENT DATE), (CURRENT DATE + 1 DAY))
        AS Dienos(Vakar, Šiandien, Rytoj).
```

Sakinys `VALUES` dažniausiai naudojamas vardinių konstantų (sistemos laiko, datos, vartotojo vardo ir kt.) reikšmėms sužinoti.

3.2a. Reliacinė algebra

Jau minėjome, kad lentelės reliacinė schema pagrindinai nusakoma lentelės vardu ir jos stulpelių vardais. Norėdami pabrėžti pirminio rakto svarbą, jį taip pat priskyrėme reliacinei schemai. Kai pirminio rakto vaidmuo nėra esminis, nurodydami lentelės schemą, jį praleisime. Prisilaikydami reliacinėje teorijoje naudojamais terminais, lentelės stulpelius taip pat vadinsime **atributais**. Tarkime, $A_1, A_2,..., A_n$ yra lentelės L atributų (stulpelių) aibė, čia A_i – atributas, $i = 1,...,n$. Tuomet $L(A_1, A_2,..., A_n)$ žymi lentelės schemą. Lentelės atributų aibę pažymėję $R = \{A_1, A_2,..., A_n\}$, jos schemą galime užrašyti taip $L(R)$.

Lentele l, kurios schema $L(A_1, A_2,..., A_n)$, yra vadinama sutvarkytų reikšmių rinkinių (eilučių) aibė $l = \{e_1, e_2,..., e_m\}$. Konkreti lentelė l su schema $L(A_1, A_2,..., A_n)$ dar žymima taip $l(A_1, A_2,..., A_n)$. Kiekviena eilutė e_j yra sutvarkytas reikšmių rinkinys $e_j = <a_1, a_2,..., a_n> \in l$, čia $a_i \in dom(A_i)$, $dom(A_i)$ žymi atributo A_i galimų reikšmių aibę, t.y. jo domeną, $i = 1,...,n$; $j = 1,...,m$. $e(A_i)$ žymi atributo A_i reikšmę eilutėje e. Atributų aibės R poaibio A, $A \subseteq R$, reikšmių rinkinys eilutėje e žymimas panašiai: $e(A)$.

Pasinaudodami pateiktaisiais apibrėžimais, galime sakyti, kad lentelės $l(R) = \{e_1, e_2,..., e_m\}$ atributų aibės R poaibis $V \subseteq R$ yra viršraktis, jei $\forall\ i, j = 1,...,m$ ir $i \neq j$ yra teisinga nelygybė $e_i(V) \neq e_j(V)$.

Lentelę $l(A_1, A_2,..., A_n)$ galima apibrėžti ir kaip Dekarto sandaugos poaibį

$$l(A_1, A_2,..., A_n) \subseteq (dom(A_1) \times dom(A_2) \times ... \times dom(A_n)).$$

Suprantama, lentelės schema (sandara) $L(R)$ ir konkretus jos turinys $l(R)$ nėra tas pat. Lentelė $l(R)$ – tai lentelės, kurios schema yra $L(R)$, konkretus turinys (užpildymas konkrečiais duomenimis) konkrečiu momentu. Dėl šių sąvokų dualumo mes dažnai naudosime tą patį žymenį tiek lentelei, tiek jos schemai žymėti, laikydami kad kontekstas apsprendžia, kas konkrečiai turima omenyje.

Sudarinėdami užklausas, mes jau manipuliavome lentelėmis. Dabar formaliai apibrėšime pagrindines lentelių operacijas, kurios sudaro reliacinę algebrą. Neformaliai, **reliacine algebra** vadinama formali operacijų kalba (sistema), kuria iš vienos ar kelių lentelių (santykių), nekeičiant jų turinio, galima gauti kitas lenteles. Kadangi operacijos rezultatas yra lentelė, tai su juo taip pat galima atlikti reliacinės algebros operacijas. Taip konstruojami **reliacinės algebros reiškiniai**.

Reliacinės algebros operacijos dažniausiai yra skirstomos į dvi grupes. Pirmąją grupę sudaro matematinės aibių teorijos operacijos: sąjunga ($\cup$), sankirta ($\cap$), skirtumas ($-$) ir Dekarto sandauga ($\times$). Šias operacijas galima taikyti lentelėms, nes jos yra eilučių aibės. Antrąją reliacinės algebros operacijų grupę sudaro specifinės duomenų bazėms operacijos: išrinkimas (σ), projekcija (π), jungimas ($\bowtie$) ir kitos. Aibių operacijos mums yra žinomos iš aibių teorijos pagrindų, be to, neformaliai mes jas jau aptarėme ankstesniame šios knygos skyriuje. Detaliau aptarsime išrinkimą, projekciją ir jungimą, kurios yra vienos iš pagrindinių reliacinėje algebroje.

3.2.1. Pagrindinės operacijos su lentelėmis

Išrinkimas yra unarinė operacija su lentele. Pritaikius šią operaciją lentelei, gaunama kita lentelė, kurią sudaro pradinės lentelės eilučių poaibis su konkrečia konkretaus atributo reikšme. Tarkime, $l = l(R)$ yra lentelė, A – atributas iš aibės R ir $a \in dom(A)$. Tuomet $\sigma_{A=a}(l)$ žymi išrinkimo operaciją „iš l išrinkti visas eilutes, kuriose atributo A reikšmė yra lygi a“. Operacijos rezultatas yra lentelė $l'(R)$

$$\sigma_{A=a}(l) = l'(R) = \{e \in l : e(A) = a\}.$$

Išrinkimo operacija yra komutatyvi jų kompozicijos atžvilgiu

$$\sigma_{A=a}(\sigma_{B=b}(l)) = \sigma_{B=b}(\sigma_{A=a}(l)).$$

Ši savybė išplaukia iš išrinkimo operacijos apibrėžimo

$\sigma_{A=a}(\sigma_{B=b}(l)) = \sigma_{A=a}(\{e\in l : e(B) = b\}) = \{e' \in \{e\in l : e(B) = b\} : e'(A) = a\} =$
$= \{e\in l : e(A) = a \text{ ir } e(B) = b\} = \{e' \in \{e\in l : e(A) = a\} : e'(B) = b\} = \sigma_{B=b}(\sigma_{A=a}(l)).$

Kadangi išrinkimo tvarka nesvarbi, tai kompoziciją $\sigma_{A=a} \circ \sigma_{B=b}$ galima išreikšti taip $\sigma_{A=a,B=b}$. Išrinkimo operacija yra distributyvi aibių binarinių operacijų ($\cup$, $\cap$, $-$) atžvilgiu. Šiai savybei įrodyti taip pat pakanka išrinkimo operacijos apibrėžimo.

Projekcijos operacija taip pat yra unarinė. Priešingai išrinkimo operacijai, projekcijos operacija yra išrenkamas ne eilučių, bet stulpelių poaibis. Lentelės $l = l(R)$ projekcija aibėje A, $\pi_A(l)$, $A \subseteq R$, yra lentelė $l'(A)$, kuri gaunama iš $l(R)$ išbraukiant stulpelius $R-A$,

$\pi_A(l) = l'(A) = \{e(A) : e\in l\}$.

Jei dvi projekcijos atliekamos paeiliui ir antrojoje išrenkamųjų stulpelių aibė yra pirmosios stulpelių aibės poaibis, tai pirmoji operacija yra nereikšminga, ją galima praleisti. Tiksliau, jei $A \subseteq B \subseteq R$, tai lentelei $l(R)$ galioja lygybė

$\pi_A(\pi_B(l)) = \pi_A(l)$.

Projekcija komutuoja su išrinkimu, kai išrinkimo atributas yra vienas iš projekcijos atributų. Jei $A \in B$, $B \subseteq R$ ir $l(R)$ yra lentelė, tai

$\pi_B(\sigma_{A=a}(l)) = \pi_B(\{e\in l : e(A) = a\}) = \{e'(B) : e' \in \{e\in l : e(A) = a\}\} =$
$= \{e(B) : e\in l \text{ ir } e(A) = a\} = \sigma_{A=a}(\{e(B) : e\in l\}) = \sigma_{A=a}(\pi_B(l))$.

Kita labai svarbi operacija su lentelėmis yra jų **jungimas**. Tai binarinė operacija, kuria kombinuojamos dvi lentelės. Dviejų lentelių $l_1(R_1)$ ir $l_2(R_2)$ **junginiu** $l_1 \bowtie l_2$ yra vadinama lentelė $l_3(R_3)$, $R_3 = R_1 \cup R_2$, tokia, kad

$l_1 \bowtie l_2 = l_3(R_3) =$
$= \{e : \exists\, e_1,e_2\, (e_1\in l_1,\ e_2\in l_2,\ e(R_1) = e_1,\ e(R_2) = e_2,\ e_1(R_1\cap R_2) = e_2(R_1\cap R_2))\}$.

Jungimo operacijos apibrėžime nereikalaujama, kad $R_1\cap R_2$ būtų netuščia aibė. Jeigu $R_1\cap R_2 = \varnothing$, tai jungimas yra ekvivalentiškas Dekarto sandaugai, $l_1 \bowtie l_2 = l_1 \times l_2$. Taigi, Dekarto sandauga yra ypatingasis jungimo operacijos atvejis.

Yra ir daugiau reliacinės algebros operacijų. Tačiau čia pateiktosios yra laikomos vienomis pagrindinių. Pavyzdžiui, yra įrodyta, kad operacijų aibė $\{\sigma, \pi, \cup, -, \times\}$ yra **pilnoji**. Tai reiškia, kad bet kurią kitą reliacinę operaciją galima išreikšti šios aibės operacijų seka. Pavyzdžiui, sankirtos operaciją ($\cap$) galima išreikšti per sąjungą ($\cup$) ir skirtumą ($-$)

$l_1 \cap l_2 = (l_1\cup l_2) - ((l_1 - l_2) \cup (l_2 - l_1))$.

Lentelių $l_1(R_1)$ ir $l_2(R_2)$ junginį $l_1 \bowtie l_2$ galima išreikšti per σ, π ir $\times$

$$l_1 \bowtie l_2 = \pi_{R_1\cup R_2}(\sigma_{l_1.A_1=l_2.A_1,\ldots,l_1.A_N=l_2.A_N}(l_1\times l_2)),$$

čia $\{A_1, A_2,..., A_N\} = R_1 \cap R_2$. Jungimo operacijos savybes aptarsime detaliau.

3.2.2. Jungimo savybės

Jungimo operacija galima išreikšti išrinkimo operaciją. Tarkime, lentelėje $l(R)$ reikia atlikti išrinkimo operaciją $\sigma_{A=a}(l)$. Apibrėžiame naują lentelę $l'(A)$ su vienintele eilute e, kurioje $e(A) = a$. Tuomet yra teisinga lygybė $\sigma_{A=a}(l) = l \bowtie l'$, nes

$l \bowtie l' = \{e : \exists\, e_1,e_2\, (e_1\in l,\ e_2\in l',\ e(R) = e_1,\ e(A) = e_2)\} =$
$= \{e : \exists\, e_1\, (e_1\in l,\ e(R) = e_1,\ e(A) = a)\} = \sigma_{A=a}(l)$.

Kad lentelėje $l(R)$ sumodeliuoti bendresnę išrinkimo operaciją $\sigma_{A=a_1,A=a_2}(l)$, papildomoje lentelėje l' reikia įrašyti dvi eilutes e_1,e_2, kurioms $e_1(A) = a_1$, $e_2(A) = a_2$.

Jei papildomoje dviejų stulpelių lentelėje $l'(AB)$ yra viena eilutė e, kuriai $e(A) = a$, $e(B) = b$, tai $l \bowtie l' = \sigma_{A=a,B=b}(l)$. Taip galima modeliuoti išrinkimą pagal konkrečias dviejų stulpelių reikšmes.

Dėl simetrijos jungimo operacijos apibrėžime, gauname, kad ši operacija yra asociatyvi

$(l_1 \bowtie l_2) \bowtie l_3 = l_1 \bowtie (l_2 \bowtie l_3)$.

Todėl tokiuose reiškiniuose skliaustelius galima praleisti.

Sujunkime, pavyzdžiui, tris lenteles

l_1

A	B
a_1	b_1
a_1	b_2
a_2	b_1

l_2

B	C
b_1	c_2
b_2	c_1

l_3

A	C
a_1	c_2
a_2	c_2

Jų junginys yra lentelė

$l_1 \bowtie l_2 \bowtie l_3$

A	B	C
a_1	b_1	c_2
a_2	b_1	c_2

Jungiant šias lenteles, lentelės l_1 eilutė $<a_1, b_2>$ ir lentelės l_2 eilutė $<b_2, c_1>$ liko nesujungtos. Lentelės $l_1, l_2, ..., l_m$ vadinamos **visiškai sujungiamos**, jei kiekviena kiekvienos lentelės eilutė yra lentelių junginio konkrečios eilutės dalis. Jei lentelę l_3 papildytume eilute $<a_1, c_1>$, tai pateiktosios lentelės l_1, l_2, l_3 taptų visiškai sujungiamomis. Tuomet jų junginys atrodytų taip

$l_1 \bowtie l_2 \bowtie l_3$

A	B	C
a_1	b_1	c_2
a_1	b_2	c_1
a_2	b_1	c_2

Jungimo ir projekcijos operacijos nėra viena kitai visiškai atvirkščios. Tarkime, $l_1(R_1)$ ir $l_2(R_2)$ – lentelės. Nesunku įsitikinti, kad

$\pi_{R_1}(l_1 \bowtie l_2) \subseteq l_1$.

Čia poaibio ženklas tampa lygybe, kai kiekvienai lentelės l_1 eilutei e_1 lentelėje l_2 yra eilutė e_2 tokia, kad $e_1(R_1 \cap R_2) = e_2(R_1 \cap R_2)$. Poaibis gali tapti lygybe ir tuo atveju, kai l_1 ir l_2 nėra visiškai sujungiamos. Imkime dvi pateiktąsias lenteles l_2 ir l_3, kiekvienoje iš kurių yra po dvi eilutes. l_2 ir l_3 nėra visiškai sujungiamos, nes lentelės l_2 eilutės $<b_2, c_1>$ negalima sujungti nei su viena lentelės l_3 eilute. Šių dviejų lentelių junginį $l_2 \bowtie l_3$, sudarys tos pačios dvi eilutės, kaip ir jau pateiktąjį junginį $l_1 \bowtie l_2 \bowtie l_3$, gautą prieš papildant l_3 trečiąja eilute. Nesunku įsitikinti, kad $l_3 = \pi_{AB}(l_2 \bowtie l_3)$.

Tarkime $l_3(R_1 \cup R_2)$ yra lentelė, o $l_1(R_1)$ ir $l_2(R_2)$ – jos projekcijos, $l_1 = \pi_{R_1}(l_3)$, $l_2 = \pi_{R_2}(l_3)$. Jeigu e yra l_3 eilutė, tai $e(R_1) \in l_1$ ir $e(R_2) \in l_2$. Vadinasi, $l_3 \subseteq l_1 \bowtie l_2$. Jeigu galioja lygybė $l_3 = l_1 \bowtie l_2$, tai toks lentelės l_3 skaidymas į dvi lenteles l_1 ir l_2 yra vadinamas lentelės (l_3) **dekompozicija be praradimo**. Taip skaidant lentelę l_3 į l_1 ir l_2 jokie duomenys nėra prarandami, nes pradinę lentelę l_3 galima gauti jungiant l_1 ir l_2.

Tarkime, turime lentelę

l_4

A	B	C
a_1	b	c_1
a_2	b	c_2

Sudarome dvi šios lentelės projekcijas

$\pi_{AB}(l_4)$

A	B
a_1	b
a_2	b

$\pi_{BC}(l_4)$

B	C
b	c_1
b	c_2

Akivaizdu, kad šių projekcijų junginį $\pi_{AB}(l_4) \bowtie \pi_{BC}(l_4)$ sudaro net keturios eilutės. Tai reiškia, kad $l_4 \subset \pi_{AB}(l_4) \bowtie \pi_{BC}(l_4)$. Todėl lentelės l_4 suskaidymas į dvi jos projekcijas: $\pi_{AB}(l_4)$ ir $\pi_{BC}(l_4)$ nėra dekompozicija be praradimo. Nesunku įsitikinti, jeigu lentelės l_4 stulpelyje B esančios reikšmės būtų skirtingos, tai atitinkamų projekcijų junginys sutaptų su pradine lentele ir gautume dekompoziciją be praradimo.

3.2.3. Palyginimo apibendrinimas operacijose

Reliacinės algebros operacijose dviems domeno (atributo reikšmių aibės) reikšmėms palyginti iki šiol naudojome tik vieną lygybės (=) operaciją. Reliacinėje teorijoje laikomasi prielaidos, kad bet kurias konkretaus domeno reikšmes galima įvertinti ar jos tarpusavyje lygios (=), ar nelygios (≠). Pakankamai dažnai domenų reikšmės būna sutvarkytos taip, kad bet kurioms dviems nelygioms domeno reikšmėms galima nustatyti, kuri iš jų yra didesnė, o kuri – mažesnė. Daugelyje praktikoje naudojamų domenų (skaičiai, simboliniai duomenys, datos, laikai ir pan.) naudojamos palyginimo operacijos: =, ≠, <, ≤, ≥, >. Šią palyginimo operacijų aibę žymėsime Θ. Bendruoju atveju gali būti lyginamos skirtingų domenų reikšmės, pvz. sveikąjį skaičių galime lyginti su realiuoju. Jei $\theta \in \Theta$ - palyginimo operacija, o A ir B – atributai, tai A ir B – vadinsime θ–palyginamais, jei θ yra apibrėžta aibėje $dom(A) \times dom(B)$.

Apibrėžtosiose išrinkimo ir jungimo operacijose naudojome lygybės operaciją. Jų apibrėžimą galima išplėsti ir kitoms palyginimo operacijoms. Tarkime, $l = l(R)$ yra lentelė, A, $B \in R$, $a \in dom(B)$, $\theta \in \Theta$, A ir B yra θ–palyginami. Tuomet $\sigma_{A\theta a}(l)$ žymi **θ – išrinkimo operaciją**

$$\sigma_{A\theta a}(l) = \{e \in l : e(A)\ \theta\ a\}.$$

Vietoje atributo lyginimo su konkrečia reikšme, galima lyginti du atributus

$$\sigma_{A\theta B}(l) = \{e \in l : e(A)\ \theta\ e(B)\}.$$

Patogumo dėlei, išrinkimo operaciją galima dar labiau **apibendrinti**, leidžiant išrinkimo sąlygoje naudoti logines operacijas bei skliaustelius, pvz. $\sigma_{A \le B \wedge (A=a \vee B \ne b)}(l)$.

Jungiant lenteles taip pat galima naudoti bet kurią palyginimo operaciją $\theta \in \Theta$. Dviejų lentelių $l_1(R_1)$ ir $l_2(R_2)$ **θ** – j**unginiu** $l_1 \bowtie_\theta l_2$ yra vadinama lentelė, sudaryta iš stulpelių $R_3 = R_1 \cup R_2$, ir kurios eilutės apibrėžiamos taip

$$l_1 \bowtie_\theta l_2 = \{e : \exists\, e_1, e_2\ (e_1 \in l_1,\ e_2 \in l_2,\ e(R_1) = e_1,\ e(R_2) = e_2,\ e_1(R_1 \cap R_2)\ \theta\ e_2(R_1 \cap R_2))\}.$$

Praktikoje, tiek θ – išrinkimas, tiek ir apibendrintasis išrinkimas yra labai plačiai naudojami, θ – jungimas sutinkamas žymiai rečiau. Jungimo operaciją galima dar labiau apibendrinti, tačiau praktikoje to beveik neprireikia.

Svarbu tai, kad visos anksčiau apibrėžtos išrinkimo ir jungimo operacijų savybės būdingos ir jų pateiktiesiems apibendrinimas.

3.2.4. Operacijos su lentelėmis SQL kalboje

Apibrėžtosios operacijos mums nėra visiškai naujos. Sudarinėjant užklausas SQL kalba mes jau susidūrėme su panašiomis operacijomis. Jau minėjome, kad iš pradžių buvo sukurta reliacinė teorija, vėliau buvo sudaryta SQL kalba ir dar vėliau pasirodė pirmosioms reliacinių duomenų bazių valdymo sistemos. Visas pagrindines reliacines operacijas DB sistemose realizuoja SQL sakinys `SELECT`.

Lentelės $l(R)$ išrinkimo operacija $\sigma_{A=a}(l)$ išreiškiama tokia užklausa

`SELECT` * `FROM` l `WHERE` $A = a$

Nesunku išreikšti ir apibendrintuosius išrinkimus, pvz. išrinkimą $\sigma_{A \le B \wedge (A=a \vee B \ne b)}(l)$ galima užrašyti tokiu SQL sakiniu

`SELECT` * `FROM` l `WHERE` $A <= B$ `AND` (A = a `OR` $B <>$ b)

Lentelės $l(R)$ projekciją $\pi_A(l)$ galima išreikšti sakiniu

`SELECT DISTINCT` A `FROM` L

Kadangi reliacinėje teorijoje lentelėje vienodų eilučių nėra, tai projekcijai išreikšti prireikė raktažodžio `DISTINCT`.

Dviejų lentelių $l_1(R_1)$ ir $l_2(R_2)$ junginį $l_1 \bowtie l_2$ galima gauti SQL užklausa, kurią struktūriškai galima užrašyti taip

`SELECT` $R_1 \cup R_2$ `FROM` l_1, l_2 `WHERE` $l_1.A_1 = l_2.A_1$ `AND` ... `AND` $l_1.A_N = l_2.A_N$

čia $(A_1, A_2, ..., A_N) = R_1 \cap R_2$. θ–junginio $l_1 \bowtie_\theta l_2$ rezultatas išreiškiamas labai panašiai – tereikia visus lygybės ženklus SQL sakinyje pakeisti palyginimo operacijos θ SQL ženklu.

Trys aibių operacijos: sąjunga ($\cup$), sankirta ($\cap$), skirtumas ($-$) turi tiesioginius atitikmenis SQL kalboje: `UNION`, `INTERSECT` ir `EXCEPT`. Dekarto sandaugą ($\times$) galima laikyti ypatinguoju jungimo operacijos atveju, kai abiejų lentelių (operacijos operandų) visų stulpelių pavadinimai yra laikomi skirtingais. Dviejų lentelių $l_1(R_1)$ ir $l_2(R_2)$ Dekarto sandaugą SQL kalba galima išreikšti taip

`SELECT` R_1, R_2 `FROM` l_1, l_2

Dviejų lentelių Dekarto sandaugą SQL kalboje užrašėme kaip ypatingąjį jungimo atvejį. Galima daryti ir atvirkščiai. Iš pradžių galima apibrėžti Dekarto sandaugą SQL kalba. Tada dviejų lentelių junginys SQL kalboje išreiškiamas per išrinkimo, projekcijos ir Dekarto sandaugą.

3.7'. Funkcinių priklausomybių ir atributų uždariniai

Algoritmas atributų aibės S uždariniui S^+ FP F atžvilgiu rasti.

$S^+ := S$;
repeat
  $T := S^+$;
  for each $X \to Y \in F$
    if $X \subseteq S^+$ then $S^+ := S^+ \cup Y$;
  endfor
until $T <> S^+$.

Kad patikrinti, ar dvi FP aibės F ir G yra ekvivalenčios, pakanka patikrinti, ar kiekviena FP $X \to Y \in F$ yra išvedama aibėje G, t.y. $X \to Y \in G^+$ ir atvirkščiai, ar kiekviena FP $U \to V \in G$ yra išvedama aibėje F, t.y. $U \to V \in F^+$.

FP uždarinyje yra galimos perteklinės FP, kurios išvedamos iš kitų uždarinio FP. Tai gali labai apsunkinti FP savybių tyrimą. Todėl įvedama minimalaus (standartinio) denginio sąvoka.

FP aibė F vadinama **minimaliąja**, jeigu ji tenkina reikalavimus:

a) kiekvienai FP $X \to Y \in F$, dešinioji pusė Y yra sudaryta tik iš vieno atributo;
b) aibėje F nėra FP, kurią pašalinę, gautume aibę, ekvivalentišką duotajai aibei F;
c) aibėje F nėra FP $X \to A$, kurią pakeitus FP $Y \to A$, $Y \subset X$, gautume FP aibę, ekvivalentišką aibei F.

FP aibės F **minimaliu denginiu** vadinama minimalioji FP aibė F_{min}, kuri yra ekvivalenti aibei F. FP aibei F gali egzistuoti kelios F_{min}. Tačiau visada galima surasti bent vieną jų.

Algoritmas FP aibės F minimaliajam denginiui F_{min} rasti.

$F_{min} := F$;
for each $X \to A_1 A_2 \ldots A_n \in F_{min}$
  $F_{min} := F_{min} - (X \to A_1 A_2 \ldots A_n)$;
  for $i := 1$ to n
    $F_{min} := F_{min} \cup X \to A_i$;
  endfor
endfor
for each $X \to A \in F_{min}$
  $T := X^+$ aibės $(F_{min} - (X \to A))$ atžvilgiu;
  if $A \in T$ then $F_{min} := F_{min} - (X \to A)$ endif;
endfor
for each $X \to A \in F_{min}$
  for each $B \in X$
    $T := (X - B)^+$ aibės $((F_{min} - (X \to A)) \cup ((X - B) \to A))$ atžvilgiu;
    if $A \in T$ then
      $F_{min} := F_{min} - (X \to A)$;
      $F_{min} := F_{min} \cup ((X - B) \to A)$;
    endif
  endfor
endfor.

Atliekant šio algoritmo pirmąjį ciklą `for` visos FP yra suskaidomos taip, kad dešiniosiose FP pusėse būtų tik po vieną atributą. Antrajame cikle `for` iš pirmajame etape sudarytos aibės F_{min} yra pašalinamos visos perteklinės FP, kurių dešiniosios pusės gali būti išvestos iš likusių FP nepanaudojant šalinamosios. Tuomet analizuojamos visos likusios FP pašalinant perteklinius determinantų atributus. Determinanto atributas yra perteklinis, jei FP dešinėje

esantį atributą galima išvesti net ir tuomet, kai perteklinis atributas pašalinamas iš determinanto.

Turėdami algoritmą atributų uždariniui rasti, galima nesudėtingai rasti lentelės raktą.

Algoritmas lentelės $L(R)$ raktui K FP aibės F atžvilgiu rasti.

$K := R$;
`for each` $A \in K$
  $T := (K - A)^+$ aibės F atžvilgiu;
  `if` $T = R$ `then` $K := K - A$ `endif`;
`endfor.`

Šio algoritmo pradžioje yra tariama, kad raktą sudaro visi lentelės atributai. Tuomet iš rakto pašalinami visi atributai, be kurių vis tiek galima išvesti visus lentelės atributus. Šiuo algoritmu surandamas tik vienas raktas. Jei lentelė turi kelis raktus, visiems jiems efektyviai surasti reikalingas šiek tiek sudėtingesnis algoritmas.

3.8'. Antroji norminė forma

Gali atrodyti, kad naujosiose lentelėse galiojančios FP skiriasi nuo galiojusiųjų pradinėje lentelėje. Tačiau taip nėra. Suskaidę lentelę nepraradome jokių savybių, nes funkcinė priklausomybė $AB \to CD$ yra išvedama iš $AB \to C$ ir $A \to D$, panaudojant kompozicijos taisyklę. Lentelių skaidymas, kai išsaugomos visos atributų tarpusavio priklausomybės yra vadinamas **dekompozicija išsaugant priklausomybes**.

Suskaidžius pradinę lentelę L į dvi lenteles L_1 ir L_2, lentelės L_2 raktas A gali būti pirminiu, o lentelėje L_1 galima apibrėžti išorinį raktą, susiejant jos atributą A su L_2 pirminiu raktu.

Abi naujosios lentelės L_1 ir L_2 yra pradinės lentelės L projekcijos. Pradinės lentelės L duomenis galima atkurti jungiant jos projekcijas L_1 ir L_2. Aptariant reliacinės algebros projekcijos ir jungimo operacijas, minėjome, kad lentelių skaidymas, kai pradinės lentelės duomenys sutampa su jos projekcijų junginiu, vadinamas dekompozicija be praradimo. Tai, kad pasiūlytas lentelės skaidymas, siekiant užtikrinti 2NF, yra dekompozicija be praradimo, garantuoja Hezo (*I.J. Heath*) teorema.

Hezo (*I.J. Heath*) teorema. Tarkime, lentelėje $L(A, B, C)$ galioja funkcinė priklausomybė $A \to B$ arba funkcinė priklausomybė $A \to C$, kur A, B ir C – lentelės atributų aibės poaibiai. Tuomet lentelę L galima gauti jungiant jos projekcijas $L_1(A, B)$ ir $L_2(A, C)$.

Įrodymas. Teoremą įrodysime prieštaros būdu. Tarkime, $A \to B$ arba $A \to C$ ir lentelė $L(A, B, C)$ nėra jos projekcijų $L_1(A, B)$ ir $L_2(A, C)$ junginys, t.y. $L \neq L_1 \bowtie L_2$. Kadangi kiekvienai lentelei L ir jos projekcijoms L_1 ir L_2 galioja sąryšis $L \subseteq L_1 \bowtie L_2$, tai $L \neq L_1 \bowtie L_2$ reiškia, kad lentelių L_1 ir L_2 junginyje $L_1 \bowtie L_2$ yra eilutė $<a, b, c>$, kurios nėra lentelėje L. Jei eilutė $<a, b, c>$ priklauso L_1 ir L_2 junginiui, tai lentelėje L_1 yra eilutė $<a, b>$, o lentelėje L_2 – eilutė $<a, c>$. Vadinasi, lentelėje L yra eilutės $<a, b', c>$ ir $<a, b, c'>$, kuriose $b' \neq b$ ir $c' \neq c$. Todėl joje negalioja nei $A \to B$, nei $A \to C$. Tai prieštarauja prielaidai.

Hezo teorema garantuoja, jei lentelėje $L(A, B, C)$ galioja $A \to B$ arba $A \to C$, tai $L = \pi_{AB}(L) \bowtie \pi_{AC}(L)$. Jungdami lentelės L projekcijas, gauname tą pačią lentelę L. Kitaip tariant, jei $A \to B$ arba $A \to C$, tai lentelės L skaidymas į $\pi_{AB}(L)$ ir $\pi_{AC}(L)$ yra dekompozicija be praradimo.

Hezo teoremos teiginys nėra apverčiamas. Tai, kad $\pi_{AB}(L)$ ir $\pi_{AC}(L)$ yra lentelės L dekompozicija be praradimo, nereiškia, kad $A \to B$ arba $A \to C$. Pavyzdžiui,

L

A	B	C
a	b	c
a	b'	c
a	b	c'
a	b'	c'

L_1

A	B
a	b
a	b'

L_2

A	C
a	c
a	c'

Nors $L_1 = \pi_{AB}(L)$, $L_2 = \pi_{AC}(L)$ ir $L = L_1 \bowtie L_2$, tačiau šiems duomenims negalioja nei $A \to B$, nei $A \to C$.

Ne kiekvienas lentelės skaidymas yra skaidymas be praradimo. Pavyzdžiui, išskaidykime lentelę *Projektai_Vykdymas* į tokias dvi lenteles

Projektai_Vykdymas1(*Projektas, Pavadinimas, Svarba, Trukmė, Pradžia, Vykdytojas*),
Projektai_Vykdymas2(*Vykdytojas, Statusas, Valandos*).

Tuomet, net tik negalėtume antrajai lentelei apibrėžti teisingo rakto, bet taip skaidydami prarandame dalį informacijos. Nesunku įsitikinti, kad

Projektai_Vykdymas $\subset$ *Projektai_Vykdymas1* $\bowtie$ *Projektai_Vykdymas2*.

Jungdami lenteles *Projektai_Vykdymas1* ir *Projektai_Vykdymas2* negauname tikslių duomenų apie vykdytojų dalyvavimą konkrečiuose projektuose. Taip atsitinka dėlto, kad

kiekviena lentelės *Projektai_Vykdymas1* eilutė jungiama su tiek *Projektai_Vykdymas2* eilučių, kiek kartų toje lentelėje yra paminėtas konkretus vykdytojas. Junginyje gausime, pavyzdžiui, kad vykdytojas Nr. 2 projektą Nr. 1 vykdo ir dokumentuotojo, ir analitiko, ir vadovo statusuose, nors iš tiesų jis tame projekte yra tik dokumentuotojas. Kadangi teoremos teiginys nėra apverčiamas, tai Hezo teorema nepaaiškina, kodėl taip atsitinka. Neformaliai galime tvirtinti, kad taip atsitinka todėl, kad taip skaidydami lentelę *Projektai_Vykdymas* prarandame funkcinę priklausomybę {*Projektas, Vykdytojas*} $\rightarrow$ {*Statusas, Valandos*}. Vėliau suformuluosime bendresnę teoremą su tokia sąlyga, kuri bus apverčiama.

3.9'. Trečioji norminė forma

(Įterpiama prieš pastraipą, prasidedančią žodžiu „Literatūroje“)

Hezo (*I.J. Heath*) teorema mus užtikrina, kad toks skaidymas yra dekompozicija be praradimo. Akivaizdu, kad taip skaidant yra išsaugomos visos funkcinės priklausomybės.

Duotąją lentelę $L(\underline{A, B, C})$, kurioje galioja dvi FP $A \to B$ ir $B \to C$, galima skaidyti ir kitaip, pvz. į tokias dvi jos projekcijas: $L_3(\underline{A, B})$ ir $L_4(\underline{A, C})$. Lentelėje L_3 galioja $A \to B$, o L_3 – galioja $A \to C$. Hezo (*I.J. Heath*) teorema garantuoja, kad ir šis skaidymas yra dekompozicija be praradimo. Tačiau tai nėra dekompozicija išsaugant priklausomybes, nes FP $B \to C$ nėra išvedama iš lentelėse L_3 ir L_4 galiojančių FP. Nepageidaujamas skaidymo, kai neišsaugomos FP, savybes detaliau aptarsime kitame šios knygos skyriuje.

Naudojant pateiktąjį skaidymą, kuriuo sudaroma 3NF reikalavimus tenkinančios lentelės, galime gauti ir pakankamai neefektyvią DB schemą. Tarkime, turime lentelę $L(\underline{A, B, C, D})$, kurioje galioja FP $A \to B$, $B \to C$, $B \to D$. Tokia lentelė nėra 3NF, nes joje du atributai C ir D tranzityviai priklauso nuo rakto A. Jei lentelę skaidysime stengdamiesi panaikinti atributų CD tranzityviąją priklausomybę nuo rakto, tai gausime dvi lenteles: $L_1(\underline{A, B})$ ir $L_2(\underline{B, C, D})$. Tačiau, jei iš pradžių likviduojame vieno atributo tranzityviąją priklausomybę nuo rakto, o po to kito atributo, tai gauname tris lenteles: $L_1(\underline{A, B})$, $L_3(\underline{B, C})$ ir $L_4(\underline{B, D})$. Abiem skaidymais visos gautosios lentelės yra 3NF. Nesunku įsitikinti, kad abudu skaidymai yra dekompozicijos be praradimo ir išsaugančios funkcines priklausomybes. Tačiau pirmasis skaidymas yra efektyvesnis, nes lentelės L_2 saugojimui kompiuterio atmintyje reikės mažiau vietos negu saugant dvi lenteles: L_3 ir L_4.

Egzistuoja nesudėtingas algoritmas, užtikrinantis efektyvią lentelių dekompoziciją be praradimo kai išsaugomos funkcinės priklausomybės.

Algoritmas lentelei, kurioje galioja FP aibė F, skaidyti, užtikrinant 3NF.

$G := F_{min}$;
$i := 0$;
`for each` $X : \exists(X \to Y) \in G$
  $R := X$;
  `for each` $Y : \exists(X \to Y) \in G$
    $R := R \cup Y$;
  `endfor`
  $i := i + 1$;
  Į DB reliacinę schemą įtraukti $L_i(R)$;
`endfor`

Šio algoritmo pagrinde yra FP minimaliojo denginio F_{min} radimas. Toliau siekiama, kad visi atributai, priklausantys nuo to pačio atributų rinkinio, patektų į tą pačią lentelę. Pagal šį algoritmą, gausime tiek lentelių, kiek minimaliajame FP aibės denginyje yra skirtingų determinantų (kairiųjų FP pusių). Pasibaigus algoritmui kintamasis i yra lygus sudarytų lentelių skaičiui.

Pritaikykime šį algoritmą jau minėtai lentelei $L(\underline{A, B, C, D})$, kurioje galioja FP aibė F: $A \to B$, $B \to C$, $B \to D$, skaidyti. Nesunku įsitikinti, kad ši FP aibė tenkina minimaliosios FP aibės reikalavimus, todėl ji kartu yra ir minimalusis denginys, $F_{min} = F$. Kadangi aibėje F_{min} yra tik du skirtingi determinantai (A ir B), tai išorinio ciklo kūną reikia atlikti 2 kartus. Pirmąjį kartą imame $X := A$. Įvykdę vidinį ciklą gauname pirmosios lentelės schemą $L_1(\underline{A, B})$. Vykdant išorinį ciklą antrą kartą, imame $X := B$. Įvykdę vidinio ciklo kūną du kartus (pirmąjį kartą $Y := C$, o antrąjį $Y := D$) gauname antrosios lentelės schemą $L_2(\underline{B}, C, D)$, kurią sudaro visi atributai, esantys funkcinėse priklausomybėse su determinantu B.

3.10'. Boiso-Kodo norminė forma

Abu pasiūlytieji lentelės *Projektų_Etapai* pertvarkymai yra skaidymai be praradimo. Nesunku įsitikinti, kad abiem lentelės skaidymo variantais yra išsaugojamos visos FP.

Prisiminus visiškai netrivialiosios priklausomybės ir lentelės viršrakčio apibrėžimus, BKNF apibrėžimą galima šiek tiek performuluoti. Lentelė yra BKNF FP aibės F atžvilgiu, jei kiekvienoje tenkinamoje FP $X \to A$ atributai X yra lentelės viršraktis.

BKNF apibrėžimas pastebimai skiriasi nuo 3NF apibrėžimo. Iš apibrėžimų nėra lengva nustatyti šių norminių formų tarpusavio priklausomybę. Parodysime, kad BKNF reikalavimai yra griežtesni už 3NF reikalavimus.

Teiginys. Jei lentelė yra BKNF, tai ji yra ir 3NF.

Įrodymas. Įrodysime prieštaros būdu. Tarkime lentelė $L(R)$ yra BKNF, bet ji nėra 3NF. Jei lentelė $L(R)$ nėra 3NF, tai joje būtinai egzistuoja atributų aibės poaibiai K, A, B tokie, kad K – raktas, A – nepirminiai, $A \not\subset KB$, kuriem galioja FP $K \to B$, $B \to A$ bei negalioja FP $B \to K$. Kadangi negalioja FP $B \to K$, tai B nėra raktas. Kadangi $A \not\subset B$, tai $B \to A$ yra visiškai netriviali, t.y. determinante nėra atributų, esančių dešiniojoje FP pusėje. Todėl FP $B \to A$ netenkina BKNF reikalavimų. Tai reiškia, kad lentelė $L(R)$ nėra BKNF.

Praktikoje dauguma lentelių, esančių 3NF, yra ir BKNF. Tik tuomet, kai lentelėje yra tenkinama FP $X \to A$, kurioje X nėra viršraktis ir A yra pirminis atributas, lentelė yra 3NF, bet nėra BKNF. Formalizuokime šį teiginį.

Teiginys. Jei lentelė yra 3NF ir nėra BKNF FP aibės F atžvilgiu, tai egzistuoja $X \to A \in F^+$, kurioje X nėra viršraktis ir A yra pirminis atributas.

Įrodymas. Tarkime, lentelė $L(R)$ yra 3NF ir nėra BKNF. Jei nėra BKNF, tai egzistuoja netrivialioji FP $X \to A$, kurioje X nėra viršraktis. Prieštaros būdu parodysime, kad A yra pirminis atributas. Tarkime, A – nepirminis. Jei X - ne viršraktis, tai bet kuriam lentelės L raktui K negalioja FP $X \to K$. Kadangi K - raktas, tai galioja $K \to X$. Kadangi FP $X \to A$ yra netrivialioji, tai $A \notin X$. Gavome, kad nepirminis atributas A tranzityviai priklauso nuo rakto K ($K \to X \in F^+$, $X \to K \notin F^+$, $X \to A$, ir $A \notin KX$). Vadinasi L nėra 3NF, o tai prieštarauja prielaidai.

Teiginys. Jei lentelė yra 3NF FP aibės F atžvilgiu ir egzistuoja FP $X \to A \in F^+$, kurioje X nėra viršraktis ir A yra pirminis atributas, tai ta lentelė nėra BKNF.

Įrodymas. Tarkime, lentelė $L(R)$ yra 3NF FP F atžvilgiu ir iš aibės F galima išvesti FP $X \to A$, kurioje X nėra viršraktis ir A yra pirminis atributas. Kadangi FP $X \to A$ determinantas X nėra viršraktis, tai ši lentelė netenkina BNKF reikalavimų, t.y. ji nėra BKNF.

3.10a. Nedalomosios lentelės

Tarkime, kiekvienas firmos darbuotojas gali turėti kelias kvalifikacijas, kurios įgyjamos sėkmingai baigus mokymo įstaigoje konkrečią mokymo programą, o informacijai apie tai yra skirta lentelė *Studijos*(*Nr*, *Kvalifikacija*, *Mokymo_programa*), kurios stulpeliai reiškia darbuotojo numerį, įgytos kvalifikacijos pavadinimą ir mokymo programos, kurią baigus buvo įgyta kvalifikacija, pavadinimą. Tarkime, ta pati kvalifikacija gali būti suteikta baigus kelias skirtingas mokymo programas, tačiau kiekvienai konkrečiai mokymo programai atitinka tik viena kvalifikacija. Papildomai tarkime, kad kiekvienas darbuotojas konkrečią kvalifikaciją gali įgyti tik vieną kartą baigęs vieną konkrečią mokymo programą. Tuomet ši lentelė turi du raktus: {*Nr*, *Kvalifikacija*} ir {*Nr*, *Mokymo_programa*}. Kadangi kvalifikacijos pavadinimas dažniausiai būna trumpesnis už programos pavadinimą, pirmąjį raktą parenkame pirminiu. Pateiksime šios lentelės duomenų fragmentą.

Studijos

Nr	*Kvalifikacija*	*Mokymo_programa*
1	Informatikas	Kompiuterių mokslas
1	Statistikas	Ekonometrija
5	Informatikas	Programų sistemos
5	Statistikas	Matematinė statistika

Šioje lentelėje galioja trys FP:

{*Nr*, *Mokymo_programa*} → *Kvalifikacija*,
{*Nr*, *Kvalifikacija*} → *Mokymo_programa*,
Mokymo_programa → *Kvalifikacija*.

Lentelėje *Studijos* visi atributai yra pirminiai, todėl ji yra 3NF. Kadangi trečiojoje visiškai netrivialioje (nesuprastinamoje) FP determinantas nėra raktas, tai lentelė nėra BKNF. Todėl lentelę *Studijos* reikia skaidyti į dvi:

Vykdytojų_studijos(*Nr*, *Mokymo_programa*),
Mokymo_programos(*Mokymo_programa*, *Kvalifikacija*).

Anksčiau pateiktieji lentelės *Studijos* duomenys pasiskirsto taip

Vykdytojų_studijos

Nr	*Mokymo_programa*
1	Kompiuterių mokslas
1	Ekonometrija
5	Programų sistemos
5	Matematinė statistika

Mokymo_programos

Mokymo_programa	*Kvalifikacija*
Kompiuterių mokslas	Informatikas
Ekonometrija	Statistikas
Programų sistemos	Informatikas
Matematinė statistika	Statistikas

Tačiau dabar pirmoje lentelėje galime apibrėžti tik trivialiąją FP

{*Nr*, *Mokymo_programa*} → {*Nr*, *Mokymo_programa*},

o antrojoje – tik vieną FP, išreiškiančią lentelės raktą,

Mokymo_programa → *Kvalifikacija*.

Iš šių dviejų FP mes negalime išvesti FP {*Nr*, *Kvalifikacija*} → *Mokymo_programa*, kuri galioja pradinėje lentelėje *Studijos*. Todėl gautąsias dvi lenteles, kurios yra pradinės lentelės *Studijos* projekcijos, negalima atnaujinti nepriklausomai. Pavyzdžiui, esant apibrėžtiesiems šių lentelių raktams, nėra kliūčių įterpti į lentelę *Vykdytojų_studijos* eilutę, kurioje nurodytas darbuotojo Nr. 1 ir mokymo programa „Programų sistemos“. Tačiau toks įterpimas turėtų būti atmestas, kadangi baigusiesiems šią programą yra suteikiama informatiko kvalifikacija, kurią darbuotojas Nr. 1 jau turi. Pastarajam faktui patikrinti, papildomai reikia kreiptis į kitą lentelę *Mokymo_programos*. Pradinėje lentelėje *Studijos*, nepažeidžiant raktų vientisumo, tokių duomenų įterpti nebuvo galima. Suskaidydami lentelę *Studijos* į dvi jos projekcijas: *Vykdytojų_studijos* ir *Mokymo_programos* sudarėme DB schemą tenkinančią BKNF reikalavimus, tačiau praradome vieną svarbią duomenų tarpusavio priklausomybę. Pastebėsime, kad šis suskaidymas yra be praradimo, t.y. lentelės *Studijos* duomenys yra visiškai atstatomi jungiant jos projekcijas. Tai mes žinome iš Hezo (*I.J. Heath*) teoremos. Netgi skaidymas be praradimo ne visada užtikrina atributų tarpusavio FP nepraradimą.

Lentelė *Studijos* yra nedalomosios lentelės pavyzdys. Lentelė vadinama **nedalomąja lentele**, jeigu jos negalima suskaidyti į tarpusavyje nepriklausomas lenteles, formaliau, jos negalima suskaidyti taip, kad išsaugotume visas priklausomybes. Jei lentelę *Studijos* neskaidome, tai susiduriame su duomenų pertekliumi, kuris sukelia duomenų anomalijas. Tačiau, jei ją skaidome, prarandame FP. Lentelei *Studijos* negalime rasti visais atžvilgiais teisingo sprendimo. Tai reliacinio modelio ribotumas. Konkrečiose DBVS lentelę *Studijos* reikėtų skaidyti, kad ji tenkintų BKNF reikalavimus, o dėl skaidymo prarandamas FP galima užtikrinti reliaciniam modeliui nebūdingomis priemonėmis, pvz. trigeriais, kuriuos aptarsime kituose knygos skyriuose.

Priminsime, kad ankstesniajame skyrelyje atliktas lentelės *Projektų_Etapai* pertvarkymas abiem būdais yra skaidymai be praradimo ir išsaugojantys FP. Todėl lentelei, esančiai 3NF, bet neesančiai BKNF, gali egzistuoti suskaidymas be praradimo kai išsaugojamos visos FP.

3.11'. Ketvirtoji norminė forma

Daugiareikšmės priklausomybės (DRP) apibrėžimas yra simetriškas. Tarkime, lentelėje $L(A, B, C)$ galioja DRP $A \rightarrow\!\!\!\rightarrow B$. Tuomet lentelėje L yra trys eilutės <a, b, c>, <a, b′, c′>, <a, b′, c>. Pritaikykime DRP apibrėžimą toms pačioms pradinėms eilutėms, tik sukeiskime jas vietomis. Gauname: jei lentelėje yra <a, b′, c′> ir <a, b, c>, tai lentelėje turi būti ir <a, b, c′>. Todėl, jei lentelėje $L(A, B, C)$ galioja DRP $A \rightarrow\!\!\!\rightarrow B$, tai bet kurioms dviem lentelės L eilutėms <a, b, c>, <a, b′, c′> lentelėje yra dar dvi eilutės <a, b′, c> ir <a, b, c′>. Šią simetriškumo savybę galima suformuluoti teiginiu.

Teiginys. Jei lentelėje $L(A, B, C)$ galioja DRP $A \rightarrow\!\!\!\rightarrow B$, tai joje galioja ir DRP $A \rightarrow\!\!\!\rightarrow C$.

Įrodymas. Tarkime, lentelėje $L(A, B, C)$ galioja $A \rightarrow\!\!\!\rightarrow B$, bet negalioja $A \rightarrow\!\!\!\rightarrow C$. Jei negalioja $A \rightarrow\!\!\!\rightarrow C$, tai atributo A reikšmei a lentelėje galima rasti 2 eilutes: <a, b, c>, <a, b′, c′>, bet nėra bent vienos iš šių dviejų eilučių: <a, b′, c>, <a, b, c′>. Tai reiškia, kad lentelėje L negalioja $A \rightarrow\!\!\!\rightarrow B$. Prieštaravimas.

Pateiksime, dar vieną DRP apibrėžimo galimą formuluotę. Lentelėje $L(A, B, C)$ galioja DRP $A \rightarrow\!\!\!\rightarrow B$ tada ir tik data, kai bet kurioms dviems lentelės L eilutėms: e_1 ir e_2, kurioms $e_1(A) = e_2(A)$, lentelėje L egzistuoja eilutės e_3 ir e_4 tokios, kad $e_3(A) = e_1(A)$, $e_4(B) = e_1(B)$, $e_4(C) = e_2(C)$, $e_4(A) = e_1(A)$, $e_3(B) = e_2(B)$, $e_3(C) = e_1(C)$.

Lentelė $L(R)$ yra **ketvirtosios norminės formos** (**4NF**) FP ir DRP aibės F atžvilgiu, tada ir tik tada, kai kiekviena DRP $A \rightarrow\!\!\!\rightarrow B \in F^+$, $A \subset R$, $B \subset R$, arba yra triviali, arba jos determinantas A yra lentelės L viršraktis.

Jeigu DRP $A \rightarrow\!\!\!\rightarrow B$ determinantas A yra lentelės raktas, tai ši DRP faktiškai yra FP. Todėl, 4NF apibrėžime vietoje reikalavimo, kad kiekvienos netrivialios DRP determinantas yra lentelės viršraktis, galima pakeisti reikalavimu, kad kiekviena netriviali DRP būtų FP. Tam, kad lentelė L būtų 4NF, lentelėje turi galioti tik FP, kuriose determinantas yra viršraktis, arba, kitaip tariant, tik visiškai netrivialios FP, kuriose determinantas yra raktas. Todėl lentelė, esanti 4NF yra ir BKNF.

Jau minėjome, kad kiekviena FP yra ir DRP. Tačiau ne kiekviena DRP yra FP. Paimkime, pavyzdžiui tokią paprastą lentelę *Pavyzdys*.

Pavyzdys

A	*B*	*C*
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_2	c_1

Šios lentelės duomenys tenkina FP $A \rightarrow B$ reikalavimus. Vadinasi, yra tenkinama ir DRP $A \rightarrow\!\!\!\rightarrow B$. Iš pirmo žvilgsnio gali atrodyti, kad taip nėra, bet ši DRP visiškai tenkina DRP apibrėžimo reikalavimus. Iš aukščiau įrodyto teiginio seka, kad šioje lentelėje galioja ir $A \rightarrow\!\!\!\rightarrow C$. Nesunku įsitikinti, kad ir ši DRP tenkina jai keliamus reikalavimus. Tačiau atitinkama FP $A \rightarrow C$ negalioja.

Tarkime, papildomai kalbų, kurias moka darbuotojai, pavadinimams duomenų bazėje reikia įsiminti ir jų mokėjimo lygmenį. Tam papildykime lentelę *Kalbos* stulpeliu *Lygmuo*

Kalbų_mokėjimas

Nr	*Kalba*	*Lygmuo*
1	Anglų	Puikiai
1	Vokiečių	Gerai
2	Anglų	Gerai

Naujojoje lentelėje galioja tik viena FP $\{Nr, Kalba\} \rightarrow Lygmuo$, kurios determinantas yra lentelės raktas. Lentelėje *Kalbos*, kurioje nėra stulpelio *Lygmuo*, galioja DRP $Nr \rightarrow\!\!\!\rightarrow Kalba$,

tačiau naujojoje lentelėje ji negalioja. Ši lentelė yra 4NF, nes {*Nr, Kalba*} $\rightarrow\!\!\!\rightarrow$ *Lygmuo* yra trivialioji DRP, be to ji yra ir FP, kurios determinantas yra raktas.

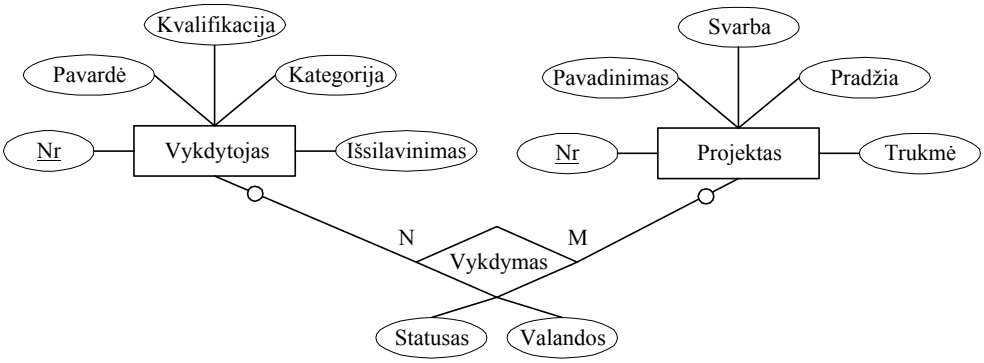
4. Semantinis modeliavimas

4.5". DB „Darbai“ ER modelis

Pavaizduokime ER diagrama anksčiau jau nagrinėtą dalykinę sritį, atitinkančią duomenų bazę *Darbai*. Tiksliau, pakartokime DB *Darbai* projektavimą – sudarykime šios srities ER modelį ir po to atvaizduokime jį reliacine schema.

Kaip ir anksčiau, nagrinėkime paprasčiausią įstaigos veiklos atvejį, kai įstaigos darbuotojai vykdo projektus, kiekvienas turi konkretų vaidmenį ir skiria projektui tam tikrą laiką. Praktiškai nedvejojant galima išskirti dvi esybes: *Vykdytojas* ir *Projektas*. Kiekvienos iš šių esybių atributų rinkinys priklauso nuo įstaigos veiklos parametrų. Apsiribokime ankstesniame skyriuje nagrinėtais vykdytojo atributais: *Nr*, *Pavardė*, *Kvalifikacija*, *Kategorija*, *Išsilavinimas*, ir projekto atributais: *Nr*, *Pavadinimas*, *Svarba*, *Pradžia*, *Trukmė*.

Į vykdytojų dalyvavimą projektuose galima žiūrėti kaip į N:M tipo ryšį *Vykdymas* tarp esybių *Vykdytojas* ir *Projektas*. Abu ryšio dalyviai yra sąlygiški. Šis ryšys turi du atributus *Statusas* ir *Valandos*. Taip sudaryto ER modelio *Darbai* schema pateikta 4.4 pav.



4.4 pav. E-R modelio *Darbai* schema. Vykdytojų dalyvavimas projektuose modeliuojamas ryšiu.

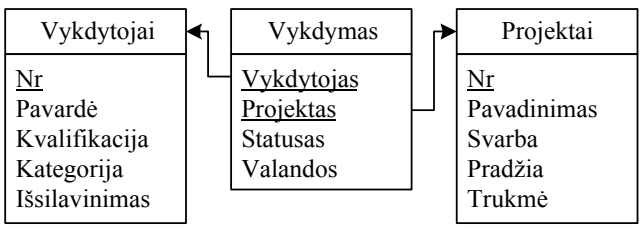
Pavaizduokime šią ER diagramą reliacine DB schema. Iš esybių *Vykdytojas* ir *Projektas* gauname dvi lenteles:

Vykdytojai (*Nr, Pavardė, Kvalifikacija, Kategorija, Išsilavinimas*);
Projektai (*Nr, Pavadinimas, Svarba, Pradžia, Trukmė*).

Sudarydami esybėms lenteles, jas pervardijome tik tam, kad lentelių vardai atitiktų anksčiau vartotus vardus. N:M tipo ryšiui *Vykdymas* sudarome atskirą lentelę *Vykdymas*, į kurią įtraukiame ryšio atributus ir jo dalyvių raktus, kurie tampa išoriniais raktais:

Vykdymas (*Projektas, Vykdytojas, Statusas, Valandos*),
išoriniai raktai: *Projektas* nukreipia į *Projektai*,
Vykdytojas nukreipia į *Vykdytojai*.

Sudarytos duomenų bazės *Darbai* reliacinę schemą galima pavaizduoti grafiškai:



4.5 pav. Duomenų bazės *Darbai* reliacinė schema, atitinkanti ER modelį, pavaizduotą 4.4 pav.

4.5a. Ryšių realizavimo ypatumai

ER modelyje, pavaizduotame 4.4 pav., vykdytojų dalyvavimas projektuose yra išreikštas N:M ryšiu *Vykdymas* su dviem atributais. Taip apibrėžiama kiekvieno vykdytojo galimybė dalyvauti keliuose projektuose ir tai, kad vieną projektą gali vykdyti keli vykdytojai. Konkretaus vykdytojo indėlis projekte apibūdinamas ryšio atributais. Šiam ryšiui reliacinėje schemoje, pavaizduotoje 4.5 pav., atitinka lentelė *Vykdymas*, kurios raktas yra iš dviejų stulpelių, atitinkančių esybių raktus.

Lentelėje *Vykdymas* konkretaus vykdytojo dalyvavimas konkrečiame projekte gali būti pateiktas tik vienoje eilutėje. Tai reiškia, kad konkretus vykdytojas konkrečiame projekte gali dalyvauti tik viename statuse ir skirti tik vieną konkretų skaičių valandų. Taip sudarytoje lentelėje *Vykdymas* negalima kaupti vykdytojų dalyvavimo projektuose istorijos. Jeigu konkrečiame vykdyme vykdytojas projektui papildomai skiria konkretų skaičių valandų, tai jas reikia pridėti prie anksčiau skirtųjų. Jei vykdytojo statusas projekte keičiasi, tai mes turime „pamiršti“ ankstesnį statusą. Tačiau kartais būna svarbu išsaugoti ankstesnę informaciją.

N:M tipo ryšį atitinkančiai lentelei galima sudaryti **dirbtinį** raktą reiškiantį vykdymo numerį. Pavadinkime šį stulpelį *Nr*, o naująją lentelę – *Vykdymai*. Visi kiti stulpeliai – tokie pat kaip anksčiau sudarytoje lentelėje *Vykdymas*. Naujosios lentelės *Vykdymai* reliacinė schema yra

Vykdymai (<u>*Nr*</u>, *Projektas, Vykdytojas, Statusas, Valandos*),
išoriniai raktai: *Projektas* nukreipia į *Projektai*,
Vykdytojas nukreipia į *Vykdytojai*.

Taip sudarytoje lentelėje, kaip ir anksčiau sudarytoje lentelėje *Vykdymas*, galima saugoti duomenis apie vykdytojų dalyvavimą keliuose projektuose, bei apie visus kiekvieno projekto vykdytojus. Papildomai, kiekvienas vykdytojas konkrečiame projekte gali dalyvauti keliuose statusuose. Be to, kiekvieno vykdytojo konkrečiam projektui papildomai skiriamas valandas galima įsiminti naujoje eilutėje. Pavaizduokime naujosios lentelės fragmentą:

Vykdymai

Nr	*Projektas*	*Vykdytojas*	*Statusas*	*Valandos*
1	1	1	Programuotojas	30
2	1	1	Programuotojas	10
3	1	1	Testuotojas	40
4	1	2	Dokumentuotojas	100
5	1	2	Dokumentuotojas	200

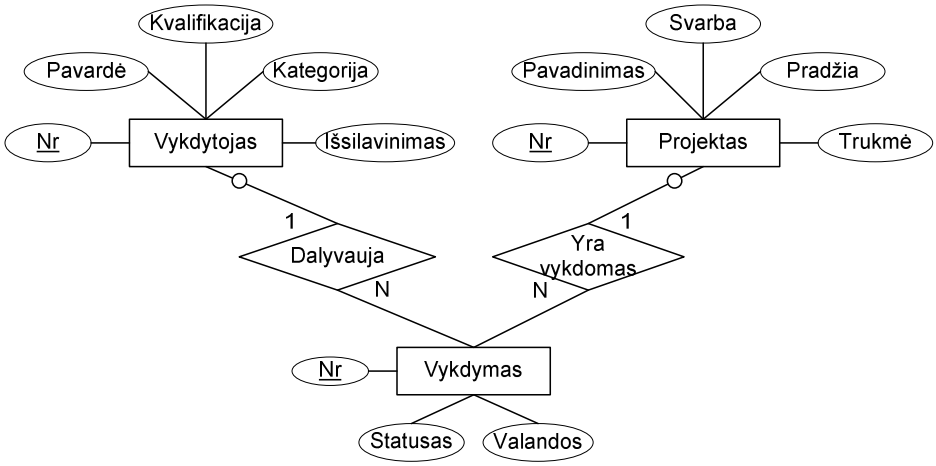
Taip sudaryta lentelė *Vykdymai* yra lankstesnė už anksčiau sudarytą lentelę *Vykdymai*. Tačiau, tai nereiškia, kad ji yra pranašesnė. Tai priklauso nuo konkrečių aplinkybių.

Tarkime, aplinkybės yra nusakomos FP {*Projektas*, *Vykdytojas*} → {*Statusas*, *Valandos*}. Tuomet, lentelė *Vykdymas* visiškai užtikrina šią FP. Lentelėje *Vykdymai* ši FP taip pat gali būti užtikrinama apibrėžiant papildomą raktą {*Projektas*, *Vykdytojas*}. Tačiau tuomet, stulpelis *Nr* lentelėje *Vykdymai* tampa beprasmiu. Šiuo atveju, anksčiau sudaryta lentelė *Vykdymas* yra pranašesnė už *Vykdymai*, nes ji yra paprastesnė.

Jei aplinkybės lemia, kad FP {*Projektas*, *Vykdytojas*} → {*Statusas*, *Valandos*} bendruoju atveju negalioja, tai lentelė *Vykdymas* negali turėti apibrėžtojo rakto. Lentelėje *Vykdymai* galioja FP *Nr* → {*Projektas*, *Vykdytojas, Statusas*, *Valandos*}, kuris pakankamai gerai atspindi tokią situaciją. Tačiau ir šiuo atveju lentelėje *Vykdymai* stulpelį *Nr* galima laikyti pertekliniu, nors dažniausiai kiekviena lentelė turi raktą. Raktas yra būtinas kai, kitoje lentelėje norime apibrėžti išorinį raktą, nukreipiantį į šią lentelę. Lentelė *Vykdymai* yra pagrįsta, kai FP {*Projektas*, *Vykdytojas*} → {*Statusas*, *Valandos*} negalioja.

Nors abiejų lentelių (*Vykdymas* ir *Vykdymai*) struktūra yra labai panaši, jų savybės yra pakankamai skirtingos. Aplinkybės, įtakojančios ryšio realizavimą, ER modelyje *Darbai* atsispindi nepakankamai. Tam mums pritrūko ER modelio raiškos priemonių. Panašias aplinkybes galima aprašyti ER schemos lydraštyje.

Kita vertus, kai lentelė *Vykdymai* labiau atitinka dalykinę sritį nei lentelė *Vykdymas*, t.y. kai negalioja FP {*Projektas*, *Vykdytojas*} → {*Statusas*, *Valandos*}, vykdytojų dalyvavimą projektuose yra tikslingiau išreikšti ne ryšiu, bet esybe *Vykdymas*. Tokias aplinkybes vienareikšmiškai aprašanti ER schema yra pateikta 4.6 pav.



4.6 pav. E-R modelio *Darbai* schema. Vykdytojų dalyvavimas projektuose modeliuojamas esybe.

DB schema, atitinkanti ER schemą, pavaizduotą 4.6 pav., skirtųsi nuo pateiktosios 4.5 pav. tik vienos lentelės schema. Vietoje lentelės *Vykdymas* duomenų bazėje *Darbai* būtų lentelė *Vykdymai*. Realizuojant ER schemą nekyla neaiškumų, būdingų realizuojant N:M ryšius. Jei kiekvienas vykdytojas kiekviename projekte gali dalyvauti tik viename statuse ir vykdytojo indėlis projekte apibūdinamas bendru valandų skaičiumi, tai tokį uždavinį tikslinga vaizduoti ER schema, pavaizduota 4.4 pav., kitaip – 4.6 pav.

Atsižvelgdami į čia pateiktas pastabas, patikslinsime N:M ryšio realizavimo taisyklę. Ryšys N:M realizuojamas sudarant atskirą lentelę. Į lentelę įtraukiami abiejų lentelių, atitinkančių ryšyje dalyvaujančias esybes, pirminiai raktai, kurie tampa išoriniais raktais. Į lentelę taip pat įtraukiami visi ryšio atributai. Lentelė gali neturėti nei vieno rakto. Tačiau dažniausiai lentelės pirminis raktas sudaromas vienu iš šių būdų:

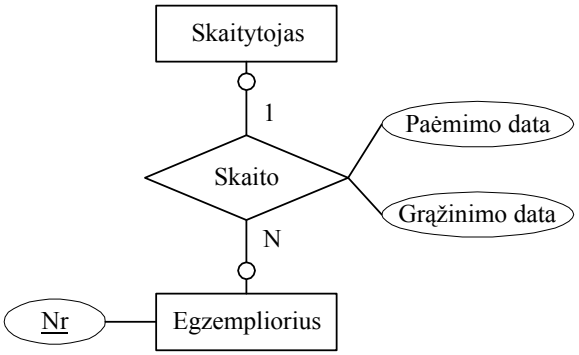
- lentelės pirminį raktą sudaro abiejų susietųjų lentelių pirminių raktų derinys;
- lentelė papildoma nauju stulpeliu, kuris tampa jos pirminiu (dirbtiniu) raktu.

Kai lentelė papildoma dirbtiniu pirminiu raktu, tenka spręsti, kaip parinkti šio stulpelio reikšmes. Dažniausiai tam naudojama natūraliųjų skaičių aibė. Kiekvieną kartą įvedant naują eilutę, parenkamas vis kitas natūrinis skaičius – eilutės numeris. Numerio parinkimas nesudaro sunkumų, nes SQL kalboje tam yra numatytos specialios priemonės, kurias aptarsime vėliau.

Papildomai panagrinėkime ER modelyje *Biblioteka* esantį ryšį *Skaito*. Šis ryšys sieja dvi esybes: *Egzempliorius* ir *Skaitytojas*. Bibliotekos modelyje, kuris pavaizduotas 4.2 pav., ir realizuotas DB schema, pavaizduota 4.3 pav., yra stebimi tik šiuo metu paimtų egzempliorių skaitytojai. Gali būti prasminga prisiminti visus ar bent kelis paskutiniuosius egzempliorių skaitytojus. Pavyzdžiui, paaiškėjus, kad konkretus egzempliorius yra sugadintas, turint duomenis apie visus skaičiusius jį skaitytojus, galima ieškoti kaltininko. Žinoti ankstesnį skaitytoją naudinga ir tuomet, kai naujasis skaitytojas randa knygoje ankstesnio skaitytojo paliktą daiktą, pvz. svarbų dokumentą. Visų skaitytojų įsiminimas taip pat yra naudingas statistiniuose tyrimuose ir kt.

ER modelyje *Biblioteka* egzemplioriaus paėmimo ir grąžinimo datos yra esybės *Egzempliorius* atributai. Vadinasi, kiekvienam egzemplioriui būdinga tik viena paėmimo ir viena grąžinimo datos bei vienas skaitytojas. Kadangi tarp lentelės *Egzemplioriaus* stulpelių galioja FP *Nr* → {*Paėmimo_data*, *Grąžinimo_data*, *ISBN*, *Skaitytojo*_Nr}, tai jau sudaryta DB schema visiškai atitinka modelį.

Sudarydami bibliotekos modelį, paėmimo ir grąžinimo datas mes galėjome priskirti ne egzemplioriui, bet ryšiui *Skaito*. Taip pakeisto modelio ER diagramos fragmentas yra pavaizduotas 4.7 pav.



4.7 pav. Ryšys *Skaito* su egzemplioriaus paėmimo ir grąžinimo datomis

4.3 pav. pavaizduota DB schema visiškai atitinka ir taip atnaujintą ER modelį *Biblioteka*. Taip sudarytą ryšį *Skaito*, kaip ir bet kurį kitą ryšį, galime realizuoti atskira lentele *Skaitymas*, įtraukiant į ją ryšio atributus atitinkančius stulpelius (*Paėmimo_data*, *Grąžinimo_data*) ir stulpelius *Egzemplioriaus_Nr*, *Skaitytojo_Nr*, kurie atitinka susijusių su ryšiu lentelių *Egzempliorius* ir *Skaitytojas* pirminius raktus. Kadangi kiekvieną egzempliorių gali skaityti tik vienas skaitytojas, tai tokios lentelės pirminis raktas yra išorinis raktas nekreipiantis į lentelę *Egzempliorius*:

Skaitymas (*Egzemplioriaus_Nr, Skaitytojo_Nr, Paėmimo_data*, *Grąžinimo_data*),
išoriniai raktai: *Egzemplioriaus_Nr* nukreipia į *Egzempliorius*,
Skaitytojo_Nr nukreipia į *Skaitytojas*.

Lentelė *Egzempliorius* žymiai supaprastėja:

Egzempliorius(*Nr*, *ISBN*, *Skaitymas*),
išoriniai raktai: *ISBN* nukreipia į *Knyga*,
Skaitymas nukreipia į *Skaitymas*.

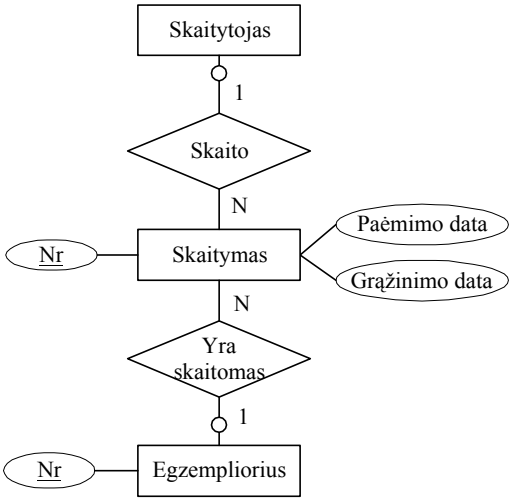
Tačiau, tokia lentelių *Skaitymas* ir *Egzempliorius* sandara nėra prasminga. Duomenų savybių prasme tokia sandara visiškai sutampa su ankstesniąja, kuri pateikta 4.3 pav., tačiau dabar mes turime viena lentele daugiau. Taip sudaryta DB tikriausiai užims daugiau vietos kompiuterio atmintyje. Tačiau yra ir teigiamų pastarosios sandaros bruožų. Jei didžioji dauguma egzempliorių bibliotekoje, dažniausiai, būna nepaimti, tai anksčiau sudarytoje lentelėje *Egzempliorius* (žr. 4.3 pav.) būtų daug `NULL` reikšmių – neefektyviai naudojami trys jos stulpeliai: *Skaitytojo_Nr*, *Paėmimo_data*, *Grąžinimo_data*. Naudojant naujesniąją DB sandarą, lentelėje *Skaitymas* `NULL` reikšmių visiškai nebus. Tiesiog lentelėje *Skaitymas* bus labai nedaug eilučių. Atsižvelgdamas į egzempliorių skaičių ir jų paėmimų dažnį, duomenų bazės administratorius turėtų įvertinti ir duomenų atnaujinimo pobūdį. Kai lentelės *Skaitymas* nėra, egzemplioriaus paėmimas ir grąžinimas yra lentelės *Egzempliorius* reikšmių atnaujinimas. Kai naudojama lentelė *Skaitymas*, tai egzemplioriaus paėmimas yra naujos eilutės įterpimas į lentelę *Skaitymas*, o grąžinimas – eilutės trynimas. Tai pakankamai subtilūs duomenų bazės administravimo klausimai. Kiekvienu atveju yra tiek teigiamų, tiek ir neigiamų aspektų. Dažniausiai, kuo mažiau nereikalingų lentelių, tuo geriau.

Jeigu realizuodami ryšį *Skaito* atskira lentele, lentelėje *Skaitymas* apibrėžtąjį pirminį raktą *Egzemplioriaus_Nr* pakeistume dirbtiniu raktu *Nr* – egzemplioriaus paėmimo registracijos numeriu, tai pažeistume ryšio pobūdį. Tuomet vienam knygos egzemplioriui lentelėje *Skaitymas* galėtų būti keli skaitytojai, ryšys 1:N taptų ryšiu N:M.

Jei norime duomenų bazėje išsaugoti duomenis apie visus visų egzempliorių paėmimus, tai mums reikia labiau pakeisti modelį *Biblioteka*. Vienas iš būdų tai padaryti yra ryšio *Skaito* pavertimas N:M ryšiu. Tokį ryšį galima pavaizduoti ER schema, labai panašia į pateiktąją 4.7

pav. Pakanka linijos žymenį „1“ reikia pakeisti žymeniu „M“. Realizuodami tokį ryšį galime elgtis panašiai kaip jau aptarto N:M ryšio *Vykdymas* atveju. Jeigu N:M ryšiui *Skaito* sukurtume atskirą lentelę su raktu iš dviejų atributų: *Egzemplioriaus_Nr*, *Skaitytojo_Nr*, tai suteiktume kitokią interpretaciją įsivaizduojamam paėmimų istorijos saugojimui. Tuomet, pavyzdžiui, joks skaitytojas negalėtų paimti tą patį egzempliorių du kartus. Tik dirbtinio rakto įvedimas atitiktų visos paėmimų istorijos įsiminimą.

ER modelis pakankamai tiksliai išreikštų siekį saugoti „istoriją“, jeigu ryšį *Skaito* pakeistume esybe *Skaitymas* suteikdami jai raktinį atributą *Nr*. Toks atributas yra prasmingas, jis atitinka paėmimo registracijos eilės numerį galimame egzempliorių paėmimo žurnale ar kartotekoje. Atnaujintos ER schemos fragmentas pavaizduotas 4.8 pav.



4.8 pav. Esybė *Skaitymas* ir jos ryšiai

Atnaujinto ER modelio *Biblioteka* esybę *Skaitymas* (4.8 pav.) bei ryšius *Skaito* ir *Yra skaitomas* galima realizuoti atskira lentele su dviem išoriniais raktais:

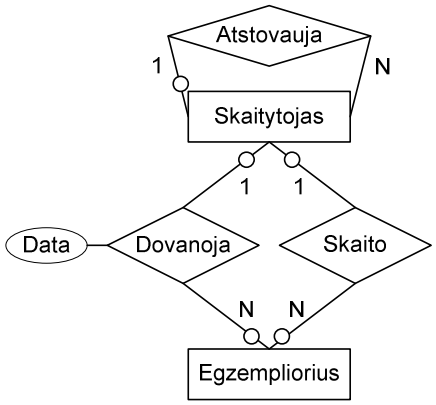
Skaitymas(*Nr*, *Egzemplioriaus_Nr*, *Skaitytojo_Nr*, *Paėmimo_data*, *Grąžinimo_data*),
išoriniai raktai: *Egzemplioriaus_Nr* nukreipia į *Egzemplioriaus*,
Skaitytojo_Nr nukreipia į *Skaitytojas*.

Lentelės *Egzempliorius*(*Nr*, *ISBN*, *Skaitymas*) sandara išlieka nepakitus. Taip sudarę duomenų bazę, lentelėje *Skaitymas* galėsime įsiminti duomenis apie visus visų egzempliorių paėmimus.

Jau minėjome, kad ryšys gali sieti esybę su ja pačia. Tarkime, kiekvienam skaitytojui yra priskiriamas kitas bibliotekos skaitytojas, į kurį bibliotekos darbuotojai gali kreiptis, jei, pavyzdžiui, skaitytojas vengia grąžinti knygą. Studentui gali būti priskirtas jo grupės seniūnas, o darbuotojui – padalinio vadovas. Tai - ryšys esybėje. 4.9 pav. pavaizduotas vienanaris ryšys *Atstovauja*, kurio dalyvis yra esybė *Skaitytojas*. Ryšys *Atstovauja* skaitytojui – atstovui priskiria jo atstovaujamus skaitytojus, arba atvirkščiai. Tarus, kad kiekvienas skaitytojas būtinai turi savo atstovą, bet ne kiekvienas skaitytojas atstovauja kurį nors skaitytoją, ryšys *Atstovauja* yra besąlygiškas iš vienos pusės ir sąlygiškas iš kitos.

Naująjį 1:N ryšį *Atstovauja* reliacinėje schemoje galima pavaizduoti lentelės *Skaitytojas* papildomu stulpeliu *Atstovas*. Šis stulpelis taip pat yra išorinis raktas į tą pačią lentelę *Skaitytojas*. Jau sudarytos lentelės *Skaitytojas* atnaujinta schema yra

Skaitytojas(*Nr*, *AK*, *Vardas*, *Pavardė*, *Gatvė*, *Namas*, *Butas, Atstovas*),
išorinis raktas: *Atstovas* nukreipia į *Skaitytojas*.



4.9 pav. Dvinaris ryšys *Atstovauja* ir vienanaris ryšys *Dovanoja*, papildantys ryšį *Skaito*.

Tarp tų pačių esybių gali būti ir keli ryšiai. Tarkime, skaitytojai gali dovanoti bibliotekai knygų egzempliorius. Taip tarp esybių *Skaitytojas* ir *Egzempliorius*, greta jau turimo ryšio *Skaito*, atsiranda ryšys *Dovanoja*, kuriuo egzempliorius siejamas su jį dovanojusiu skaitytoju. Tarkime, šis ryšys papildomai charakterizuojamas dovanojimo data. Naujasis ryšys yra sąlygiškas iš abiejų pusių. Jis pavaizduotas 4.9 pav.

Reliacinėje DB schemoje ryšį *Dovanoja* atitinka du papildomi stulpeliai lentelėje *Egzempliorius*. Vienas iš jų - *Dovanotojas* yra egzempliorių dovanojusio skaitytojo numeris, o kitas - *Dovanojimo_data*. Papildžius ER schemą *Biblioteka* (4.2 pav.) ryšiu *Dovanoja* (4.9 pav.), atnaujiname lentelės *Egzempliorius* reliacinę schemą:

Egzempliorius(<u>*Nr*</u>, *Paėmimo_data*, *Grąžinimo_data*, *ISBN*, *Skaitytojo_Nr, Dovanotojas, Dovanojimo_data*),
išoriniai raktai: *ISBN* nukreipia į *Knyga*,
Skaitytojo_Nr nukreipia į *Skaitytojas,*
Davanotojas nukreipia į *Skaitytojas*.

Dabar lentelėje *Egzempliorius* yra trys išoriniai raktai, du iš kurių nukreipia į tą pačią lentelę *Skaitytojas*.

5. Duomenų bazės sukūrimas ir užpildymas duomenimis

5.3a. Duomenų tipų suderinamumas

Praktikoje dažnai pasitaiko, kad atliekant operacijas su keliais argumentais, kreipiantis į funkcijas ir pan. tenka suderinti duomenų tipus. Taip yra, pvz., prie sveiko skaičiaus pridedant slankaus kablelio (realųjį) skaičių, taip pat lyginant fiksuoto ilgio simbolių eilutę su kintamojo ilgio simbolių eilute. Šiais, dažnai pasitaikančiais atvejais duomenų tipai yra suderinami automatiškai pagal plačiai taikomas taisykles, pvz., prieš sudedant sveikąjį ir realųjį skaičius, sveikasis yra pervedamas į realųjį. Konkrečios DBVS dokumentacijoje galima rasti tikslias tipų suderinamumo lenteles.

Tačiau automatinis tipų suderinamumas ne visada yra tikslingas ir įmanomas. Pvz., jei reiškinys *Vykdytojai.Kategorija* || 'kategorija' yra panaudotas užklausoje duomenų bazei *Darbai*, tai tikslinga jį laikyti klaidingu sintaksės požiūriu, nes vienas simbolių jungimo operacijos operandas yra skaičius. Norint užklausos rezultate šalia skaičiaus, atitinkančio darbuotojo kategoriją, turėti žodį „kategorija“, reikia reiškinyje išreikštai nurodyti skaičiaus transformavimą į simbolių eilutę. Vieno tipo reikšmės transformavimas į kito tipo reikšmę SQL kalboje išreiškiamas specifikacija `CAST`

`CAST` (<reiškinys> AS <duomenų tipas>)

Šia fraze SQL sakinyje nurodoma apskaičiuotąjį reiškinio rezultatą transformuoti į pasirinkto tipo reikšmę, pvz. jau suformuluotą užklausą informatikų pavardėms ir kategorijoms gauti, galima suformuluoti ir taip

`SELECT` *Pavardė,* `CAST(`*Kategorija* `AS CHAR(1)) ||` ' kategorija' `AS` *Kategorija*
`FROM` *Vykdytojai* `WHERE` *Vykdytojai.Kvalifikacija* = 'Informatikas'.

Jei duomenų bazėje yra antrajame skyriuje pateiktieji duomenys, tai šios užklausos rezultate bus dvi eilutės ir du stulpeliai

Pavardė	*Kategorija*
Jonaitis	2 kategorija
Antanaitis	3 kategorija

Tarkime, mums reikia apskaičiuoti visų vykdytojų kategorijų vidurkį. Paprasčiausiai tai galima išreikšti sakiniu

`SELECT AVG(`*Kategorija*`) AS "`*Kategorijų vidurkis*`" FROM` *Vykdytojai*.

Nesunku apskaičiuoti mintinai, kad vidurkis yra 2,8, tačiau šios užklausos rezultatas yra

Kategorijų vidurkis
2

Rezultate gavome sveikąjį skaičių, nes funkcijos `AVG` argumentas yra sveikojo skaičiaus tipo. Kad rezultate gauti trupmeninį skaičių, reikia argumentą *Kategorija* transformuoti į trupmeninį skaičių, pvz. `CAST(`*Kategorija* `AS FLOAT`). Tačiau taip gausime vidurkį, kuriame po dešimtainio taško bus daug skaitmenų. Žinoma, didelis tikslumas yra svarbu, bet, kalbant apie kategorijas, tikriausiai pakanka vieno ar kelių dešimtainės trupmenos skaitmenų. Apskaičiuotasis vidurkis bus pavaizduotas penkiais dešimtainiais skaitmenimis, iš kurių du po kablelio, jei vykdysime tokią užklausą

`SELECT CAST(AVG(CAST(`*Kategorija* `AS FLOAT)) AS DECIMAL(5,2))`
`AS "`*Kategorijų vidurkis*`" FROM` *Vykdytojai*.

Pastebėsime, kad specifikacija `CAST` galimas ne bet koks tipų transformavimas. Pavyzdžiui, negalima skaitinius duomenis transformuoti į dvejetainius ir datos bei laiko duomenis. Leistinų transformacijų rinkinį galima rasti konkrečios DBVS dokumentacijoje.

Daugelyje plačiai naudojamų DBVS tipų reikšmes taip pat galima transformuoti skaliarinėmis funkcijomis. Pvz., DBVS DB2 darbuotojų kategorijų vidurkį šimtųjų tikslumu galima apskaičiuoti tokia užklausa

```
SELECT DECIMAL(AVG(FLOAT(Kategorija)), 5, 2)) AS "Kategorijų vidurkis"
FROM Vykdytojai.
```

Kai standartinės DBVS galimybės vartotoją netenkina, SQL2 standartą užtikrinančiose sistemose vartotojas gali pats apibrėžti reikiamas tipų transformavimo funkcijas. Tokių funkcijų naudojimą aptarsime tolimesniuose skyreliuose.

6. Virtualiosios lentelės ir duomenų nepriklausomumo lygiai

6.9. Materializuotosios virtualios lentelės

Vykdant užklausas virtualiosioms lentelėms, apibrėžiančioji užklausa yra vykdoma kiekvieną kartą. Taip užtikrinama, kad per „langą“ visą laiką matytųsi užklausos metu pirminėse lentelėse esantys duomenys. Kai duomenys konkrečioje virtualiojoje lentelėje ieškomi pakankamai dažnai, tikslinga apibrėžiančiosios užklausos rezultatą įsiminti, kad panaudoti jį vykdant kitą užklausą tai pačiai virtualiajai lentelei. Įsimintas konkretus užklausos rezultatas yra vadinamas **materializuotąja virtualiąja lentele (MVL) (materializuotuoju rodiniu)** (angl. *materialized view, materialized query table*).

Jei paprastai virtualiajai lentelei tinka „lango“ įvaizdis, tai materializuotajai galima taikyti „fotografijos“ įvaizdį. Sukuriant MVL yra fiksuojamas ir įsimenamas momentinis užklausą atitinkančių duomenų „vaizdas“. Taip įsiminami MVL sukūrimo metu per „langą“ matomi duomenys. Į vėlesnes užklausas, formuluojamas panaudojant MVL, galima žiūrėti kaip į „fotografiją“ peržiūrą, t.y. paiešką duomenų, kurie buvo užfiksuoti sudarant MVL.

Užfiksuotas vaizdas dažniausiai tenkina vartotojų poreikius tol, kol jis morališkai nepasensta. Pasikeitus duomenims pirminėje lentelėje (lentelėse), duomenys materializuotoje lentelėje gali prarasti aktualumą, jei jie jau neatspindi tikrosios padėties. Taip atsitinka gyvenime, kai, praėjus konkrečiam laikui, reikia pakeisti asmens dokumentą, pvz. pasą, nes pagal jame esančią nuotrauką tampa sunku nustatyti asmens tapatybę. Duomenims pasenus, juos reikia atnaujinti.

Jei MVL duomenų nebūtų galima atnaujinti, t.y. MVL butų visiškai statiškos, tai jų panauda būtų labai ribota. Kad užtikrinti MVL gyvybingumą, pasikeitus pradinių lentelių duomenims, būtina atnaujinti MVL duomenis. Kitaip tariant, pasikeitus realiems objektams, reikia atnaujinti jų fotografijas. Šiuolaikinėse DBVS MVL duomenų atnaujinimui yra numatytos pakankamai lanksčios strategijos.

DBVS IBM DB2 MVL sudaromos sakiniu `CREATE TABLE`. MVL duomenys apibrėžiami užklausa, t.y. taip pat, kaip paprastos virtualiosios lentelės. Šiame skyriuje jau naudota virtualioji lentelė apie visų projektų vykdymą yra pakankamai gera kandidatė tapti materializuotąja. Sukurkime MVL su panašiais statistiniais duomenimis apie visus vykdytojus:

```
CREATE TABLE ApieVykdytojus(Vykdytojas, VisosValandos, Projektai)
    AS SELECT Vykdytojas, SUM(Valandos), COUNT(*)
       FROM Vykdymas GROUP BY Vykdytojas
    DATA INITIALLY DEFERRED
    REFRESH IMMEDIATE.
```

Lyginant šį lentelės apibrėžimą su atitinkamu virtualiosios lentelės (`VIEW`) apibrėžimu, galima nesunkiai pastebėti du papildomus parametrus: `DATA INITIALLY DEFERRED` ir `REFRESH IMMEDIATE`. Pirmuoju jų (`DATA INITIALLY`) apibrėžiama pradinio duomenų užpildymo strategija. Kadangi MVL gali turėti labai daug duomenų, tai duomenų įkėlimas į jas gali užtrukti pakankamai ilgai. Todėl duomenų įkėlimas papildomu raktiniu žodžiu `DEFERRED` atidedamas vėlesniam laikui. Duomenims įkelti į MVL naudojamas kitas SQL sakinys `REFRESH TABLE`.

Sudarant MVL, fraze `REFRESH` apibrėžiama MVL duomenų atnaujinimo strategija. Dažniausiai, MVL duomenis reikalinga atnaujinti kiekviena kartą, kai atnaujinami pradinės lentelės (lentelių) duomenys. Jeigu MVL yra naudojama labai intensyviai ir yra labai svarbu, kad joje esantys duomenys visada tiksliai atspindėtų tikruosius duomenis, duomenų atnaujinimą galima patikėti sistemai. Fraze `REFRESH IMMEDIATE` DB valdymo sistemai nurodoma MVL duomenis atnaujinti iš karto, kai pasikeičia pradinėse lentelėse esantys duomenys. Taip kiekvienas pradinės lentelės atžvilgiu įvykdytas sakinys `INSERT`, `UPDATE` ar `DELETE`

automatiškai iššaukia MVL duomenų atnaujinimą. Tai ne visada yra patogu. Jei MVL yra naudojama santykinai retai ar nėra labai svarbu, kad peržiūrimi statistiniai duomenys visada tiksliai atitiktų esamą akimirką, MVL atnaujinimą galima tvarkyti patiems sistemos vartotojams jų pasirinktu metu. Vietoje frazės `REFRESH IMMEDIATE` panaudojus frazę `REFRESH DEFERRED`, MVL duomenys bus atnaujinami tik tuomet, kai vartotojas įvykdys duomenų atnaujinimo sakinį

`REFRESH TABLE` <MVL vardas>.

Pradiniam duomenų įkėlimui šį sakinį būtina įvykdyti visoms MVL. Vėliau šį sakinį reikia kartoti tik toms MVL, už kurių atnaujinimą atsakingas yra pats sistemos vartotojas, t.y. jei lentelė buvo sukurta nurodant `REFRESH DEFERRED`. Šio sakinio vykdymo dažnis priklauso nuo konkrečių aplinkybių.

Duomenų įvedimas, atnaujinimas ir šalinimas materializuotose virtualiosiose lentelėse dažniausiai yra neprasmingas ir negalimas, nors kartais tai yra galima ir taikoma.

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`.

8. SQL sakiniai taikomosiose programose

8.6'. Statinių užklausų apdorojimas

Pakankamai dažnai programose prireikia vykdyti užklausas vienai rezultato eilutei išrinkti. Taip yra, pavyzdžiai, kai lentelėje ieškoma duomenų pagal lentelės raktą. Kai iš anksto žinome, kad užklausos rezultatą sudarys ne daugiau kaip viena eilutė, programose galima taikyti supaprastintą užklausą, kuri išreiškiama SQL sakiniu `SELECT INTO`

`SELECT` <stulpelių vardai> `INTO` <programos kintamieji> <`FROM` frazė>
[`WHERE` frazė] [`GROUP BY` frazė] [`ORDER BY` frazė] [`FETCH FIRST ROW ONLY`].

Vykdytojo, kurio numeris jau yra priskirtas programos kintamajam *nr*, pavardę ir kategoriją galima sužinoti taip:

```
EXEC SQL SELECT Pavardė, Kategorija INTO :name, :category :ind
         FROM Vykdytojai WHERE Nr = :nr;
if (0 > SQLCODE)
  printf("Įvyko klaida, jos kodas: %ld\n", SQLCODE) ;
else if (100 == SQLCODE)
  printf("Vykdytojas Nr.: %d yra nežinomas\n", nr) ;
else if (0 == SQLCODE)
  printf("Nr.: %d, pavardė: %s, kategorija: %d\n", nr, name, (ind < 0 ? "-" : category)) ;
```

Jei, įvykdžius sakinį `SELECT INTO`, užklausos rezultatą sudaro daugiau negu 1 eilutė, tai DBVS nepateikia išrinktų duomenų ir praneša apie klaidą. Kartais užklausoje suformuluotus reikalavimus gali tenkinti kelios eilutės, bet mus domina tik pirmoji. Tuomet fraze `FETCH FIRST ROW ONLY` galima nurodyti sistemai pateikti tik pirmąją eilutę, nepriklausomai nuo to, kiek eilučių tenkina paieškos sąlygą.

8.10. Dinaminių užklausų vykdymas

Sudarysime žymiai detalesnę C funkciją, turinčią tik vieną parametrą – simbolių eilutę, kuria pateikiama užklausa vykdymui. Atliekant šią funkciją, užklausa yra įvykdoma ir jos rezultatas išvedamas į standartinį išvedimo įrenginį.

```
void SelectUsingDescribe(char *selectStmt)
{
  EXEC SQL BEGIN DECLARE SECTION;
    char *strStmt;          /* - kintamasis SELECT sakiniui */
  EXEC SQL END DECLARE SECTION;
  EXEC SQL WHENEVER SQLERROR      GOTO error;
  EXEC SQL WHENEVER NOT FOUND     GOTO end;
  struct sqlda *pSqlda = NULL;
  int nofColumns ;
  strStmt = selectStmt ;
  EXEC SQL PREPARE stmt FROM :strStmt;
  /* Ruošiame atmintį stulpelių skaičiui */
  SqldaInit(&pSqlda, 1);
  /* Sužinome stulpelių skaičių */
  EXEC SQL DESCRIBE stmt INTO :*pSqlda;
  nofColumns = (int) pSqlda->sqld;
  free(pSqlda);
  /* Ruošiame atmintį duomenims apie visus stulpelius */
  SqldaInit(&pSqlda, nofColumns);
  /* Sužinome visų stulpelių aprašus */
```

```
  EXEC SQL DESCRIBE stmt INTO :*pSqlda;
  EXEC SQL DECLARE curs CURSOR FOR stmt;
  EXEC SQL OPEN curs;
  /* Ruošiame atmintį visų stulpelių reikšmėms */
  RowDataMemoryAlloc(pSqlda);
  while( 0 == SQLCODE ) {
    EXEC SQL FETCH c2 USING DESCRIPTOR :*pSqlda;
    RowDataDisplay(pSqlda);
  }
  EXEC SQL WHENEVER SQLERROR     CONTINUE;
  EXEC SQL WHENEVER NOT FOUND    CONTINUE;
  error:
    printf("SQL klaida: %ld\n", SQLCODE) ;
  end:
    EXEC SQL CLOSE curs ;
}
/************************************************/
void SqldaInit(struct sqlda **ppSqlda, int nofColumns)
{
  int memSize = offsetof(struct sqlda, sqlvar)+
                        nofColumns * sizeof(struct sqlvar) ;
  *ppSqlda = (struct sqlda *) malloc(memSize);
  memset(*ppSqlda, '\0', memSize ) ;
  memcpy((*ppSqlda)->sqldaid, "SQLDA   ", 8);
  (*ppSqlda)->sqldabc = memSize ;
  (*ppSqlda)->sqln = nofColumns;
  (*ppSqlda)->sqld = 0;
}
/************************************************/
void RowDataMemoryAlloc(struct sqlda *pSqlda)
{
  short iCol;
  unsigned int memSize;
  for( iCol = 0; iCol < pSqlda->sqld; iCol++) {
    /* Išskiriame atmintį indikatoriui */
    pSqlda->sqlvar[iCol].sqlind = (short *) malloc(sizeof(short));
    memset(pSqlda->sqlvar[iCol].sqlind, '\0', sizeof(short));
    /* Išskiriame atmintį reikšmei */
    switch (pSqlda->sqlvar[iCol].sqltype) {
      case SQL_TYP_DATE:
      case SQL_TYP_TIME:
      case SQL_TYP_STAMP:
      case SQL_TYP_VARCHAR:
      case SQL_TYP_CHAR:
      case SQL_TYP_FLOAT:
      case SQL_TYP_SMALL:
        memSize = pSqlda->sqlvar[iCol].sqllen + 1;
        break;
      /* Išskiriame atmintį kitų, sudėtingesnių tipų reikšmėms */
      case SQL_TYP_DECIMAL:
      ...
    }
    pSqlda->sqlvar[iCol].sqldata = (char *) malloc(memSize);
```

```
      memset(pSqlda->sqlvar[iCol].sqldata, '\0', memSize);
   }
}
/************************************************/
void RowDataDisplay(struct sqlda *pSqlda)
{
   short iCol;
   for( iCol = 0;  iCol < pSqlda->sqld;  iCol++ )
     CellDataDisplay(&pSqlda->sqlvar[iCol]) ;
   printf("\n");   /* - eilutės pabaiga */
}
/************************************************/
void CellDataDisplay( struct sqlvar *pSqlvar )
{
   printf("%s:", pSqlvar->sqlname.data);  /* - stulpelio vardas */
   if(pSqlvar->sqlind < 0)
     printf("-");      /* - NULL reikšmė */
   else {
      switch (pSqlvar->sqltype) {
        case SQL_TYP_DATE:
        case SQL_TYP_TIME:
        case SQL_TYP_STAMP:
        case SQL_TYP_CHAR:
        case SQL_TYP_VARCHAR:
          printf("%s", pSqlvar->sqldata);
          break;
        case SQL_TYP_SMALL:
          printf("%d", *((short *)pSqlvar->sqldata));
          break;
        case SQL_TYP_FLOAT:
           printf("%f", *((double *)pSqlvar->sqldata));
          break;
        /* Kitų tipų reikšmių išvedimas */
        ...
      }
   }
}
/************************************************************/
void RowDataMemoryFree(struct sqlda *pSqlda)
{
   short  iCol;
   for( iCol = 0; 0 != pSqlda && iCol < pSqlda->sqld;  iCol++ ) {
      if( 0 != pSqlda->sqlvar[iCol].sqldata )
         free(pSqlda->sqlvar[iCol].sqldata);
      if( 0 != pSqlda->sqlvar[iCol].sqlind )
         free(pSqlda->sqlvar[iCol].sqlind);
   }
   if( 0 != pSqlda )
      free(pSqlda);
}
```

Taupydami vietą, dviejų funkcijų *RowDataMemoryAlloc* ir *CellDataDisplay* kamienus pateikėme nepakankamai detaliai. Šių funkcijų kamienuose, vietoje daugtaškių, dar reikia

detaliai užrašyti atminties išskyrimą rečiau naudojamų duomenų tipų reikšmėms (funkcijoje *RowDataMemoryAlloc*) ir tokių tipų reikšmių išvedimą (funkcijoje *CellDataDisplay*).

8.11. Sąsaja JDBC

Taikomųjų programų sąsaja pateikia programuotojams funkcijas-paprogrames, kuriuos kviečiant užtikrinamas SQL sakinių vykdymas. Sąsaja JDBC (angl. *Java Database Connectivity*) yra taikoma JAVA programavimo kalba sudaromose programose.

Jau aptarėme, kad, taikant taikomųjų programų sąsają, programos nereikia prekompiliuoti. Visi SQL sakiniai, kuriais kreipiamasi į DBVS reikiamų paslaugų, vykdant tokias programas paruošiami programos vykdymo metu, t.y. dinamiškai. Todėl programų sąsaja yra mažiau efektyvi už statinius programų SQL sakinius. Tačiau dėl patogaus naudojimo ir nereikalingo prekompiliavimo, programų sąsajos naudojamos pakankamai plačiai. Aptarsime pagrindinius JDBC bruožus.

8.11.1. Ryšys su DB

Ryšys tarp programos ir duomenų bazės nustatomas dviem etapais:

- Įkeliama konkrečiai DBVS būdinga tvarkyklė. Tvarkyklė užtikrina tolimesnių JDBC paslaugų nepriklausomumą nuo naudojamos DBVS ypatumų. IBM DB2 tvarkyklę galima aktyvuoti taip

  ```
  Class.forName( "com.ibm.db2.jdbc.app.DB2Driver" ).newInstance();
  ```

- Ryšys su konkrečia DB nustatomas sukuriant klasės `Connection` objektą, pvz.,

  ```
  Connection con = DriverManager.getConnection( "jdbc:db2:Darbai",
                                                username, password ) ;
  ```

Pirmuoju metodo `getConnection` parametru nurodoma duomenų bazės URL (angl. *Unified Resource Location*), kurį sudaro: protokolas (jdbc), DBVS (db2) ir DB vardas (*Darbai*). Kituose dviejuose parametruose (programos kintamuosiuose *username* ir *password*) nurodoma sistemos vartotojo vardas ir jo slaptažodis.

8.11.2. Paprastų SQL vykdymas

SQL sakinių vykdymui sąsajoje JDBC yra numatyta objektų klasė `Statement`. Sukuriant šios klasės objektą, ryšys su konkrečia DB jau turi būti nustatytas. Turint `Connection` klasės objektą *con*, klasės `Statement` objektą galima sukurti kviečiant metodą `createStatement`

```
Statement stmt = con.createStatement() ;
```

Konkretų klasės `Statement` objektą galima naudoti daugeliui SQL sakinių vykdyti. Paprasčiausiai vykdomi DDL sakiniai ir lentelių duomenų atnaujinimo sakiniai (`INSERT`, `DELETE`, `UPDATE`). Šie sakiniai vykdomi kviečiant klasės `Statement` objektui metodą `executeUpdate`. Kviečiant šį metodą parametru pateikiama simbolių eilutė – SQL sakinys, pvz.,

```
Statement stmt = con.createStatement();
stmt.executeUpdate( "INSERT INTO Vykdytojai " +
                    "VALUES (6, 'Baltakis', 'Informatikas', 2, NULL)" ) ;
String str = "UPDATE Vykdytojai SET Kategorija = Kategorija + 1" ;
stmt.executeUpdate(str) ;
```

Kad tą patį SQL sakinį būtų galima efektyviai atlikti keletą kartų, gal net keičiant parametrų reikšmes, iš pradžių, sakinį galima paruošti, o po to jį pakartotinai vykdyti su reikiamomis parametrų reikšmėmis. Ruošiamuose sakiniuose parametrai nurodomi klaustukais, pvz.,

```
PreparedStatement stmt = con.prepareStatement(
```

```
                "UPDATE Vykdytojai SET Kategorija = ? WHERE Pavardė = ?" ) ;
stmt.setInt(1, 5);
stmt.setString(2, "Jonaitis");
stmt.executeUpdate() ;  // - atnaujinam kategoriją Jonaičiui
stmt.setInt(1, 4);
stmt.setString(1, "Petraitis");
stmt.executeUpdate() ;  // - atnaujinam kategoriją Petraičiui
```

Čia metodais `setInt` ir `setString` yra priskiriamos reikšmės parametrams. Parenkant metodą reikšmei priskirti reikia atsižvelgti į priskiriamosios reikšmės tipą.

Transakcijos yra valdomos `Connection` objekto metodais. JDBC numatyta, kad po kiekvieno SQL sakinio automatiškai užbaigiama transakcija, įtvirtinant atliktus pakeitimus. Kad užtikrinti kelių SQL sakinių transakcijas, reikia atšaukti numatytąjį transakcijų valdymą. Automatinis transakcijų valdymas išjungiamas ir įjungiamas metodu `setAutoCommit`, kurį kviečiant reikia nurodyti vieną parametrą - `Boolean` tipo reikšmę. SQL `COMMIT` sakinį atitinka `commit` metodas, o `ROLLBACK` – `rollback`.

8.11.3. Klaidų apdorojimas

Informacijai apie klaidas, atsirandančias programos vykdymo metu, perteikti yra numatytos specialios klasės: `SQLException` ir `SQLWarning`. Klaidos apdorojamos objektinei programavimo kalbai įprastu būdu – „gaudant“ klaidų įvykius:

```
try {
   con.setAutoCommit(false); // - atšaukiamas automatinis pakeitimų įtvirtinimas
   stmt.executeUpdate(<SQL sakinys 1>);
   …
   stmt.executeUpdate(<SQL sakinys n>);
   con.commit() ;
   con.setAutoCommit(true) ;
}
catch(SQLException ex) {
   System.err.println("SQLException: " + ex.getMessage()) ;
   con.rollback() ;
   con.setAutoCommit(true) ;
}
```

8.11.4. Užklausų apdorojimas

Užklausos vykdomos kviečiant klasės `Statement` objektui metodą `executeQuery`. Šis metodas rezultate grąžina – klasės `ResultSet` objektą. Ši klasė užtikrina rezultato eilučių peržiūrą metodu `next`. Kviečiant `next`, vidinis objekto žymuo kiekvieną kartą paslenkamas vis prie kitos eilutės. Metodais, atitinkančiais konkrečios užklausos rezultato stulpelių tipus, galima gauti žymimosios eilutės reikšmes. Sudarysime programos fragmentą visų darbuotojų vardams ir jų kategorijoms išvesti.

```
String name;
int category;
ResultSet rs = stmt.executeQuery(
                        "SELECT Pavardė, Kategorija FROM Vykdytojai" );
while( rs.next() ) {
   name     = rs.getString("Pavardė");
   category = rs.getInt("Kategorija");
   System.out.println(name + " kategorija: " + (rs.wasNull() ? "-" : category) );
}
```

Šiame pavyzdyje užklausos rezultato reikšmėms paimti metoduose `getString` ir `getInt` naudojame stulpelių vardus (*Pavardė* ir *Kategorija*). Reikšmių gavimo stulpelių pavadinimus galima pakeisti jų eilės numeriais:

```
name     = rs.getString(1);
category = rs.getInt(2);
```

Metodu `wasNull` galima sužinoti, ar rezultato reikšmė, į kurią buvo kreiptasi prieš pat kviečiant šį metodą, buvo `NULL` reikšmė.

Patogiam užklausos rezultato perrinkimui yra numatyta pakankamai daug metodų, kurių vardai atspindi jų prasmę: `getRow`, `isFirst`, `isBeforeFirst`, `isLast`, `isAfterLast`, `absolute`, `previous`, `relative` ir kt., pvz.,

```
rs.absolute(3);      // šokti prie trečios eilutės
rs.previous();       // sugrįžti vieną eilutę atgal
rs.relative(2);      // pirmyn per dvi eilutes (prie 4-os)
rs.relative(-3);     // atgal per tris eilutes (prie 1-os)
```

9. Sisteminiai duomenų bazių aspektai

9.3'. Bendro duomenų naudojimo problemos

Nesuderintų duomenų analizės problema. Tarkime, transakcija *A* sumuoja valandas, kurias vykdytojai skiria projektui Nr. 3, o transakcija *B* dalį (50 val.) vykdytojo Nr. 3 skiriamų valandų paskiria vykdytojui Nr. 2. Prisimindami duomenis, pateiktus skyrelyje 2.1, nesunkiai apskaičiuojame, kad projektui Nr. 3 iš viso yra skiriama 250+400+150 = 800 val. Lentelės *Vykdymas* eilutę, kurioje yra duomenys apie vykdytojo Nr. *i* „indėlį“ į projekto Nr. 3 vykdymą pažymėkime R_i, i = 1, 2, 3. Transakcijos *A* kintamąjį, kuriame kaupiama suma, pažymėkime *suma*. Po atliekamo duomenų atnaujinimo ši suma turi išlikti nepakitus. Kai nesirūpinama dviejų transakcijų tarpusavio sąveika, gali susidaryti nesuderintų duomenų analizės problema.

Transakcija *A*	Laikas	Transakcija *B*
-	t_1	-
FETCH R_1 *suma* = 250		
-	t_2	-
FETCH R_2 *suma* = 650		
-	t_3	FETCH R_3
-	t_4	-
-		UPDATE R_3 150 →100
-	t_5	-
-		FETCH R_1
-	t_6	-
-		UPDATE R_1 250 →300
-	t_7	-
-		COMMIT
-	t_8	-
FETCH R_3 *suma* = 750		

Transakcijos *A* pabaigoje (laiko momentu t_8) gautas rezultatas (750) yra neteisingas. Atlikus visus veiksmus, valandų suma turėtų nepasikeisti: 300+400+100 = 800 val. Tokiu atveju yra sakoma, kad transakcijos *A* susidūrė su nesuderintais duomenimis ir atliko nesuderintų duomenų analizę. Šiuo konkrečiu atveju duomenų analize buvo – paprastas reikšmių sumavimas.

9.4'. Bendro duomenų naudojimo problemų sprendimas

Veikiant duomenų blokavimui **nesuderintų duomenų analizės** padėtyje neteisinga suma negaunama. Tiksliau, negaunama jokia suma, nes transakcijos patenka į aklavietę:

Transakcija *A*	Laikas	Transakcija *B*
-	|	-
`FETCH` R_1 (prašoma ir R_1 užblokuojama `S` blokuote) *suma* = 250	t_1	-
-	|	-
`FETCH` R_2 (prašoma ir R_2 užblokuojama `S` blokuote) *suma* = 650	t_2	-
-	t_3	`FETCH` R_3 (prašoma ir R_3 užblokuojama `S` blokuote)
-	|	-
-	t_4	`UPDATE` R_3 (prašoma ir R_3 užblokuojama `X` blokuote) 150 →100
-	|	-
-	t_5	`FETCH` R_1 (prašoma ir R_1 užblokuojama `S` blokuote)
-	|	-
-	t_6	`UPDATE` R_1 (prašoma R_1 užblokuoti `X` blokuote) Laukiama
-	|	Laukiama
`FETCH` R_3 (prašoma R_3 užblokuoti `X` blokuote) Laukiama	t_7	Laukiama
Laukiama	|	Laukiama