1. Įvadas

1.1. Duomenų bazės, duomenų bazių valdymo sistemos

Duomenų baze (DB) galima vadinti rinkinį tarpusavyje susijusių duomenų, kurie apdorojami programomis. Tokia duomenų bazė, loginiu požiūru, yra panaši į elektroninę kartoteką. Studentų kartotekos pavyzdys lentele pateiktas 1.1 pav.

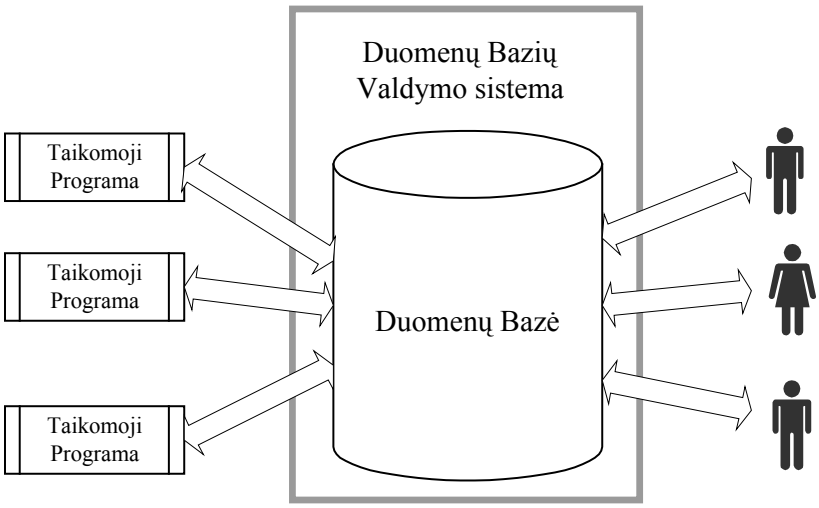
Vardas	Pavardė	Gimimo data	Studijų programa	Pakopa	Kursas
Jonas	Baltakis	1983.03.05	Informatika	Bakalaurantas	3
Ona	Gražulytė	1983.04.15	Informatika	Magistrantas	1

1.1 pav. Studentų duomenų bazė

Panašios duomenų bazės (kartotekos) vartotojai atlieka tokias operacijas:

- naujo įrašo (naujos kortelės) įtraukimas;
- esamo įrašo (kortelės duomenų) keitimas;
- esamo įrašo (kortelės) šalinimas;
- duomenų paieška duomenų bazėje (kartotekoje).

Minėtom ir panašiom operacijoms atlikti, kai duomenys saugomi kompiuterio atmintyje, reikia programinės įrangos. Tokia programinė įranga kartu su duomenų baze vadinama **duomenų bazės sistema** (DBS). Dažnai, kalbant apie kompiuterines sistemas, pastarosioms priskiriami ir jų vartotojai. DB vartotojais laikomi asmenys, betarpiškai operuojantys duomenų baze, bei taikomosios programos, kurios vykdymo metu kreipiasi į DB. Pagrindinė tokios programinės įrangos paskirtis yra suteikti DB vartotojui galimybę dirbti su DB, neatsižvelgiant į technines detales. Programinė įranga, sprendžianti tokį uždavinį vadinama **duomenų bazių valdymo sistema** (DBVS). Supaprastinta DBS pavaizduota 1.2 pav. Kitaip tariant, DBVS leidžia vartotojui žiūrėti į DB kaip į žymiai aukštesnio lygio objektą, negu į įrašų failą. Šiuo požiūriu, DBVS atlieka vartotojo sąsajos su DB vaidmenį. Vartotojas operuoja DB duomenimis formalia kalba, pvz. SQL.



1.2 pav. Supaprastinta duomenų bazės sistema

1.2. Duomenų bazių valdymo sistemų funkcijos

Be jau minėtos pagrindinės DBVS funkcijos, kuri leidžia vartotojui dirbti su DB naudojant logines sąvokas, DBVS atlieka ir kitas svarbias funkcijas. DBVS sprendžia ne tik vartotojo nepriklausomumo nuo fizinių DB realizavimo ypatybių problemą, bet ir kitus uždavinius. Tuo pačiu, galime kalbėti ir apie reikalavimus DB valdymo sistemoms. Aptarsime bene pagrindinius iš jų.

Naudojant popierinę duomenų saugojimo technologiją, norint tuos pačius duomenis vartoti sprendžiant skirtingus uždavinius, dažnai tenka dubliuoti duomenis. Pvz., studentų skyrius naudoja "savus" studentų sąrašus, o studentų atstovybė ar buhalterija "savus". Panaši situacija yra ir saugant bei apdorojant duomenis, esančius failuose. Dažna taikomoji programa, dirbdama su failais, naudoja savas kopijas ar net savus duomenų formatus. Tokia padėtis nepageidautina. DBVS keliamas uždavinys **minimizuoti duomenų perteklių**. Dažnai nepavyksta visiškai išvengti duomenų dubliavimo, bet minimizuoti jų perteklių yra būtina.

Yra žinoma, kad atidarius duomenų failą rašymui, operacijų sistema kitiems vartotojams neleidžia ne tik į jį rašyti, bet ir skaityti iš jo tol, kol rašymo operacija nebus baigta ir failas nebus uždarytas. DB yra skiriama plačiam vartotojų ratui, todėl DBVS keliamas uždavinys užtikrinti **efektyvų bendrą DB vartojimą**. DBVS pasiekia tai lanksčiai naudodama duomenų blokavimą (izoliavimą). Pagal vieną iš pagrindinių naudojamos strategijos idėjų, vienam vartotojui keičiant DB įrašą, kitiems vartotojams tuo pačiu metu leidžiama peržiūrėti ar net keisti kitus įrašus.

DB vadinama **vientisa** (integralia), jei ji tenkina tam tikrus apribojimus (sąlygas) duomenims ir išsaugo tuos apribojimus modifikuojant (keičiant, šalinant, įterpiant) duomenis. Tokio apribojimo pavyzdžiu gali būti "studento bakalauranto kursas negali būti didesnis už 4, o magistranto – už 2". DBVS, kuri užtikrina DB integralumą, atliekant kiekvieną DB modifikavimą, privalo tikrinti, ar duomenų pakeitimas nepažeis nustatytų apribojimų. Jei vartotojas bando modifikuoti DB taip, kad po to kuri nors sąlyga taptų nepatenkinta, tai DBVS privalo neatlikti tokios operacijos ir informuoti apie tai vartotoją.

Kita labai svarbi DB savybė yra **duomenų neprieštaringumas**. Tarkime, kuriam nors studentui studentų bazėje yra du įrašai, kurių viename nurodyta, kad jis studijuoja pirmame kurse, o kitame, kad - antrame. Tokiu atveju, vartotojui užklausus iš bazės duomenų apie to studento kursą, užklausos rezultatas priklausys nuo to, iš kurio įrašo duomuo bus paimtas. Aišku, tokia situacija yra neleistina. DB valdymo sistemai keliamas uždavinys užtikrinti, kad panašių prieštarų būtų išvengta. Minėta prieštara neatsiras, pavyzdžiui, jei studentų duomenų bazėje kiekvienam studentui visuomet bus skiriama tik viena eilutė.

Nemažiau svarbu apsaugoti duomenis, esančius duomenų bazėje, nuo tyčinio ar netyčinio svarbių duomenų sunaikinimo (pašalinimo) ar pakeitimo. Duomenų bazės **saugumo** sąvoka apima ir apsaugą nuo neleistinos duomenų peržiūros. Šiam tikslui pasiekti, DBVS atsakingajam DB vartotojui (administratoriui) pateikia priemones, kurios leidžia apibrėžti kiekvienam vartotojui ar jų grupei teises duomenų atžvilgiu. DBVS, turėdama teisių sąrašą, aptarnaujant vartotojų užklausas, stebi, kad niekas neviršytų jam priskirtų teisių.

Saugant duomenis failuose, įmanoma sukurti programas, kurios, atsižvelgdamos į duomenų formatą, galės labai efektyviai atlikti paiešką pagal iš anksto numatytus kriterijus net ir labai dideliuose failuose. Tačiau, dažnai negalima iš anksto numatyti visus galimus paieškos kriterijus ir, atsiradus poreikiui, tenka ieškoti duomenų pagal naują kriterijų ar jų kombinaciją. DBVS keliamas uždavinys dideliuose duomenų masyvuose efektyviai atlikti ne tik **planuotas** (pagal iš anksto numatytus kriterijus), bet ir **neplanuotas užklausas**.

Aptarėme ne visas funkcijos. Minėtas ir kitas DBVS funkcijas (reikalavimus, uždavinius) plačiau aptarsime kituose skyreliuose. Nagrinėsime ir tai, kokiomis priemonėmis DBVS keliamus jai uždavinius sprendžia.

1.3. Reliacinės ir kitokios DB

Viena iš pagrindinių DBVS funkcijų yra sąsaja tarp vartotojo ir DB. DBVS suteikia vartotojui galimybę kreiptis į duomenų bazę ne fizinėmis (aparatūros ar operacijų sistemos) sąvokomis, bet loginėmis. Sąvokų rinkinys ir jų vartojimo taisyklės sudaro **modelį**. Dauguma DBVS, sukurtų pradedant 70-jų pabaiga, vartotojo sąsajai naudoja **reliacinį duomenų modelį**. DB, kurios apibrėžiamos naudojant reliacinį modelį, vadinamos **reliacinėmis duomenų bazėmis** (RDB), o atitinkamos jų valdymo sistemos - **reliacinėmis duomenų bazių valdymo sistemomis** (RDBVS).

Reliacinį modelį 1970 m. pasiūlė E.F. Kodas (E.F. Codd), tuomet dirbęs IBM tyrimų laboratorijoje. Pateiksime supaprastintą reliacinio modelio aprašą. Formaliau reliacinį modelį išdėstysime tolimesniuose skyreliuose.

Pagrindinė reliacinio modelio sąvoka yra **lentelė**. Reliacinėse sistemose lentelę sudaro horizontalios **eilutės** (angl. *row*) ir vertikalūs **stulpeliai** (angl. *column*). **Reliacine sistema** yra vadinama sistema, besiremianti šiais pagrindiniais principais:

- visi duomenys vartotojui pateikiami lentelėmis;
- vartotojui pateikiami operatoriai generuojantys naujas lenteles iš senų. Pavyzdžiui, pateikiamas operatorius, leidžiantis gauti turimos lentelės eilučių poaibį, bei operatorius, įgalinantis gauti lentelės stulpelių poaibį.

Tokios sistemos vadinamos reliacinėmis, nes angliškas žodis *relation* (santykis) yra matematinis lentelės pavadinimas. Daugeliu atžvilgių, terminai "lentelė" ir "santykis" yra sinonimai. Literatūroje vietoje termino "eilutė", galima sutikti terminus "kortežas" (angl. *tuple*) ir "įrašas" (angl. *record*), o vietoje termino "stulpelis" – "atributas" (angl. *attribute*) ir "laukas" (angl. *field*). Taigi, reliacinė DB - tai duomenų bazė, kurią vartotojai priima kaip lentelių rinkinį. Kitaip tariant, lentelių rinkinys sudaro duomenų bazę.

Kiekviena duomenų bazės lentelė turi savo vardą (pavadinimą). Kiekviena lentelės eilutė aprašo vieną objektą – žmogų, firmą, sandėrį ar pan. Kiekvienas stulpelis aprašo vieną objekto charakteristiką – žmogaus pavardę, vardą, adresą ir pan., firmos telefono numerį, sandėrio numerį, datą ir pan. Duomenų elementas arba reikšmė (angl. *value*), nusakoma lentelės eilutės ir stulpelio susikirtimu. Stulpeliui yra priskiriama galimų reikšmių aibė, vadinama **domenu** (angl. *domain*).

Ne reliacinėse sistemose duomenys vartotojui pateikiami kitokiomis struktūromis nei reliacinės sistemos lentelės. Pavyzdžiui, **hierarchinėse** sistemose duomenys vaizduojami medžio struktūromis, o tarp operacijų yra judėjimo hierachinėmis struktūromis į viršų ir žemyn operacijos. **Tinklinės sistemos**, didele dalimi, yra hierarchinių sistemų apibendrinimas. Duomenys tinklinėse sistemose vaizduojamos orientuotais grafais. Tiek hierachinės tiek ir tinklinės sistemos priskiriamos ikireliacinėms sistemoms. Šių duomenų modelių dominavimą panaikino reliacinių sistemų atsiradimas. Tačiau ir šiandien tinklinės sistemos efektyviai taikomos specifiniams uždaviniams spręsti, pvz. geografijos informacinėms sistemoms realizuoti.

Pirmosios reliacinės sistemos atsirado 1970-jų pabaigoje – 1980-jų pradžioje. Tarp šiuo metu labiausiai paplitusių ir labiausiai išvystytų RDBVS galima paminėti firmos IBM sukurtą DBVS DB2, korporacijos Oracle produktą ORACLE, Sybase Inc. produktą SYBASE, firmos Microsoft DBVS SQL Server ir kt.

Nors reliacinis modelis pasirodė labai patogus vartojimui, tačiau jis nėra idealus. Pastaraisiais dešimtmečiais buvo gana aktyviai ieškoma naujų duomenų modelių. Paminėtina **deduktyvios, ekspertinės** ir **objektinės** duomenų bazių valdymo sistemos. Ypač aktyviai domimasi pastarosiomis. Objektinės DBVS pastaraisiais metais plinta žymiai sparčiau negu reliacinės. Tačiau reliacinės sistemos ir toliau žymiai plačiau vartojamos negu visos kitos kartu paėmus, tiek Lietuvoje, tiek ir visame pasaulyje. Dėl šios priežasties, tolimesniuose skyreliuose mes nagrinėsime tik reliacines sistemas. Pažyminį reliacinė (-inis) dažnai praleisime.

1.4. SQL

Vartotojo sąsajai organizuoti (užklausoms formuluoti) yra vartojama tam tikra formalizuota kalba. Daugumoje reliacinių sistemų yra vartojama vienas iš kalbos **SQL** (Structured Query Language) dialektų. Ši kalba buvo sukurta firmos IBM tyrimų centre 1970-jų metų pabaigoje. Praktiškai nuo pačios pirmosios firmos IBM sukurtos reliacinės sistemos System R SQL kalba tapo Amerikos nacionaliniu (ANSI) bei tarptautiniu (ISO) standartais.

Didele dalimi, SQL kalba vartojama reliacinėms operacijoms aprašyti. Šioje kalboje išskiriamos trys sakinių grupės: duomenų apibrėžimo sakiniai, kitaip dar vadinami duomenų apibrėžimo kalba (DDL, Data Definition Language); manipuliavimo duomenimis sakiniai (DML, Data Manipulation Language) ir duomenų valdymo sakiniai (DCL, Data Control Language), t.y. SQL = DDL + DML + DCL. **DDL** sakiniais, tarp kitų veiksmų, yra kuriamos duomenų bazės ir lentelės. Kuriant naują DB yra nurodomas jos vardas (pavadinimas), fizinė vieta ir kai kurios kitos savybės. Kuriant (apibrėžiant) lentelę, privaloma nurodyti lentelės vardą, jos stulpelių vardus ir stulpelių tipus. Stulpelio tipu nusakoma, kokios rūšies duomenys galės būti įrašomi stulpelyje. SQL leidžiamos tokios duomenų rūšys: tekstiniai duomenys, skaičiai, dvejetainiai duomenys, datos ir laikai. Kiekviena duomenų rūšis su tam tikromis fizinėmis realizacijos savybėmis (pvz., baitų kiekis, skiriamas reikšmės vaizdavimui) nusakoma **duomenų tipu**. Duomenų tipo sąvoka yra reliacinės teorijos domeno sąvokos atitikmuo. SQL duomenų tipus plačiau aptarsime vėliau.

DML sakiniais formuluojamos užklausos (angl. *query*) duomenims surasti, keisti, šalinti ir įterpti. **DCL** sakiniais koordinuojamas bendras efektyvus vartotojų darbas su DB. SQL - tai kalba, kuria galima bendrauti su reliacinėmis duomenų bazėmis.

Dažnai realiame informacijos valdyme duomenys būna nežinomi arba nepilni: studentas gali nesakyti mums, kaip jis išlaikė egzaminą, kiek per vasaros atostogas užsidirbo pinigų; pildant anketą galima nepildyti amžiaus ir šeimyninės padėties grafų. Įvedant informaciją į kompiuterį, į tai atsižvelgiama. Kartais tam naudojama sąvoka “nulis”. Tačiau, tai nevisada padeda, nes nežinojimas nevisada reiškia nulį. Tai, kad nesužinojome, kiek draugas užsidirbo, nereiškia, kad jis užsidirbo nulį litų. Atsižvelgiant į fakto apie duomenų nebuvimą svarbą, SQL kalboje kiekviename duomenų tipe yra speciali reikšmė `NULL`. Šia reikšme žymima, kad duomenys yra nežinomi, šiuo metu dar neįvesti, jau pašalinti ir pan. Tai lyg reikšmė, reiškianti reikšmės nebuvimą.

Formuluojant užklausas SQL kalba, reikia išreikšti tai, kas norima gauti, nesileidžiant į detales, kaip rezultatą gauti. DBVS uždavinys – nustatyti kaip, kokiu būdu, kokius algoritmus panaudoti rezultatui gauti. Todėl, SQL gali būti priskirta neprocedūrinėms (deklaratyvioms) programavimo kalboms. Kita ypatybė, kuri daro SQL kalbą netradicine programavimo kalba, yra SQL sakinių vartojimas programuojant programavimo kalbomis. SQL sakiniai gali būti įterpiami į programas, sudaromas bazine programavimo kalba. Bazinėmis programavimo kalbomis SQL kalbai gali būti daugelis plačiai vartojamų programavimo kalbų: C/C++, PL1, COBOL, FORTRAN ir pan. Kita vertus, SQL sakiniai gali būti vykdomi ir interaktyviai.

Pirmasis tarptautinis SQL kalbos standartas buvo priimtas 1989 m., kuris vadinamas SQL-89 arba SQL1. Dauguma šiuolaikinių RDBVS šį SQL standartą pilnai atitinka. Besivystančios informacinės technologijos įtakojo ir SQL vystymąsi. 1992 m. buvo priimtas kitas standartas, kuriam prigijo SQL-92 ir SQL2 pavadinimai. Šiuo metu, tai – reikalavimai, kurių prisilaiko dauguma komercinių RDBVS. 1999 m. pasirodė naujas standartas SQL3. SQL2 nuo SQL1 skiriasi, daugiausia, kiekybiniais parametrais. SQL3 standarto kalba išsiskiria gana dideliais kokybiniais pakitimais. Į SQL3 įvesti nauji duomenų tipai, numatyta galimybė apibrėžti naujus struktūrinius duomenų tipus. Naujų galimybių atsiradimą labai įtakojo objektinių technologijų paplitimas. Šiandien RDBVS vystosi, siekdamos įdiegti SQL3 standarto galimybes.

Pastebėsime, kad SQL sudaro gana daug sakinių. Kai kurių sakinių sintaksės aprašai užima ne vieną puslapį. 1992 m. išleisto SQL-92 (International Standard Database Language SQL, 1992) aprašas užima virš 600 puslapių. Todėl, nėra galimybės nedidelės apimties knygoje išdėstyti ne tik pagrindines reliacinių DB sąvokas, bet ir SQL sintaksę. Šioje mokomojoje knygoje mes apžvelgsime žymią dalį SQL2 ir tik nedidelę dalį SQL3 sakinių.

Nagrinėjamų SQL sakinių detalios sintaksės nepateiksime. Juo labiau, kad, neretai, SQL sakinių sintaksė konkrečiose DBVS skiriasi tiek nuo standarto, tiek ir tarpusavyje. Praktiškai, SQL dialektų skaičius yra lygus komercinių RDBVS skaičiui. Toliau SQL sakinius pateiksime RDBVS IBM DB2 SQL dialekte, kuris yra artimas SQL2 standartui. Pateikdami sakinių sintaksę prisilaikysime Backus-Naur notacijos. Kita vertus, šioje knygelėje, nesistengsime pateikti visų nagrinėjamų sakinių galimybių. Tikslios sakinių sintaksės siūlome ieškoti komercinio produkto dokumentacijoje. Mūsų tikslas yra išmokyti skaitytoją operuoti pagrindinėmis RDBVS ir SQL sąvokomis. Dėstoma medžiaga skirta suformuoti pagrindą, kuris leistų savarankiškai naudotis komercinio produkto dokumentacija, siekiant pasitikslinti sakinių sintaksę, galimybes ir pan.

Ši mokomoji priemonė sudaryta taip, kad skaitytojas galėtų palaipsniui susipažinti su pagrindinėmis reliacinių DBVS sąvokomis ir išmokti jomis operuoti naudojant SQL kalbą. Priemonė nėra orientuota į sudėtingų taikomųjų programų kūrimą - tai kitų informatikos sričių (informacinių sistemų, programų inžinerijos ir kt.) objektas.

2. Informacijos išrinkimas

2.1. Duomenų bazė "Darbai"

Duomenų bazės duomenys yra žymiai dažniau peržiūrimi negu keičiami ar įvedami. Dar rečiau ir tik nedaugeliui vartotojų (DB administratoriams ir taikomųjų sistemų kūrėjams) tenka kurti lenteles ir duomenų bazes. Vienas iš paprasčiausių būdų susipažinti su DB ir SQL kalba yra pradėti nuo jau esamų duomenų duomenų bazėje peržiūrėjimo.

Tarkime, egzistuoja DB *Darbai*, kurioje yra trys lentelės *Vykdytojai*, *Projektai* ir *Vykdymas*. Norėdami išskirti DB objektų vardus, rašysime juos kursyvu. Tarkime, kad šios lentelės užpildytos duomenimis, kaip parodyta 2.1 pav.

Vykdytojai

Nr	*Pavardė*	*Kvalifikacija*	*Kategorija*	*Išsilavinimas*
1	Jonaitis	Informatikas	2	VU
2	Petraitis	Statistikas	3	VU
3	Gražulytė	Inžinierius	1	NULL
4	Onaitytė	Vadybininkas	5	VDU
5	Antanaitis	Informatikas	3	VU

Projektai

Nr	*Pavadinimas*	*Svarba*	*Pradžia*	*Trukmė*
1	Studentų apskaita	Maža	2001.01.01	12
2	Buhalterinė apskaita	Vidutinė	2001.03.01	10
3	WWW svetainė	Didelė	2001.06.01	2

Vykdymas

Projektas	*Vykdytojas*	*Statusas*	*Valandos*
1	1	Programuotojas	30
1	2	Dokumentuotojas	100
1	3	Testuotojas	100
1	4	Vadovas	100
2	1	Programuotojas	300
2	2	Analitikas	250
2	4	Vadovas	100
3	1	Programuotojas	250
3	2	Vadovas	400
3	3	Dizaineris	150

2.1 pav. Duomenų bazė *Darbai*

Duomenų bazėje *Darbai* yra informacija apie realiai neegzistuojančioje informacinių technologijų firmoje vykdomus projektus (atliekamus darbus, užsakymus). Lentelėje *Vykdytojai* yra užregistruoti visi projektų vykdytojai (firmos darbuotojai). Šioje lentelėje yra surašyta vykdytojų tapatumo firmoje numeriai, vykdytojų pavardės, įgyta kvalifikacija, tam tikra kategorija, nusakanti darbuotojo kvalifikacijos lygmenį, bei kur buvo įsigytas išsilavinimas. Viena lentelės eilutė skirta vienam darbuotojui. Lentelėje *Projektai* yra informacija apie vykdomus firmoje projektus. Lentelės eilutėje yra unikalus projekto tapatumo numeris, pavadinimas, svarbos lygmuo ir vykdymo trukmė mėnesiais. Trečioje lentelėje *Vykdymas* yra informacija apie tai, kokius darbus kurie vykdytojai atlieka. Lentelės eilutėje yra projekto ir darbuotojo numeriai, statusas, kuriame darbuotojas dalyvauja projekte, ir kiekis valandų, kurias vykdytojas skiria projektui. Kiekviena reikšmių pora (*Projektas*, *Vykdytojas*) lentelėje *Vykdymas* yra unikali, t.y. jei darbuotojas dalyvauja kokiame nors projekte, tai tam skirta tik viena eilutė.

Pastebėsime, kad DB *Darbai* yra grynai mokomoji. Ji neatspindinti realios situacijos. Pateikta DB yra per daug supaprastinta, kad ji atitiktų realių projektų vykdymą realioje firmoje. Šios DB pagrindinė paskirtis – išmokti sudaryti užklausas.

Užklausose, kaip ir daugumoje kitų SQL sakinių, nėra nurodoma, kuriai DB yra skirtas sakinys. Duomenų bazės vardas nurodomas prieš veiksmus su duomenimis. DB, kuriai yra skiriami tolimesni SQL sakiniai, nurodoma sakiniu:

`CONNECT TO <`DB vardas`>.`

Čia frazė "CONNECT TO" yra bazinis (raktinis) SQL žodis. Eilutės gale padėtas taškas nėra SQL sakinio dalis, tai teksto sakinio pabaiga. SQL standartas nenumato sakinio pabaigos požymio, nors kai kuriose komercinėse DBVS yra vartojamas sakinio pabaigos požymis, pvz., kabliataškis. Pačiame SQL sakinyje gali būti vartojami įvairūs atskyrėjai. Raktinius kalbos žodžius rašysime kitu šriftu, kad šie išsiskirtų. SQL baziniai žodžiai gali būti rašomi didžiosiomis ir mažosiomis raidėmis. Tas pat taikoma ir DB objektų (lentelių, stulpelių ir kt.) pavadinimams – raidžių registras neturi įtakos. Mes bazinius žodžius rašysime didžiosiomis raidėmis.

Sakinys, kuriuo nurodoma duomenų bazė, su kuria toliau bus dirbama, yra "aktyvus" – jo vykdymo metu vyksta fizinis ir loginis ryšio nustatymas su DB. Jei šio sakinio vykdymo metu DBVS negalės nustatyti ryšio, pavyzdžiui, dėl fizinio ryšio nebuvimo su DB serveriu, tai sakinys nebus įvykdytas. DBVS informuoja vartotoją apie SQL sakinio įvykdymo sėkmę atitinkamu pranešimu. Tam, kad pradėti darbą su DB *Darbai*, reikia įvykdyti sakinį

`CONNECT TO` *Darbai.*

Pabaigus darbą, loginis ryšys su DB nutraukiamas sakiniu

`CONNECT RESET.`

Pastebėsime, kad nereikia palaikyti ryšio su DB be reikalo, kadangi tai susiję su tam tikrų resursų sąnaudomis tiek vartotojo darbo vietoje, tiek ir serveryje. Rekomenduojama nustatyti ryšį prieš pat pradedant darbą su DB ir nutraukti jį iš karto, kai tik nustojama (laikinai ar visiškai) naudotis ja.

2.2. Sakinys SELECT

SQL sakinys `SELECT` yra pagrindinis sakinys DB-je esantiems duomenims peržiūrėti. Šis sakinys turi daug įvairiausių variantų ir galimybių. Išsamiai šio sakinio sintaksei pateikti prireiktų keleto puslapių. Sakinį bendriausiu atveju galima užrašyti taip:

```
SELECT   [DISTINCT]  <stulpelių vardai>
         FROM        <lentelių vardai>
         [WHERE      <paieškos sąlyga>]
         [GROUP BY   <stulpelių vardai> [HAVING  <Paieškos sąlyga>]]
         [ORDER BY   <stulpelių vardai>] .
```

Įvykdžius šį sakinį, DBVS suformuoja ir pateikia vartotojui užklausos rezultatą - laikiną lentelę, kuri egzistuoja tik užklausos rezultato peržiūros metu. Sistemos vartotojas rezultatą gali pamatyti monitoriaus ekrane ar apdoroti jį programoje.

Nesunku pastebėti, kad pati paprasčiausia užklausa atrodo taip:

`SELECT` <stulpelių vardai> `FROM` <lentelės vardas>.

Tokios užklausos rezultatą sudaro išvardinti nurodytos lentelės stulpeliai. Į rezultatą įtraukiamos visos lentelės eilutes. Sakinio dalį iki bazinio žodžio FROM vadinsime SELECT fraze. Šia fraze nusakomi stulpeliai, kurie turi patekti į užklausos rezultatą.

Kadangi lentelėje gali būti labai daug eilučių, tai rezultatas, kurį sudaro lentelės visos eilutės, gali būti nepriimtinas peržiūrai. Sakinį papildžius fraze WHERE <paieškos sąlyga>, užklausos rezultatą sudarys tik tos eilutės, kurios tenkins nurodytą sąlygą. Detaliau paieškos sąlygų formavimą bei kitas sakinio dalis aptarsime tolimesniuose skyreliuose.

2.3. Paprasčiausios užklausos duomenims išrinkti

SQL sakinį SELECT, kaip ir daugelį kitų sakinių, paaiškinsime nagrinėdami pavyzdžius. Tarkime, mes norime sužinoti visų informatikų – projektų vykdytojų pavardes ir kategorijas. Nesunku pastebėti, kad visa ši informacija yra vienoje lentelėje *Vykdytojai*. Reikiamą informaciją galima sužinoti įvykdžius sakinį:

SELECT *Pavardė, Kategorija* FROM *Vykdytojai* WHERE *Kvalifikacija* = 'Informatikas'.

Šioje užklausoje pavartota frazė 'Informatikas' yra simbolių eilutė – SQL konstanta. Simbolių eilutė (tekstinių duomenų konstanta) SQL sakiniuose rašoma tarp apostrofų. Šios užklausos rezultatas bus laikina lentelė, kurią sudarys du stulpeliai, ir į ją įeis tos lentelės *Vykdytojai* eilutės, kuriose stulpelio *Kvalifikacija* reikšmė yra 'Informatikas':

Pavardė	*Kategorija*
Jonaitis	2
Antanaitis	3

Lentelės stulpelius užklausoje galima patikslinti, lentelės vardu:

SELECT *Vykdytojai.Pavardė, Vykdytojai.Kategorija* FROM *Vykdytojai*
WHERE *Vykdytojai.Kvalifikacija* = 'Informatikas'.

Stulpelių vardų patikslinimai nėra būtini, jei SQL sakinyje vartojama tik viena lentelė, kaip pastarajame pavyzdyje. Kaip pamatysime vėliau, stulpelių vardus gali prireikti patikslinti, kai sakinyje dalyvauja keletas lentelių. Patikslinant vardus, dažnai nėra patogu rašyti gana ilgus lentelių vardus. Šito galima išvengti, naudojant lentelių vardų sinonimus, pvz.:

SELECT *A.Pavardė, A.Kategorija* FROM *Vykdytojai* AS *A*
WHERE *A.Kvalifikacija* = 'Informatikas'.

Paminėjus lentelės *Vykdytojai* vardą frazėje FROM, čia pat jam suteikiamas sinonimas *A* (raktinis žodis AS gali būti ir praleistas). Lentelei suteiktas sinonimas galioja tik tame viename SQL sakinyje.

Tarkime, mums reikia sužinoti visas aukštąsias mokyklas, kurias yra baigę projektų vykdytojai. Šį uždavinį galime išspręsti tokia užklausa:

SELECT *Išsilavinimas* FROM *Vykdytojai* ,

kurios rezultatas bus:

Išsilavinimas
VU
VU
NULL
VDU
VU

Kadangi užklausoje nėra nurodyta jokia sąlyga, tai rezultate yra tiek eilučių, kiek jų yra lentelėje *Vykdytojai*. Šiame rezultate tekstas 'VU' kartojasi tris kartus, kas visiškai nebūtina. Vienodos eilutės galėtų netgi trukdyti, jei jų būtų daug. Vienodų eilučių galima išvengti panaudojant bazinį žodį DISTINCT. Paskutinį sakinį perrašę taip:

SELECT DISTINCT *Išsilavinimas* FROM *Vykdytojai* ,

gausime rezultatą, kurį sudarys tik trys eilutės. Taigi, neįtraukus į užklausą bazinio žodžio DISTINCT, užklausos rezultate galimos vienodos eilutės, tiksliau, vienodos eilutės nėra pašalinamos.

Kai kurie stulpeliai gali būti apskaičiuojami. Tarkime, kad lentelėje *Projektai* projektų trukmė nurodyta mėnesiais, o mes tą laiką norime sužinoti dienomis. Paprastumo dėlei tarkime, kad kiekviename mėnesyje yra 30 dienų. Uždavinį išspręsime sakiniu:

SELECT *Pavadinimas, Trukmė* * 30 FROM *Projektai* .

Šiame sakinyje yra pavartotas skaičius 30 – tai skaitinių duomenų konstanta. Skaičiai SQL sakiniuose rašomi be jokių papildomų atskyrėjų. Šios užklausos rezultatas yra laikina lentelė, turinti du stulpelius. Pirmojo stulpelio reikšmės – tai lentelės *Projektai* stulpelio *Pavadinimas* reikšmės. Antrojo stulpelio reikšmės – tai tos pačios lentelės stulpelio *Trukmė* reikšmės, padaugintos iš 30. Antrasis stulpelis, skirtingai nuo pirmojo, nėra paprastas lentelės stulpelis, tai aritmetinis reiškinys, kuriame gali būti ir keli lentelės stulpeliai. Kai užklausos SELECT frazėje vartojamas reiškinys, rezultato stulpelio pavadinimas gali būti dviprasmiškas. Tokiais atvejais sistema, vaizduodama užklausos rezultatą, stulpeliui suteikia tarnybinį pavadinimą, kuris atitinka stulpelio eilės numerį užklausoje (antrajam stulpeliui suteikiamas vardas "2"). Tai nevisada priimtina. Vartotojas gali pats nurodyti norimą stulpelio pavadinimą, pavartojant bazinį žodį AS. Suteikiamas stulpeliui pavadinimas turi tenkinti tuos pačius reikalavimus, kaip ir duomenų bazės stulpelio pavadinimas. Norint stulpeliui suteikti pavadinimą, kuriame būtų vienas ar keli specialieji simboliai, būtina stulpelio pavadinimą rašyti tarp kabučių:

SELECT *Pavadinimas, Trukmė* * 30 AS "*Trukmė dienomis*" FROM *Projektai* .

Šioje užklausoje antrojo stulpelio pavadinimas yra tarp kabučių, nes jį sudaro du žodžiai (tiksliau, pavadinime yra tarpas). Užklausos rezultatas:

Pavadinimas	*Trukmė dienomis*
Studentų apskaita	360
Buhalterinė apskaita	300
WWW svetainė	60

Pastebėsime, kad ir anksčiau mes vartojome stulpelių pavadinimus, kuriuos kai kuriose komercinėse DBVS reikėtų rašyti tarp kabučių arba jie apskritai būtų neleistini. Tai stulpelių pavadinimai, kuriuose pavartotos lietuvių kalbos alfabeto raidės, nesančios lotynų alfabete, pvz., *Pavardė*, *Pradžia*. Daugumoje komercinių DBVS duomenų bazės objektų (lentelių, stulpelių ir kt.) pavadinimuose (**SQL varduose**) leidžiama vartoti tik lotynų alfabeto raides. SQL kalboje raidėmis taip pat laikomi trys pavieniai simboliai: $, # ir @. SQL vardai turi prasidėti raide. Sudarant SQL vardus galima vartoti raides, skaitmenis bei pabraukimo ženklą (_). Mes, patogumo dėlei, laikysime, kad, sudarant SQL vardus, galima vartoti visas lietuvių kalbos raides. Tarp kabučių rašysime tik vardus, kuriuose yra specialieji simboliai.

Ankstesniajame pavyzdyje, SELECT frazėje, pavartojome aritmetinį reiškinį. Ypatingas reiškinio atvejis yra konstanta. Pateiksime užklausą su konstanta SELECT frazėje:

```
SELECT Pavadinimas,
       'Trukmė dienomis: ' AS Pastaba,
       Trukmė * 30 AS "Trukmė dienomis"
FROM Projektai .
```

Užklausos antrojo stulpelio reikšmė yra konstanta. Visos šio stulpelio reikšmės yra vienodos ir lygios tekstinei konstantai 'Trukmė dienomis: ':

Pavadinimas	*Pastaba*	*Trukmė dienomis*
Studentų apskaita	Trukmė dienomis:	360
Buhalterinė apskaita	Trukmė dienomis:	300
WWW svetainė	Trukmė dienomis:	60

Atkreipsime dėmesį į tai, kad užklausoje pavartoti tiek apostrofai, tiek ir kabutės. Priminsime, kad apostrofais žymimos tekstinės konstantos, o kabutėmis – lentelės stulpelių bei kitų duomenų bazės objektų pavadinimai.

Pastebėsime, kad konstanta gali sudaryti ir visą užklausos rezultatą, pvz., užklausos

SELECT 'Trukmė dienomis' AS *Pastaba* FROM *Projektai*

rezultatas bus suformuotas tik iš vienos konstantos:

Pastaba
Trukmė dienomis
Trukmė dienomis
Trukmė dienomis

Nežiūrint į tai, kad iš užklausos formuluotės aišku, kokia reikšmė sudarys užklausos rezultate, DBVS vistiek turi peržiūrėti lentelę *Projektai*, nes rezultatą sudaro tiek eilučių, kiek jų yra lentelėje.

Norint užklausos rezultate gauti visų lentelės stulpelių reikšmes, vietoj to, kad išvardinti juos SELECT frazėje, galima pavartoti specialųjį simbolį '*'. Visi duomenys, esantys lentelėje *Vykdytojai* bus pateikti įvykdžius vieną iš šių užklausų:

SELECT * FROM *Vykdytojai*,

SELECT *Vykdytojai*.* FROM *Vykdytojai*,

SELECT A.* FROM *Vykdytojai* A.

Dažnai didelę duomenų aibę žymiai patogiau peržiūrėti, kai ši išdėstyta pagal vieną ar kitą kriterijų. DBVS sutvarko užklausos rezultatą pagal tam tikrą kriterijų (kriterijus), kai sakinyje SELECT yra tvarką pibrėžianti frazė ORDER BY. Šioje frazėje nurodomi stulpelių vardai arba stulpelių eilės numeriai, atskiriant juos kableliais. Po kiekvieno stulpelio vardo (arba numerio) galima rašyti vieną iš bazinių žodžių: ASC (angl. *ascending*) arba DESC (angl. *descending*), kuriais nustatoma reikšmių didėjimo ar mažėjimo tvarka. Jei nepavartotas nei vienas iš šių žodžių, tai, pagal nutylėjimą, suprantama ASC. Pavyzdžiui, įvykdžiusi užklausą

SELECT *Pavardė, Išsilavinimas, Kategorija* FROM *Vykdytojai*
ORDER BY 2, *Kategorija* DESC,

sistema pateiks duomenis apie visų vykdytojų pavardes, jų išsilavinimą ir kategoriją, sutvarkytus pagal antrojo ir trečiojo stulpelių reikšmes. Rezultatas bus pateiktas leksikografine tvarka pagal išsilavinimą. Esant vienodiems aukštųjų mokyklų pavadinimams, eilučių tarpusavio padėtis bus nustatoma pagal kategoriją, jų mažėjimo tvarka. Atkreipsime dėmesį į tai, kad antrojo stulpelio reikšmės yra tekstinės. SQL kalboje tarp tekstinių duomenų yra nustatyta leksikografinė tvarka.

2.4. Reiškiniai

Užklausose jau vartojome pačius paprasčiausius reiškinius: konstantas, stulpelių pavadinimus ir aritmetinius reiškinius. Aptarsime reiškinius detaliau.

Vardinės konstantos (sisteminiai pseudokintamieji) - tai reikšmės, kurias saugo DBVS ir kurias galima vartoti SQL sakiniuose. Paprastai, tai išorinė DBVS požiūriu informacija, žyminti einamąją sisteminę datą (CURRENT DATE), laiką (CURRENT TIME), datą ir tikslų laiką (CURRENT TIMESTAMP), sistemos vartotojo vardą (USER) ir pan. Kai kuriose komercinėse DBVS, sisteminės reikšmės pasiekiamos ne per vardines konstantas, bet kviečiant specialias funkcijas.

Funkcija - tai operacija, nusakoma funkcijos vardu ir apskliaustais lenktiniais skliaustais argumentais, kurie tarpusavyje atskiriami kableliais (atskiru atveju argumentų gali ir nebūti). Funkcija visuomet grąžina tam tikrą rezultatą (tai gali būti ir specialioji reikšmė NULL). Funkcijos yra skirtomos į agregatines (stulpelių) ir skaliarines.

Agregatinės funkcijos - tai funkcijos, kurias galima pritaikyti eilučių aibėms: visoms eilutėms, tenkinančioms frazėje WHERE nurodytą sąlygą, arba eilučių grupei, apibrėžiamai fraze GROUP (eilučių grupavimą aptarsime vėliau). Agregatinės funkcijos bet kokiam eilučių rinkiniui apskaičiuoja vieną reikšmę. Šios rūšies funkcijos dažnai vartojamos SELECT frazėje, nurodant stulpelį, pagal kurio reikšmes apskaičiuojamas funkcijos rezultatas.

Agregatinės funkcijos vartojamos ir frazėje HAVING, kai funkcijos reikšmė skaičiuojama kiekvienai eilučių grupei. Keletas dažnai vartojamų agregatinių funkcijų:

Agregatinė funkcija	Rezultatas
SUM([DISTINCT] <reiškinys>)	(Skirtingų) ne NULL reikšmių suma
AVG([DISTINCT] <reiškinys>)	(Skirtingų) ne NULL reikšmių vidurkis
COUNT([DISTINCT] <reiškinys>)	(Skirtingų) ne NULL reikšmių kiekis
COUNT(*)	Eilučių kiekis aibėje
MAX(<reiškinys>)	Maksimali reikšmė
MIN(<reiškinys>)	Minimali reikšmė

Jei visos funkcijoje nurodyto stulpelio reikšmės yra NULL, arba stulpelis yra tuščias, tai funkcijų SUM, AVG, MIN, MAX rezultatas yra NULL, o funkcijos COUNT – nulis.

Pateiksime keletą užklausų su agregatinėmis funkcijomis. Visų vykdytojų skaičius arba, tiksliau, eilučių kiekis lenteleje *Vykdytojai* gali būti sužinomas sakiniu:

SELECT COUNT(*) FROM *Vykdytojai*.

Visų vykdytojų, dalyvaujančių bent viename projekte, skaičius - sakiniu:

SELECT COUNT(DISTINCT *Vykdytojas*) FROM *Vykdymas*.

Pirmosios (iš pastarųjų dviejų) užklausos rezultate gausime vieną eilutę, kurioje bus pateiktas skaičius 5. Antrosios užklausos rezultatas - taip pat viena eilutė, bet joje bus skaičius 4 – tiek skirtingų vykdytojų numerių yra paminėta lentelės *Vykdymas* stulpelyje *Vykdytojas*.

Bendrą kiekį valandų, kurias vykdytojas Nr. 1 skiria visiems projektams, galima sužinoti įvykdžius užklausą:

SELECT SUM(*Valandos*) AS “Vykdytojo Nr. 1 valandos” FROM *Vykdymas*
WHERE *Vykdytojas* = 1.

Kadangi šis vykdytojas dalyvauja trijuose projektuose, t.y. lentelėje *Vykdymas* yra 3 eilutės tenkinančios nurodytą sąlygą (*Vykdytojas* = 1), ir 30 + 300 + 250 = 580, tai pastarosios užklausos rezultatas bus:

Vykdytojo Nr. 1 valandos
580

Skaliarinės funkcijos argumentas visuomet yra viena reikšmė. Funkcija gali turėti ir kelis argumentus, tačiau kiekvienas argumentas – viena reikšmė, o ne reikšmių aibė, kaip agregatinėje funkcijoje. Vartojant skaliarines funkcijas frazėje WHERE, funkcijos rezultatas apskaičiuojamas tikrinant paieškos sąlygą kiekvienai lentelės eilutei atskirai. Šios rūšies funkcijos dažnai vartojamos ir SELECT frazėje, kuomet rezultatas skaičiuojamas ne visoms eilutėms iš karto, kaip yra agregatinių funkcijų atveju, o kiekvienai eilutei atskirai. Paprastai, DBVS leidžia vartoti gana daug skaliarinių funkcijų. Paminėsime tik keletą iš jų: DAY(<data>) – diena (skaičius nuo 1 iki 31) datoje, MONTH(<data>) – mėnesis datoje, YEAR(<data>) – metai datoje, LENGTH(<simbolių eilutė>) – simbolių eilutės ilgis, SUBSTR(<simbolių eilutė>, <pradžia>, <ilgis>) – simbolių eilutės fragmentas ir pan.

Iš konstantų, kreipinių į funkcijas bei vardų, žyminčių DB objektus, panaudojant operacijas, galima sudaryti paprasčiausius reiškinius. SQL leidžia naudoti dvi unarines operacijas: + (pliusas), - (minusas), bei keletą binarinių: + (sudėtis), - (atimtis), * (daugyba), / (dalyba), || (apjungimas). Jei bent vienas iš pateiktų operacijų argumentų yra NULL, tai ir rezultatas yra NULL.

Su tekstiniais duomenimis galima atlikti tik dviejų simbolių eilučių nuoseklų apjungimą į vieną (||). Su skaičiais, kaip įprasta, galima atlikti visas aritmetines operacijas. Sudėtį ir atimtį galima atlikti ir su datos bei laiko duomenimis, tiksliau, turimą reikšmę galima padidinti, sumažinti bei rasti dviejų reikšmių skirtumą. Prie datos galima pridėti ar iš jos atimti tam tikrą sveiką skaičių dienų, mėnesių ar metų. Laiką galima pakeisti laiko vienetais. Atliekant tokias

operacijas, vienas iš argumentų – sveikas skaičius turi turėti dimensiją – raktinį žodį, nurodantį vienetus. Dimensijų pavyzdžiai: YEARS, MONTHS, DAYS, HOURS, MINUTES, SECONDS. Reiškinio '2000.31.31' + 3 DAYS rezultatas yra '2000.01.03', o reiškinys *Projektai.Pradžia* + 3 MONTHS nusako datą, kuri bus praėjus 3 mėnesiams nuo *Projektai.Pradžia*. Atimant vieną iš kitos dvi datas ar du laikus, gaunamas laikotarpis. Galimas ir neigiamas laikotarpis.

Sudarant reiškinius, galima naudoti lenktinius skliaustus, taip keičiant operacijų atlikimo tvarką.

Reiškiniai gali būti predikatų argumentais (operandais). **Predikatas -** tai sąlyga lentelės eilutei ar eilučių grupei, kuri gali būti teisinga, neteisinga arba neapibrėžta. Visos predikate dalyvaujančios reikšmės tarpusavyje turi būti suderintos. Be to, simbolių eilutės, įeinančios į predikatus, dažnai, negali būti ilgesnės už tam tikrą konkrečiai DBVS būdingą simbolių kiekį, pvz., 4000. Paprasčiausi predikatai sudaromi panaudojant palyginimo operacijas. Palyginime visuomet dalyvauja dvi reikšmės. SQL leidžiamos tokios palyginimo operacijos: =, <, <=, >, >=, <>. Jei palyginimo operacijose vienas iš operandų yra NULL**,** tai predikato rezultatas yra neapibrėžtas. Palyginimo operacijose operandais gali būti ne tik skaitiniai duomenys, bet ir tekstiniai bei datos ir laiko duomenys.

Prie paprasčiausių predikatų priskiriama:

- x BETWEEN y AND z - rezultatas teisingas tik tuomet, kai x reikšmė yra tarp y ir z, t.y. kai x >= y ir x <= z;
- x NOT BETWEEN y AND z - rezultatas teisingas tik tuomet, kai x nėra tarp y ir z, t.y. kai x < y arba x > z;
- x IN (y1, y2,..., yn) - rezultatas teisingas tik tuomet, kai x reikšmė sutampa bent su viena iš reikšmių y1, y2,..., yn;
- x NOT IN (y1, y2,..., yn) - rezultatas teisingas tik tuomet, kai x nesutampa nei su viena iš reikšmių y1, y2, ..., yn;
- x LIKE y - rezultatas teisingas tik tuomet, kai simbolių eilutė x yra “panaši” į simbolių eilutę y. Simbolių eilutėje y galima panaudoti formato simbolius: %, kuris atitinka bet kokį kiekį, bet kokių simbolių, įskaitant tuščią (nulinio ilgio) eilutę; _(pabraukimo ženklas), kuris atitinka bet kurį vieną simbolį. Eilutėje y galimi ir bet kokie kiti simboliai, atitinkantys juos pačius;
- x NOT LIKE y - rezultatas yra teisingas tik tuomet, kai simbolių eilutė x nėra panaši į simbolių eilutę y. Eilutėje y gali būti pavartoti tie patys simboliai kaip ir predikate LIKE;
- x IS NULL - rezultatas yra teisingas tik tuomet, kai reiškinio x reikšmė yra NULL;
- x IS NOT NULL - rezultatas yra teisingas tik tuomet, kai reiškinio x reikšmė nėra NULL.

Iš predikatų, panaudojant logines operacijas: AND (loginis “ir”), OR (loginis “arba”) arba NOT (loginis “ne”), galima konstruoti paieškos sąlygas. Paieškos sąlygos rezultatas gali būti: “tiesa” (sąlyga patenkinta, teisinga), “netiesa” (sąlyga nepatenkinta, neteisinga) arba neapibrėžtas.

Loginių operacijų panaudojimas SQL kalboje atitinka priimtus matematinėje logikoje susitarimus, tačiau atsižvelgiama į tai, kad argumentų reikšmės gali būti neapibrėžtos. Pateiksime loginių operacijų AND ir OR teisingumo lentelę:

X	Y	X AND Y	X OR Y
Tiesa	Tiesa	Tiesa	Tiesa
Tiesa	Netiesa	Netiesa	Tiesa
Tiesa	Neapibrėžta	Neapibrėžta	Tiesa
Netiesa	Tiesa	Netiesa	Tiesa
Netiesa	Netiesa	Netiesa	Netiesa
Netiesa	Neapibrėžta	Netiesa	Neapibrėžta
Neapibrėžta	Tiesa	Neapibrėžta	Tiesa
Neapibrėžta	Netiesa	Netiesa	Neapibrėžta
Neapibrėžta	Neapibrėžta	Neapibrėžta	Neapibrėžta

Loginė operacija NOT apibėžiama taip: NOT (Tiesa) yra Netiesa, NOT (Netiesa) yra Tiesa ir NOT(Neapibėžta) yra Neapibėžta. Jei paieškos sąlygoje yra pavartoti skliausteliai, tai sąlyga, esanti tarp jų įvertinama pirmiau.

Pavyzdžiui, projektų, kurių pavadinime yra frazė "apskaita", jų svarba yra vidutinė ar didelė, ir kurie turėjo būti pabaigti iki šiandien, pavadinimus, vykdymo pradžios bei pabaigos datas galima sužinoti tokia užklausa:

```
SELECT Pavadinimas, Pradžia, Pradžia+Trukmė MONTHS AS Pabaiga
FROM Projektai
WHERE Pavadinimas LIKE '%apskaita%' AND
      Svarba IN ('Vidutinė', 'Didelė') AND
      Pradžia+Trukmė MONTHS < CURRENT DATE.
```

Informacija apie vykdytojus – informatikus arba vykdytojus, baigusius Vilniaus universitetą (nepriklausomai nuo kvalifikacijos) ir turinčius aukštesnę nei trečią kategoriją, bus pateikta įvykdžius užklausą:

```
SELECT * FROM Vykdytojai
WHERE Kvalifikacija = 'Informatikas' OR (Išsilavinimas = 'VU' AND Kategorija > 3).
```

2.5. Kelių lentelių jungimas

Viena iš svarbiausių SELECT sakinio galimybių yra dviejų ar daugiau lentelių jungimas (angl. *join*). Tarkime, mums reikia sužinoti pavardes vykdytojų, vykdančių projektą Nr. 1. Visa reikalinga informacija yra dviejose lentelėse: *Vykdytojai* ir *Vykdymas,* todėl užklausoje turi dalyvauti ne viena, o dvi lentelės. Šios lentelės turi bendrą dalį, kuri jas sieja - tai vykdytojo numeris. Pagal šį numerį lenteles galima logiškai jungti. SQL kalba šį uždavinį galima užrašyti taip:

```
SELECT Pavardė FROM Vykdytojai, Vykdymas
WHERE Vykdytojas = Nr AND Projektas = 1.
```

Šioje užklausoje, sąlyga *Vykdytojas = Nr* yra nusakytas loginis ryšys tarp dviejų lentelių.

Jungiant lenteles, kiekviena vienos lentelės eilutė siejama su kiekviena kitos lentelės eilute. Bendru atveju, jei užklausoje dviems lentelėms nėra jokios sąlygos ir vienoje iš lentelių yra n eilučių, o kitoje - m eilučių, tai rezultatą sudaro $m \times n$ eilučių. Tarkime, turime dvi lenteles:

LentelėA

A1	*B1*
a_1	b_1
a_2	b_1
a_2	b_2

LentelėB

A2	*B2*	*C2*
a_1	b_1	c_1
a_2	b_2	c_2

Tuomet užklausos

SELECT *A1, B1, A2, B2, C2* FROM *LentelėA, LentelėB*

rezultatas atrodys taip

A1	*B1*	*A2*	*B2*	*C2*
a_1	b_1	a_1	b_1	c_1
a_2	b_1	a_1	b_1	c_1
a_2	b_2	a_1	b_1	c_1
a_1	b_1	a_2	b_2	c_2
a_2	b_1	a_2	b_2	c_2
a_2	b_2	a_2	b_2	c_2

Paprastai užklausoje dviems lentelėms yra paieškos sąlyga, kuri logiškai susieja vienos lentelės vieną ar kelis stulpelius su kitos lentelės atitinkamais stulpeliais. Papildžius pastarąją užklausą paieškos sąlyga, kuri sujungtų abi lenteles, sulyginant dviejų vienos lentelės stulpelių (*A1*, *B1*) reikšmes su kitos lentelės dviejų stulpelių (*A2*, *B2*) reikšmėmis:

SELECT *A1, B1, C2* FROM *LentelėA, LentelėB*
WHERE *A1= A2* AND *B1 = B2*,

gausime tik dvi eilutės:

A1	*B1*	*C2*
a_1	b_1	c_1
a_2	b_2	c_2

Suprantama, stulpelių porų (*A1*, *A2*) ir (*B1*, *B2*) reikšmės turi būti tarpusavyje palyginamos, t.y., kaip minimum, jų tipai (reikšmių aibės) neturi konfliktuoti. Paskutinės užklausos SELECT frazėje, vietoje anksčiau pavartotų 5 stulpelių, turime tik 3, nes likusieji 2 stulpeliai naujų reikšmių neduoda.

Užrašysime užklausą apie tai, kokie vykdytojai kokius projektus vykdo ir kiek kiekvienam projektui skiria valandų. Jei rezultate norime matyti vykdytojo pavardę ir projekto pavadinimą, o ne jų numerius, tai mums reikia formuluoti užklausą net trims lentelėms:

SELECT *Pavardė, Pavadinimas, Valandos*
FROM *Vykdytojai, Projektai, Vykdymas*
WHERE *Projektas = Projektai.Nr* AND *Vykdytojas = Vykdytojai.Nr*.

Šioje užklausoje dvi lentelės (*Vykdytojai* ir *Projektai*) turi po vieną stulpelį tuo pačiu pavadinimu *Nr*, todėl šio stulpelio pavadinimą būtina patikslinti lentelės pavadinimu.

Lentelę galima sujungti ir su ja pačia. Sudarykime poras vykdytojų, turinčių tą pačią kvalifikaciją:

SELECT *A.Pavardė, B.Pavardė* FROM *Vykdytojai A, Vykdytojai B*
WHERE *A.Kvalifikacija = B. Kvalifikacija* AND *A.Nr < B.Nr*

Jungiant lentelę su ja pačia, galime laikyti, kad turime du tos pačios lentelės egzempliorius (kopijas). Tam, kad būtų galima kreiptis ir į abudu vienos lentelės egzempliorius, būtina kiekvienam egzemplioriui suteikti pavadinimą (*A* ir *B*). Kadangi abiejų lentelių (tiksliau dviejų vienos lentelės egzempliorių) stulpelių vardai sutampa, tai užklausoje negalime apsieiti be stulpelių vardų patikslinimo lentelės egzemplioriaus pavadinimu. Predikatas *A.Kvalifikacija = B. Kvalifikacija* atlieka dviejų lentelių loginio susiejimo funkciją. Sąlyga *A.Nr < B.Nr* pavartota tik tam, kad išvengti vykdytojo poros su juo pačiu ir porų pasikartojimo. Jei rezultate turime porą (x, y), tai pora (y, x) nereikalinga, nes ji nepateikia naujos informacijos. Pastebėsime, kad bazinio žodžio DISTINCT pavartojimas čia nepadeda, nes (x, y) ir (y, x) yra skirtingos eilutės, jei tik $x \neq y$.

2.6. Struktūrinės užklausos

Vienoje užklausoje galima pavartoti ir kitą užklausą. Todėl, žodis "struktūrinė" įeina į kalbos pavadinimą. Keli SELECT sakiniai, pavartoti vienoje užklausoje, visuomet yra griežtoje priklausomybėje. Ši priklausomybė yra hierarchinė - viena užklausa yra išorinė kitos atžvilgiu, o ši yra vidinė (dalinė, angl. *subquery*) išorinės atžvilgiu. Tačiau tai tik sintaksinė bei loginė priklausomybė, užklausų vykdymo tvarka nebūtinai yra tokia.

Apibrėšime pavardes vykdytojų, dalyvaujančių projekte Nr. 1, pasitelkdami vidinę užklausą:

SELECT *Pavardė* FROM *Vykdytojai*
WHERE *Nr* IN (SELECT *Vykdytojas* FROM *Vykdymas* WHERE *Projektas* = 1).

Šioje užklausoje pavartotas bendresnis predikato IN variantas, kai antrasis parametras yra nurodytas vidine užklausa. Vykdant tokią užklausą, DBVS, loginiu požiūriu (realiai gali būti ir kitaip), iš pradžių įvykdo vidinę (esančią sąlygoje) užklausą, suformuodama jos rezultatą – aibę reikšmių. Po to vykdoma išorinė užklausa, kiekvienai išorinės užklausos eilutei tikrinant, ar vykdytojo numeris priklauso aibei, gautai įvykdžius vidinę užklausą.

Pastarojoje užklausoje pavartoti du SELECT sakiniai nėra riba - jų gali būti ir daugiau. Pavyzdžiui, užklausą "pavardės vykdytojų, dalyvaujančių bent viename didelės svarbos projekte" galimai suformuluoti trimis SELECT sakiniais:

```
SELECT Pavardė FROM Vykdytojai
WHERE Nr IN
        (SELECT Vykdytojas FROM Vykdymas
         WHERE Projektas IN
                 (SELECT Nr FROM Projektai WHERE Svarba = 'Didelė')).
```

Priminsime, kad SQL yra deklaratyvi kalba, t.y. formuluojant užklausas reikia apibrėžti, kas norima gauti užklausos rezultate, nesirūpinant, kaip rezultatą gauti. Pastaroji užklausa tėra formalus ir detalus užrašas taip perfrazuotos užduoties: "pavardės vykdytojų, kurių numeriai yra tarp numerių vykdytojų, dalyvaujančių didelės svarbos projekte". Pastarojoje užklausoje visi SELECT sakiniai gali būti vykdomi nuosekliai, pradedant nuo pačios giliausios vidinės užklausos ir baigiant išorine.

Nors pastaroji užklausos formuluotė gana gerai atspindi žodinę užduoties formuluotę, didelis dalinių užklausų kiekis, nereiškia gerą užklausų sudarymo stilių. Dažnai analogiška užklausa, užrašyta vienu sakiniu SELECT, jungiant keletą lentelių, nors, iš pirmo žvilgsnio, ir nėra aiškesnė, tačiau yra trumpesnė, o galiausiai ir aiškesnė. Tačiau lentelių jungimas reikalauja daugiau įgūdžių, nei dalinių užklausų vartojimas. Pateiksime uždavinio apie vykdytojus, dalyvaujančius bent viename didelės svarbos projekte, sprendinį be dalinių užklausų:

```
SELECT DISTINCT Pavardė FROM Vykdytojai, Vykdymas, Projektai
WHERE Projektas = Projektai.Nr AND Vykdytojas = Vykdytojai.Nr  AND
        Svarba = 'Didelė'.
```

Tai, kad lentelių jungimas gali būti žymiai pranašesnis už dalines užklausas, galima įsitikinti palyginus pastarąją užklausos formuluotę su ankstesnio skyrelio užklausa, kurioje jungiamos tos pačios trys lentelės. Pastebėsime, kad abiejų užklausų paieškos sąlygos yra labai panašios, skiriasi tik papildoma sąlyga, lentelių jungimo sąlyga nesikeičia. Vieną kartą išsiaiškinus, kaip logiškai lentelės siejamos tarpusavyje, vėliau, daug užklausų galima parašyti tik papildant paieškos sąlygą.

Atkreipsime dėmesį, kad ta pati lentelė gali būti tiek vidinėje, tiek ir išorinėje užklausoje. Uždavinį "numeriai vykdytojų, kurie dalyvauja bent viename projekte, kuriame dalyvauja vykdytojas Nr. 1" galime išreikšti ir taip:

```
SELECT DISTINCT Vykdytojas FROM Vykdymas
WHERE Vykdytojas <> 1 AND
        Projektas IN (SELECT Projektas FROM Vykdymas WHERE Vykdytojas = 1).
```

Nors šioje užklausoje ta pati lentelė pavartota du kartus, tačiau nėra būtina patikslinti stulpelių vardų dėl panaudotų skliaustų ir galiojančios taisyklės: jei stulpelio vardas nėra tikslus, tai jis priklauso lentelei iš einamosios (vidinės ar išorinės) užklausos.

Anksčiau jau keletą kartų formuluotą uždavinį "pavardės vykdytojų, dalyvaujančių projekte Nr. 1" galima suformuluoti ir taip:

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

Šią užklausą neformaliai galima perfrazuoti taip: vykdytojai, dalyvaujantys projekte Nr.1, tai tokie vykdytojai, kuriems tarp visų projektų, kuriuose jie dalyvauja, yra projektas Nr. 1. Šioje, gana painioje, užklausoje, dalinė užklausa negali būti apskaičiuojama iš anksto, prieš pradedant vykdyti išorinę. Dalinė užklausa vykdoma kiekvienai pagrindinės užklausos eilutei. Tai - priklausomos dalinės užklausos pavyzdys. **Priklausoma užklausa** - tai užklausa, kurios vidinės užklausos rezultatas priklauso nuo išorinės užklausos rezultato. Priklausomoje užklausoje, vidinė užklausa yra parametrizuota, į ją įeina parametras (*Nr*), įgyjantis reikšmę išorinėje užklausoje. Vidinę užklausą tenka vykdyti kiekvienai to kintamojo reikšmei. Priklausomose užklausose negalima abi užklausas (vidinę ir išorinę) vykdyti nuosekliai. Priklausomos užklausos dažnai yra gana painios ir neefektyvios, todėl jų reikia vengti.

Pateiksime ankstesnio uždavinio "pavardės vykdytojų, dalyvaujančių projekte Nr. 1" 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, pavartotas predikatas, kurio sintaksė yra `EXISTS (SELECT * FROM…)`. Šio predikato reikšmė yra "tiesa" tuomet, ir tik tuomet, kai vidinės užklausos rezultatas yra netuščia aibė. Atkreipsime dėmesį į tai, kad tai nėra vienintelis predikatas, realizuojantis kvantoriaus sąvoką. Be egzistavimo kvantoriaus užklausose galima vartoti ir visuotinumo kvantorių. Pastaroji užklausa taip pat yra priklausoma. Vidinės, predikate pavartotos užklausos rezultatas priklauso nuo parametro *Vykdytojai.Nr*, kuris įgyja reikšmę išorinėje užklausoje.

2.7. Laikinos lentelės

Kadangi užklausos rezultatas yra lentelė (nors ir laikina), tai užklausą galima vartoti ir sakinio `SELECT` frazėje `FROM`. Pavyzdžiui, anksčiau suformuluotą užklausą "numeriai vykdytojų, kurie dalyvauja bent viename projekte, kuriame dalyvauja vykdytojas Nr. 1", galima užrašyti ir taip:

```
SELECT DISTINCT Vykdytojas
FROM Vykdymas,
        (SELECT Projektas FROM Vykdymas WHERE Vykdytojas = 1) AS Projektai1
WHERE Vykdymas.Vykdytojas <> 1 AND Projektai1.Projektas = Vykdymas.Projektas .
```

Šio SQL sakinio `FROM` frazėje pavartota užklausa, kuriai suteiktas vardas *Projektai1*. Tai - laikinos lentelės apibrėžimas. Taip apibėžta lentelė egzistuoja tik užklausos vykdymo metu. Į laikiną lentelę galima kreiptis tik užklausoje, kurioje ji yra apibrėžta. Į laikinos lentelės apibrėžimą galima žiūrėti kaip į vidinę užklausą, kuriai suteiktas vardas.

Laikinos lentelės apibrėžimas `FROM` frazėje, ypač kai tokių apibrėžimų yra keletas, gali padaryti užklausą sunkiai suprantama. Kad taip neatsitiktų, galima pasinaudoti specialia, sudėtingoms užklausoms formuluoti skirta konstrukcija `WITH`. Taip formuluojant užklausas elgiamasi panašiai, kaip ir apibrėžiant laikiną lentelę frazėje `FROM`. Iš pradžių apibėžiama

(“sukuriama”) laikina lentelė suteikiant jai vardą, o paskui rašomas SELECT sakinys, kuriame kreipiamasi į aukščiau apibrėžtą laikiną lentelę. Užklausa formuluojama nuosekliai: laikina lentelė yra apibrėžiama prieš (užklausos tekste) jos pavartojimą. Ankstesniąją užklausą galima suformuluoti taip:

```
WITH Projektai1 AS (SELECT Projektas FROM Vykdymas WHERE Vykdytojas = 1)
   SELECT DISTINCT Vykdytojas
   FROM Vykdymas, Projektai1
   WHERE Vykdymas.Vykdytojas <> 1 AND Projektai1.Projektas = Vykdymas.Projektas.
```

Viena fraze WITH galima apibėžti kelias laikinas lenteles, atskiriant jas tarpusavyje kableliais. Apibrėžiant laikiną lentelę, iš karto po jos pavadinimo, tarp skliaustų galima išvardinti ir jos stulpelių pavadinimus. Formuluojant sudėtingas užklausas, ši konstrukcija leidžia uždavinį spręsti, skaidant jį į keletą smulkesnių. Suformulavus užklausą tarpiniams rezultatams gauti, jai suteikiamas vardas ir toliau formuluojama aukštesnio lygio užklausa.

2.8. Duomenų grupavimas

Tarkime, kiekvienam projektui reikia apskaičiuoti, kiek visi vykdytojai bendrai skiria laiko jo vykdymui. Šiam uždaviniui spręsti iki šiol išdėstytų kalbinių priemonių nepakanka. Nesunku apskaičiuoti bendrą valandų kiekį kuriam nors vienam projektui, pvz. Nr. 1:

```
SELECT SUM(Valandos) FROM Vykdymas WHERE Projektas = 1.
```

Šiame SQL sakinyje paieškos sąlygoje yra fiksuotas vienas projektas. Visoms tenkinančioms sąlygą eilutėms yra pritaikyta funkcija valandoms sumuoti. Pradiniam mūsų uždaviniui spręsti, šią stulpelių funkciją reikia pritaikyti kiekvienai eilučių grupei, atitinkančiai visus skirtingus projektus. Eilučių grupavimą užklausose realizuoja speciali konstrukcija GROUP BY**,** kurią pavartojus uždavinio sprendinys atrodo taip:

```
SELECT Projektas, SUM(Valandos) AS Valandos FROM Vykdymas
GROUP BY  Projektas.
```

Visos lentelės *Vykdymas* eilutės grupuojamos taip, kad į kiekvieną grupę patenka eilutės su vienodomis stulpelio *Projektas* reikšmėmis. Po grupavimo, SELECT frazė yra taikoma kiekvienai grupei (o ne eilutei), suformuojant vieną užklausos rezultato eilutę. Vykdant pateiktą užklausą, DBVS kiekvienam projektui (kiekvienai grupei, kurią sudaro duomenys apie vieno projekto vykdymą) įtraukia į užklausos rezultatą projekto numerį ir sumą valandų, kurias skiria visi vykdytojai einamajam projektui vykdyti.

Pastebėsime, kad nežymiai pakeitus pastarąją užklausą:

```
SELECT  Projektas, Vykdytojas, SUM(Valandos) AS Valandos FROM Vykdymas
GROUP BY  Projektas
```

gauname sintaksiškai neteisingą užklausą. Sugrupavus lentelės *Vykdymas* eilutes taip, kad į vieną grupę patenka visos eilutės su ta pačia stulpelio *Projektas* reikšme, grupėje gali atsirasti kelios eilutės su skirtingomis stulpelio *Vykdytojas* reikšmėmis. Tuomet taps neaišku, kuri *Vykdytojo* reikšmė turėtų “atstovauti” grupę užklausos rezultate.

Kadangi, pritaikant SELECT frazę grupei eilučių, suformuojama tik viena rezultato eilutė, tai visi šios frazės reiškiniai turi būti vienareikšmiškai apskaičiuojami, t.y. turi įgyti vieną reikšmę visoje grupėje. Užklausos su grupavimu SELECT frazėje, su nedidelėmis išimtimis, tegali būti:

- stulpelis, paminėtas frazėje GROUP BY (grupavimo stulpelis);
- konstanta;
- reiškinys pagal konstantas ir grupavimo stulpelius;
- agregatinė (stulpelių) funkcija (SUM, MIN, MAX, COUNT, AVG ir kt.), kuri eilučių grupei pateikia vieną reikšmę.

Prieš GROUP BY frazę galima vartoti paieškos sąlygą, kuri tikrinama prieš eilučių grupavimą. Jei norime paieškos sąlygą pritaikyti ne kiekvienai lentelės eilutei, bet kiekvienai grupei, tai frazė GROUP BY vartojama kartu su fraze HAVING. Sąlyga grupėms yra tikrinama po grupavimo. Grupės "atstovas" į užklausos rezultatą įtraukiamas tik tuomet, kai grupė tenkina paieškos sąlygą, nurodytą frazėje HAVING.

Tokiu būdu, jei užklausoje su grupavimu yra pavartota frazė WHERE, tai, iš pradžių, kiekvienai eilutei tikrinama ši sąlyga ir visos eilutės, tenkinančios šią sąlygą, yra sugrupuojamos. Jei užklausoje yra pavartota ir frazė HAVING**,** tai pastarojoje frazėje nurodyta sąlyga yra tikrinama kiekvienai grupei. Pavyzdžiui, numeriai projektų, kuriuos vykdo daugiau negu vienas vykdytojas, gali būti sužinomi užklausa:

SELECT *Projektas* FROM *Vykdymas*
GROUP BY *Vykdytojas* HAVING COUNT(*) > 1.

Galima grupuoti ir pagal kelis stulpelius. Tuomet sakoma, kad grupuojama grupės viduje. Tarkime, kad mums reikia sužinoti, kiek yra vykdytojų pagal kiekvieną baigtą aukštąją mokyklą ir turimą kategoriją.Užklausai sudaryti, reikia apibrėžti grupavimą pagal du lentelės *Vykdytojai* stulpelius: *Išsilavinimas* ir *Kategorija*:

SELECT *Išsilavinimas*, *Kategorija*, COUNT(*) AS "*Vykdytojų Skaičius*"
FROM *Vykdytojai* GROUP BY *Išsilavinimas*, *Kategorija*
ORDER BY *Išsilavinimas*.

Šios užklausos rezultatas, išdėstytas pagal išsilavinimą leksikografine tvarka, atrodo taip:

Išsilavimas	*Kategorija*	*Vykdytojų Skaičius*
NULL	5	1
VDU	6	1
VU	2	1
VU	3	2

Tarkime, reikia sužinoti, kiek kiekvienas vykdytojas skiria valandų visiems projektams vykdyti. Be to, mus domina tik tie vykdytojai, kurie visų projektų vykdymui skiria daugiau nei 300 valandų. Jeigu mums pakanktų vykdytojo numerio, tai uždavinį galėtume spręsti panašiai kaip uždavinį, kuriame valandos buvo skaičiuojamos kiekvienam projektui. Jei mus domina vykdytojų pavardės, užklausą reikia formuluoti dviems lentelėms:

SELECT *Pavardė*, SUM(*Valandos*) AS "*Visos Valandos*"
FROM *Vykdytojai*, *Vykdymas* WHERE *Nr* = *Vykdytojas*
GROUP BY *Pavardė* HAVING SUM(*Valandos*) > 300
ORDER BY "*Visos Valandos*".

Sąlygoje grupei (frazėje HAVING) negalima vartoti SELECT frazėje apibrėžto stulpelio pavadinimo "*Visos Valandos"*. Tokio stulpelio pavadinimas grupėse negalioja. Jo tiesiog nėra grupėse. Stulpelio pavadinimas prasmingas tik užklausos rezultato kontekste, todėl jį galima vartoti apibrėžiant eilučių tvarką.

Jei užklausos rezultate norime pamatyti tiek vykdytojų pavardes, tiek ir jų numerius, tai, atsižvelgiant į jau pateiktus apribojimus SELECT frazei, tenka grupuoti pagal du stulpelius:

SELECT *Pavardė*, *Nr*, SUM(*Valandos*) AS "*Visos Valandos*"
FROM *Vykdytojai*, *Vykdymas* WHERE *Nr* = *Vykdytojas*
GROUP BY *Pavardė, Nr* HAVING SUM(*Valandos*) > 300
ORDER BY *Pavardė*.

Tai, kad kiekvienam vykdytojui (jo pavardei) mes gausime ne daugiau kaip po vieną eilutę, lems savybė (žinoma, jei tokia savybė tenkinama), kad nėra dviejų vykdytojų vienodomis pavardėmis. Jei keli vykdytojai galėtų turėti vienodas pavardes, bet skirtingus numerius, tai pastaroji užklausa išliktų teisinga, kai tuo tarpu ankstesnė, kurioje grupuojama tik pagal pavardes, logiškai būtų klaidinga (sintaksiškai išliktų teisinga), nes visi bendrapavardžiai būtų laikomi tuo pačiu asmeniu. Taigi, ir ankstesnėje užklausoje būtų buvę teisingiau (užklausos teisingumas nepriklausytų nuo apribojimų duomenims) pavartoti grupavimą pagal abudu vykdytojo tapatumo atributus: numerį ir pavardę.

2.9. Aibių operacijos

Kadangi užklausos rezultatas yra eilučių aibė, tai prasminga dviejų užklausų rezultatams taikyti aibių operacijas: sąjungą, sankirtą, skirtumą. Kiekvienai iš šių trijų aibių teorijos operacijų SQL kalboje yra po dvi savas operacijas. Taip yra todėl, kad užklausų rezultatuose, skirtingai negu aibių teorijoje, yra galimos vienodos eilutės. SQL kalboje yra šešios aibių operacijos: UNION, UNION ALL, INTERSECT, INTERSECT ALL, EXCEPT ir EXCEPT ALL. Apibrėšime šių operacijų rezultatus.

UNION ir UNION ALL - rezultatas yra dviejų užklausų rezultatų (*R1* ir *R2*) eilučių sąjunga. Operacijos rezultatas yra formuojamas iš visų *R1* ir *R2* eilučių, po to, jei nepavartotas bazinis žodis ALL, pašalinamos pasikartojančios eilutės, paliekant po vieną kiekvienos eilutės egzempliorių.

INTERSECT ir INTERSECT ALL - rezultatas yra dviejų užklausų rezultatų (*R1* ir *R2*) eilučių sankirta. Rezultatas formuojamas iš eilučių, kurios įeina tiek į *R1*, tiek ir į *R2*, po to, jei nepavartotas bazinis žodis ALL, pašalinamos pasikartojančios eilutės, paliekant po vieną kiekvienos eilutės egzempliorių.

EXCEPT ir EXCEPT ALL - rezultatas yra dviejų užklausų rezultatų (*R1* ir *R2*) eilučių skirtumas. Rezultatas formuojamas iš *R1* eilučių, kurių nėra rezultate *R2*, atliekant tai kiekvienam eilutės egzemplioriui atskirai, bet prieš tai, jei nepavartotas bazinis žodis ALL, tiek iš *R1,* tiek ir iš *R2* yra pašalinamos pasikartojančios eilutės, paliekant po vieną kiekvienos eilutės egzempliorių.

Tam, kad būtų galima atlikti aibių operacijas su užklausų rezultatais, būtina, kad stulpelių skaičius operacijų argumentuose (užklausų rezultatuose *R1* ir *R2*) sutaptų, o atitinkamų stulpelių tipai nekonfliktuotų. Jeigu, pavyzdžiui, vienos užklausos rezultate būtų vienas stulpelis, o kitos – du, tai taptų neaišku, kiek stulpelių turėtų būti tokių užklausų rezultatų sankirtoje. Todėl panašios situacijos SQL sintaksė neleidžia.

Visas SQL aibių operacijas paaiškinsime schematiškai. Tarkime, turime du vienodos struktūros užklausų rezultatus (lenteles) *R1* ir *R2*, ir iš viso yra tik penkios skirtingos eilutės, kurias pažymėsime numeriais nuo 1 iki 5. Tarkime, pirmosios užklausos rezultate *R1* yra dešimt eilučių: tris kartus pasikartoja eilutė, pažymėta Nr. 1, tris kartus – Nr. 2, vieną kartą – Nr. 3, du kartus – Nr. 4 ir vieną kartą – Nr. 5, o *R2* sudaro du kartus pasikartojanti eilutė Nr. 1, keturis kartus – Nr. 3 ir viena eilutė, pažymėta Nr. 4. Tuomet, visų šešių aibių operacijų rezultatus galima pavaizduoti tokia lentele:

R1	*R2*	UNION ALL	UNION	EXCEPT ALL	EXCEPT	INTERSECT ALL	INTERSECT
1	1	1	1	1	2	1	1
1	1	1	2	2	5	1	3
1	3	1	3	2		3	4
2	3	1	4	2		4	
2	3	1	5	4			
2	3	2		5			
3	4	2					
4		2					
4		3					
5		3					
		3					
		3					
		3					
		4					
		4					
		4					
		5					

Užklausos rezultatą, gautą aibių operacijos rezultate galima rūšiuoti. Tokiu atveju, eilučių tvarką apibrėžianti frazė yra rašoma po antrojo aibių operacijos operando. Pavyzdžiui, visus vykdytojus (jų numerius), kurie nedalyvauja nei viename projekte, galima sužinoti, įvykdžius užklausą su aibių skirtumo operacija EXCEPT:

```
SELECT Nr FROM Vykdytojai
EXCEPT
SELECT Vykdytojas FROM Vykdymas
ORDER BY 1.
```

Kadangi operacijoje EXCEPT dalyvaujančių užklausų rezultatuose stulpelių pavadinimai (*Nr* ir *Vykdytojas*) yra skirtingi, tai rūšiavimo stulpelis šioje užklausoje nurodytas eilės numeriu (1). Pastebėsime, kad šioje užklausoje operaciją EXCEPT pakeitę operacija EXCEPT ALL gausime tą patį rezultatą. Taip yra todėl, kad lentelėje *Vykdytojai* nėra vienodų stulpelio *Nr* reikšmių.

2.10. Sąlyginiai reiškiniai

Sudarykime projektų sąrašą, kuriame šalia projekto pavadinimo būtų pažymėta, ar jis trumpalaikis, ar ilgalaikis. Projektą laikysime trumpalaikiu, jei jo vykdymo trukmė neilgesnė už 6 mėn., kitaip projektą laikysime ilgalaikiu. Tokį sąrašą galima gauti užklausa:

```
SELECT Pavadinimas, 'Trumpalaikis' FROM Projektai WHERE Trukmė <= 6
UNION
SELECT Pavadinimas, 'Ilgalaikis'     FROM Projektai WHERE Trukmė > 6 .
```

Šioje užklausoje, priklausomai nuo sąlygos teisingumo, stulpelio reikšmė (bendruoju atveju - reiškinys) keičiama viena ar kita reikšme (kitu reiškiniu). Jei mes norėtume projektų vykdymo trukmę skirstyti į daugiau intervalų, tai, taip sprendžiant uždavinį, kiekvienam papildomam trukmės intervalui reikėtų pavartoti naują aibių sąjungos operaciją. Tokio uždavinio sprendimą galima supaprastinti pavartojus konstrukciją CASE, įgalinančią užrašyti sąlyginį reiškinį. Sąlyginio reiškinio sintaksę galima užrašyti taip:

```
CASE WHEN <paieškos sąlyga> THEN NULL | <reiškinys>
     {WHEN <paieškos sąlyga> THEN NULL | <reiškinys>}
     [ELSE NULL | <reiškinys>] END.
```

Sąlyginio reiškinio reikšmė priklauso nuo paieškos sąlygų teisingumo. Vykdant užklausą, kiekvienai eilutei paeiliui peržiūrimos visos paieškos sąlygos tikrinant jų teisingumą, kol randama teisinga sąlyga. Tuomet, viso reiškinio reikšme tampa reikšmė, esanti teisingos sąlygos dešinėje, už bazinio žodžio THEN. Jei tokios sąlygos nėra, viso sąlyginio reiškinio reikšmė apskaičiuojama pagal reiškinį, esantį dešinėje raktinio žodžio ELSE, jei tik toks yra, kitaip viso reiškinio reikšmė yra NULL.

Pastarąją užklausą galima užrašyti be aibių operacijos vienu sakiniu SELECT:

```
SELECT Pavadinimas,
        CASE WHEN Trukmė <= 6 THEN 'Trumpalaikis' ELSE 'Ilgalaikis' END
FROM Projektai.
```

Pateiksime dar vieną sąlyginio reiškinio vartojimo pavyzdį. Tarkime, kiekvienam vykdomam projektui mums reikia pateikti statistinius duomenis: bendrą visų projekto vykdytojų kiekį ir jų skiriamą laiką, bei tokius pat duomenis apie informatikų bei statistikų dalyvavimą projekte atskirai. Visą šią informaciją galima sužinoti trimis labai panašiomis užklausomis:

```
SELECT  Pavadinimas, COUNT(DISTINCT Vykdytojas) AS "Visi vykdytojai",
        SUM(Valandos) AS "Visų valandos"
FROM Projektai, Vykdymas WHERE Nr = Projektas GROUP BY Pavadinimas;

SELECT  Pavadinimas, COUNT(DISTINCT Vykdytojas) AS Informatikai,
        SUM(Valandos) AS "Informatikų valandos"
FROM Projektai, Vykdymas, Vykdytojai
WHERE Projektai.Nr = Projektas AND Vykdytojai.Nr = Vykdytojas AND
      Kvalifikacija = 'Informatikas'
GROUP BY Pavadinimas;

SELECT  Pavadinimas, COUNT(DISTINCT Vykdytojas) AS Statistikai,
        SUM(Valandos) AS "Statistikų valandos"
FROM Projektai, Vykdymas, Vykdytojai
WHERE Projektai.Nr = Projektas AND Vykdytojai.Nr = Vykdytojas AND
      Kvalifikacija = 'Statistikas'
GROUP BY Pavadinimas.
```

Netgi visas šias užklausas apjungus į vieną, pasinaudojant aibių sąjungos operacija, rezultatą nepatogu peržiūrėti, nes informacija apie vieną projektą bus trijose eilutėse. Informaciją apie projektą būtų patogiau peržiūrėti, jei visa ji būtų vienoje eilutėje. Tokį rezultatą gausime pavartoję sąlyginius reiškinius:

```
SELECT  Pavadinimas,
   COUNT(DISTINCT Vykdytojas) AS "Visi vykdytojai",
   SUM(Valandos) AS "Visų valandos",
   COUNT(DISTINCT CASE WHEN Kvalifikacija = 'Informatikas' THEN Vykdytojas)
         AS "Visi informatikai",
   SUM(CASE WHEN Kvalifikacija = 'Informatikas' THEN Valandos)
         AS "Informatikų valandos"
   COUNT(DISTINCT CASE WHEN Kvalifikacija = 'Statistikas' THEN Vykdytojas)
         AS "Visi statistikai",
   SUM(CASE WHEN Kvalifikacija = 'Statistikas' THEN Valandos)
         AS "Statistikų valandos"
FROM Projektai, Vykdymas, Vykdytojai
WHERE Projektai.Nr = Projektas AND Vykdytojai.Nr = Vykdytojas
GROUP BY Pavadinimas.
```

Dažnai vartojamiems sąlyginiams reiškiniams užrašyti yra numatyti sutrumpinimai - skaliarinės funkcijos: `NULLIF` ir `COALESCE`. Šios funkcijos apibrėžiamos taip:

Sąlyginis reiškinys	Ekvivalenti funkcija
`CASE WHEN` e1 = e2 `THEN NULL ELSE` e1 `END`	`NULLIF`(e1,e2)
`CASE WHEN` e1 `IS NOT NULL` `THEN` e1 `ELSE` e2 `END`	`COALESCE`(e1,e2)
`CASE WHEN` e1 `IS NOT NULL` `THEN` e1 `ELSE COALESCE`(e2,...,eN) `END`	`COALESCE`(e1,e2,...,eN)

Tarkime, mums reikia sužinoti, kokios bus vykdytojų kategorijos, jei visiems vykdytojams jas padidintume vienetu, o tiems vykdytojams, kuriems iki šiol kategorija nebuvo suteikta (tarkime, tokių gali būti), priskirsime pirmą kategoriją. Šiam uždaviniui spręsti pavartosime sutrumpintą sąlyginį reiškinį:

```
SELECT Pavardė, Kategorija AS "Esama kategorija",
       COALESCE(Kategorija, 0) + 1 AS "Naujoji kategorija"
FROM Vykdytojai.
```

2.11. Sisteminis katalogas

DBVS tam, kad galėtų sėkmingai vykdyti vartotojo užklausas ir valdyti duomenis, turi "prisiminti" gana didelį kiekį informacijos, susijusios su DB struktūra. Reliacinėje DB tokia informacija saugoma **sisteminiame kataloge** – sistemos lentelėse, kurias DBVS naudoja savo vidiniams poreikiams. Sisteminiame kataloge saugomas ir DB struktūros (lentelių, stulpelių ir kt.) aprašas.

Nors sisteminis katalogas skirtas sistemos vidiniams poreikiams, jame esanti informacija prieinama ir DBVS vartotojams. Reliacinėje DB yra gana detalus jos pačios aprašas, kurį vartotojas gali sužinoti užklausomis.

Sisteminio katalogo lentelės yra sukuriamos automatiškai sukuriant DB. Vartotojui nereikia rūpintis jų turiniu. Sistema pati rūpinasi, kad sisteminis katalogas atspindėtų einamąją DB būseną. Vartotojui griežtai uždrausta keisti sisteminio katalogo turinį – tai išimtinė DBVS privilegija. Vartotojas gali tik užklausti sisteminio katalogo duomenų.

Vykdydama SQL sakinius DBVS pastoviai kreipiasi į sisteminį katalogą. Pavyzdžiui, kad įvykdyti užklausą dviem lentelėms, DBVS turi:

- patikrinti, ar egzistuoja tos dvi lentelės;
- patikrinti, ar einamasis vartotojas, turi teisę kreiptis į abi lenteles;
- patikrinti, ar lentelėse yra stulpeliai, nurodyti užklausoje;
- nustatyti, kuriai lentelei priklauso stulpeliai, kurių vardai užklausoje nurodyti be lentelės pavadinimo;
- kiekvienam stulpeliui nustatyti duomenų tipą.

Visa ši informacija yra iš anksto apibrėžtose sisteminio katalogo lentelėse. Todėl DBVS informacijos paieškai sisteminiame kataloge gali naudoti ypač efektyvius metodus ir algoritmus.

Praktiškai visose komercinėse RDBVS, tarp kitų sisteminio katalogo lentelių, yra lentelės, aprašančios visas DB lenteles ir visus visų lentelių stulpelius. DBVS DB2 tai lentelės *SYSCAT.TABLES* ir *SYSCAT.COLUMNS*. Šios dvi lentelės, kaip ir visos kitos sisteminio katalogo lentelės, taip pat yra aprašytos jose pačiose. Todėl, užklausa:

```
SELECT * FROM SYSCAT.COLUMNS
```

galima sužinoti informaciją apie visų DB lentelių (tame tarpe ir pačios lentelės *SYSCAT.COLUMNS*) stulpelius. Detalesne užklausa:

```
SELECT NAME FROM SYSCAT.COLUMNS
WHERE TABSCHEMA = 'SYSCAT' AND TABNAME = 'TABLES'
```

sužinosime sisteminio katalogo lentelės, kurioje yra DB lentelių aprašai, stulpelių pavadinimus. Sisteminio katalogo lentelių stulpelių pavadinimai yra gana informatyvūs – iš stulpelio pavadinimo galima suprasti jo paskirtį. Visų sisteminio katalogo lentelių pavadinimus, išdėstytus leksikografine tvarka, galima sužinoti įvykdžius užklausą:

```
SELECT TABNAME FROM SYSCAT.TABLES
WHERE TABSCHEMA = 'SYSCAT'
ORDER BY 1.
```

Toks pat lentelių sąrašas bus gautas įvykdžius ir tokią užklausą:

```
SELECT DISTINCT TABNAME FROM Syscat.Columns
WHERE TABSCHEMA = 'SYSCAT'
ORDER BY 1.
```

Lentelės pavadinimą *Syscat.Columns* šioje užklausoje užrašėme pavartodami ir didžiąsias, ir mažąsias raides – tai neturi įtakos. Tačiau sąlygoje pavartotas schemos pavadinimas, būtinai turi būti užrašytas didžiosiomis raidėmis, nes tai tekstinė konstanta. Kadangi lentelėje *SYSCAT.COLUMNS* kiekvienai duomenų bazės lentelei gali būti po keletą eilučių (tiek, kiek lentelėje yra stulpelių), tai bazinis žodis `DISTINCT` užklausoje užtikrina, kad rezultate bus pateikti tik skirtingi lentelių pavadinimai.

Duomenų bazėje gali būti keletas lentelių tuo pačiu pavadinimu, bet skirtingose schemose. Pilną lentelės pavadinimą sudaro schemos ir lentelės pavadinimai kartu paėmus. Pilnus visų duomenų bazės lentelių pavadinimus galima sužinoti užklausa:

```
SELECT TABSCHEMA, TABNAME FROM SYSCAT.TABLES
ORDER BY 1, 2.
```

Detalesnį sisteminio katalogo aprašą, galima rasti konkrečios DBVS dokumentacijoje.

3. Duomenų bazių projektavimas

3.1. Įvadas

Šiame skyriuje aptarsime reliacinių duomenų bazių projektavimą. Duomenų bazės projektavimo rezultatas yra loginė DB struktūra. Sprendžiant šį uždavinį ieškoma ne bet kokios DB struktūros, o tokios, kuri turėtų kiek galima mažiau blogų ir kiek galima daugiau gerų savybių. Duomenų bazės kokybei įvertinti taikomi formalūs metodai. Kita vertus, formaliai aptikus nepageidautinas DB struktūros savybes, gana dažnai, galima jas formaliai ir pašalinti. Šiame skyriuje aptarsime, kaip įvertinti DB struktūros kokybę ir kaip projektuoti DB, kad ji neturėtų žinomų nepageidautinų savybių.

Kai kurios reliacinio modelio sąvokos jau buvo pateiktos ankstesniuose skyriuose. Šiame skyriuje reliacinį modelį aptarsime detaliau ir formaliau.

3.2. Pagrindinės reliacinio modelio sąvokos

Reliaciniame modelyje duomenys pateikiami lentelėmis. Matematikoje lentelę atitinka santykis. **Santykis** (angl. *relation*) šiuo atveju išreiškiamas dvimate lentele, susidedančia iš eilučių ir stulpelių. Lentelės stulpelis reliaciniame modelyje dažnai vadinamas **atributu**. Stulpelis turi pavadinimą - tai atributo vardas. Reliacinėje teorijoje laikomasi nuostatos, kad tiek stulpelių, tiek ir eilučių tvarka lentelėje yra neapibrėžta. Visų leistinų atributo reikšmių aibė vadinama **domenu**. **Tuščioji reikšmė** (žymima `NULL`) – tai reikšmė, kuri priskiriama atributui eilutėje, kai atributo reikšmė nežinoma arba atributas yra neprasmingas. Tarkime, duomenų bazėje *Darbai* kai kurie asmenys (projektų vykdytojai) yra be aukštojo išsilavinimo. Tada atributas "baigta aukštoji mokykla" tokiems asmenims netenka prasmės.

Atributų rinkinys (aibė), vienareikšmiškai apibrėžiantis (nusakantis) kiekvieną lentelės eilutę, vadinamas **viršrakčiu**. Lentelės **raktu** vadinamas viršraktis, iš kurio pašalinus bent vieną atributą jis nustoja būti viršrakčiu. Taigi raktas – tai minimalus viršraktis. Raktą taip pat galima apibrėžti kaip minimalią atributų aibę, vienareikšmiškai apibrėžiančią (kitaip dar sakoma funkciškai apibrėžiančią) kiekvieno atributo reikšmę eilutėje. Kitaip tariant, **lentelės** *L* **raktu** vadinamas toks jos atributų aibės poaibis *K*, kad:

1) rakto atributų reikšmės vienareikšmiškai apibrėžia visų lentelės *L* atributų reikšmes, t.y. dvi skirtingos lentelės eilutės negali sutapti pagal visų aibės *K* atributų reikšmes (vienareikšmiška identifikacija),
2) joks aibės *K* poaibis neturi vienareikšmiškos identifikacijos savybės (pertekliaus nebuvimas).

Duomenų bazės *Darbai* lentelėje *Vykdytojai* atributų rinkinys {*Nr*, *Pavardė*} yra lentelės viršraktis, kadangi pagal bet kokias šių atributų reikšmes lentelėje galima surasti ne daugiau kaip vieną eilutę. Tačiau, ši aibė nėra raktas, nes jos poaibis {*Nr*} irgi yra viršraktis. Kadangi pastarąją aibę sudaro tik vienas atributas, tai ji yra lentelės raktas. Užrašant aibę, riestinius skliaustelius praleisime, kai aibę sudarys tik vienas elementas.

Lentelėje gali būti keli atributų rinkiniai, turintys rakto savybę. Visi jie vadinami **galimais raktais**, arba tiesiog raktais. Pavyzdžiui, stulpelis *Pavardė* yra galimas lentelės raktas. Tačiau, taip bus tik tuomet, kai pavardės nesikartos, t.y. nebus dviejų asmenų vienodomis pavardėmis. Paprastai tokios prielaidos daryti negalima, todėl *Pavardė* neturėtų būti raktu. Raktas, kurį sudaro du ar daugiau atributų vadinamas **sudėtiniu raktu**.

Paprastai vienas iš raktų paskelbiamas **pirminiu raktu**. Jei lentelėje yra keli raktai, tai pirminiu parenkamas tas, kuriuo paprasčiausia naudotis, pavyzdžiui, trumpiausias. Dažniausiai, raktu vadinsime pirminį raktą. Pavyzdžiui, lentelė *Vykdymas* turi vieną sudėtinį raktą {*Vykdytojas*, *Projektas*}, kuris yra ir vienintelis galimas lentelės pirminis raktas.

Lentelės reliacinė schema (struktūra, aprašas) - tai lentelės ir visų jos stulpelių pavadinimai, su pažymėtu (išskirtu, pvz., pabrauktu) pirminiu raktu. Pavyzdžiui, DB *Darbai* lentelių reliacines schemas galima pavaizduoti taip:

Vykdytojai (*Nr, Pavardė, Kvalifikacija, Kategorija, Išsilavinimas*);
Projektai (*Nr, Pavadinimas, Svarba, Pradžia, Trukmė*);
Vykdymas (*Projektas, Vykdytojas, Statusas, Valandos*).

Atkreipsime dėmesį į tai, kad lentelės *Projektai* raktą sudaro du stulpeliai, todėl ir pabraukti. Tai jokiu būdu nereiškia, kad kiekvienas iš šių stulpelių, atskirai paimtas, irgi yra raktas. Lentelė turi tik vieną raktą, sudarytą iš dviejų stulpelių.

Dažnai dviejose DB lentelėse yra vienas ar net keli stulpeliai, turintys tą pačią prasmę abiejose lentelėse. Pavyzdžiui, lentelės *Vykdytojas* stulpelio *Nr* ir lentelės *Vykdymas* stulpelio *Vykdytojas* galimų reikšmių aibės pagal stulpelių prasmę sutampa. Abiejų šių stulpelių reikšmės yra vykdytojų tapatumo numeriai. Gana dažnai atitinkami stulpeliai turi panašius, ar net vienodus, pavadinimus. Tokia atributų pora yra išorinio rakto pavyzdys. **Išoriniu raktu** vadinamas vienos lentelės atributų rinkinys, kuris kitoje lentelėje (ar net toje pačioje) yra pirminis raktas. Išorinio rakto atributų pavadinimai nebūtinai turi sutapti su pirminio rakto atributų pavadinimais. Kadangi išorinis raktas paprastai siejamas su lentelės pirminiu raktu, tai jam apibrėžti visiškai pakanka nurodyti išorinį raktą sudarančius atributus ir pavadinimą lentelės, į kurios pirminį raktą išorinis raktas rodo.

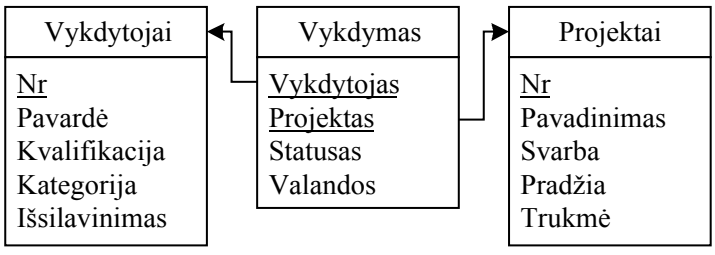
Išorinis raktas vartojamas loginiams ryšiams tarp lentelių apibrėžti. Informacija apie išorinius raktus yra svarbi DB loginei struktūrai. Todėl **duomenų bazės reliacine schema** vadinsime visų jos lentelių reliacinių schemų rinkinį kartu su lentelių išoriniais raktais. Pavyzdžiui, DB *Darbai* reliacinę schemą galima pavaizduoti taip:

Vykdytojai (*Nr, Pavardė, Kvalifikacija, Kategorija, Išsilavinimas*);
Projektai (*Nr, Pavadinimas, Svarba, Pradžia, Trukmė*);
Vykdymas (*Projektas, Vykdytojas, Statusas,Valandos*),
išoriniai raktai: *Projektas* nurodo į *Projektai*,
Vykdytojas nurodo į *Vykdytojai*.

Pastebėsime, kad duomenų bazėje *Darbai* tik viena lentelė *Vykdymas* turi du išorinius raktus, kitos dvi lentelės išorinių raktų neturi.

DB reliacinę schemą galima pavaizduoti ir grafiškai. Grafiškai pavaizduota DB reliacinė schema tampa aiškesnė, ypač kai duomenų bazę sudaro daug išoriniais raktais tarpusavyje susijusių lentelių. Kiekvienos lentelės schema vaizduojama stačiakampiu, kurio viršuje užrašomas lentelės pavadinimas. Žemiau išvardijami visi lentelės atributai (stulpelių pavadinimai). Pirminį raktą sudarantys atributai pabraukiami. Pagrindinis grafinio vaizdavimo privalumas - galimybė grafiškai pavaizduoti ryšius tarp lentelių. Lentelės išoriniai raktai vaizduojami rodykle iš išorinį raktą sudarančių atributų į lentelės, į kurią išorinis raktas nurodo, pavadinimą.

DB *Darbai* reliacinę schemą galima pavaizduoti taip:



3.1 pav. Duomenų bazės *Darbai* reliacinė schema.

3.3. Duomenų vientisumo sąlygos

Reliacinėje teorijoje yra keletas reikalavimų, kuriuos turi atitikti duomenų bazės duomenys. Viena reikalavimų rūšis vadinama **duomenų vientisumo sąlygomis**. Aptarsime tris iš jų:

- kategorijų vientisumas;
- nuorodų vientisumas;

- funkciniai sąryšiai.

Reliacinės lentelės eilutės yra konkrečių realaus pasaulio objektų atvaizdai duomenų bazėje. Pavyzdžiui, kiekviena lentelės *Vykdytojai* eilutė atitinka konkretų firmos darbuotoją. Realaus pasaulio objektai, kurie vaizduojami lentelės eilutėmis, reliacinėje teorijoje vadinami **kategorijomis**. Lentelės raktas vienareikšmiškai nusako lentelės eilutę, tuo pačiu ir kategoriją. Norėdamas rasti duomenis apie konkrečią kategoriją (objektą), vartotojas privalo žinoti rakto reikšmes. Tai reškia, kad kategorija neturi prasmės duomenų bazėje, jei bent vieno rakto atributo reikšmė yra nežinoma. Todėl pagal **kategorijų vientisumo reikalavimą** joks lentelės rakto atributas nė vienoje eilutėje negali turėti `NULL` reikšmės.

Jau minėjome, kad reliacinėje duomenų bazėje vienos lentelės eilučių ryšiui su kitos lentelės eilutėmis vartojami išoriniai raktai. Pavyzdžiui, duomenų bazės *Darbai* lentelės *Vykdymas* kiekvienoje eilutėje yra duomenys apie tai, kiek konkretus vykdytojas konkrečiam projektui vykdyti skiria valandų. Stulpelis *Vykdytojas* šioje lentelėje vartojamas konkrečiam vykdytojui nurodyti. Pagal jo reikšmę lentelėje *Vykdytojai* galima rasti (naudojantis šios lentelės raktu) visus duomenis apie tą vykdytoją. Todėl labai svarbu, kad kiekvienos eilutės atributo *Vykdytojas* reikšmė lentelėje *Vykdymas* atitiktų kurią nors atributo *Nr* reikšmę lentelėje *Vykdytojai*. Kitaip lentelėje *Vykdymas* turėsime nuorodą į nežinomą (neapibrėžtą) vykdytoją. Duomenų bazė, kurioje visi netušti išoriniai raktai nurodo į egzituojančią pirminio rakto reikšmę, tenkina nuorodų vientisumo reikalavimą. Pastebėsime, kad lentelėje išorinis raktas gali būti tuščias (jo atributų reikšmės yra `NULL`), jei tik jo atributai neįeina į pirminį raktą. **Nuorodų vientisumo reikalavimas** gali būti suformuluotas taip: kiekvieno išorinio rakto reikšmė duomenų bazėje turi būti arba tuščia, arba sutapti su viena pirminio rakto reikšme lentelėje, į kurią išorinis raktas nurodo.

Trečiąjį reikalavimą duomenims - funkcinius sąryšius - detaliau aptarsime kituose skyreliuose.

3.4. Duomenų anomalijos

Tarkime, vietoje dviejų DB *Darbai* lentelių *Vykdymas* ir *Projektai* turime vieną lentelę *Projektai_Vykdymas*. Tuomet DB reliacinė schema yra tokia:

Vykdytojai (*Nr, Pavardė, Kvalifikacija, Kategorija, Išsilavinimas*),
Projektai_Vykdymas (*Projektas, Pavadinimas, Svarba, Trukmė, Pradžia, Vykdytojas, Statusas, Valandos*),
išorinis raktas: *Vykdytojas* nurodo į *Vykdytojai*.

Užpildžius lentelę *Projektai_Vykdymas* duomenimis, kurie buvo dviejose lentelėse *Projektai* ir *Vykdymas*, naujoji lentelė bus tokia:

Projektai_Vykdymas

Projektas	*Pavadinimas*	*Svarba*	*Trukmė*	*Vykdytojas*	…
1	Studentų apskaita	Maža	12	1	
1	Studentų apskaita	Maža	12	2	
1	Studentų apskaita	Maža	12	3	
1	Studentų apskaita	Maža	12	4	
2	Buhalterinė apskaita	Vidutinė	10	1	
2	Buhalterinė apskaita	Vidutinė	10	2	
2	Buhalterinė apskaita	Vidutinė	10	4	
3	WWW svetainė	Didelė	6	1	
3	WWW svetainė	Didelė	6	2	
3	WWW svetainė	Didelė	6	3	

Lentelę supaprastinome pavaizduodami ne visus jos duomenis. Dešiniojo stulpelio, kuris pažymėtas daugtaškiu, vietoje turėtų būti pavaizduotos atributų *Pradžia, Statusas* ir *Valandos* reikšmės.

Nesunku pastebėti, kad lentelė *Projektai_Vykdymas* sudaryta nesėkmingai. Pavyzdžiui, keturiose eilutėse, atitinkančiose projektą Nr. 1, kartojasi tas pats projekto pavadinimas, jo

svarba ir trukmė. Tokie **pertekliniai duomenys** ne tik užima papildomą vietą kompiuterio atmintyje – dėl jų gali būti prarastas duomenų vientisumas, arba, kitaip tariant, atsirasti duomenų prieštaringumas. Mat tada vieną projektą gali vykdyti keli vykdytojai. Tarkime, projekto Nr. 1 trukmė pasikeitė, ir naujoji trukmė buvo pakeista tik vienoje eilutėje. Tuomet tarp eilučių, kuriose yra informacija apie tą patį projektą Nr. 1, atsiranda neatitikimas, kuris vadinamas atnaujinimo anomalija. **Atnaujinimo anomalija** – tai duomenų prieštaringumas, atsirandantis dėl duomenų pertekliaus, atnaujinus tik dalį jų.

Tarkime, vykdytojai, kurių numeriai yra 1, 2 ir 3 išeina iš darbo. Tada iš lentelės pašalinamos visos eilutės, atitinkančios šiuos vykdytojus. Todėl iš lentelės *Projektai_Vykdymas*, o kartu ir iš duomenų bazės, pašalinami visi duomenys apie projektą Nr. 3, nes jį vykdė tik tie iš darbo išėję darbuotojai. Tai nėra gerai, nes projektą vis tiek reikės tęsti, paskyrus dirbti kitus darbuotojus. Tai **pašalinimo anomalija** – nenumatytas duomenų praradimas, susijęs su kitų duomenų pašalinimu.

Kita vertus, firma gali gauti užsakymą naujam projektui vykdyti, kai dar nėra paskirtas nė vienas projekto vykdytojas. Kadangi vykdytojo numeris, tiksliau, atributas *Vykdytojas*, yra lentelės *Projektai_Vykdymas* rakto dalis, tai, kol nebus paskirtas bent vienas vykdytojas, mes negalėsime įvesti į duomenų bazę duomenų apie naująjį projektą. Tai **įvedimo anomalija** – nebuvimas galimybės įvesti duomenis dėl to, kad trūksta kitų duomenų.

Duomenų bazė turi būti sudaryta taip, kad atnaujinimo, pašalinimo ir įvedimo anomalijų nebūtų. Todėl duomenų bazėje *Darbai* vietoj lentelės *Projektai_Vykdymas* turi būti dvi lentelės *Projektai* ir *Vykdymas,* kaip tai buvo padaryta anksčiau. Tačiau tai tik intuityvus problemos sprendimas, kuris, esant duomenų bazėje daug lentelių ir atributų, gali nebūti toks akivaizdus. Tolimesniuose skyreliuose mes apibrėšime formalesnį metodą, vadinamą lentelių skaidymu, kuris duos tokį patį rezultatą. **Lentelių skaidymu** vadinamas lentelės padalijimas į kelias lenteles, siekiant išvengti anomalijų ir neprarasti duomenų vientisumo. Tam naudojamos normalinės formos ir lentelių struktūrizavimo taisyklės.

3.5. Pirmoji normalinė forma

Normaline forma (sutrumpintai, **NF**) vadinami apribojimai duomenų bazės reliacinei schemai, kuriuos taikant DB išvengia tam tikrų nepageidaujamų savybių.

Lentelė yra **pirmos normalinės formos** (1NF), jei visų jos atributų reikšmės yra atomai. **Atomu** laikoma reikšmė, kuri nėra nei aibė, nei sąrašas. Paprastai apibėžiant reliacinę lentelę reikalaujama, kad visos visų atributų reikšmės būtų atomai. Todėl toliau visas nagrinėjamas lenteles laikysime attinkančiomis 1NF. 1NF sąvoką paaiškinsime, nagrinėdami pavyzdį lentelės, kuri neatitinka 1NF.

Imkime ankstesniame skyrelyje įvestą lentelę *Projektai_Vykdymas.* Nustatėme, kad šiai lentelei būdingos duomenų anomalijos. Visos įvardytos duomenų anomalijos gali būti pašalintos, išskaidžius lentelę į dvi lenteles. Pabandykime problemą spręsti kitaip. Anomalijų priežastis yra ta, kad vieną projektą gali vykdyti keli vykdytojai, ir dėl to lentelėje *Projektai_Vykdymas* atsiranda duomenų perteklius. Šią lentelę pertvarkykime į lentelę *Projektai_Vykdytojai*:

Projektai_Vykdytojai(*Projektas, Pavadinimas, Svarba, Trukmė, Pradžia, Vykdytojai, Statusas, Valandos*).

Pastarojoje lentelėje atributo *Vykdytojai* reikšmės yra visų projekto vykdytojų numerių aibė. Naująją lentelę galima pavaizduoti taip:

Projektai_Vykdytojai

Projektas	*Pavadinimas*	*Svarba*	*Trukmė*	*Vykdytojai*	…
1	Studentų apskaita	Maža	12	{1,2,3,4}	
2	Buhalterinė apskaita	Vidutinė	10	{1,2,4}	
3	WWW svetainė	Didelė	6	{1,2,3}	

Iš pirmo žvilgsnio gali pasirodyti, kad ši lentelė sudaryta gerai, nes joje nėra duomenų pertekliaus. Tikriausiai tai vienintelis jos privalumas. Visų pirma, šiai lentelei negalima

apibrėžti išorinio rakto, nes lentelės atributas *Vykdytojai* negali būti susietas su lentelės *Vykdytojai* pirminiu raktu, kaip buvo anksčiau, nors ryšys tarp jų yra. Mat vykdytojų numerių aibė nėra palyginama su vienu numeriu. Nesant išorinio rakto, stulpelyje *Vykdytojai* gali atsirasti numeriai, neatitinkantys nė vieno vykdytojo, užregistruoto lentelėje *Vykdytojai*.

Lentelė *Projektai_Vykdytojai* nėra pirmos normalinės formos, nes jos stulpelyje *Vykdytojai* yra aibės, kurių elementai kitoje lentelėje yra atributo reikšmės. Jei lentelę *Projektai_Vykdytojai* pakeistume lentele *Projektai_Vykdymas*, turėtume lentelę, atitinkančią 1NF. Tačiau jau anksčiau padarėme išvadą, kad lentelė *Projektai_Vykdymas* nėra gerai sudaryta, nors ji ir yra 1NF. Taigi reikalavimas lentelei atitikti 1NF yra per silpnas, kad lentelės schemą būtų galima vadinti pakankamai gera. Todėl nagrinėjamos ir aukštesnės eilės normalinės formos. Kuo aukštesnė NF, tuo aukštesni reikalavimai lentelės schemai. Taigi kuo aukštesnės NF reikalavimus tenkina lentelės schema, tuo mažiau blogų savybių ji turi. Duomenų bazė yra tam tikros NF, jei visos jos lentelės yra tos NF. Aukštesnėms NF apibrėžti vartojama funkcinio sąryšio sąvoka.

3.6. Funkciniai sąryšiai

Jau aptarėme kategorijų ir nuorodų vientisumo sąlygas. Dar griežtesnius reikalavimus reliacinei schemai apibrėžia funkciniai sąryšiai. Tai bene svarbiausias apribojimas. Jo esmė - vienų atributų reikšmės eilutėje gali vienareikšmiškai apibrėžti kitų atributų reikšmes. Pavyzdžiui, kiekvienoje lentelės *Projektai_Vykdymas* eilutėje *Projektas* vienareikšmiškai apibrėžia atributų *Pavadinimas*, *Svarba, Trukmė* ir *Pradžia* reikšmes.

Tokiu būdu, jei atributų aibės A reikšmės eilutėje vienareikšmiškai apibrėžia atributų aibės B reikšmes eilutėje, tai sąryšį tarp A ir B vadiname funkciniu. Formaliau, **funkcinis sąryšis** (**f-sąryšis**) gali būti apibrėžtas taip: jei A ir B yra lentelės L atributų aibės, tai užrašas $\boldsymbol{A \rightarrow B}$ reiškia, jog jei dvi lentelės L eilutės turi vienodas atributų A reikšmes, tai atributų B reikšmės taip pat sutampa. Užrašą $A \rightarrow B$ galima skaityti dvejopai: B **funkciškai priklauso** nuo A, taip pat A **funkciškai apibrėžia** B. F-sąryšio kairiosios dalies atributai vadinami **determinantu**. Determinantas – tai atributai, kurių reikšmės apibrėžia kitų atributų reikšmes.

Jau minėti lentelės *Projektai_Vykdymas* atributų tarpusavio sąryšiai yra f-sąryšių pavyzdžiai, kuriuos galima užrašyti taip:

$Projektas \rightarrow Pavadinimas,$
$Projektas \rightarrow Svarba,$
$Projektas \rightarrow Trukmė,$
$Projektas \rightarrow Pradžia.$

Šį keturių funkcinių sąryšių užrašą galima pakeisti vienu trumpesniu:

$Projektas \rightarrow \{Pavadinimas, Svarba, Trukmė, Pradžia\}$.

Be šių f-sąryšių, lentelėje *Projektai_Vykdymas* galioja dar ir toks:

$\{Projektas, Vykdytojas\} \rightarrow \{Statusas, Valandos\}$.

Funkcinio sąryšio sąvoka yra sąvokos "lentelės viršraktis" apibendrinimas. Lentelės L viršraktis – tai toks jos atributų aibės R poaibis S ($S \subseteq R$), kad $S \rightarrow R$. Faktą, kad lentelės L atributų aibė yra R, žymesime $L(R)$, kuris atitinka lentelės L schemos užrašą be lentelės pirminio rakto.

Atkreipsime dėmesį į tai, kad f-sąryšis, kaip ir kiti apribojimai duomenims, galioja reliacinei schemai, o ne apie konkretiems lentelės duomenims. Pagal einamąjį lentelės turinį neįmanoma nustatyti, ar koks nors f-sąryšis teisingas (būdingas) schemai. Pavyzdžiui, iš turimų duomenų lentelėje *Projektai_Vykdymas* būtų galima daryti išvadą, kad šios lentelės schemai būdingas f-sąryšis: $Svarba \rightarrow Trukmė$, kadangi jis tinka visoms lentelės eilutėms. Tačiau taip tvirtinti negalima, nes kitu metu gali būti vykdomi du (ar daugiau) projektai, kurių svarba vienoda, tačiau trukmė skirtinga. Taigi kai sakoma, kad lentelei būdingas konkretus f-sąryšis, tai turima omenyje, kad tas sąryšis galioja bet kuriuo momentu.

F-sąryšis $A_1, A_2,\ldots, A_n \to B$ vadinamas **trivialiu**, jei B sutampa su vienu iš aibės $\{A_1, A_2,\ldots, A_n\}$ atributų. Pastebėsime, kad bet kuris trivialus f-sąryšis yra teisingas bet kuriam santykiui. Bendru atveju f-sąryšis $A_1, A_2,\ldots, A_n \to B_1, B_2,\ldots, B_m$:

- **trivialus**, jei $\{B_1, B_2,\ldots, B_m\}$ yra aibės $\{A_1, A_2,\ldots, A_n\}$ poaibis;
- **netrivialus**, jei egzistuoja bent vienas $B_i \in \{B_1, B_2,\ldots, B_m\}$, toks, kad $B_i \notin \{A_1, A_2,\ldots, A_m\}$;
- **visiškai netrivialus**, jei $\forall i : 1,\ldots,m : B_i \notin \{A_1, A_2,\ldots, A_m\}$.

Labai svarbi *f*-sąryšių savybė yra ta, kad nesudėtinga sistema gali nuolat automatiškai sekti, ar apibrėžti (žinomi sistemai) sąryšiai yra tenkinami (nėra pažeisti). Pvz., kiekvieną kartą įterpiant naujus duomenis į lentelę bei keičiant esamus, galima patikrinti, ar įvykdžius operaciją nebus pažeistas kuris nors f-sąryšis. Pateiksime paprastą algoritmą, sprendžiantį šį uždavinį vienam atskirai paimtam sąryšiui.

Algoritmas patikrinti, ar lentelė L konkrečiu momentu tenkina *f*-sąryšį $A \to B$.

1) lentelės L eilutes sugrupuojame pagal atributų A reikšmes taip, kad eilutės su vienodomis A reikšmėmis priklausytų tai pačiai grupei.
2) jei kiekvienai eilučių grupei su vienodomis A reikšmėmis atributų B reikšmės taip pat sutampa, tai L tenkina f-sąryšį $A \to B$, kitaip - netenkina.

Visiškai nesunku pastarąjį algoritmą apibendrinti, kad vietoj vieno sąryšio būtų tikrinama, ar lentelė tenkina pasirinktą *f*-sąryšių aibę.

3.7. Funkcinių sąryšių uždarinys

Jau minėjome, kad vieni f-sąryšiai gali būti pakeisti kitais. Tarkime, turime lentelę su trimis atributais *A, B* ir *C*, tarp kurių galioja sąryšiai: $A \to B$ ir $B \to C$. Nesunku pastebėti, kad tuomet toje lentelėje galios ir sąryšis $A \to C$, vadinamas **tranzityviuoju** f-sąryšiu, t.y. C tranzityviai priklauso nuo A. Aibė visų galimų f-sąryšių, kurie apibrėžiami duotąja f-sąryšių aibe F, vadinama **aibės *F* uždariniu** ir žymima F^+. Armstrongas (W.W. Armstrong) pirmasis suformulavo išvedimo taisykles, kuriomis galima gauti visus aibės F^+ f-sąryšius. Suformuluosime šias taisykles, kurios dar vadinamos Armstrongo aksiomomis.

Armstrongo aksiomos. Tarkime *A, B* ir *C* yra reliacinės lentelės (santykio) *L(R)* atributų aibės R poaibiai, o užrašas AB reiškia aibių A ir B sąjungą.

1. Refleksyvumas: jei $B \subseteq A$, tai $A \to B$.
2. Papildymas: jei $A \to B$, tai $AC \to BC$.
3. Tranzityvumas: jei $A \to B$ ir $B \to C$, tai $A \to C$.

Kiekviena iš šių taisyklių gali būti įrodyta remiantis f-sąryšio apibrėžimu. Be to, ši aibė yra pilnoji, t.y. duotai f-sąryšių aibei F bet koks f-sąryšis, priklausantis F^+, gali būti išvestas naudojant tik šias taisykles. Tam, kad turimai aibei F būtų paprasčiau išvesti aibę F^+, šios trys taisykės yra papildomos dar keliomis.

4. Savęs apibrėžimas: $A \to A$.
5. Dekompozicija: jei $A \to BC$, tai $A \to B$ ir $A \to C$.
6. Apjungimas: jei $A \to B$ ir $A \to C$, tai $A \to BC$.
7. Kompozicija: jei $A \to B$ ir $C \to D$, tai $AC \to BD$, čia $D \subset R$.

Pastarosios keturios taisyklės gali būti išvestos iš pirmųjų trijų, visos jos įvestos tik patogumo dėlei.

Turint duotai reliacinei lentelei $L(R)$ galiojančių f-sąryšių aibę F, galima ieškoti lentelės raktų. Išspręsime šiek tiek paprastesnį uždavinį - sudarysime procedūrą, kuria patikrinsime, ar pasirinktas lentelės atributų aibės poaibis yra viršraktis. Priminsime, kad lentelės raktas – tai virsraktis, iš kurio negalima pašalinti jokio atributo, išsaugant viršrakčio savybę, o viršraktis – tai jos atributų aibės R poaibis S ($S \subseteq R$), kuriam $S \to R$. Galimybė patikrinti, ar atributų aibės poaibis S yra viršraktis, labai palengvintų raktų paieškos uždavinio sprendimą. Tam reikia rasti būdą sudaryti aibę visų atributų, kurie funkciškai priklauso nuo S, aibę, kuri

vadinama a**tributų aibės *S* uždariniu** f-sąryšių aibei *F*. Atributų aibės *S* uždarinys žymimas S^+.

Algoritmas atributų aibės *S* uždariniui S^+ f-sąryšių aibės *F* atžvilgiu rasti.
$S^+ := S$;
while (true)
  $T := S^+$;
  for each $X \to Y \in F$
    if $X \subseteq S^+$ then $S^+ := S^+ \cup Y$;
  endfor
  if $T \equiv S^+$ then
    leave loop ;
endwhile .

Pastebėsime, kad duotai f-sąryšių aibei *F* nesunku patikrinti, ar f-sąryšį $X \to Y$ galima išvesti iš *F*, nes taip gali būti tada ir tik tada, kai $Y \subseteq X^+$.

Dvi f-sąryšių aibės *F* ir *G* yra vadinamos **ekvivalenčiomis**, jei $F^+ \equiv G^+$.

Naudodamiesi, naujai įvestomis sąvokomis, dar kartą apibrėšime lentelės raktą. Lentelės *L*(*R*), kurioje galioja f-sąryšių aibė *F*, raktas yra toks jos atributų aibės *R* poaibis *K* ($K \subseteq R$), kad:

1) $K \to R$ (vienareikšmiška identifikacija),
2) $\forall B \subset K : B \to R \notin F^+$ (pertekliaus nebuvimas).

F-sąryšis yra semantinė sąvoka. F-sąryšiai susiję su duomenų prasme. Nurodysime svarbiausias priežastis, kodėl f-sąryšiai yra svarbūs reliacinėse DBVS:

- f-sąryšiais galima aprašyti gana svarbius reikalavimus duomenims (loginius ryšius tarp duomenų), kurių nepaisant duomenys gali tapti prieštaringais;
- f-sąryšiais galima išreikšti svarbias DB loginės struktūros savybes, kurios palengvina DB struktūros projektavimą ir įgalina gauti geresnį rezultatą;
- f-sąryšius gana paprasta išreiškšti, todėl atitinkamos priemonės įtrauktos į SQL kalbą, naudojant juos kaip pirminius ir išoriniais raktus bei indeksus;
- DBVS, prieš atlikdama kiekvieną duomenų modifikavimą, gali patikrinti, ar operacijos rezultate nebus pažeistas kuris nors f-sąryšis, tuo pačiu užkirsti galimybę duomenų bazei patekti į būseną, kai kuris nors f-sąryšis bus nepatenkintas.

3.8. Antroji normalinė forma

Atributų aibė *B* yra **visiškai priklausoma** nuo atributų aibės *A*, *f*-sąryšių aibės *F* atžvilgiu, jei *B* - funkciškai priklausoma nuo visos aibės *A*, bet nėra funkciškai priklausoma nei nuo jokio *A* poaibio. Formaliau, atributų aibė *B* visiškai priklausoma nuo atributų aibės *A*, jei $A \to B \in F^+$, tai $\forall C \subset A : C \to B \notin F^+$.

Lentelės *L*(*R*) atributas *a* ($a \in R$) vadinamas **pirminiu** lentelės *L* **atributu** *f*-sąryšių aibės *F* atžvilgiu, jei *a* priklauso kuriam nors lentelės *L* raktui, priešingu atveju *a* vadinamas **nepirminiu atributu**.

Lentelė *L*(*R*) yra **antros normalinės formos** (2NF) *f*-sąryšių aibės *F* atžvilgiu, jei ji yra 1NF ir kiekvienas jos nepirminis atributas visiškai priklauso nuo kiekvieno lentelės *L* rakto. **Duomenų bazė yra 2NF** *f*-sąryšių aibės *F* atžvilgiu tada ir tik tada, kai visos jos lentelės yra 2NF atžvilgiu *F*.

Paanalizuokime aukščiau pateiktą lentelę *Projektai_Vykdymas:*

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

Kaip jau nustatėme, šioje lentelėje galioja tokie f-sąryšiai:

$Projektas \to \{Pavadinimas, Svarba, Trukmė, Pradžia\}$, (3.1)
$\{Projektas, Vykdytojas\} \to \{Statusas, Valandos\}$. (3.2)

Ši lentelė turi vienintelį raktą {*Projektas, Vykdytojas*}. Pateiktieji du sąryšiai vienareikšmiškai nusako šį lentelės raktą, nes:

pagal refleksyvumo taisyklę –
$$\{Projektas, Vykdytojas\} \rightarrow Projektas, \quad (3.3)$$
iš (3.1) ir (3.3) pagal tranzityvumo taisyklę –
$$\{Projektas, Vykdytojas\} \rightarrow \{Pavadinimas, Svarba, Trukmė, Pradžia\}, \quad (3.4)$$
pagal refleksyvumo taisyklę –
$$\{Projektas, Vykdytojas\} \rightarrow Vykdytojas, \quad (3.5)$$
iš (3.2) - (3.5) pagal apjungimo taisyklę -
$$\{Projektas, Vykdytojas\} \rightarrow \{ Projektas, Pavadinimas, Svarba, Trukmė, Pradžia, Vykdytojas, Statusas, Valandos\} . \quad (3.6)$$

F-sąryšis (3.6) reiškia, kad atributų aibė {*Projektas, Vykdytojas*} yra lentelės *Projektai_Vykdymas* viršraktis. Kitaip tariant, $\{Projektas, Vykdytojas\}^+ \equiv \{Projektas, Pavadinimas, Svarba, Trukmė, Pradžia, Vykdytojas, Statusas, Valandos\}$. Kadangi atributų aibė {*Projektas, Pavadinimas, Svarba, Trukmė, Pradžia, Vykdytojas, Statusas, Valandos*} nėra nei atributo *Projektas*, nei atributo *Vykdytojas* uždarinys, tai {*Projektas, Vykdytojas*} yra ne tik viršraktis, bet ir raktas.

Vadinasi, lentelės *Projektai_Vykdymas* atributai *Projektas, Vykdytojas* yra pirminiai atributai, o visi kiti atributai: *Pavadinimas, Svarba, Trukmė, Pradžia, Statusas, Valandos* – nepirminiai. Atributai *Pavadinimas, Svarba, Trukmė, Pradžia* nevisiškai priklauso nuo lentelės rakto, jie funkciškai priklauso nuo atributo *Projektas*, kuris yra rakto dalis. Todėl lentelė *Projektai_Vykdymas* nėra 2NF. Jau anksčiau pastebėjome, kad šiai lentelei būdinga pertekliniai duomenys bei duomenų įvedimo, atnaujinimo ir šalinimo anomalijos. Šių nepageidautinų savybių nelieka išskaidžius lentelę į dvi lenteles *Projektai* ir *Vykdymas*. Abi lentelės *Projektai* ir *Vykdymas* yra 2NF. Tokiu būdu, 2NF sumažina duomenų perteklių ir pašalina duomenų anomalijas. Kita vertus, darbas su dviem lentelėmis yra šiek tiek sudėtingesnis, negu su viena. Prisiminkime, kad duomenų paieškai keliose lentelėse SQL turi galimybę jungti lenteles. Be to, kaip vėliau pamatysime, SQL leidžia apibrėžti virtualią lentelę, kuri didele dalimi imituoja neišskaidytą (apjungtą) lentelę. Tuomet darbas su dviem lentelėmis nedaug tesiskiria nuo darbo su viena lentele.

Lentelės, neatitinkančios 2NF, skaidymo procesas į kelias lenteles, atitinkančias 2NF, susideda iš kelių nesudėtingų žingsnių:

1) sukuriama nauja lentelė, kurios atributai yra pradinės lentelės atributai, įeinantys į f-sąryšį tarp nepirminių atributų (atributo) ir rakto dalies. Šio f-sąryšio determinantas tampa naujos lentelės raktu;
2) atributai, esantys dešinėje f-sąryšio pusėje, pašalinami iš pradinės lentelės;
3) jei pradinė lentelė nėra 2NF dėl daugiau nei vieno f-sąryšio, tai žingsniai 1 ir 2 kartojami kiekvienam tokiam sąryšiui.

Pavaizduosime pirmųjų dviejų žingsnių schemą. Tarkime, turime lentelę $L(\underline{A, B}, C, D)$, kurioje galioja f-sąryšiai: $AB \rightarrow CD$ ir $A \rightarrow D$. Tuomet lentelę L skaidome į dvi: $L_1(\underline{A, B}, C)$ ir $L_2(\underline{A}, D)$. Lentelėje L_1 galios f-sąryšis $AB \rightarrow C$, o lentelėje $L_2 - A \rightarrow D$.

Gali pasirodyti, kad naujose lentelėse galiojantys sąryšiai skiriasi nuo pradinėje lentelėje galiojusių. Tačiau taip nėra. Suskaidę lentelę nepraradome savybių, nes sąryšis $AB \rightarrow CD$ yra išvedamas iš $AB \rightarrow C$ ir $A \rightarrow D$, panaudojant kompozicijos taisyklę. Toks lentelių skaidymas, kai informacija neprarandama, vadinamas **dekompozicija be praradimo**. 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.

Lentelė, kurią sudaro tik kai kurie kitos lentelės stulpeliai, vadinama **projekcija**. Taigi ir lentelė L_1, ir lentelė L_2 yra lentelės L projekcijos. Tai, kad išskaidžius lenteles neprarandama informacijos, garantuoja Hezo (I.J. Heath) teorema.

Hezo teorema. Tarkime, lentelėje *L(A, B, C)* galioja f-sąryšis $A \to B$, kur *A, B* ir *C* – lentelės atributų aibės. Tuomet lentelę *L* galima gauti jungiant jos projekcijas $L_1(A, B)$ ir $L_2(A, C)$.

Šią teoremą įrodyti paliksime skaitytojui, kaip uždavinį.

3.9. Trečioji normalinė forma

Lentelės $L(R)$ atributų aibės R poaibis C ($C \subset R$) **tranzityviai priklauso** nuo atributų aibės A *f*-sąryšių aibės F atžvilgiu, jei egzistuoja toks atributų aibės R poaibis B, kad A funkciškai apibrėžia B, bet ne atvirkščiai, bei B funkciškai apibrėžia C atžvilgiu F, t.y. $\exists B \subset R : A \to B \in F^+$, $B \to A \notin F^+$, $B \to C \in F^+$ ir $C \not\subset A \cup B$.

Lentelė $L(R)$ yra **trečios normalinės formos (3NF)** *f*-sąryšių aibės F atžvilgiu, jei ji yra 1NF ir nėra nepirminių atributų, tranzityviai priklausančių nuo rakto. **DB yra 3NF** *f*-sąryšių aibės F atžvilgiu tada ir tik tada, kai visos jos lentelės yra 3NF F atžvilgiu.

Tarkime, duomenų bazėje *Darbai* mums reikia saugoti ir aukštųjų mokyklų, kurias baigė darbuotojai – projektų vykdytojai, adresus. Vienas iš būdų tam pasiekti - papildyti lentelę *Vykdytojai* nauju stulpeliu, pvz., *AM_Adresas*. Tarkime, vietoj lentelės *Vykdytojai* turime lentelę *Vykdytojai_AM*:

Vykdytojai_AM (*Nr, Pavardė, Kvalifikacija, Kategorija, Išsilavinimas, AM_Adresas*).

Užpildę šią lentelę duomenimis gausime:

Vykdytojai_AM

Nr	*Pavardė*	**…**	*Išsilavinimas*	*AM_Adresas*
1	Jonaitis		VU	Universiteto 3, Vilnius
2	Petraitis		VU	Universiteto 3, Vilnius
3	Gražulytė		NULL	NULL
4	Onaitytė		VDU	Donelaičio 58, Kaunas
5	Antanaitis		VU	Universiteto 3, Vilnius

Kad būtų paprasčiau, pavaizdavome ne visus lentelės stulpelius.

Kaip ir lentelėje *Vykdytojai,* stulpelis *Nr* yra vienintelis šios lentelės raktas. Lentelėje *Vykdytojai_AM* atributas *AM_Adresas* funkciškai priklauso nuo atributo *Išsilavinimas*. Taigi lentelėje *Vykdytojai_AM* turime du f-sąryšius:

$Nr \to \{Pavardė, Kvalifikacija, Kategorija, Išsilavinimas, AM\_Adresas\}$,
$Išsilavinimas \to AM\_Adresas$.

Kadangi lentelės *Vykdytojai_AM* raktą sudaro tik vienas atributas, tai negali egzistuoti nepirminių atributų, priklausančių nuo rakto dalies. Tai reiškia, kad ši lentelė yra 2NF. Nors lentelė *Vykdytojai_AM* yra 2NF, jai būdinga duomenų perteklius bei anomalijos.

Kadangi lentelėje *Vykdytojai_AM* galioja f-sąryšis $Nr \to Išsilavinimas$, negalioja f-sąryšis $Išsilavinimas \to Nr$ ir bet to *Išsilavinimas* funkciškai apibrėžia *AM_Adresas*, tai šioje lentelėje nepirminis atributas *AM_Adresas* tranzityviai priklauso nuo lentelės rakto *Nr*. Todėl, ši lentelė nėra 3NF. Duomenų pertekliaus bei anomalijų nelieka, išskaidžius lentelę į dvi: *Vykdytojai* ir naują lentelę *AM_Adresai*(*Pavadinimas*, *Adresas*):

AM_Adresai

Pavadinimas	*Adresas*
VU	Universiteto 3, Vilnius
VDU	Donelaičio 58, Kaunas

Tiek lentelė *Vykdytojai,* tiek ir lentelė *AM_Adresai* yra 3NF. 3NF sumažino duomenų perteklių ir panaikino duomenų anomalijas.

Pavaizduosime tranzityvios vienos nepirminio atributo priklausomybės nuo rakto likvidavimo schemą. Tarkime, turime lentelę *L*(*A, B, C*), kurioje nepirminis atributas (atributų aibė) *C* tranzityviai priklauso nuo rakto *A,* kitaip tariant, galioja f-sąryšiai: $A \to B$ ir $B \to C$,

bei negalioja f-sąryšis $B \rightarrow A$. Tokia lentelė nėra 3NF. Tam, kad DB būtų 3NF, lentelę L išskaidome į dvi: $L_1(\underline{A, B})$ ir $L_2(\underline{B, C})$. Lentelėje L_1 galios f-sąryšis $A \rightarrow B$, o lentelėje L_2 - $B \rightarrow C$. Lentelės L_2 raktas B gali būti pirminiu raktu, o lentelėje L_1 galima apibrėžti išorinį raktą, susiejant jos atributą B su L_2 pirminiu raktu.

Literatūroje galima aptikti 3NF apibrėžimą, kuriame vietoj reikalavimo lentelei atitikti 1NF, yra reikalavimas atitikti 2NF. Galima įsitikinti, kad šie du reikalavimai, apibrėžiant 3NF, yra ekvivalentiški.

Teorema. Bet kuri lentelė, esanti 3NF *f*-sąryšių aibės F atžvilgiu, yra ir 2NF F atžvilgiu.

Įrodymas. Įrodysime prieštaros būdu. Tarkime, lentelė $L(R)$ yra 3NF, bet nėra 2NF *f*-sąryšių aibės F atžvilgiu. Tuomet $\exists a \in R : a$ - nepirminis ir a funkciškai priklauso nuo kurio nors rakto K dalies K', $K' \subset K \subseteq R$. Tai reiškia, kad iš F galima išvesti $K' \rightarrow a$ ($K' \rightarrow a \in F^+$) ir negalima išvesti $K' \rightarrow K$ ($K' \rightarrow K \notin F^+$), nes kitaip K' būtų L raktas, o ne K. Be to, $a \notin K$, kadangi K - raktas, o a - nepirminis. Taigi turime: $K \rightarrow K'$, ne $K' \rightarrow K$, $K' \rightarrow a$ bei $a \notin K$ ir juo labiau $a \notin K'$, t.y. $a \notin K \cup K'$. Tokiu būdu įrodėme, kad nepirminis atributas a tranzityviai priklauso nuo rakto K, o tai reiškia, kad lentelė $L(R)$ nėra 3NF; tai, savo ruožtu, prieštarauja prielaidai, padarytai įrodymo pradžioje.

Įrodę teoremą įsitikinome, kad dalinė priklausomybė reiškia ir tranzityvią priklausomybę. Kitaip tariant, jei lentelė nėra 2F, tai joje egzistuoja nepirminis atributas, tranzityviai priklausantis nuo rakto.

3.10. Boiso-Kodo normalinė forma

3NF, kaip ir 1NF bei 2NF, pirmasis apibrėžė Kodas (E.F.Codd). Kurį laiką buvo skaitoma, kad 3NF yra praktiškai pakankama. Vėliau paaiškėjo, kad 3NF nevisiškai tinka lentelėms, kurios turi du ar daugiau raktų, bent du iš jų yra sudėtiniai (sudaryti iš keleto atributų) raktai ir jie tarpusavyje kertasi (turi bent vieną bendrą atributą). Todėl tas 3NF apibrėžimas buvo pakeistas griežtesniu Boiso-Kodo (Boyce-Codd) apibrėžimu. Lentelė yra **Boiso-Kodo normalinės formos (BKNF)**, jei kiekvieno netrivialaus f-sąryšio determinantas yra raktas. Gana nesunku įsitikinti, kad lentelė, esanti BKNF, yra ir 3NF. BKNF įgavo tokį pavadinimą dėl to, kad jos suformulavimo metu jau buvo suformuluota 4NF. **DB yra BKNF** *f*-sąryšių aibės F atžvilgiu tada ir tik tada, kai visos jos lentelės yra BKNF F atžvilgiu.

Tarkime, projektai vykdomi etapais ir lentelėje *Etapai* yra saugoma informacija apie projektų etapų pabaigas:

Projektų_Etapai (*Nr, Pavadinimas, Etapas, Pabaiga*).

Šioje lentelėje pažymėtas vienas (pirminis) raktas, kurį sudaro du pabraukti stulpeliai. Tarus, kad negali būti dviejų projektų su vienodais pavadinimais, lentelė turi ir antrą raktą {*Pavadinimas, Etapas*}. Matome, kad abu raktai turi vieną bendrą stulpelį *Etapas*. Šiai lentelei būdingi tokie f-sąryšiai:

{*Pavadinimas, Etapas*} $\rightarrow$ {*Nr, Pabaiga*},
{*Nr, Etapas*} $\rightarrow$ {*Pavadinimas, Pabaiga*},
Nr $\rightarrow$ *Pavadinimas*,
Pavadinimas $\rightarrow$ *Nr*.

Išanalizavę šiuos f-sąryšius pastebėsime, kad lentelė *Projektų_Etapai* yra 3NF, tačiau jai būdingi aukščiau minėti požymiai. Užpildžius lentelę duomenimis nesunku pastebėti, kad lentelėje duomenys dubliuojami.

Projektų_Etapai

Nr	*Pavadinimas*	*Etapas*	*Pabaiga*
1	Studentų apskaita	1	2001.06.30
1	Studentų apskaita	2	2001.12.31
2	Buhalterinė apskaita	1	2001.05.31
2	Buhalterinė apskaita	2	2001.12.31
3	WWW svetainė	1	2001.07.31

Tarp lentelės *Projektų_Etapai* atributų galioja net du f-sąryšiai ($Nr \to Pavadinimas$ ir $Pavadinimas \to Nr$), kurių determinantai nėra raktai. Todėl lentelė *Projektų_Etapai* nėra BKNF. Siekiant BKNF, lentelę reikia suskaidyti į dvi lenteles. Galimi du skaidymo variantai:

1) *Pavadinimai_Etapai*(*Pavadinimas, Etapas, Pabaiga*),
   Nr_Pavadinimai(*Nr, Pavadinimas*);
2) *Nr_Etapai*(*Nr, Etapas, Pabaiga*),
   Nr_Pavadinimai(*Nr, Pavadinimas*).

Abu lentelių rinkiniai yra BKNF. Teoriškai jie lygiaverčiai. Viena lentelė *Nr_Pavadinimai* yra bendra abiem atvejams. Ši lentelė turi du raktus ir nė vieno nepirminio atributo. Kitos dvi lentelės skiriasi tik vienu stulpeliu. Kadangi stulpelio *Pavadinimas* reikšmės yra ilgesnės negu stulpelio *Nr*, tai galima sakyti, kad lentelė *Nr_Etapai* yra pranašesnė už *Pavadinimai_Etapai*. Tas pranašumas dar sustiprėtų tarus, kad projektų pavadinimai gali keistis. Taigi grynai praktiniu požiūriu antrasis variantas yra geresnis už pirmąjį.

Jei lentelė nėra BKNF, tai egzituoja netrivialus f-sąryšis, kurio determinantas nėra raktas. Tai, kad netrivialaus f-sąryšio $A \to B \in F$ determinantas A nėra lentelės $L(R)$ ($A, B \subset R$) raktas, reiškia, kad iš atributų aibės A sąryšių aibėje F negalima išvesti visų lentelės atributų, t.y. $A^+ \subset R$. Norint patikrinti, ar f-sąryšio $A \to B \in F$ determinantas A nėra lentelės $L(R)$ raktas, galima pasinaudoti ankščiau pateiktu algoritmu atributų aibės A uždariniui A^+ f-sąryšių aibės F atžvilgiu rasti.

Jei lentelėje egzistuoja f-sąryšis, kurio determinantas nėra raktas, tai iš pradinės lentelės pašalinami atributai esantys tokio sąryšio dešinėje pusėje, o iš visų dalyvaujančių sąryšyje atributų sudaroma nauja lentelė. Tarkime, lentelėje $L(A, B, C)$ galioja netrivialus f-sąryšis $A \to B$ ir A nėra šios lentelės raktas. Tuomet lentelę L skaidome į dvi: $L_1(A, C)$ ir $L_2(\underline{A}, B)$. Lentelės L_2 raktas A gali būti pirminiu, o lentelėje L_1 galima apibrėžti išorinį raktą, susiejant jos atributus A su L_2 pirminiu raktu.

3.11. Ketvirtoji normalinė forma

Ketvirtajai normalinei formai apibrėžti funkcinio sąryšio sąvokos nepakanka. Tam įvedama daugiareikšmio sąryšio sąvoka, apibendrinanti funkcinį sąryšį.

Lentelėje $L(A, B, C)$ yra **daugiareikšmis** (angl. *multi valued*) **sąryšis**, sutrumpintai **MV-sąryšis**, $A \to\!\!\to B$ tada ir tik tada, kai atributų B reikšmių aibė, atitinkanti bet kurias atributų aibių A ir C reikšmes, priklauso tik nuo A ir nepriklauso nuo atributų C reikšmių. MV-sąryšis gali būti ir kitaip apibrėžiamas. Lentelėje $L(A, B, C)$ galioja MV-sąryšis $A \to\!\!\to B$, jei, egzistuojant dviem lentelės eilutėms (a, b, c) ir (a, b′, c′), būtinai egzistuoja ir trečia eilutė (a, b′, c), kur a, b, b′, c, c′ žymi atitinkamų atributų reikšmes.

Funkcinio sąryšio atveju kiekvieną determinanto reikšmę atitinka tik viena nuo jo priklausomo atributo reikšmė. Daugiareikšmio sąryšio atveju atributų reikšmę gali atitikti kelios kito atributo reikšmės. MV-sąryšio sąvoka yra f-sąryšio apibendrinimas ta prasme, kad kiekvienas f-sąryšis kartu yra ir MV-sąryšis. Tiksliau, f-sąryšis – tai MV-sąryšis, kuriame determinantą atitinkanti reikšmių aibė visada yra iš vieno elemento. Kaip ir f-sąryšių atveju, mus domina tik tokie MV-sąryšiai, kurie lentelėje galioja visada, o ne kuriuo nors konkrečiu momentu. **MV-sąryšis $A \to\!\!\to B$ yra netrivialus** lentelėje $L(R)$, jei nė vienas atributas iš B neįeina į A ir, be atributų A ir B, lentelėje yra dar ir kitų atributų, t.y. jei $A \cap B = \varnothing$ ir $A \cup B \subset R$.

Tarkime, kiekvienas firmos darbuotojas gali turėti kelias kvalifikacijas ir mokėti kelias užsienio kalbas, o informacijai apie tai skirta lentelė *Išsilavinimas*(*Nr*, *Kvalifikacija*, *Kalba*), kurios stulpeliai atitinkamai reiškia darbuotojo numerį, įgytos kvalifikacijos pavadinimą ir užsienio kalbą, kurią jis moka. Ši lentelė turi vieną raktą, kurį sudaro visi trys lentelės atributai. Tarkime, darbuotojas Nr. 1 turi informatiko ir fiziko kvalifikaciją, taip pat moka dvi užsienio kalbas: anglų ir vokiečių. Lentelės fragmentas su duomenimis apie šį darbuotoją gali būti toks:

Išsilavinimas

Nr	*Kvalifikacija*	*Kalba*
1	Informatikas	Anglų
1	Informatikas	Vokiečių
1	Fizikas	Anglų
1	Fizikas	Vokiečių

Lentelėje *Išsilavinimas* yra du MV-sąryšiai:

$Nr \rightarrow\!\!\!\rightarrow Kvalifikacija,$
$Nr \rightarrow\!\!\!\rightarrow Kalba.$

Iš pirmo žvilgsnio gali pasirodyti, kad turimai informacijai išreikšti pakanka tik dviejų eilučių lentelėje, pavyzdžiui, pirmos ir paskutinės. Tačiau tada galėtų atrodyti, kad žmogus moka tik anglų kalbą, kai dirba informatiko darbą, ir tik vokiečių kalbą, kai fiziko. Taip, žinoma, neturėtų būti. Vienintelis būdas išreikšti, kad kvalifikacijos niekaip nesusiję su užsienio kalbomis, yra kiekvieną kalbą pakartoti su kiekviena kvalifikacija. Tai akivaizdžiai sukelia duomenų perteklių ir su tuo susijusias duomenų anomalijas. Pastebėsime, kad lentelėje nėra nepirminių atributų, todėl ji yra BKNF.

Reliacinėje teorijoje nesunkiai įrodoma, kad jei lentelėje $L(A, B, C)$ galioja MV-sąryšis $A \rightarrow\!\!\!\rightarrow B$, tai galioja ir MV-sąryšis $A \rightarrow\!\!\!\rightarrow C$. Tokiu būdu netrivialūs daugiareikšmiai sąryšiai visuomet sudaro poras. Todėl MV-sąryšių pora dažnai vaizduojama taip: $A \rightarrow\!\!\!\rightarrow B \mid C$. Lentelėje *Išsilavinimas* galioja $Nr \rightarrow\!\!\!\rightarrow Kvalifikacija \mid Kalba$.

Lentelė $L(R)$ yra **ketvirtosios normalinės formos** (**4NF**) tada ir tik tada, kai egzistuojant netrivialiam MV-sąryšiui $A \rightarrow\!\!\!\rightarrow B$, $A \subset R$, $B \subset R$, visi kiti atributai, funkciškai priklauso nuo A. Neformaliai lentelė yra 4NF tik tuomet, kai kiekvienas joje galiojantis netrivialus sąryšis išreiškia atributų funkcinę priklausomybę nuo (galimo) rakto. **DB yra 4NF** tada ir tik tada, kai visos jos lentelės yra 4NF.

Teiginys. Lentelė L yra 4NF, jei visi lentelės netrivialūs daugiareikšmiai sąryšiai faktiškai yra funkciniai sąryšiai, apibrėžiantys priklausomybę nuo lentelės galimo rakto.

Paėmus kurį nors vieną iš dviejų MV-sąryšių, galiojančių lentelėje *Išsilavinimas*, pavyzdžiui, $Nr \rightarrow\!\!\!\rightarrow Kvalifikacija,$ likęs atributas *Kalba* nėra f-priklausantis nuo *Nr*. Todėl ši lentelė nėra 4NF.

Feigino teorema. Tarkime, A, B ir C yra lentelės $L(A, B, C)$ atributų aibės. Tuomet lentelė L gali būti gaunama sujungiant jos projekcijas $L_1(A, B)$ ir $L_2(A, C)$ tada ir tik tada, kai $A \rightarrow\!\!\!\rightarrow B \mid C$.

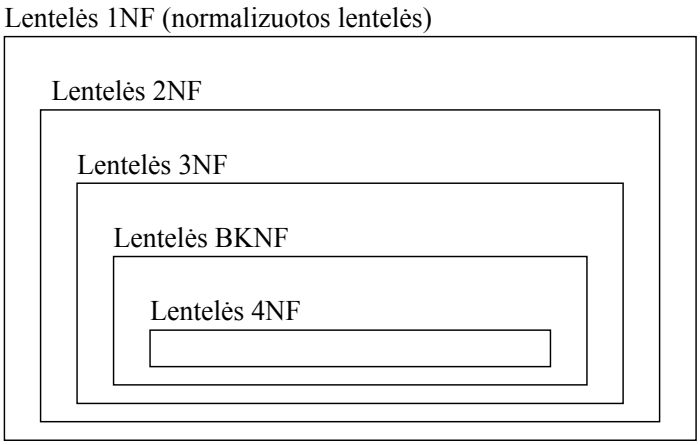
Prisiminus, kad MV-sąryšis yra f-sąryšio apibendrinimas, nesunku pastebėti, jog Feigino (R. Fagin) teorema yra Hezo teoremos apibendrinimas. Sekant Feigino teorema, lentelę *Išsilavinimas* galima suskaidyti į dvi projekcijas:

Kvalifikacijos(*Nr*, *Kvalifikacija*),
Kalbos(*Nr*, *Kalba*).

Abiejose šiose lentelėse yra po vieną MV-sąryšį, tačiau abu jie yra trivialūs, t.y. juose dalyvauja visi atitinkamos lentelės atributai. Todėl abi šios dvi lentelės yra 4NF.

3.12. Normalinių formų tarpusavio ryšys

Kiekvienas f-sąryšis yra ir MV-sąryšis. Tai reiškia, kad jei lentelė nėra BKNF, tai ji nėra ir 4NF. Kitaip tariant, kiekviena lentelė, esanti 4NF, kartu yra ir BKNF. BKNF pavadinime neatsispindi normalinės formos vieta jų hierarchijoje, kadangi 4NF jau buvo anksčiau apibrėžta. Taigi kiekviena aukštesnė NF patenkina vis griežtesnius reikalavimus lentelės reliacinei schemai.



3.2 pav. Normalinių formų tarpusavio padėtis

Realiuose uždaviniuose, jei duomenų bazė yra 3NF ar BKNF ar net 4NF, sakoma, kad to pakanka. Reliacinėje teorijoje nagrinėjamos ir aukštesnės normalinės formos nei ketvirtoji. Tačiau aukštesnės normalinės formos svarbios daugiau teoriniu, negu praktiniu požiūriu.

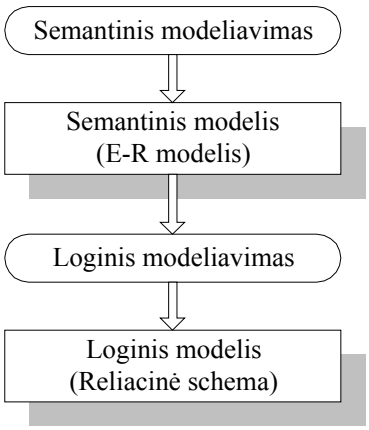
Kaip matyti iš šiame skyriuje pateiktų pavyzdžių, sudarant duomenų bazės schemą, įmanoma pasiekti BKNF, netgi 4NF, ir be formalių metodų. Į visas nagrinėtas normalines formas galima žiūrėti kaip į sukauptų praktinių žinių formalizavimą. Tačiau šis formalizavimas nėra savitikslis. Formalizuotos žinios leidžia automatizuoti šį procesą. Dabar praktikoje taikoma ne viena automatizuoto DB sudarymo priemonė, kuriomis generuojamos DB struktūros, esančios 3NF, BKNF ar net 4NF.

4. Semantinis modeliavimas

4.1. Įvadas

Duomenų bazės projektavimas yra gana sudėtingas ir ilgas procesas, kuriame būtinai turėtų dalyvauti tvarkomo dalyko specialistai. DB loginė struktūra (schema) yra visos informacinės sistemos pagrindas. Todėl projektuojant dideles DB svarbu sudaryti tikslų dalykinės srities aprašą. Formalizuotas dalykinės srities aprašas, kuris gali būti suprantamas tiek DB specialistams, tiek ir dalyko specialistams, vadinamas **semantiniu (**arba **sampratos, konceptualiu) modeliu**. Reliacinis modelis yra lakoniškas ir jame nelabai atsispindi dalykinės srities semantika. Pastebėsime, kad modelyje vaizduojami ne visi realaus pasaulio objektai, reiškiniai, sąryšiai ir pan., o tik atskiros detalės. Šiame skyriuje aptarsime vieną populiariausių semantinių modelių – E–R (esybė – sąryšis, angl. *Entity-Relationship*) modelį, kurį pasiūlė 1976 m. Čenas (P. P.-S. Chen). Po to sužinosime kaip E-R modelį atvaizduoti DB reliacine schema.

E-R modelis plačiai taikomas sudėtingoms kompiuterinėms sistemoms projektuoti pagal objektų technologiją. Semantinio modeliavimo idėjos taip pat vertingos projektuojant DB. DB schemos projektavimo technologija, pagrįsta semantiniu modeliu, vadinama "iš viršaus į apačią" (angl. *top-down*), kadangi projektavimas pradedamas nuo realaus pasaulio aukščiausio abstrakcijos lygio ir baigiamas palyginti žemo abstrakcijos lygio konkrečia logine DB schema. Semantinio modelio sudarymas yra pirmasis DB projektavimo etapas. Turint semantinį modelį galima sudaryti DB reliacinę schemą.



4.1 pav. DB projektavimo etapai

Semantinio modeliavimo metodą išsiaiškinsime modeliuodami supaprastintą biblioteką. Tarkime, bibliotekoje laikomos tik knygos. Žurnalų, pavyzdžiui, nenagrinėsime. Biblioteka gali turėti keletą kiekvienos knygos egzempliorių. Kiekviena knyga turi unikalų ISBN numerį, o kiekvienas bibliotekos turimas knygos egzempliorius turi dar ir unikalų tapatumo (registracijos) numerį. Bibliotekos knygomis naudojasi skaitytojai, turintys skaitytojo pažymėjimą su numeriu. Kiekvienam skaitytojui suteikiama galimybė pasiimti kiek nori knygų. Imant knygą nustatoma jos grąžinimo data. Bibliotekoje yra sisteminis katalogas, kurio požymis – žinių sritis. Visos bibliotekos knygos suskirstytos atsižvelgiant į žinių sritį.

4.2. Pagrindinės E-R modelio sąvokos

Aptarkime pagrindines E-R modelio sąvokas.

Esybė. Esybe (angl. *entity*) grindžiama vieno tipo objektų klasė. Esybės - tai fiziškai egzistuojantys ar mintyse suvokiami ir lengvai skiriami modeliuojamo pasaulio bendrieji vaizdai (sąvokos, konceptai). Kiekvienai esybei modeliuojamoje sistemoje suteikiamas unikalus tapatumo vardas. Pavyzdžiui, modeliuojant biblioteką, galima išskirti tokias esybes: *Knyga* – abstraktus, užregistruotas bibliotekoje leidinys; *Katalogas* – sisteminis knygų katalogas, kuriame žinių sričiai priskiriama dalis bibliotekos knygų; *Skaitytojas* – bibliotekos skaitytojas; *Egzempliorius* – bibliotekos knygos egzemplioriaus abstrakcija. Konkrečios knygos – tai esybės *Knyga* objektai, konkretūs knygų egzemplioriai – esybės *Egzempliorius* objektai ir pan.

Atributai. Visi objektai turi tam tikrus požymius – **atributus** (angl. *attributes*). Visi vienos esybės objektai turi tuos pačius atributus. Kiekvienam atributui priskiriama galimų reikšmių aibė. Paprastai objekto atributai yra **vienareikšmiai** (kiekvienas objektas gali turėti tik vieną atributo reikšmę), pavyzdžiui, kiekvienas skaitytojas turi vieną asmens kodą. Tačiau kai kurie atributai gali būti **daugiareikšmiai** (angl. *multivalued*), pavyzdžiui, knyga gali būti parašyta keleto autorių. Taigi esybės *Knyga* atributas *Autorius* yra daugiareikšmis. Atributas gali būti **paprastas** (angl. *simple*) arba **sudėtinis** (angl. *composite*). Sudėtinį atributą sudaro keletas paprastų atributų. Pavyzdžiui, esybės *Skaitytojas* objektai gali turėti sudėtinį atributą *Adresas*, kurį, tarus, kad visi skaitytojai gyvena tame pačiame mieste, sudaro trys paprasti atributai: *Gatvė, Namas* ir *Butas*. Atributai, vienareikšmiškai atitinkantys esybės objektą, vadinami **raktu** (raktiniais atributais). Pvz., esybės *Knyga* objektų (knygų) atributas *ISBN* yra raktas, kadangi kiekviena knyga turi unikalų ISBN numerį. Kai kurie atributai gali būti **išvestiniai**, apskaičiuojami pagal kitų atributų reikšmes, pvz., skaitytojo amžių galima apskaičiuoti pagal jo gimimo ir einamąją datas.

Silpnoji esybė. Kai kurių esybių objektų prasmingumas priklauso nuo kitų esybių objektų. Pavyzdžiui, negali būti neegzistuojančios knygos egzemplioriaus. Todėl esybė *Egzempliorius* priklauso nuo esybės *Knyga*. Tačiau galima laikyti, kad knygos bibliotekoje nėra, jei nėra nei vieno jos egzemplioriaus. Nors šios esybės priklauso viena nuo kitos, tačiau jos yra gana savarankiškos. Kai kurios esybės gali būti tiek priklausomos nuo kitos (pagrindinės) esybės, kad priklausomos esybės objektai vienareikšmiškai nustatomi tik panaudojant pagrindinės esybės raktą. Tokio lygio priklausoma esybė vadinama **silpnąja** (angl. *weak*). Jei esybė nėra tokia priklausoma, tai ją galima vadinti **stipriąja** (angl. *strong*), jei ji nėra silpnoji. Silpnosios esybės pavyzdys - knygos skyrius. Knygos skyrius vienareikšmiškai identifikuojamas skyriaus numeriu kartu su knygos raktu (ISBN numeriu). Tačiau knygų skyriai bibliotekoje paprastai nėra tvarkomi (nagrinėjami). Todėl tokios esybės į bibliotekos modelį neįtrauksime.

Sąryšis. ER modelio sąvokų autorius sąryšį (angl. *relationship*) apibrėžė kaip dvinarę asociaciją, nusakančią esybių tarpusavio santykį arba sąveiką. Sąryšio pavadinimas turi atspindėti jo esmę. Pavyzdžiui, tarp esybių *Knyga* ir *Katalogas* objektų egzistuoja sąryšis, nusakantis knygos priklausomybę sisteminio katalogo skirsniams, t.y. žinių sritims. Pavadinkime šį sąryšį *Priklauso*. Esybės, dalyvaujančios sąryšyje vadinamos **dalyviais**, o dalyvių kiekis – **sąryšio laipsniu**. Vienanariai sąryšiai – tai sąryšiai vienoje esybėje. Tokie sąryšiai taip pat vadinami **rekursyviais** (angl. *recursive*). Dažniausiai sutinkami **dvinariai** sąryšiai, kuriuose dalyvauja dvi esybės. Matematine prasme, sąryšis R tarp esybių E_1, E_2,…, E_n yra Dekarto sandaugos poaibis:

$$R \subseteq E_1 \times E_2 \times \ldots \times E_n,$$

čia n – sąryšio laipsnis. Sąryšio (Dekarto sandaugos) elementas $(e_1, e_2, \ldots, e_n) \in R$ vadinamas sąryšio egzemplioriumi (angl. *instance*), kur $\forall i : 1 \le i \le n : e_i \in E_i$.

Yra trys pagrindinės sąryšių rūšys: **vienas su vienu** (sutrumpintai žymima **1:1**), **vienas su daug** (**1:N**), **daug su daug** (**N:M**). Tarp dviejų esybių egzistuoja sąryšis 1:1, jei vienas vienos esybės objektas yra susijęs su vienu kitos esybės objektu. Kitaip tariant, sąryšis $R \subseteq E_1 \times E_2$

yra 1:1 sąryšis, jei $\forall e_1 \in E_1$ priskiriamas daugiausiai vienas objektas $e_2 \in E_2$ ir atvirkščiai: $\forall e_2 \in E_2$ priskiriamas daugiausiai vienas objektas $e_1 \in E_1$. Sąryšio 1:M atveju vieną vienos esybės objektą gali atitikti daugiau nei vienas kitos esybės objektas. Kai tarp dviejų esybių egzistuoja N:M sąryšis, tai vieną pirmos esybės objektą gali atitikti keletas antros esybės objektų ir vieną antros esybės objektą gali atitikti keletas pirmos esybės objektų. Pavyzdžiui, tiek sąryšis *Yra egzempliorius* tarp esybių *Knyga* ir *Egzempliorius*, tiek ir sąryšis *Skaito* tarp esybių *Skaitytojas* ir *Egzempliorius* yra 1:N sąryšiai, nes bibliotekoje gali būti keletas konkrečios knygos egzempliorių, o vienas skaitytojas tuo pačiu metu gali būti pasiėmęs keletą knygų (egzempliorių). Be to, konkretus egzempliorius negali būti kelių knygų egzemplioriumi, kaip ir vieno egzemplioriaus vienu metu negali skaityti (pasiimti) keli skaitytojai. Tarp esybių *Knyga* ir *Katalogas* egzistuoja N:M tipo sąryšis *Priklauso*, nes vienai žinių sričiai gali priklausyti keletas bibliotekos knygų ir ta pati knyga gali priklausyti keletui sričių.

Kartais patogu atributus priskirti ne esybėms, bet sąryšiui. **Sąryšio atributai** apibūdina sąryšį. Pavyzdžiui, sąryšį *Skaito* galima papildomai nusakyti knygos egzemplioriaus paėmimo ir grąžinimo data. Taip atributai *Paėmimo data* ir *Grąžinimo data* gali būti sąryšio *Skaito* atributai. Tačiau sąryšiai su atributais nėra būtini E-R modelyje. Visuomet galima sudaryti naują esybę su atributais, nusakančiais sąryšį, ir įtraukti naująją esybę į sąryšį. Knygos egzemplioriaus paėmimo ir grąžinimo datas priskirsime esybei *Egzempliorius*, kuri dalyvauja sąryšyje *Skaito*.

Kiekvienas iš trijų tipinių sąryšių gali būti **besąlygišku** (**stipriu**, angl. *regular*, *non-weak*) arba **sąlygišku (silpnu**, angl. *weak*). Sąryšis yra besąlygiškas, jei sąryšyje dalyvauja visi esybės objektai. Sąlygiškame sąryšyje gali dalyvauti ne visi esybės objektai. Sąryšis gali būti sąlygiškas iš vienos pusės ir besąlygiškas iš kitos pusės. Pavyzdžiui, sąryšis *Yra egzempliorius* tarp esybių *Knyga* ir *Egzempliorius* yra besąlygiškas iš esybės *Egzempliorius* pusės, nes kiekvienas egzempliorius yra kažkurios knygos egzempliorius. Sąryšis *Yra egzempliorius* yra besąlygiškas ir iš kitos pusės, nes bibliotekoje nėra knygos, jei nėra nė vieno jos egzemplioriaus. Sąryšis *Priklauso* tarp esybių *Katalogas* ir *Knyga* yra besąlygiškas iš esybės *Knyga* pusės, nes kiekvieną knygą galima priskirti bent vienai žinių sričiai. Sąryšis *Priklauso* besąlygiškas iš esybės *Katalogas* pusės, nes kataloge nėra atitinkamos skilties (žinių srities), jei bibliotekoje nėra nei vienos knygos, priklausančios tai sričiai. Sąryšis *Skaito* yra sąlygiškas iš abiejų pusių, nes knygos egzempliorius nebūtinai turi būti skaitomas ir skaitytojas nebūtinai turi būti paėmęs bent vieną knygą.

Tarp stipriosios ir silpnosios esybių dažniausiai egzistuoja sąryšis 1:N, rečiau 1:1. Silpnąją ir stipriąją esybes negali sieti N:M sąryšis.

Ryšio priklausomumas kuriai nors rūšiai yra integralumo sąlyga, kuri modeliuojamame pasaulyje turi būti patenkinta visada, ne tik kurioje nors konkrečioje situacijoje.

Kategorizavimas. E-R modelyje galima taikyti kategorizacijos (tipizacijos) principą, pagal kurį įvedama esybės potipio ir viršTipio sąvokos. Kiekviena esybė gali turėti keletą potipių. Daugelyje uždavinių tarp kelių esybių galima išskirti bendrų atributų aibę. Tuomet bendrus tokių esybių atributus galima išskirti į atskirą esybę – **viršTipį**. Esybės iš kurių buvo išskirtos bendrosios dalys, tampa viršTipio **potipiais** (viršTipio kategorijomis). Procesas, kai išskiriamas viršTipis vadinamas **apibendrinimu**. Galimas ir atvirkščias, **specializavimo** procesas, kurio metu esybėje išskiriami keli atributų poaibiai ir esybė suskaidoma į bendrąją dalį – viršTipį bei kelias specializuotas dalis – potipius. Esminė sąryšio viršTipis-potipiai savybė – paveldimumas. Potipiai paveldi visus savo viršTipio atributus ir sąryšius. Paveldimumas išreiškiamas specialia sąryšio rūšimi "yra" (angl. *is-a*). Pavyzdžiui, esybę *Skaitytojas* galima specializuoti išskiriant du potipius: *Studentas* ir *Dėstytojas*. Pirmosios esybės-potipio atributais gali būti studijų knygelės numeris *SKNR* ir *Specialybė*. Esybė *Dėstytojas* gali būti charakterizuojama atributais *Katedra* ir *Pareigos*. Savo ruoštu, esybę *Dėstytojas* galima apibendrinti esybe *Darbuotojas* ar specializuoti į esybes *Asistentas*, *Profesorius* ir pan. Taip gaunama **tipų hierarchija**. Priklausomai nuo skaitytojų rato, esybę *Skaitytojas* galima specializuoti ir kitaip.

Pastebėsime, kad pagrindines E-R modelio sąvokas aptarėme tik bendriausia jų prasme. Kai kurias pateiktas sąvokas galima detalizuoti. Pavyzdžiui, 1:N ir N:M tipų sąryšius galima

charakterizuoti tiksliau, sąvoką “daug” (“N”, “M”) patikslinant minimalia ir maksimalia ribomis, nurodant, kiek kartų mažiausiai ir kiek kartų daugiausiai esybė gali dalyvauti sąryšyje. Tai sąryšio **kardinalumas** (angl. *cardinality*). Tačiau detalesnes E-R modelio charakteristikas paliksime savarankiškam darbui.

4.3. E-R diagramos

E-R modelį vaizdžiai atspindi **E-R diagrama**. Pateiksime pagrindinių E-R modelio sąvokų vaizdavimo E-R diagrama principus.

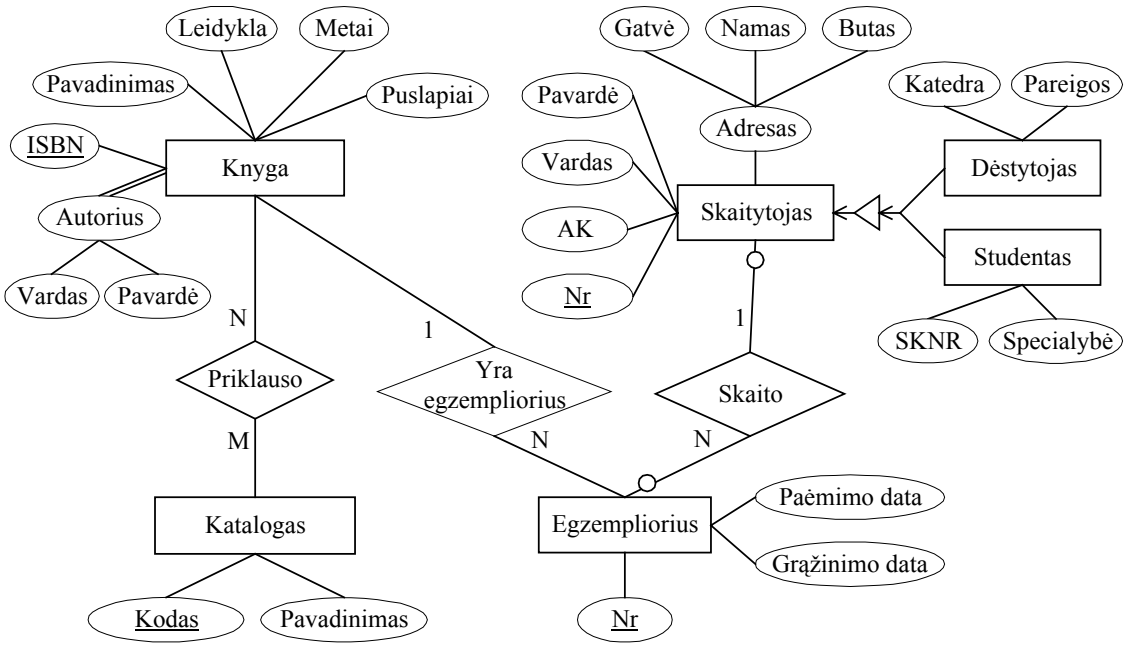
Esybės. Kiekviena esybė vaizduojama stačiakampiu, kurio viduje rašomas esybės vardas. Silpnosios esybės vaizduojamos stačiakampiu su dvigubu rėmeliu.

Atributai. Atributai E-R diagramoje vaizduojami ovalais, susiejant juos su atitinkamos esybės stačiakampiu ištisine linija. Ovalo viduje rašomas atributo vardas. Išvestinių (apskaičiuojamų) atributų ovalai brėžiami punktyrine linija. Sudėtinio atributo sudedamosios dalys vaizduojamos ovalais, sujungtais ištisine linija su sudėtinio atributo ovalu. Raktinių atributų vardai pabraukiami. Daugiareikšmiai atributai jungiami su esybe dviguba linija.

Sąryšiai. Kiekvienas sąryšis vaizduojamas rombu su sąryšio vardu viduje. Rombas apvedamas dviguba linija, jei sąryšis yra tarp silpnosios esybės ir pagrindinės esybės. Sąryšį vaizduojantis rombas sujungiamas ištisinėmis linijomis su visais sąryšio dalyviais. Kiekviena tokia linija pažymima “1”, “N” ar “M” sąryšio tipui pažymėti. Linija, jungianti sąryšio rombą su silpnąja esybe, yra dviguba. Jei sąryšis turi atributus, tai šie vaizduojami atskirais ovalais ir sujungiami linija su sąryšį žyminčiu rombu. Jei sąryšis yra sąlyginis, tai linija, jungianti jį su esybe, iš kurios pusės sąryšis sąlyginis, yra pažymima tuščiu skrituliuku.

Potipiai ir **virštipiai**. Jeigu esybės X_1, X_2,.., X_n yra esybės Y potipiai, tai iš kiekvienos esybės X_i ($1 \le i \le n$) stačiakampio brėžiama linija į grafinį elementą, vadinamą **deskriptoriumi** (jį vaizduosime trikampiu), o iš jo į esybės Y stačiakampį su rodykle linijos gale.

Pavaizduokime, bibliotekos E-R modelį *Biblioteka* E-R diagrama:



4.2 pav. Semantinio modelio *Biblioteka* E-R diagrama

4.4. E-R modelio vaizdavimas reliaciniu duomenų modeliu

Semantinis modelis vartojamas ankstyvosiose projektavimo stadijose. Juo gali naudoti kiti sistemos kūrėjai vėlesniuose etapuose. DB projektavimas yra gana subjektyvus procesas, nes priklauso nuo projektuotojo patirties ir dalykinės srities supratimo. Tačiau E-R modeliui egzistuoja algoritmas, vienareikšmiškai siejantis semantinį modelį su reliaciniu.

Apžvelkime ER modelio suvedimo į reliacinį duomenų modelį **bendrąsias taisykles**.

1. Kiekvieną E-R modelio **esybę** reliaciniame modelyje atitinka lentelė. Lentelės vardas turi atitikti konkrečios RDBVS reikalavimus, todėl jis gali skirtis nuo esybės vardo.
2. Kiekvienam **esybės vienareikšmiam atributui** atitinkamoje reliacinėje lentelėje turi būti stulpelis. Stulpelio vardas gali sutapti su atributo vardu arba skirtis nuo jo. Stulpelio vardas formuojamas pagal RDBVS reikalavimus. Išvestiniai (apskaičiuojami) atributai į lentelę neįtraukiami. **Sudėtiniams** atributams į lentelę įtraukiami tik juos sudarantys paprastieji atributai.
3. Kadangi reliaciniame modelyje aibė negali būti reikšme, tai **daugiareikšmio** atributo negalima atvaizduoti lentelės stulpeliu. Galimi du daugiareikšmių atributų realizavimo būdai:
   - Jei žinoma daugiareikšmio atributo A maksimalus paprastųjų reikšmių kiekis n, tai į esybę atitinkančią lentelę įtraukiama n stulpelių A_1, A_2 ,…, A_n – po vieną kiekvienai galimai atributo reikšmės dedamajai. Šiuo atveju jei n yra gana didelis, tai esybės lentelė turės daug stulpelių. Be to, dauguma lentelės stulpelių A_1, A_2 ,…, A_n reikšmių gali būti `NULL`.
   - Daugiareikšmiam atributui A sukuriama atskira lentelė, į kurią įtraukiamas atitinkamas stulpelis A. Jei A yra sudėtinis, tai įtraukiami visi jį sudarantys paprastieji atributai. Į lentelę taip pat įtraukiamas esybės, kuriai priklauso A, raktas K, kuris tampa lentelės išoriniu raktu. Lentelės pirminiu raktu tampa A ir K kartu paėmus. Šio būdo trūkumas – atributui kuriama atskira lentelė, tačiau šioje lentelėje nebus `NULL` reikšmių.
4. **Esybės raktas** tampa pirminiu lentelės raktu. Semantiniame modelyje raktiniai atributai esybei gali būti ir neapibėžti. Tokiu atveju iš esybės atributų išskiriama vienas ar keli atributai, sudarysiantys pirminį raktą. Arba galima parinkti visiškai naują papildomą atributą, kuris taptų pirminiu raktu. Jei esybė turi keletą raktų, tai vienas iš jų parenkamas pirminiu.
5. Į kiekvieną lentelę, atitinkančią **silpnąją esybę**, įtraukiami pagrindinę (stipriąją) esybę atitinkančios lentelės pirminio rakto atributai. Šie atributai silpnąją esybę atitinkančioje lentelėje tampa išoriniu raktu į pagrindinę lentelę. Silpnąją esybę atitinkančioje lentelėje pirminis raktas sudaromas iš atributų (stulpelių), vienareikšmiškai identifikuojančių silpnąją esybę pagrindinės esybės atžvilgiu, ir visų pagrindinės lentelės pirminio rakto stulpelių.
6. Bendriausiu atveju kiekvieną **sąryšį** galima realizuoti atskira lentele. Į lentelę įtraukiami abiejų sąryšyje dalyvaujančių esybių raktai, kurie tampa lentelės išoriniais raktais. Jei esybė yra privaloma sąryšyje, tai atitinkamo išorinio rakto stulpeliai negalės įgyti `NULL` reikšmės. Realizuojančios sąryšį lentelės pirminiu raktu gali būti visi išorinių raktų stulpeliai. Dvinarius sąryšius 1:1 ir 1:N galima realizuoti ir neįvedant specialios lentelės. Šių tipų dvinarius sąryšius, o taip pat ir N:M tipo sąryšius, galima realizuoti pagal specializuotas žemiau pateiktas taisykles.
7. Kiekvienam **1:N sąryšiui** į lentelę L_N, atitinkančią esybę, pažymėtą “N”, įtraukiamas lentelės, atitinkančios esybę, pažymėtą “1”, pirminis raktas ir jis tampa lentelės L_N išoriniu raktu. Jei esybė, atitinkanti “1”, yra privaloma sąryšyje, tai išorinio rakto stulpeliai negalės įgyti `NULL` reikšmės. Į lentelę L_N taip pat įtraukiami visi sąryšio atributai, jei tokie yra.
8. Kadangi **sąryšis 1:1** yra simetrinis, tai laisvai pasirenkama, kuri esybė bus “pagrindinė”, o kuri “priklausoma”. Tokie pat vaidmenys suteikiami ir atitinkamoms lentelėms. Jei vienos esybės dalyvavimas sąryšyje yra būtinas, o kitos ne, tai rekomenduojama “priklausoma”

lentele pasirinkti būtinąją esybę atitinkančią lentelę. Į "priklausomą" lentelę įtraukiami "pagrindinės" lentelės pirminio rakto stulpeliai ir jie paskelbiami išoriniu raktu. Jei "priklausoma" esybė yra privaloma sąryšyje, tai išorinio rakto stulpeliai negalės įgyti `NULL` reikšmės. Jei sąryšis turi atributus, tai jie įtraukiami į "pagrindinę" lentelę.

9. Kiekvienas **N:M sąryšis** realizuojamas sudarant atskirą lentelę. Į lentelę įtraukiami abiejų lentelių, atitinkančių sąryšyje dalyvaujančias esybes, pirminiai raktai, kurie tampa išoriniais raktais. Į lentelę taip pat įtraukiami visi sąryšio atributai. Lentelės pirminiu raktu tampa abiejų susietų lentelių pirminių raktų kombinacija.
10. Ryšį **virštipis-potipis** galima atvaizduoti keliais būdais:
    - Visoms susietoms šio tipo sąryšiu esybėms yra sukuriama viena lentelė. Į lentelę įtraukiami visi virštipio ir visų potipių atributai. Tokiu atveju, kiekvienam esybės egzemplioriui (objektui) dalis atributų bus neprasmingi. Visi potipių atributus atitinkantys stulpeliai galės įgyti `NULL` reikšmę. Šiuo atveju galima situacija, kai dauguma lentelės reikšmių bus `NULL`. Taip prarandama apibendrinimo sąvokos prasmė. Be to, teks įvesti papildomas taisykles potipiams atskirti vieną nuo kito. Šio būdo privalumas – sukuriama tik viena lentelė.
    - Tiek virštipiui, tiek ir kiekvienam potipiui sukuriamos atskiros lentelės. Į visas potipius atitinkančias lenteles įtraukiamas virštipio pirminis raktas, kuris tampa potipio lentelės tiek pirminiu raktu, tiek ir išoriniu raktu. Šio būdo trūkumas – sukuriama daugiau lentelių.

Atvaizduokime E-R diagramą *Biblioteka* duomenų bazės *Biblioteka* reliacine schema.
Pagal taisykles 1, 2 ir 4 kiekvienai pagrindinei esybei gauname po lentelę:

Knyga(*ISBN*, *Pavadinimas*, *Leidykla*, *Metai*, *Puslapiai*);

Egzempliorius(*Nr*, *Paėmimo_data*, *Grąžinimo_data*);

Katalogas(*Kodas*, *Pavadinimas*);

Skaitytojas(*Nr*, *AK*, *Vardas*, *Pavardė*, *Gatvė*, *Namas*, *Butas*).

Pagal taisyklę 3 esybės *Knyga* daugiareikšmiui atributui *Autorius* gauname naują lentelę:

Autorius(*ISBN*, *Vardas*, *Pavardė*),
išorinis raktas: *ISBN* nurodo į *Knyga*.

Silpnųjų esybių E-R diagramoje nėra, todėl taisyklės 5 netaikome. Kadangi mūsų modelyje yra tik 1:N, N:M bei virštipis-potipis tipo sąryšiai, tai taisyklių 6 ir 8 netaikome. Pagal taisyklę 7, lentelę *Egzempliorius* papildome stulpeliais – lentelių *Knyga* ir *Skaitytojas* pirminiais raktais, kurie lentelėje *Egzempliorius* tampa išoriniais raktais. Kadangi stulpelis *Nr* jau yra lentelėje, lentelės *Skaitytojas* pirminį raktą įtraukdami jį į lentelę *Egzempliorius* pervardijame. Papildyta lentelės *Egzempliorius* schema:

Egzempliorius(*Nr*, *Paėmimo_data*, *Grąžinimo_data*, *ISBN*, *Skaitytojo_Nr*),
išoriniai raktai: *ISBN* nurodo į *Knyga*,
Skaitytojo_Nr nurodo į *Skaitytojas*.

Panaudoję taisyklę 9 suformuojame naują lentelę

Priklauso(*ISBN*, *Kodas*),
išoriniai raktai: *ISBN* nurodo į *Knyga*,
Kodas nurodo į *Katalogas*.

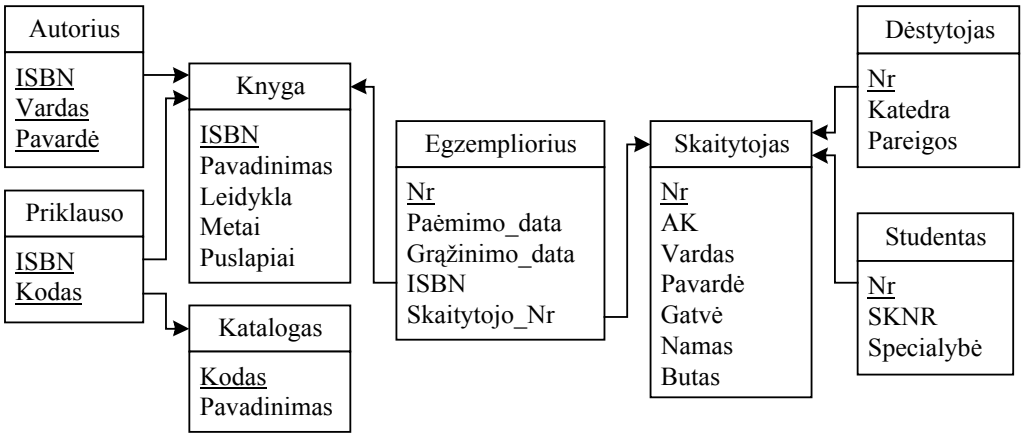
Jau minėjome, kad atributus *Paėmimo data* ir *Grąžinimo data* E-R modelyje galima buvo priskirti ne esybei *Egzempliorius*, bet sąryšiui *Skaito*. Tačiau ir tokiu atveju, realizuojant sąryšį *Skaito* pagal taisyklę 7, atitinkami stulpeliai *Paėmimo_data* ir *Grąžinimo_data* taptų lentelės *Egzempliorius* dalimi. Todėl, ir tokiu atveju gautume tokią pačią lentelės *Egzempliorius* schemą.

Pagal taisyklę 10 kiekvienam esybės-virštipio *Skaitytojas* potipiui *Dėstytojas* ir *Studentas* suformuojame po atskirą lentelę:

Dėstytojas(*Nr*, *Katedra*, *Pareigos*),
išorinis raktas: *Nr* nurodo į *Skaitytojas*;

Studentas(*Nr*, *SKNR*, *Specialybė*),
išorinis raktas: *Nr* nurodo į *Skaitytojas*.

Pavaizduokime DB *Biblioteka* reliacinę schemą grafiškai:



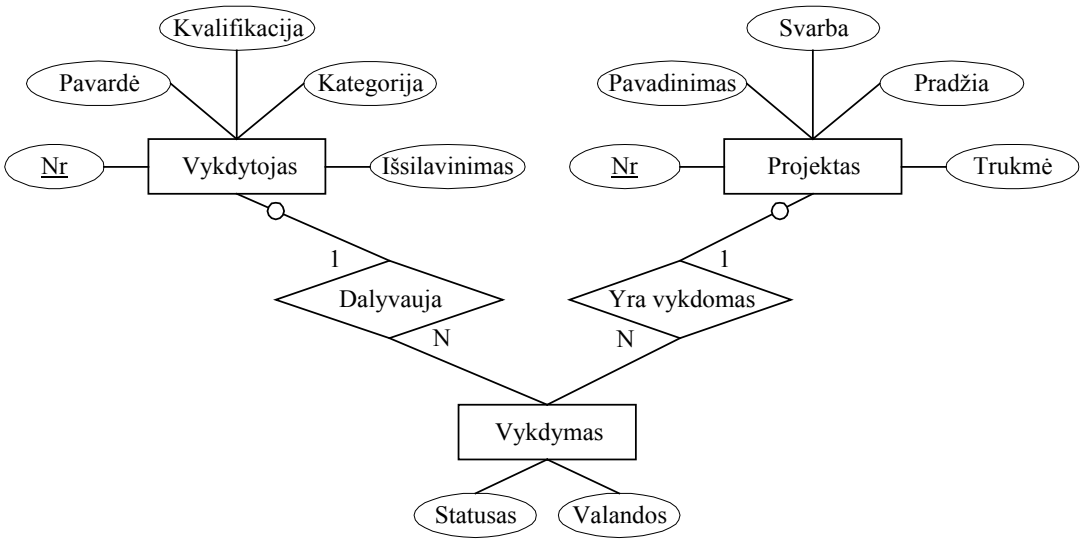
4.3 pav. Duomenų bazės *Biblioteka* reliacinė schema.

4.5. DB "Darbai" ER modelis

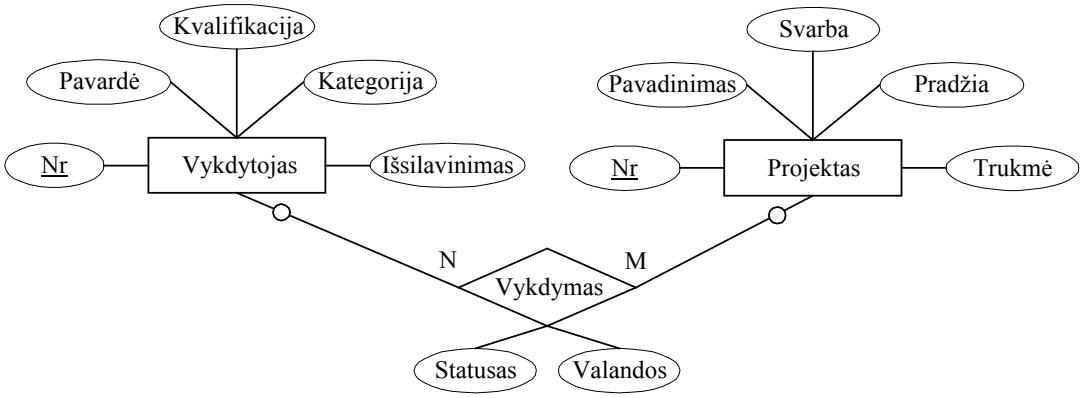
Pavaizduokime E-R diagrama anksčiau nagrinėtą dalykinę sritį, atitinkančią duomenų bazę *Darbai*. Tiksliau, pakartokime DB *Darbai* projektavimą – sudarykime šios srities E-R modelį ir po to atvaizduokime jį reliacine schema.

Kaip ir anksčiau, nagrinėkime paprasčiausią įstaigos veiklos atvejį, kai įstaigos darbuotojai vykdo projektus, atlikdami juose tam tikrą vaidmenį ir skirdami jo vykdymui 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*. Sumodeliuokime darbuotojų (vykdytojų) dalyvavimą projektuose. Šį uždavinį galima spręsti keliais būdais.

- Į vykdytojų dalyvavimą projektuose galima žiūrėti kaip į atskirą esybę *Vykdymas* su atributais *Statusas* ir *Valandos*, susiejant šią esybę tiek su esybe *Vykdytojas,* tiek ir su *Projektas*. Kadangi vienas vykdytojas gali dalyvauti vykdant kelis projektus, tai tarp esybės *Vykdytojas* ir *Vykdymas* egzistuoja 1:N tipo sąryšis *Dalyvauja*. Tarus, kad vykdytojas gali nedalyvauti nė viename projekte, *Vykdytojas* yra nebūtinas šio sąryšio dalyvis. Panašiai tarp esybių *Projektas* ir *Vykdymas* taip pat yra 1:N sąryšis ir jis yra neprivalomas iš esybės *Projektas* pusės. Taip sudaryto E-R modelio *Darbai* schema pateikta 4.4 pav.
- Į vykdytojų dalyvavimą projektuose galima žiūrėti ir kaip į N:M tipo sąryšį *Vykdymas* tarp esybių *Vykdytojas* ir *Projektas*. Abu sąryšio dalyviai yra neprivalomi. Šis sąryšis turi du atributus *Statusas* ir *Valandos*. Taip sudaryto E-R modelio *Darbai* schema pateikta 4.5 pav.



4.4 pav. E-R modelio *Darbai* schema, modeliuojant vykdytojų dalyvavimą projektuose esybe.



4.5 pav. E-R modelio *Darbai* schema, modeliuojant vykdytojų dalyvavimą projektuose sąryšiu.

Atvaizduokime abi E-R diagramas reliacinėmis schemomis. Visų pirma, abiejose schemose (abiejuose modeliuose) esybės *Vykdytojas* ir *Projektas* yra modeliuojamos vienodai. Abiem atvejais gauname dvi tokias pačias lenteles:

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

Atvaizduojant esybes lentelėmis, jas pervardijome tik tam, kad lentelių vardai atitiktų anksčiau vartotus vardus. Esybė *Vykdymas* atvaizduojama lentele su dviem atributais:

Vykdymas (*Statusas, Valandos*).

Modeliuojant sąryšį *Dalyvauja*, į pastarąją lentelę įtraukiamas lentelės *Vykdytojas* pirminis raktas, kuris tampa išoriniu raktu:

Vykdymas (*Vykdytojas*, *Statusas,Valandos*).
išorinis raktas: *Vykdytojas* nurodo į *Vykdytojai*.

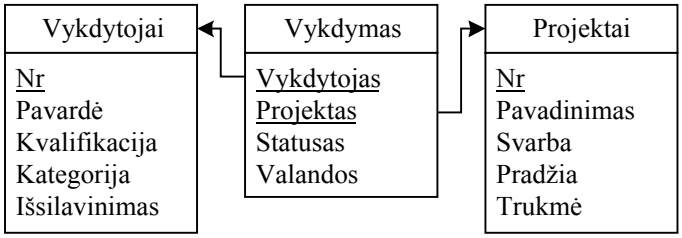
Sąryšis *Vykdomas* modeliuojamas į lentelę *Vykdymas* įtraukiant lentelės *Projektas* pirminį raktą atitinkantį stulpelį, kuris tampa jos išoriniu raktu. Mūsų modelyje neatsispindėjo faktas, kad vykdytojas, vykdant konkretų projektą, gali dalyvauti ne daugiau kaip viename vaidmenyje. Šis faktas reliacinėje schemoje gali būti atvaizduotas lentelės *Vykdymas* pirminiu raktu. Galutinė lentelės *Vykdymas* reliacinė schema:

Vykdymas (*Projektas, Vykdytojas, Statusas,Valandos*),

išoriniai raktai: *Projektas* nurodo į *Projektai*,
Vykdytojas nurodo į *Vykdytojai*.

Antruoju atveju, kai E-R modelyje *Darbai* vykdytojų dalyvavimas projektuose vaizduojamas sąryšiu, lentelės *Vykdymas* reliacinę schemą pagal taisyklę 9 gauname vienu žingsniu. Be to, lentelės pirminis raktas nustatomas pagal tą pačią taisyklę.

Abu E-R modelius *Darbai* atvaizdavome tokia pačia reliacine schema. Tačiau antruoju atveju prireikė mažiau žingsnių tikslui pasiekti. Sugeneruotos duomenų bazės *Darbai* reliacinę schemą galima pavaizduoti grafiškai:



4.6 pav. Duomenų bazės *Darbai* reliacinė schema, gauta iš E-R modelio

4.6. ER modelis ir duomenų normalizavimas

Sudarę E-R modelio reliacinę schemą, šiai galime dar pritaikyti duomenų normalizavimo teoriją, kurią aptarėme ankstesniame skyrelyje. Tačiau duomenų normalizavimą galima taikyti ir semantinio modelio lygyje. Pateiksime E-R modelio normalizavimo algoritmą, užtikrinantį 4NF.

1. Išanalizuoti E-R modelį siekiant nustatyti esybes, kurios savo turiniu modeliuoja kelias skirtingas realaus pasaulio objektų klases, tarpusavyje susijusias tam tikrais sąryšiais. Jei yra tokių esybių, tai visas jas reikia išskaidyti į naujas esybes, apibrėžiant tarp jų reikiamus sąryšius. Gautas modelis bus 1NF.
2. Išanalizuoti visas esybes su sudėtiniais raktais. Jei yra esybių, kuriose nepirminiai atributai funkciškai priklauso nuo rakto dalies, tai kiekvieną tokią esybę suskaidyti į dvi ir apibrėžti tarp jų reikiamus sąryšius. Gautas modelis bus 2NF.
3. Išanalizuoti visus neraktinius visų esybių atributus, ieškant jų tranzityvių priklausomybių nuo esybės rakto. Kiekvienai tranzityviai priklausomybei suskaidyti esybę į dvi, nepaliekant tranzityvios priklausomybės, ir apibrėžti tarp naujųjų esybių reikiamus sąryšius. Gautas modelis bus 3NF.
4. Išanalizuoti visus visose esybėse egzistuojančius f-sąryšius, ieškant tokių, kurių determinantas nėra galimas raktas. Radus tokį f-sąryšį, esybę suskaidyti į dvi ir apibrėžti tarp naujųjų esybių reikiamus sąryšius. Gautas modelis bus BKNF.
5. Išanalizuoti visus visose esybėse egzistuojančius netrivialius daugiareikšmius sąryšius, ieškant tokių, kurie nėra f-sąryšiai. Radus tokį MV-sąryšį, esybę suskaidyti į dvi ir apibrėžti tarp naujųjų esybių atitinkamus sąryšius. Gautas modelis bus 4NF.
6. Visi nurodyti žingsniai yra vykdomi nuosekliai. Kadangi kiekvienos aukštesnės normalinės formos reikalavimai apima žemesnės formos reikalavimus, kai kuriuos žingsnius galima praleisti. Taip apdorojus E-R modelį ir vėliau jį atvaizdavus reliacine schema pagal anksčiau pateiktas taisykles, gausime duomenų bazės reliacinę schemą, esančią 4NF.

5. Duomenų bazės sukūrimas ir užpildymas duomenimis

5.1. Įvadas

Jau aptarėme duomenų bazių loginės struktūros sudarymą. Sudarius DB gauname jos reliacinę schemą: visų atributų padalijimą į lenteles, lentelių pirminius bei išorinius raktus. Turint DB schemą, konkrečios DBVS terpėje galime fiziškai sukurti sudarytas lenteles ir kitus DB objektus.

Sukūrus lenteles, į jas galima įvesti duomenis. Duomenys įvedami duomenų įvedimo sakiniu. Tai - duomenų modifikavimo sakinys. Duomenų modifikavimo sakiniais duomenys taip pat keičiami bei šalinami. Priešingai užklausoms, kurios gali būti sudaromos kelioms lentelėms, duomenų modifikavimo sakiniais keičiama tik viena lentelė.

Esamų duomenų keitimo bei šalinimo sakiniai turi dvi formas: paieškos ir pozicinę. Šiame skyriuje aptarsime tik paieškos formos sakinius. Pozicinę sakinių formą aptarsime vėliau.

Prieš sukuriant lenteles, reikia sukurti pačią duomenų bazę.

5.2. Duomenų bazės sukūrimas

DB sistemose, funkcionuojančiose kompiuterių tinkluose, naujų DB kūrimu rūpinasi sistemos administratorius. Tam, kad būtų galima naudotis duomenų baze, esančia serveryje, vartotojo darbo vietoje reikia įdiegti DBVS vartotojo (kliento) posistemę. Tipinė DBVS vartotojo posistemė neleidžia darbo vietoje kurti savų DB. Tačiau, įdiegus reikiamus sistemos modulius, darbo vietos vartotojui – administratoriui leidžiama turėti savas DB.

Prieš kuriant DB, reikia įvertinti fizinės kompiuterio atminties poreikį būsimos DB duomenims saugoti ir parinkti tinkamą vietą. Priklausomai nuo DBVS ir operacijų sistemos ypatybių, fizinei DB galima skirti vietą kompiuterio diske, disko dalyje ar operacijų sistemos failuose. Visas tinkamas duomenų bazei fizines kompiuterio atminties vietas vadinsime **DB įrenginiu** (angl. *database device*).

SQL standarte nėra sakinio duomenų bazei kurti. Taip yra dėl DB įrenginio priklausomybės nuo konkrečios operacijų sistemos ir komercinėse DBVS esančių priemonių duomenų bazėms kurti įvairovės. Tačiau daugelyje DBVS yra komanda CREATE DATABASE ar atitinkama tarnybinė programa duomenų bazėms sukurti. DB valdymo sistemoje IBM DB2, duomenų bazę galima paprasčiausiai sukurti nurodžius DB vardą ir jos įrenginį. Pavyzdžiui,

CREATE DATABASE *Darbai* ON D:

Taip sukurtą DB *Darbai* galima sunaikinti sakiniu:

DROP DATABASE *Darbai* .

Reliacinėse DBVS svarbiausia duomenų bazės dalis yra lentelė. Savo ruožtu, lentelės pagrindinis struktūrinis elementas - stulpelis, kurio pagrindinė charakteristika yra jo duomenų tipas. SQL duomenų tipas yra reliacinės teorijos domeno atitikmuo. Prieš aptardami lentelės sukūrimą, susipažinsime su duomenų tipais.

5.3. Duomenų tipai

Stulpelio duomenų tipas apibrėžia stulpelio duomenų rūšį (simboliai, skaičiai, datos ir pan.). Reliacinės sistemos leidžia naudoti įvairius duomenų tipus. Labai svarbu stulpeliui parinkti tinkamą duomenų tipą, nes daugumoje DBVS yra gana sudėtinga stulpelio duomenų tipą pakeisti. Kita vertus, daugumoje DBVS yra gana daug funkcijų reikšmių tipui keisti. Pavyzdžiui, jei simbolių tipo stulpelyje yra tik skaičiai, pakeisti stulpelio tipą į skaičių tipą gali būti neįmanoma. Tačiau, specialiomis funkcijomis galima atlikti aritmetines operacijas su tokiais simboliniais duomenimis.

Daugelyje reliacinių DBVS naudojami tokie duomenų tipai:

- **Simbolių duomenų tipai** (angl. *character datatypes*). Šių tipų reikšmės yra raidžių, skaitmenų ir specialųjų ženklų sekos – simbolių eilutės. Simbolių kiekis eilutėje vadinamas eilutės ilgiu. Kiekvienam simbolių duomenų stulpeliui būtina nurodyti didžiausią leistiną eilutės ilgį. Skiriamos fiksuoto ir kintamo ilgio simbolių eilutės. Fiksuoto ir kintamo ilgio simbolių eilutės skiriasi jų fiziniu vaizdavimu ir panaudojimo ypatybėmis.

  DB valdymo sistemoje IBM DB2 fiksuoto ilgio simbolių eilutės didžiausias ilgis gali svyruoti nuo 1 iki 254. Fiksuoto ilgio simbolių eilučių tipas žymimas `CHAR`(n), kur n yra didžiausias leistinas ilgis. Fiksuoto ilgio simbolių eilutei fizinėje kompiuterio atmintyje visuomet skiriama tiek baitų, kiek jų reikia didžiausio leistino ilgio simbolių eilutei, nepriklausomai nuo faktinio eilutės ilgio.

  Kintamo ilgio simbolių eilutės gali būti vieno iš trijų tipų: `VARCHAR`(n), `LONG VARCHAR`(n) ir `CLOB(`n[K|M|G]), kur n – didžiasias leistinas eilutės ilgis, kuris gali būti nurodytas K-kilo, M-mega ar G-giga baitais. Pagal nutylėjimą ilgis skaičiuojamas baitais. Šie trys tipai skiriasi didžiausiu eilutės ilgiu: `VARCHAR` (angl. *variable character*) tipo eilutė gali užimti atmintyje iki 4000 baitų, `LONG VARCHAR` – iki 32.700, o `CLOB` (angl. *character large object*) - net iki 2 gigabaitų (2GB).

  SQL riboja ilgų simbolių eilučių panaudojimą. Pvz., simbolių duomenų tipo stulpelių, kurių reikšmių didžiausias leistinas ilgis yra didesnis už 254, negalima naudoti `SELECT` sakinyje su fraze `DISTINCT.` Tokių stulpelių negalima naudoti ir `GROUP BY` bei `ORDER BY` frazėse. Panašių apribojimų yra ir daugiau. Dauguma šių apribojimų paaiškinami šių operacijų realizacijos sudėtingumu. Baitų kiekis, kurie išskiriami atmintyje kintamo ilgio simbolių eilutei, priklauso nuo faktinio jos ilgio.

  SQL sakiniuose simbolių duomenų tipų reikšmės (konstantos) yra išskiriamos apostrofais, pvz., ‘Matematikos ir informatikos fakultetas’, ‘Naugarduko 24, Vilnius’.

- **Skaičių duomenų tipai (**angl. *number datatypes***)**. Šių tipų reikšmės **– s**veiki ir trupmeniniai skaičiai. Šie tipai visuomet turi tikslumą - skaičiaus skaitmenų arba baitų kiekį.

  `SMALLINT` žymi “mažus” sveikus skaičius. Tai - skaičiai, kuriuos galima pavaizduoti dviejuose baituose: nuo -32768 iki 32767, pavyzdžiui, 23, +56, -789, 8965.

  `INTEGER` - “dideli” sveiki skaičiai, kurių vaizdui kompiuteryje reikia 4 baitų: nuo -2.147.483.648 iki 2.147.483.647. Pvz.: 589654, 23, -545545445.

  `REAL` - slankaus kablelio skaičiai, kuriems vaizduoti atmintyje skiriami 4 baitai. Pvz.: 1.02, -2E5, 5.555E-18, -.655645e8.

  `DOUBLE` arba `FLOAT` - dvigubo tikslumo skaičiai (skiriama 8 baitai) su slankiu kableliu. Tai – skaičiai, patenkantys į intervalus [-1.79769e308, -2.225e307] ir [2.225e-307, 1.79769e308]. Pvz.: 23E5, -2.555E-58, .655645e98.

  `DECIMAL` arba `NUMERIC` - dešimtainiai skaičiai, kurių tikslumas priklauso nuo jų apibrėžimo, bet ne didesnis nei 31 dešimtainis skaitmuo. Apibrėžiant tokio tipo lentelės stulpelį reikia nurodyti bendrą skaitmenų kiekį ir skaitmenų kiekį po dešimtainio kablelio (taško). Bendras dešimtainių skaičių tipo žymuo yra `DECIMAL(`n, m**)**, kur n - bendras skaitmenų kiekis ir m - skaitmenų po kablelio kiekis. Dešimtainių konstantų pavyzdžiai: 15, -125.5, 1234569012345.123.

- **Dvejetainių duomenų tipai** (angl. *binary datatypes*). Šių tipų stulpeliai skirti dvejetainiams kodams saugoti. Kaip ir simbolių eilutės, dvejetainiai kodai gali būti fiksuoto (`BIT`(n)) ir kintamo (`BIT VARYING`(n) ir `BLOB(`n[K|M|G])) ilgio, kur n – didžiausias leistinas dvejetainio kodo ilgis baitais. `BLOB` (angl. *binary large object*) tipo duomenys gali būti iki 2 gigabaitų (2GB). Dvejetainiais kodais saugomi grafiniai vaizdai, audio ar video įrašai ir pan.

- **Datos ir laiko duomenų tipai**. Daugelyje DBVS yra trys šios rūšies duomenų tipai: `DATE` – datos duomenų tipas, `TIME` – laiko duomenų tipas ir `TIMESTAMP` - tikslaus laiko duomenų tipas. Šių tipų duomenys kompiuterio atminyje vaizduojami vidiniu formatu,

kuris vartotojui neprieinamas. DATE tipo reikšmėms (datoms) kompiuterio atmintyje skiriama 4 baitai, TIME (laikui) - 3 baitai, o TIMESTAMP (data ir tikslus laikas) - 10 baitų. SQL sakiniuose data ir laikas vaizduojami ne vidiniame formate, bet vartotojui įprastai - simbolių eilutėmis. Datos ir laiko vaizdavimas priklauso nuo vartotojo terpės bei DB parametrų, pavyzdžiui, šalies kodo (angl. *country code*). Lietuvoje data paprastai vaizduojama 10 simbolių eilute, o laikas - 8 simboliais, pvz., '2001.01.01', '12:00:00'. Tikslaus laiko duomenų formatas nepriklauso nuo terpės. DBVS DB2 - tai 26 simbolių eilutė, pvz.,' 2001-01-01-12.15.55.330000'.

Visi SQL duomenų tipai turi ypatingą reikšmę NULL, skirtą pažymėti, kad stulpelis eilutėje neturi prasminės reikšmės.

Parinkus stulpeliui tinkamą duomenų tipą, reikia parinkti duomenų tipo ilgį ir tikslumą. Parenkant duomenų tipui ilgį reikia atsižvelgti į faktiškai išskiriamos atminties dydžio priklausomybę nuo reikšmės, t.y. ar reikšmei išskiriamas baitų kiekis priklauso nuo konkrečios reikšmės. Pavyzdžiui, visoms stulpelio, apibrėžto CHAR(40), reikšmėms kompiuterio atmintyje bus skiriama tiek baitų, kiek jų reikia keturiasdešimčiai simbolių kodų (40 baitų), nepriklausomai nuo to, kiek faktiškai simbolių įvedama. Ilgesnės įvestos reikšmės bus patrumpinamos, o trumpesnės – papildomos tarpais. Reikšmei 'Jonaitis' skiriama 40 baitų. Jei iš anksto žinoma, kad dauguma reikšmių bus trumpesnės už apibrėžime nurodytą ilgį, tai galimas didelis atminties pereikvojimas. Taip atsitinka parenkant tipą stulpeliui, skirtam pavardėms. Žinoma, kad lietuvišką pavardę sudaro iki 40 simbolių, nors daugumai pavardžių užrašyti užtenka 20 raidžių.

Kintamo ilgio duomenys atmintyje užima tiek baitų, kiek jų reikia faktinei reikšmei užrašyti kompiuterio atmintyje. Pavyzdžiui, parinkus stulpeliui VARCHAR(40) tipą, ilgesnės reikšmės, kaip ir fiksuoto ilgio atveju, patrumpinamos. Tačiau trumpesnės reikšmės nepapildomos tarpais. Kintamo ilgio duomenų tipo atveju, reikšmei 'Jonaitis' skiriami 8 baitai.

Parenkant stulpeliui kintamo ilgio duomenų tipą, reikia prisiminti, kad kiekvienai reikšmei reikia saugoti ne tik ją pačią, bet ir faktišką jos ilgį arba jos pabaigos požymį. Todėl apibrėžimas VARCHAR(1) yra teisingas, bet neprasmingas. Be to, kintamo ilgio duomenų apdorojimas yra mažiau efektyvus negu fiksuoto ilgio.

Dauguma skaičių duomenų tipų yra fiksuoto ilgio ir tikslumo. Tik dešimtainių skaičių apibrėžime galima nurodyti tikslumą. Dešimtainiai skaičiai yra dažnai naudojami finansiniuose uždaviniuose. Pastebėsime, kad mūsų šalies biudžetą negalima išreikšti slankaus kablelio skaičiumi centų tikslumu, nes netgi dvigubo tikslumo reikšmėje neužtenka reikšminių skaitmenų. Dešimtainiuose skaičiuose gali būti net 31 reikšminis skaitmuo. DECIMAL(31,2) tipo reikšme galėsime tiksliai išreikšti net ir didelės šalies biudžetą.

5.4. Lentelių apibrėžimas

Lentelės (struktūros) apibrėžimas yra "aktyvus". SQL DDL sakiniu CREATE TABLE ne tik apibrėžiama duomenų struktūra, bet ir sukuriamas realus DB objektas. Lentelės apibrėžimas (kūrimas) yra vienkartinis veiksmas. DBVS neleidžia kurti lentelės su tokiu pat vardu kaip jau egzistuojančios. Kuriant (apibrėžiant) lentelę, būtinai nurodoma:

- lentelės vardas, kurį galima patikslinti schemos vardu;
- lentelės stulpelių vardai ir jų tipai.

Bendruoju atveju, sakinyje CREATE TABLE galima nurodyti daugiau lentelės savybių, pavyzdžiui, ar stulpelyje yra galimos NULL reikšmės. Kiekvienam stulpeliui galima nurodyti reikšmę pagal nutylėjimą (angl. *default value*), kuri priskiriama stulpeliui, kai įvedamoje eilutėje jam nenurodyta jokia reikšmė.

DB *Darbai* lenteles *Vykdytojai*, *Projektai* ir *Vykdymas* galima sukurti taip:

```
CREATE TABLE Vykdytojai (
    Nr          INTEGER     NOT NULL,
    Pavardė     CHAR(30)    NOT NULL,
```

```
    Kvalifikacija   CHAR(16)      DEFAULT 'Informatikas',
    Kategorija      SMALLINT,
    Išsilavinimas   CHAR(10)),

CREATE TABLE Projektai (
    Nr              INTEGER       NOT NULL,
    Pavadinimas     VARCHAR(254)  NOT NULL,
    Svarba          CHAR(10)      DEFAULT 'Vidutinė',
    Pradžia         DATE,
    Trukmė          SMALLINT),

CREATE TABLE Vykdymas (
    Projektas       INTEGER       NOT NULL,
    Vykdytojas      INTEGER       NOT NULL,
    Statusas        VARCHAR(32)   DEFAULT 'Programuotojas',
    Valandos        SMALLINT).
```

Šiuose trijuose SQL sakiniuose fraze `NOT NULL` pažymima, kad stulpelyje negali būti `NULL` reikšmės. Tuomet DBVS pasirūpins, kad jokiomis aplinkybėmis, jokioje eilutėje taip neatsitiktų. Stulpelyje galimos `NULL` reikšmės, jei frazė `NOT NULL` nenurodyta. Fraze `DEFAULT` apibrėžiama reikšmė pagal nutylėjimą.

Apibrėžiant lentelę, papildomai galima nurodyti pirminį raktą bei išorinius raktus. Šias ir kitas galimybes aptarsime vėliau. Daugumą lentelės savybių galima apibrėžti vėliau, jau sukurtai lentelei. Tai atliekama sakiniu `ALTER TABLE`. Juo galima papildyti lentelę, pavyzdžiui, nauju stulpeliu. Tačiau keičiant lentelės struktūrą, susiduriama su apribojimais. Dažnai DB valdymo sistemos neleidžia naujiems stulpeliams nurodyti savybės `NOT NULL`. Papildykime lentelę *Vykdytojai* gimimo datos stulpeliu:

```
ALTER TABLE Vykdytojai ADD Gimtadienis DATE.
```

SQL standarte numatyta galimybė sakiniu `ALTER TABLE` pašalinti lentelės stulpelį, pavyzdžiui,

```
ALTER TABLE Vykdytojai DROP Gimtadienis.
```

Tačiau konkrečios DBVS neleidžia šalinti stulpelių. Sakiniu `ALTER TABLE` taip pat galima keisti stulpelio savybes, pavyzdžiui, reikšmę pagal nutylėjimą:

```
ALTER TABLE Projektai ALTER Svarba DROP DEFAULT;
ALTER TABLE Projektai ALTER Svarba SET DEFAULT 'Didelė'.
```

5.5. Naujų duomenų įvedimas

Nauji duomenys į DB bazę įvedami sakiniu `INSERT`, įterpiant eilutes į pasirinktą lentelę. Šis sakinys yra dviejų formų. Pirmosios formos sakiniu įterpiama viena nauja eilutė. Stulpelių reikšmės nurodomos pačiame sakinyje:

```
INSERT INTO <lentelės vardas> [(<stulpelio vardas>{, <stulpelio vardas>})]
VALUES     (<reikšmė> {,<reikšmė>})
<reikšmė> ::= <reiškinys> | NULL | DEFAULT .
```

Šiuo sakiniu į lentelę įterpiama viena nauja eilutė. Išvardintiems stulpeliams priskiriamos reikšmes, esančios už bazinio žodžio `VALUES`. Tarp stulpelių ir jų reikšmių nustatoma atitinkamybė "iš kairės į dešinę". Jei stulpelių vardai praleisti, laikoma, kad stulpeliai išvardinti ta tvarka, kuria jie buvo išvardinti sukuriant lentelę. Jei kuriam nors lentelės stulpeliui nenurodyta reikšmė, tai jis įgyja `NULL` reikšmę arba reikšmę pagal nutylėjimą, jei tokia reikšmė stulpeliui buvo priskirta. Sakinio vykdymas baigiasi nesėkme, jei stulpeliui su

savybe NOT NULL, sakinyje nenurodyta jokia prasminė reikšmė ir jam neapibrėžta reikšmė pagal nutylėjimą arba tiesiog nurodyta reikšmė NULL.

Užregistruokime naują besimokantį bendradarbį pavarde Baltakis. Suteikime jam informatiko kvalifikaciją ir antrą kategoriją. Jo duomenis galima įvesti vienu iš šių sakinių:

```
INSERT INTO Vykdytojai (Nr, Pavardė, Kvalifikacija, Kategorija, Išsilavinimas)
            VALUES (6, 'Baltakis', 'Informatikas', 2, NULL),

INSERT INTO Vykdytojai VALUES (6, 'Baltakis', 'Informatikas', 2, NULL),

INSERT INTO Vykdytojai (Nr, Pavardė, Kategorija) (6, 'Baltakis', 2),
```

Antrosios formos INSERT sakiniu galima įterpti į lentelę užklausos rezultatą:

```
INSERT INTO <lentelės vardas> [(<stulpelio vardas>{, <stulpelio vardas>})]
            <užklausa> .
```

Šiuo sakiniu visos užklausos rezultato eilutės įtraukiamos į pasirinktą lentelę. Atitinkamybė tarp lentelės stulpelių ir užklausos rezultato stulpelių nustatoma taip pat, kaip pirmosios formos sakinyje.

Galimybė įvesti duomenis, kurie jau yra kurioje nors vienoje ar keliose lentelėse, gali pasirodyti gana keista. Ji naudinga duomenims administruoti. Kartais lentelėse susikaupia labai daug duomenų. Dėl to darbas su lentele gali pastebimai sulėtėti. Tarkime, kad seni duomenys naudojami daug rečiau nei nauji. Tuomet senuosius duomenis galima perkelti į archyvo lentelę. Susikaupus labai daug duomenų, paieška archyve sulėtės, tačiau duomenų paieška pagrindinėje lentelėje, kuri naudojama dažniau, išliks greita. Be to, po tam tikro laiko gali tapti netikslinga saugoti didelį kiekį senų duomenų. Dažnai jau po kelerių ar net vienerių metų išsamūs duomenys tampa nereikalingais, nes pakanka tik mėnesinių ataskaitų. Todėl patogu turėti atskirą lentelę ataskaitoms kaupti, į kurią sugrupuoti pagrindinės lentelės duomenys yra periodiškai perkeliami.

Tarkime, lentelėje *Projektai* saugomi duomenys tik apie projektus, kurie vykdomi dabar ar pasibaigė ne seniau nei prieš metus. Duomenys apie senesnius projektus kaupiami kitoje lentelėje *Seni_Projektai*, kurios struktūra yra tokia pat kaip lentelės *Projektai*. Įtraukime į lentelę *Seni_Projektai* duomenis apie prieš metus ar anksčiau pasibaigusius projektus:

```
INSERT INTO Seni_Projektai
SELECT * FROM Projektai
WHERE Pradžia + Trukmė MONTHS < CURRENT DATE – 1 YEAR.
```

Taip lentelės *Projektai* duomenys kopijuojami į lentelę *Seni_Projektai*. Seni duomenys išlieka ir lentelėje *Projektai*. Kaip pašalinti duomenis iš lentelės sužinosime vėliau.

Duomenis į lentelę galima įvesti ir specialiomis programomis. Duomenų, esančių faile, įterpimas į lentelę vadinamas **duomenų importu**. Įvairiose DBVS vartojami skirtingi duomenų failų formatai. Greta duomenų importo programų yra ir duomenų eksporto programos. **Duomenų eksportas** leidžia užklausos rezultatą išsaugoti faile pasirinktame formate. Vėliau duomenis galima importuoti iš failo į pasirinktą DB lentelę. Kai kurie duomenų formatai, pavyzdžiui, ASCII, yra bendrai naudojami daugelyje komercinių DBVS. Taip galima apsikeisti duomenimis tarp DB, funkcionuojančių skirtingose DBVS.

5.6. Duomenų šalinimas

Kai realaus pasaulio objektas "išnyksta", prireikia pašalinti eilutę iš lentelės. Pavyzdžiui darbuotojui išėjus iš darbo, saugoti jo duomenis bazėje tampa netikslinga. Objektui išnykus, pašalinami ir jį atitinkantys duomenys, kad DB išliktų tiksliu realaus pasaulio modeliu. Panašiai yra su projektais. Užbaigus vykdyti projektą, tikslinga atitinkamas eilutes pašalinti iš lentelių *Projektai* ir *Vykdymas*. Tačiau nepašalinus šių eilučių nereiškia, kad DB tampa prieštaringa. Tiesiog tuomet laikoma, kad bazėje saugomi duomenys ne tik apie šiuo metu vykdomus projektus, bet ir apie anksčiau vykdytus.

Lentelės eilutės šalinamos SQL sakiniu:

DELETE FROM <lentelės vardas> [WHERE <paieškos sąlyga>]

Frazėje FROM nurodoma lentelė, iš kurios eilutes reikia šalinti. Fraze WHERE nurodoma šalintinų eilučių atrinkimo sąlyga. Atliekant sakinį visos (!) tenkinančios paieškos sąlygą eilutės pašalinamos iš lentelės. Iš lentelės pašalinamos visos eilutės, jei frazės WHERE nėra. Sakiniu

DELETE FROM *Vykdymas*

pašalinamos visos lentelės *Vykdymas* eilutės. Lentelė (jos apibrėžimas), žinoma, išlieka. Lentelė tampa tuščia. Todėl, vartojant duomenų šalinimo sakinį, reikia būti ypač atidiems nurodant paieškos sąlygą, kad nepašalinti perdaug ar ne tas eilutes.

Jau pateikėme sakinį lentelės *Projektai* eilutėms, atitinkančioms prieš metus pabaigtus projektus, įvesti į tokios pat struktūros lentelę *Seni_Projektai*. Užrašykime sakinį perkeltoms į "archyvą" eilutėms pašalinti iš pradinės lentelės:

DELETE FROM *Projektai*
WHERE *Pradžia* + *Trukmė* MONTHS < CURRENT DATE – 1 YEAR.

Tarkime, Baltakis išėjo iš darbo. Pašalinkime eilutę, atitinkančią šį žmogų, iš lentelės *Vykdytojai*, be to pašalinkime visas eilutes, susijusias su jo dalyvavimu vykdant projektus:

DELETE FROM *Vykdymas*
WHERE *Vykdytojas* = (SELECT *Nr* FROM *Vykdytojai* WHERE *Pavardė* = 'Baltakis'),

DELETE FROM *Vykdytojai* WHERE *Pavardė* = 'Baltakis'.

Pastebėsime, kad šiuos du sakinius reikia vykdomi būtent tokia tvarka. Vėliau sužinosime, kad papildžius lentelių *Vykdymas* ir *Vykdytojai* apibrėžimus, pirminiais ir išoriniais raktais, pirmasis šalinimo sakinys iš pastarųjų dviejų taps nereikalingu. Dėl išorinio rakto, pašalinus eilutę iš lentelės *Vykdytojai*, lentelės *Vykdymas* reikiamos eilutės bus pašalintos automatiškai.

Pateiktas sakinys lentelės *Vykdymas* eilutėms šalinti nėra vienintelis. Užrašysime dar kelis sakinius, lentelės *Vykdymas* eilutėms, susijusioms su Baltakio dalyvavimu projektuose, šalinti:

DELETE FROM *Vykdymas*
WHERE 'Baltakis' = (SELECT *Pavardė* FROM *Nr* = *Vykdytojas*),

DELETE FROM *Vykdymas*
WHERE EXISTS (SELECT * FROM *Vykdytojai*
WHERE *Pavardė* = 'Baltakis' AND *Nr* = *Vykdytojas*).

Abiejuose šiuose sakiniuose pavartotos užklausos yra priklausomos (žr. skyrelį 2.6), kadangi turi parametrą (*Vykdytojas*), kuris įgyja reikšmę užklausos išorėje. Todėl, pastarieji du sakiniai rodo kaip nereiktų daryti!

5.7. Esamų duomenų keitimas

Duomenų bazėje saugomus duomenis reikia atnaujinti, kai įvyksta pasikeitimai atitinkamuose realaus pasaulio objektuose. Tarkime, firmos darbuotoja Gražulytė baigė Vilniaus universitetą ir dėl to jai buvo pakelta kategorija vienetu. Tam, kad DB išliktų tiksliu realaus pasaulio modeliu, reikia pakeisti lentelės *Vykdytojai* eilutę.

Pasirinktos lentelės duomenys keičiami tokiu sakiniu:

UPDATE <lentelės vardas> SET <stulpelio vardas> = <reiškinys>
{,<stulpelio vardas> = <reiškinys>}
[WHERE <paieškos sąlyga>]

Fraze WHERE nurodoma keistinų eilučių atrinkimo sąlyga. Atliekant sakinį atnaujinamos visos (!) atitinkančios paieškos sąlygą lentelės eilutės. Fraze SET nurodomi stulpeliai, kurių reikšmes norime keisti, bei naujosios reikšmės.

Minėti Gražulytės gyvenimo pasikeitimai atsispindės duomenų bazėje įvykdžius sakinį

UPDATE *Vykdytojai* SET *Išsilavinimas* = 'VU', *Kategorija* = *Kategorija* + 1
WHERE *Pavardė* = 'Gražulytė' .

Nurodant atnaujinamų duomenų paieškos sąlygą, reikia būti nemažiau atidiems, negu šalinant duomenis. Jei pastarąjį sakinį, pavyzdžiui, įvykdytume, nenurodę jokios paieškos sąlygos, tai kategorija būtų padidinta bei išsilavinimas taptų vienodu visiems vykdytojams.

Dar kartą pašalinti jau pašalintus duomenis neįmanoma. Pakartotinas duomenų šalinimo sakinio įvykdymas lentelės turinio nepakeis. Keičiant duomenis, lentelės turinys gali keistis po kiekvieno sakinio įvykdymo. Pavyzdžiui, įvykdžius pastarąjį UPDATE sakinį du kartus, kategorija padidės ne vienetu, bet dviem.

Tarkime, anksčiau minėtas Baltakis išėjo iš darbo ir dėlto, projektų, kuriuose jis dalyvavo, vykdymas sulėtėja. Realiai, darbuotojo išėjimas iš darbo negali būti priežastimi pratęsti projektų vykdymo terminą. Visgi, pratęskime visų projektų, kuriuose dalyvavo Baltakis, vykdymo terminą dešimtadaliu:

UPDATE *Projektai* SET *Terminas* = *Terminas* * 1.1
WHERE *Projektai.Nr* IN (SELECT *Projektas* FROM *Vykdymas*, *Vykdytojai*
WHERE*Vykdytojas* = *Vykdytojai.Nr* AND *Pavardė* = 'Baltakis').

Paieškos sąlygoje yra pavartota užklausa, kurios rezultatas yra projektų, kuriuose dalyvavo Baltakis, numeriai. Šiuo sakiniu duomenys keičiami tik vienoje lentelėje *Projektai*, tačiau apibrėžiant atnaujinamas šios lentelės eilutes kreipiamasi į kitas dvi lenteles.

Padidinkime kategoriją visiems darbuotojams, kurie dalyvauja bent dviejuose projektuose:

UPDATE *Vykdytojai* SET *Kategorija* = *Kategorija* + 1
WHERE (SELECT COUNT(*) FROM *Vykdymas* WHERE *Vykdytojas* = *Nr*) >= 2.

Užklausa, esanti paieškos sąlygoje, yra priklausoma. Tai pavyzdys, kad užklausoje kartais yra sunku išvengti parametro. Net ir šiame uždavinyje išvengsime priklausomos užklausos, jei pasinaudosime duomenų grupavimu:

UPDATE *Vykdytojai* SET *Kategorija* = *Kategorija* + 1
WHERE *Nr* IN (SELECT *Vykdytojas* FROM *Vykdymas*
GROUP BY *Vykdytojas* HAVING COUNT(*) >= 2).

Šio sakinio vidinės užklausos rezultatas yra visų vykdytojų, kurie dalyvauja bent dviejuose projektuose, numeriai. Kadangi tokia užklausa neturi parametro, tai jos rezultatą galima sudaryti prieš duomenų keitimą.

5.8. Lentelių ir DB šalinimas

Paprastai, lentelė duomenų bazėje sukuriama vieną kartą ir išlieka joje ilgą laiką. Lentelės egzistavimo metu į ją įvedamos naujos, keičiamos bei šalinamos anksčiau įvestos eilutės. Ypatingais atvejais, pavyzdžiui keičiant DB struktūrą (reliacinę schemą), prireikia pašalinti lentelę. Lentelės pašalinimas reiškia, ne tik visų jos eilučių pašalinimą, bet ir lentelės apibrėžimo pašalinimą. Tai atliekama SQL sakiniu:

DROP TABLE <lentelės vardas>.

Įvykdžius tokį sakinį, lentelė nustoja egzistavusi. Lentelę *Vykdymas* galima sunaikinti taip:

DROP TABLE *Vykdymas*.

Praktikoje prireikia pašalinti ir visą DB. SQL standarte nėra sakinio, kuriuo tai būtų galima atlikti. Panašiai kaip ir DB kūrimo atveju, dauguma DBVS turi komandą `DROP DATABASE`, kuri leidžia tai padaryti. Pavyzdžiui, sakiniu

```
DROP DATABASE Darbai
```

DB *Darbai* pašalinsime iš kompiuterio atminties. Dirbti su šia DB negalėsime.

6. Virtualios lentelės ir duomenų nepriklausomumo lygiai

6.1. Įvadas

Virtualia lentele (**vaizdu**) (angl. *view*) vadinama užklausa, kuriai suteikiamas vardas ir ji įsimenama duomenų bazėje. Vartotojui leidžiama peržiūrėti įsimintos užklausos rezultatą, kreipiantis į jį kaip į paprastą lentelę.

Antrame skyriuje mes susipažinome su laikinomis lentelėmis. Virtuali lentelė, kaip ir laikina lentelė, yra užklausos rezultatas, kuriam suteiktas vardas. Tačiau, laikina lentelė yra tik bendresnės užklausos tarpinis rezultatas, kuris nustoja egzistuoti įvykdžius užklausą. Sukūrus virtualią lentelę, ji išlieka tol, kol nebus sunaikinta. Virtualios lentelės turi laikinos lentelės savybes, bet jų vaidmuo ir pritaikymo sritis yra žymiai platesni.

Pagrindinės virtualių lentelių vartojimo priežastys yra šios:

- jomis galima sudaryti vartotojams savitą duomenų bazės sandaros įvaizdį;
- jomis galima paslėpti lentelių eilutes ir stulpelius;
- jos supaprastina vartotojo darbą, nes supaprastėja duomenų struktūra.

Paskutinė priežastis kartu yra ir laikinų lentelių naudingumo priežastis.

6.2. Virtualių lentelių sukūrimas

Virtuali lentelė neturi savo duomenų, ji remiasi kitų lentelių duomenimis. Apie virtualios lentelės egzistavimą sistema žino tik iš sisteminio katalogo, kuriame saugomi duomenys apie ją. Tikrai egzistuoja tik virtualios lentelės apibrėžimas, bet ne jos duomenys. Virtuali lentelė sukuriama (apibrėžiama) SQL duomenų apibrėžimo (DDL) sakiniu `CREATE VIEW`:

```
CREATE VIEW <virtualios lentelės vardas> [(<stulpelio vardas> {,<stulpelio vardas> })]
            AS <užklausa> [WITH CHECK OPTION].
```

Jei virtualios lentelės stulpelių vardai, nėra išvardyti už jos pavadinimo, tai jie paveldimi iš užklausos. Užklausoje išvardintas lenteles vadinsime pradinėmis vitualiai lentelei. Virtuali lentelė nustoja egzistavusi tik tuomet, kai ji sunaikinama sakiniu

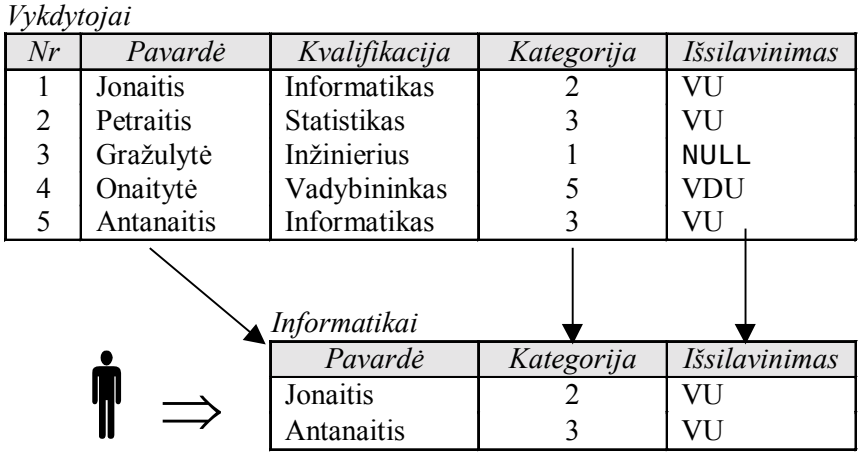
```
DROP VIEW <virtualios lentelės vardas>.
```

Apibrėžkime virtualią lentelę su duomenimis apie informatikus:

```
CREATE VIEW Informatikai (Pavardė, Kategorija, Išsilavinimas)
    AS SELECT Pavardė, Kategorija, Išsilavinimas
        FROM Vykdytojai WHERE Kvalifikacija = 'Informatikas'.
```

DB valdymo sistema, vykdydama šį sakinį, užklausos, esančios frazėje `AS`, nevykdo. Ši užklausa tik apibrėžia virtualios lentelės duomenis. Sakinio rezultatas – virtualios lentelės apibrėžimas duomenų bazės sisteminiame kataloge. Apibrėžus virtualią lentelę, vartotojas gali elgtis su ja kaip su paprasta lentele.

Virtuali lentelė yra dinamiška. Visi pasikeitimai pradinėje lentelėje *Vykdytojai* atsispindi virtualioje lentelėje (pradinės lentelės *Vykdytojai* vaizde) *Informatikai*, ir atvirkščiai, pakeitus duomenis virtualioje lentelėje, pakeitimai atsispindi ir pradinėje. Keičiant duomenis virtualioje lentelėje, iš tikrųjų, bus keičiami pradinės lentelės duomenys, nes virtualioji savų duomenų neturi. Virtuali lentelė yra lyg "langas" į pradinę lentelę. *Informatikai* yra "langas" į firmos darbuotojus (į lentelės *Vykdytojai* duomenis), per kurį matosi tik informatikai. Kitaip tariant, virtuali lentelė *Informatikai* – tai požiūris į darbuotojus, kai matomi tik informatikai ir nekreipiama dėmesio į jų numerius:



6.1 pav. Virtuali lentelė *Informatikai* – "langas" į lentelę *Vykdytojai*.

Virtualiai lentelei *Informatikai* priklauso (matoma per "langą") tik tos lentelės *Vykdytojai* eilutės ir stulpeliai, kurie atitinka paieškos sąlygą, nurodytą virtualios lentelės apibrėžime. Papildžius lentelę *Vykdytojai* naujais duomenimis, priklausomai nuo naujų darbuotojų kvalifikacijos, jie bus "matomi" per "langą" *Informatikai* (jei naujiems darbuotojams *Kvalifikacija* = 'Informatikas') arba "nematomi".

Užklausos virtualiai lentelei sudaromos kaip ir paprastai lentelei, pavyzdžiui, antros kategorijos informatikus sužinosime tokia užklausa:

SELECT * FROM *Informatikai* WHERE *Kategorija* = 2.

Kai DBVS aptinka SQL sakinyje kreipinį į virtualią lentelę, ji susiranda virtualios lentelės apibrėžimą sisteminiame kataloge. Pagal tai, DBVS perdaro užklausą virtualiai lentelei į užklausą pradinei lentelei ir vykdo ją. Taip DBVS sudaro virtualiai lentelei paprastos lentelės įvaizdį. Sistema, vietoje pateiktos užklausos, vykdys tokią užklausą:

SELECT *Pavardė*, *Kategorija*, *Išsilavinimas* FROM *Vykdytojai*
WHERE *Kvalifikacija* = 'Informatikas' AND *Kategorija* = 2.

Jei virtualios lentelės apibrėžimas toks nesudėtingas, kaip *Informatikai*, DBVS gali formuoti užklausos rezultatą iš karto, perrinkdama pradinės lentelės eilutes. Jei apibrėžianti užklausa yra sudėtinga, pvz. su duomenų grupavimu, sistemai tenka **materializuoti** virtualią lentelę. Pradžioje, DBVS įvykdo apibrėžiančiąją užklausą suformuodama ir įsimindama jos rezultatą. Tik po to vartotojo užklausa vykdoma pagal tarpinį rezultatą.

Apibrėžiančioje užklausoje neleidžiama naudoti duomenų rūšiavimo frazės ORDER BY. Virtualios lentelės duomenų sutvarkymą galima nurodyti užklausoje, pavyzdžiui,

SELECT * FROM *Informatikai* ORDER BY *Pavardė*.

6.3. Virtualių lentelių rūšys

Virtualios lentelės dažnai taikomos vartotojų galimybėms (teisėms) valdyti. Pavyzdžiui, darbuotojams, dirbantiems konkrečiame projekte turėtų "nerūpėti" tai, kaip vykdomi kiti projektai. Lentelę *Vykdymas*, galima "padalyti" į tris virtualias lenteles, kurių kiekviena atitiktų vieną iš trijų projektų:

CREATE VIEW *Vykdymas1* AS SELECT * FROM *Vykdymas* WHERE *Projektas* = 1,
CREATE VIEW *Vykdymas2* AS SELECT * FROM *Vykdymas* WHERE *Projektas* = 2,
CREATE VIEW *Vykdymas3* AS SELECT * FROM *Vykdymas* WHERE *Projektas* = 3.

Virtualios lentelės, kurias sudaro pradinės lentelės dalis eilučių, vadinamos **horizontaliomis**. Lentelės dažnai skaidomos horizontaliai organizacijose, turinčiose keletą

padalinių. Tuomet kiekvieno padalinio darbuotojai dirba tik su savo padalinio duomenimis. Visų padalinių veiklos duomenys prieinami tik visos organizacijos valdybai.

Lentelės duomenis galima padalyti horizontaliai kiekvienam asmeniui atskirai. Sukurkime virtualią lentelę su duomenimis apie Jonaičio dalyvavimą projektuose:

```
CREATE VIEW Vykdo_Jonaitis AS
SELECT * FROM Vykdymas
WHERE Vykdytojas = (SELECT Nr FROM Vykdytojai WHERE Pavardė = 'Jonaitis').
```

Horizontaliomis virtualiomis lentelėmis paslepiamos pradinės lentelės eilutės. Nemažiau svarbu paslėpti duomenis, esančius stulpeliuose. Pavyzdžiui, paprastam firmos darbuotojui galima uždrausti žinoti, kiek uždirba jo kolegos. Uždrauskime firmos darbuotojams matyti kolegų kategorijas bei išsilavinimą, sukurdami virtualią lentelę, į kurią šie duomenys nepatenka:

```
CREATE VIEW Bendri_Duomenys SELECT Nr, Pavardė, Kvalifikacija FROM Vykdytojai.
```

Virtualios lentelės, kuriose yra visos pradinės lentelės eilutės, bet ne visi stulpeliai, yra vadinamos **vertikaliomis**.

Praktikoje dažnai vartojamos **mišrios** virtualios lentelės, kurios yra lentelės horizontalus ir vertikalus padalijimai. Jau apibrėžta virtuali lentelė *Informatikai* yra mišrios virtualios lentelės pavyzdys.

Jei virtualios lentelės apibrėžime duomenys grupuojami, tai tokia virtuali lentelė vadinama **grupine**. Grupinės virtualios lentelės vartojamos dažnoms ataskaitoms įsiminti. Sukurkime virtualią lentelę, surenkančią statistiką apie kiekvieno projekto vykdymą:

```
CREATE VIEW  Apie_Vykdymą(Projektas, Visos_Valandos, Vidurkis) AS
SELECT Projektas, SUM(Valandos), AVG(Valandos) FROM Vykdymas
GROUP  BY Projektas.
```

Tokia virtuali lentelė supaprastina daugelio statistikos uždavinių sprendimą. Pavyzdžiui, kiek vidutiniškai kiekvienam projektui skiriama valandų, sužinosime tokia nesudėtinga užklausa:

```
SELECT AVG(Visos_Valandos) FROM Apie_Vykdymą.
```

Jungtinė virtuali lentelė, apibrėžiama užklausa, kurioje jungiamos kelios lentelės. Sukurkime virtualią lentelę, kuri apimtų visus DB duomenis apie projektą Nr. 1:

```
CREATE VIEW Projektas1 AS
SELECT Projektai.*, Vykdytojas, Pavardė, Kvalifikacija, Kategorija, Išsilavinimas,
        Statusas, Valandos FROM Projektai, Vykdytojai, Vykdymas
WHERE Projektas = Projektai.Nr AND Vykdytojas = Vykdytojai.Nr  AND Projektas = 1.
```

Apibrėžiančioje virtualią lentelę užklausoje galima kreiptis ne tik į paprastą lentelę, bet ir į virtualią. Taigi, virtuali lentelė gali būti pradine kitai virtualiai lentelei. Apibrėžkime virtualią lentelę, kuri apimtų duomenis apie projektą Nr. 2. Tam galime pasinaudoti jau apibrėžta virtualia lentele *Vykdymas2*:

```
CREATE VIEW Projektas2
AS SELECT Projektai.*, Vykdytojas, Pavardė, Kvalifikacija, Kategorija, Išsilavinimas,
        Statusas, Valandos FROM Projektai, Vykdytojai, Vykdymas2
WHERE Projektas = Projektai.Nr AND Vykdytojas = Vykdytojai.Nr.
```

6.4. Virtualių lentelių atnaujinimas

Virtualios lentelės duomenų atnaujinimas pakeičiamas paprastos lentelės atnaujinimu. Padidinkime Jonaičiui kategoriją vienetu virtualioje lentelėje *Informatikai*:

UPDATE *Informatikai* SET *Kategorija* = *Kategorija* + 1 WHERE *Pavardė* = ‘Jonaitis’.

DBVS elgiasi panašiai, kaip vykdydama užklausą virtualiai lentelei: virtualios lentelės *Informatikai* duomenų atnaujinimas pakeičiamas pradinės lentelės *Vykdytojai* duomenų atnaujinimu. Taigi, iš tikrųjų vykdomas toks sakinys:

UPDATE *Vykdytojai* SET *Kategorija* = *Kategorija* + 1
WHERE *Pavardė* = ‘Jonaitis’ AND *Kvalifikacija* = ‘Informatikas’.

Informatikai yra mišri virtuali lentelė. Ji turi ir horizontalios, ir vertikalios virtualios lentelės savybes. Tiek horizontaliose, tiek ir vertikaliose virtualiose lentelėse reikšmes galima keisti, jei tai leidžiama pradinėje lentelėje.

Tarp grupinės virtualios lentelės ir jos pradinės lentelės eilučių nėra atitinkamybės “vienas su vienu”. Todėl grupinės virtualios lentelės duomenų negalima nei keisti, nei šalinti. Į grupinę virtualią lentelę negalima įterpti naujų eilučių. Tarp jungtinės virtualios lentelės ir pradinių lentelių eilučių kartais galima rasti vienareikšmišką atitinkamybę. Tačiau daugelyje komercinių DBVS, jungtinės virtualios lentelės duomenis keisti negalima, kadangi SQL sakiniu galima keisti tik vienos lentelės duomenis.

SQL1 standarte suformuluotos sąlygos virtualiai lentelei, kurios duomenis galima atnaujinti. Pagal standartą, virtualią lentelę galima atnaujinti, jei ją apibrėžianti užklausa tenkina šiuos reikalavimus:

- nėra frazės DISTINCT;
- frazėje FROM yra tik viena pradinė lentelė ir ją galima atnaujinti. Jei pradinė lentelė yra virtuali, tai ji turi tenkinti visus šiuos reikalavimus;
- kiekvienas stulpelis yra paprastas lentelės stulpelis, o ne reiškinys;
- sąlygoje nėra kitos užklausos;
- nėra duomenų grupavimo.

Išvardyti reikalavimai remiasi paprastu principu: virtualią lentelę galima atnaujinti tik tuomet, kai kiekvienai atnaujinamai virtualios lentelės eilutei DBVS gali vienareikšmiškai nustatyti vieną eilutę pradinėje lentelėje, ir kiekvienam stulpeliui - pradinės lentelės stulpelį. Jei virtuali lentelė tenkina išvardytus reikalavimus, tai ir joje, ir pradinėje lentelėje galima keisti reikšmes, įterpti naujas bei šalinti esamas eilutes.

6.5. Virtualių lentelių atnaujinimo valdymas

Jei virtualią lentelę apibrėžiančioje užklausoje yra paieškos sąlyga, tai virtualioje lentelėje bus “matomos” tik sąlygą tenkinančios eilutės. Kitos pradinės lentelės eilutės bus “nematomos”. Pavyzdžiui, taip apibrėžtoje virtualioje lentelėje:

CREATE VIEW *Gudručiai*
AS SELECT * FROM *Vykdytojai* WHERE *Kategorija* > 2,

“matysime” tik tų vykdytojų duomenis, kurių kategorija aukštesnė nei 2. Taip apibrėžta virtuali lentelė tenkina visus duomenų atnaujinimo reikalavimus. Į šią virtualią lentelę galima įterpti naujas eilutes, pavyzdžiui,

INSERT INTO *Gudručiai* VALUES (7, ‘Uždavinys’, ‘informatikas’, 3, ‘VU’).

DBVS įterps naują eilutę į pradinę lentelę *Vykdytojai*. Panagrinėkime, kas atsitiks įvykdžius tokį sakinį:

INSERT INTO *Gudručiai* VALUES (8, ‘Juozaitis’, ‘informatikas’, 2, ‘VDU’).

Šis duomenų įvedimo sakinys yra sintaksiškai teisingas, be to virtualią lentelę galima atnaujinti, todėl DBVS įterps naują eilutę į lentelę *Vykdytojai*. Tačiau įvykdžius šį sakinį ir pažiūrėjus į virtualią lentelę *Gudručiai*, įvestos eilutės nepamatysime. Sakinys

`SELECT * FROM` *Gudručiai*

pateiks tokį pat rezultatą, kaip ir prieš duomenų įvedimą. Taip atsitinka todėl, kad stulpelio *Kategorija* reikšmė 2 netenkina virtualią lentelę apibrėžiančios sąlygos: *Kategorija* > 2.

Įvykdžius sakinį:

`UPDATE` *Gudručiai* `SET` *Kategorija* = 2 `WHERE` *Pavardė* = 'Jonaitis'

bus dar keisčiau. Virtualios lentelės eilutėje pakeitus vieną reikšmę, ši eilutė išnyksta iš jos, lyg būtų įvykdytas `DELETE` sakinys, o ne `UPDATE`. Tiksliau, "matoma" eilutė pavirto "nematoma". Kad taip neatsitiktų, galima uždrausti įvesti "nematomas" eilutes į virtualią lentelę. Taip pat galima uždrausti atnaujinti duomenis, kai šie iš "matomų" tampa "nematomais". Tam reikia virtualios lentelės apibrėžimą papildyti fraze `WITH CHECK OPTION`. Jei virtualią lentelę *Gudručiai* sukursime taip:

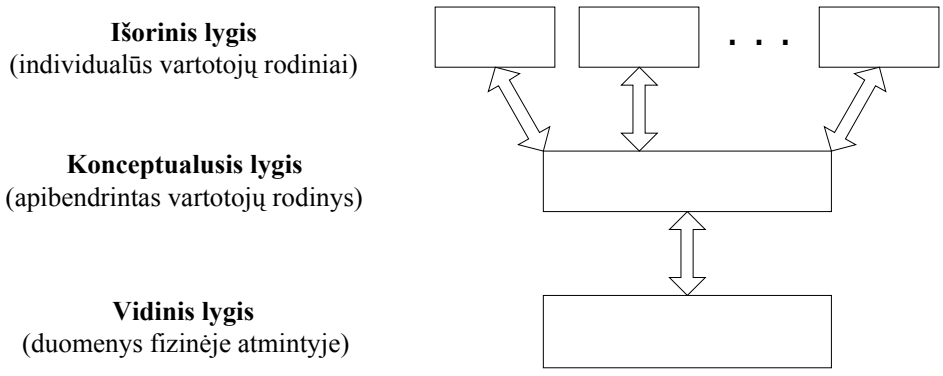
```
CREATE VIEW Gudručiai
AS SELECT * FROM Vykdytojai WHERE Kategorija > 2 WITH CHECK OPTION,
```

tai DBVS neleis įvesti į ją naujų eilučių, netenkinančių nurodytos paieškos sąlygos. Kiekvieną kartą atnaujinant duomenis taip apibrėžtoje virtualioje lentelėje, DBVS tikrina, ar neatsiras "nematomų" eilučių. Duomenis atnaujinti neleidžiama, jei tokios eilutės gali atsirasti.

Atnaujinimo tikrinimu negalima piknaudžiauti. Šis režimas neigiamai atsiliepia SQL sakinių `UPDATE` ir `INSERT` vykdymo efektyvumui. Kiekvieną kartą atliekant šiuos sakinius, DB valdymo sistemai reikia papildomai tikrinti "matomumo" sąlygą.

6.6. Duomenų nepriklausomumo lygiai

Viena iš pagrindinių virtualių lentelių paskirčių yra loginio duomenų nepriklausomumo užtikrinimas. Sistema užtikrina **loginį duomenų nepriklausomumą**, jei vartotojas ir jo vykdomos programos nepriklauso nuo loginės duomenų bazės struktūros (sandaros) pasikeitimų. Atitinkamai **fizinis duomenų nepriklausomumas** reiškia vartotojų ir jų programų nepriklausomumą nuo fizinės DB ypatybių. Abiems duomenų nepriklausomumams užtikrinti ANSI (American National Standards Institute) XX a. aštuntojo dešimtmečio viduryje pasiūlė trijų sluoksnių duomenų bazių valdymo sistemų architektūrą. Ši architektūra yra daugelio komercinių DBVS pagrindas. Architektūroje yra išskiriami vidinis, išorinis ir konceptualus lygiai (sluoksniai):



6.2 pav. Trys DBVS architektūros lygiai

Bendrais bruožais šiuos tris lygius galima apibūdinti taip:

- **Vidinis** lygis yra artimiausias fiziniam duomenų saugojimui. Jis susijęs su duomenų saugojimo fiziniuose nešėjuose būdais.

- **Išorinis** yra artimiausias vartotojui lygis, kuris susijęs su duomenų vaizdavimu atskiriems vartotojams. Tai vartotojų individualus DB loginės struktūros įvaizdis.
- **Konceptualusis** – tai tarpinis lygis, kuriame atsispindi visų duomenų, saugomų fizinėje atmintyje, loginė struktūra. Tai apibendrintas dalykinės srities modelis.

Loginis duomenų nepriklausomumas – tai nepriklausomumas tarp išorinio ir konceptualaus lygių, o fizinis nepriklausomumas – tarp konceptualaus ir vidinio lygių. Fizinis nepriklausomumas numato saugomų duomenų pernešimą iš vienų nešėjų į kitus, nekeičiant taikomųjų programų, dirbančių su DB. Loginis duomenų nepriklausomumas leidžia išlikti vartotojų programoms veikiančiomis, netgi pakeitus konceptualų duomenų modelį.

Fizinis nepriklausomumas pasiekiamas, daugiausiai, operacijų sistemos, kurioje funkcionuoja DBVS, priemonėmis. Plačiau aptarsime loginį nepriklausomumą.

6.7. Loginio duomenų nepriklausomumo užtikrinimas

Su loginiu duomenų nepriklausomumu yra susiję du pagrindiniai aspektai – DB **struktūros augimas** ir **keitimas**.

DB **struktūros augimas** galimas dviem būdais:

- papildant lentelę nauju stulpeliu;
- sukuriant naują lentelę.

Naujos lentelės sukūrimas neturi įtakos nei patiems vartotojams, nei jų programoms. Papildomos lentelės atsiradimas duomenų bazėje negali pakeisti naudojamų užklausų ir duomenų atnaujinimo.

Naujas stulpelis lentelėje gali daryti įtaką tik užklausoms, kurių `SELECT` frazėje yra nuoroda į visus lentelės stulpelius. Iš anksto neperspėjus vartotoją apie lentelės papildymą nauju stulpeliu, jis, eilinį kartą įvykdęs užklausą `SELECT * FROM`…, gali nemaloniai nustebinti, pamatęs nelauktą stulpelį. Nepakankamai kvalifikuotai sudaryta programa, kurioje tikimasi konkrečių užklausos rezultato stulpelių, gali nustoti veikti gavusi daugiau stulpelių.

Ankstesniame skyrelyje pateikėme duomenų įterpimo sakinį, kuriame duomenys apibrėžiami užklausa:

`INSERT INTO` *Seni_Projektai* `SELECT * FROM` *Projektai.*

Papildžius lentelę *Projektai* nauju stulpeliu ir nepadarius to paties su lentele *Seni_Projektai*, šio sakinio negalėsime vykdyti. Todėl, sudarant užklausas, ypač, jei jos vykdomos daugelį kartų, reikia vengti neišreikštinio stulpelių nurodymo.

Jei nelaukto stulpelio problema iškyla, tai naująjį stulpelį galima "paslėpti" virtualia lentele. Tarkime, lentelę $L(R)$ reikia papildyti naujais stulpeliais A. Pradžioje, sukuriame naują lentelę $L_1(R, A)$, į kurią perkeliame visus lentelės L duomenis, po to lentelę L sunaikiname ir sukuriame virtualią lentelę, atitinkančią pradinę lentelę L. Išreiškime tai SQL sakiniais:

`CREATE TABLE` $L_1(R, A)$;
`INSERT INTO` $L_1(R)$ `SELECT` R `FROM` L;
`DROP TABLE` L;
`CREATE VIEW` $L(R)$ `AS SELECT` R `FROM` L_1.

Su sukurta virtualia lentele L bus galima dirbti taip pat, kaip su ankstesne lentele L. Sukurdami virtualią lentelę mes išvengėme nepatogumų, kilusių dėl būtinumo papildyti lentelę nauju stulpeliu. Sprendžiant naujus uždavinius, kuriuose naujieji stulpeliai yra prasmingi, dirbsime su lentele L_1.

Kartais tenka keisti DB struktūrą taip, kad bendras duomenų kiekis išlieka toks pat, bet jų išdėstymas pasikeičia. Tuomet lentelių stulpeliai (atributai) yra pertvarkomi. Taip atsitinka kai pasikeičia atributų tarpusavio priklausomybės ir DB struktūra pradeda neatitikti konkrečios normalinės formos reikalavimų. Tarkime, mums reikia (priežastis nesvarbi) lentelę $L(A, B, C)$ pakeisti dviem lentelėmis $L_1(A, B)$ ir $L_2(A, C)$. Tuomet pradinės lentėlės L įvaizdį atstatysime sukurdami virtualią lentelę tuo pačiu vardu kaip pradinė lentelė:

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

Nuo šiol bet kuri programa ir interaktyvi operacija, kurioje vartojamas senasis lentelės vardas *L*, dabar kreipsis į virtualią lentelę tuo pačiu vardu *L*. Tačiau virtualioje lentelėje, kuri jungia dvi lenteles, negalima atnaujinti duomenų. Todėl sukūrę virtualią lentelę, mes pasiekėme tik užklausų loginį nepriklausomumą. Visiško loginio nepriklausomumo pasiekti nepavyksta.

Kita vertus, keičiant DB struktūrą, loginis duomenų nepriklausomumas turi prasmę tik tuomet, kai DB prieš pakeitimą ir DB po pakeitimo yra tapačios saugomais duomenimis. Pavyzdžiui, negalime kalbėti apie loginio duomenų nepriklausomumo užtikrinimą, kai pašalinami lentelės stulpeliai ar visos lentelės, t.y. kai duomenų bazės duomenys prarandami. Todėl, DB struktūros keitimo sąvoka neapima stulpelių ir lentelių šalinimo.

Loginio nepriklausomumo nuo struktūros keitimo užtikrinimas yra tik vienas virtualių lentelių panaudojimas. Susisteminsime virtualių lentelių privalumus.

6.8. Virtualių lentelių trūkumai ir privalumai

Pagrindiniai virtualių lentelių **privalumai** yra šie:

- **Saugumas**. Kiekvienam vartotojui galima sudaryti ir pateikti nedaug virtualių lentelių, kurios apimtų tik jam skirtus duomenis. Taip galima apriboti duomenų naudojimą. Sukuriant virtualias lenteles galima "paslėpti" realių lentelių stulpelius bei eilutes. Tuomet vartotojas "mato" tik tuos duomenis, kurie yra "matomi" per jam skirtą "langą". Tai nevisiškai išsprendžia duomenų apsaugos problemą, nes vartotojui neuždraudžiama peržiūrėti pradines lenteles. Virtualios lentelės tik papildo specialias duomenų apsaugos priemones, kurias aptarsime vėliau.
- **Užklausų paprastumas.** Dažnai vartojamoms sudėtingoms užklausoms galima apibrėžti virtualias lenteles. DBVS tokias užklausas "įsimina" sisteminiame kataloge. Tuomet vietoje sudėtingos užklausos pradinėms lentelėms, galima sudaryti paprastesnes užklausas virtualiai lentelei. Virtualios lentelės, kurios kuriamos naudoti užklausose, dažnai vadinamos **makrosais**. Šiame taikyme, virtualios lentelės apibrėžimas yra vardo suteikimas užklausai. Vėlesnėse užklausose tas vardas naudojamas užklausai sutrumpinti.
- **Struktūros paprastumas.** Virtualios lentelės sukuria kiekvienam vartotojui "savitą" DB struktūrą. Jungiant kelias lenteles į vieną virtualią, esama DB struktūra labai supaprastėja.
- **Loginis nepriklausomumas**. Duomenų bazės struktūra (reliacinė schema) gali keistis, pavyzdžiui, atsirasti naujų stulpelių. Atributų savybės bei jų tarpusavio ryšiai taip pat gali keistis. Tuomet lenteles prireikia skaidyti. Keičiantis DB struktūrai, galima sukurti virtualias lenteles, kurios "paslepia" daugelį pasikeitimų. Taip virtualios lentelės dalinai užtikrina loginį nepriklausomumą.

Virtualios lentelės keletą **trūkumų**. Štai jie:

- **Našumas**. Virtualios lentelės sudaro lentelės egzistavimo iliuziją. DB valdymo sistema užklausas virtualiai lentelei pakeičia užklausomis pradinėms (paprastoms) lentelėms. Jei virtuali lentelė jungia daug lentelių, tai paprasta užklausa virtualiai lentelei gali tapti sudėtingu lentelių jungimu, kuriam atlikti reikės daug kompiuterio resursų.
- **Ribotas atnaujinimas**. Duomenų atnaujinimas yra galimas tik gana paprastose virtualiose lentelėse. Virtualiose lentelėse, kurios jungia kelias lenteles, yra galima tik duomenų paieška.

Išvardyti trūkumai reiškia, kad negalima piktnaudžiauti virtualiomis lentelėmis. Kiekvienu konkrečiu atveju, kuriant ar vartojant virtualią lentelę, reikia atsižvelgti tiek į privalumas, tiek ir į trūkumus.

7. Duomenų vientisumo užtikrinimas

7.1. Reikalavimai duomenų vientisumui

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

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

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

7.2. Apribojimai reikšmėms

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

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

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

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

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

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

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

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

7.3. Lentelės raktų vientisumas

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

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

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

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

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

`DROP INDEX` *Raktas_Pavadinimas*.

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

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

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

7.4. Išoriniai raktai

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

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

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

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

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

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

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

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

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

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

Pagrindinė eilutė gali būti šalinama pagal vieną iš keturių taisyklių:

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

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

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

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

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

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

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

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

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

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

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

7.5. Dalykinės taisyklės ir trigeriai

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

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

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

Reikalavimų tikrinimas taikomosiomis programomis turi kelis svarbius trūkumus:

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

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

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

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

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

Trigeriai kuriami sakiniu `CREATE TRIGGER`, kuriame nurodoma:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Tačiau trigeriai turi ir trūkumų:

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

Trigeris nustoja egzistavęs, kai jis sunaikinamas sakiniu

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

7.6. Transakcijos

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Pabaigai, keletas papildomų pastabų:

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

8. SQL sakiniai taikomosiose programose

8.1. Įvadas

SQL yra bendravimo su reliacinėmis duomenų bazėmis kalba. Ją galima vartoti dviem būdais: **interaktyviai** ir **taikomosiose programose**. Toks kalbos dvilypumas turi keletą svarbių privalumų:

- visos galimybės, kurios yra prieinamos interaktyviai, yra prieinamos ir taikomosiose programose;
- pagrindines duomenų apdorojimo operacijas, iš pradžių, galima suderinti interaktyviai, o po to naudoti taikomosiose programose.

SQL nėra pilnavertė programavimo kalba, joje nėra, pavyzdžiui, veiksmų sekos valdymo sakinių. SQL efektyviai papildo konkrečią programavimo kalbą. SQL kalbos sakiniai taikomosiose programose atlieka patogios, aukšto lygio sąsajos su duomenų baze vaidmenį, o programavimo kalbos sakiniais yra patogu valdyti veiksmų seką, organizuoti vartotojo sąsają ir pan.

Šiuolaikinėse komercinėse RDBVS SQL sakinius programose galima vartoti dviem būdais:

- **Programų SQL** (angl. *embedded* SQL). SQL sakiniai vartojami programoje kaip programavimo kalbos sakiniai. Prieš kompiliuojant tokią programą programavimo kalbos kompiliatoriumi, išeities tekstas apdorojamas SQL preprocesoriumi.
- **Taikomųjų programų sąsaja** (angl. *application program interface* - API). DBVS pateikia programuotojui funkcijų rinkinį. Kviečiant tokias funkcijas, programa perduoda DB valdymo sistemai SQL sakinius, kuriuos reikia įvykdyti. SQL sakinio rezultatas grąžinamas programai kaip funkcijos rezultatas. Jokio preprocesoriaus nereikia, nes vartojamos funkcijos visiškai atitinka programavimo kalbos sintaksę.

Programų SQL pirmą kartą pavartotas firmos IBM DB valdymo sistemoje DB2. Devintajame dešimtmetyje ši galimybė jau buvo daugelyje komercinių RDBVS. Standartas SQL2 apibrėžia programų SQL tokioms programavimo kalboms: Ada, C, COBOL, FORTRAN, Pascal ir PL/1.

Pirmąją taikomųjų programų sąsają, kuri iki šiol sėkmingai vartojama, pateikė firma Microsoft savo produkte SQL Server. Programų sąsajai buvo suteiktas ODBC (*Open Database Connectivity*) pavadinimas. Įvertinę šios sąsajos patogumą, daugelis komercinių RDBVS netrukus pateikė sąsajas, artimas ODBC. Neseniai pasirodė sąsaja JDBC (*Java Database Connectivity*) leidžianti kreiptis į DB iš Java programų.

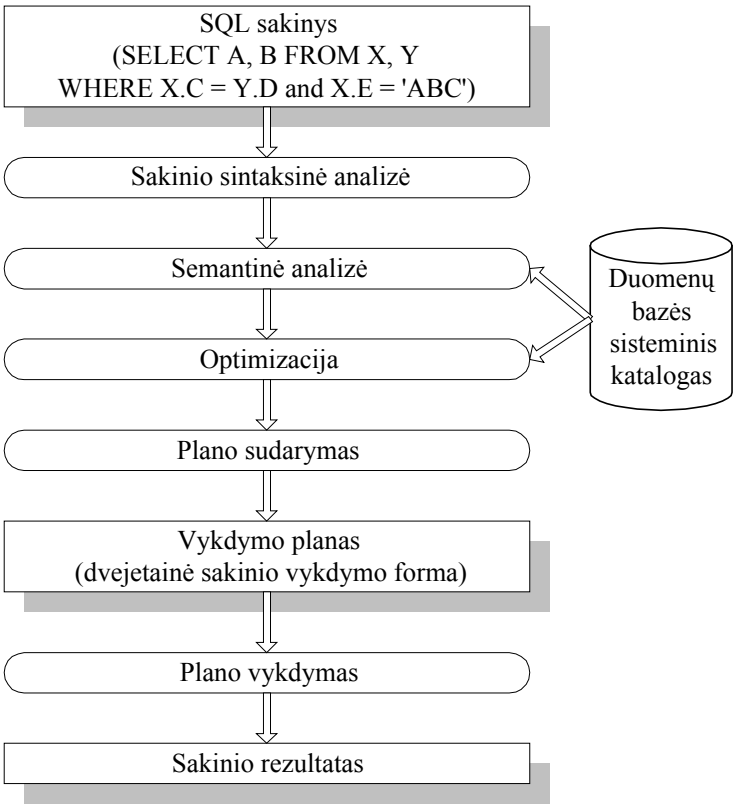
Praktikoje sėkmingai naudojama tiek programų SQL, tiek ir taikomųjų programų sąsaja. Šioje knygelėje aptarsime tik programų SQL.

8.2. SQL sakinio vykdymo etapai

Prieš pradedant nagrinėti programų SQL, išsiaiškinkime, kaip DBVS vykdo SQL sakinius. SQL sakinių vykdyme išskiriami 5 pagrindiniai etapai:

1. **Sintaksinė analizė**. Šiame etape tikrinama, ar SQL sakinys atitinka kalbos sintaksės reikalavimus, yra aptinkamos sintaksinės klaidos.
2. **Semantinė analizė.** Šiame etape DBVS tikrina SQL sakiniose pavartotų parametrų teisingumą. Pasinaudojant sisteminiu katalogu, yra tikrinama SQL sakiniuose vartojamų lentelių, stulpelių ir kitų DB objektų egzistavimas. Tikrinamos vartotojo teisės šių DB objektų atžvilgiu. Šiame etape aptinkamos semantinės klaidos.
3. **Optimizacija.** DBVS ištiria sakinio įvykdymo galimybes. Nustatoma galimybė panaudoti duomenų paieškai konkretų indeksą. Pavyzdžiui, jei užklausa yra kelioms lentelėms, nustatoma kas geriau: ar iš pradžių atrinkti reikiamas eilutes kiekvienoje lentelėje atskirai, o po to jas jungti, ar iš pradžių jungti lenteles, o po to atrinkti reikiamas eilutes, ir pan. DBVS, išanalizavusi alternatyvas, parenka tinkamiausią.

4. **Plano sudarymas.** DBVS sudaro SQL sakinio vykdymo dvejetainį (mašininį) planą. Vykdymo planas yra programos objektinio kodo atitikmuo.
5. **Plano vykdymas.** DBVS vykdo sudarytą planą, realizuojantį užklausą.



8.1 pav. SQL sakinio vykdymo etapai

SQL sakinio vykdymo etapai skiriasi tarpusavyje pagal kreipinių į DB kiekį bei vykdymo laiką. Sintaksinė analizė, kuriai nereikia kreiptis į DB, paprastai atliekama labai greitai. Optimizacija, atvirkščiai, reikalauja daug kreipinių į DB ir negali būti atlikta labai greitai. Tačiau, sugaištas laikas optimaliam sakinio įvykdymo keliui surasti dažnai atsiperka vykdant sakinį.

Vartojant SQL sakinius interaktyviai, visi penki etapai yra vykdomi nuosekliai kiekvienam SQL sakiniui. DBVS neturi kito pasirinkimo, kaip vykdyti sakinius interpretuojant juos. Tačiau, vartojant SQL sakinius programose, aplinkybės keičiasi. Dalis etapų gali būti atliekami programos kompiliavimo metu. Kiekvieno programos vykdymo metu tereikia atlikti tik likusius (neatliktus kompiliuojant) etapus. DBVS siekia kaip galima daugiau nuveikti kompiliuojant programą ir kuo mažiau veiksmų palikti programos vykdymui. Kartais DBVS kompiliavimo metu sudaro ir SQL sakinio vykdymo planą.

8.3. Programų SQL ypatybės

SQL sakiniai su programavimo kalbos sakiniais derinami prisilaikant tokių bendriausių reikalavimų:

- SQL sakiniai vartojami betarpiškai tarp programavimo kalbos sakinių. Programa, kurioje pavartoti SQL sakiniai, yra apdorojama SQL preprocesoriumi, kuris kompiliuoja tik SQL sakinius ir neliečia programavimo kalbos sakinių.
- SQL sakiniuose yra galimi kreipiniai į programavimo kalbos kintamuosius. Taip programos duomenis galima panaudoti SQL sakiniuose.
- SQL sakinio rezultatas perduodamas programai per jos kintamuosius, apibrėžtus programavimo kalba. Taip duomenys, esantys duomenų bazėje, gali būti apdorojami programa.

- `NULL` reikšmei perduoti ir priimti iš DB yra naudojami specialūs programos kintamieji.
- Nuosekliam užklausos rezultato perrinkimui yra vartojami papildomi SQL sakiniai, kurių nėra interaktyviame SQL.

Kadangi SQL sakinius kompiliuoja SQL preprocesorius, tai jie išskiriami specialiais atskyrėjais. SQL sakinių užrašymo programose taisyklės priklauso nuo konkrečios programavimo kalbos. Mes nagrinėsime, kaip SQL sakiniai yra vartojami kartu su programavimo kalba C. SQL sakiniai C programavimo kalba sudaromose taikomosiose programose rašomi pagal tokias taisykles:

- sakinys pradedamas prefiksu `EXEC SQL`;
- SQL sakinys išdėstomas tekste prisilaikant C programavimo kalbos sintaksės;
- SQL sakinys baigiamas C kalbos sakinių atskyrėju, t.y. kabliataškiu.

Visi programų SQL sakiniai yra skirstomi į **statinius** ir **dinaminius**. Ar SQL sakinys yra statinis, ar dinaminis priklauso nuo galimybės sudaryti vykdymo planą preprocesavimo (kompiliavimo) metu. Statiniams SQL sakiniams planas gali būti sudarytas preprocesuojant programą. Tam, kad sudaryti vykdymo planą, būtina tiksliai žinoti visus DB objektus (pavyzdžiui, lentelių ir stulpelių pavadinimus). Tarkime, mums reikia sudaryti programą, kuri išvestų į kompiuterio ekraną lentelės, kurios pavadinimą įveda programos vartotojas jos vykdymo metu, duomenis. Tokios programos sudarymo metu, programuotojas negali parašyti SQL sakinio, kurį reikės vykdyti programos vykdymo metu. Vadinasi, DBVS negali sudaryti vykdymo plano. Tokie uždaviniai sprendžiami dinaminiais SQL sakiniais. Pradžioje, detaliau aptarsime statinių SQL sakinių vartojimą programose.

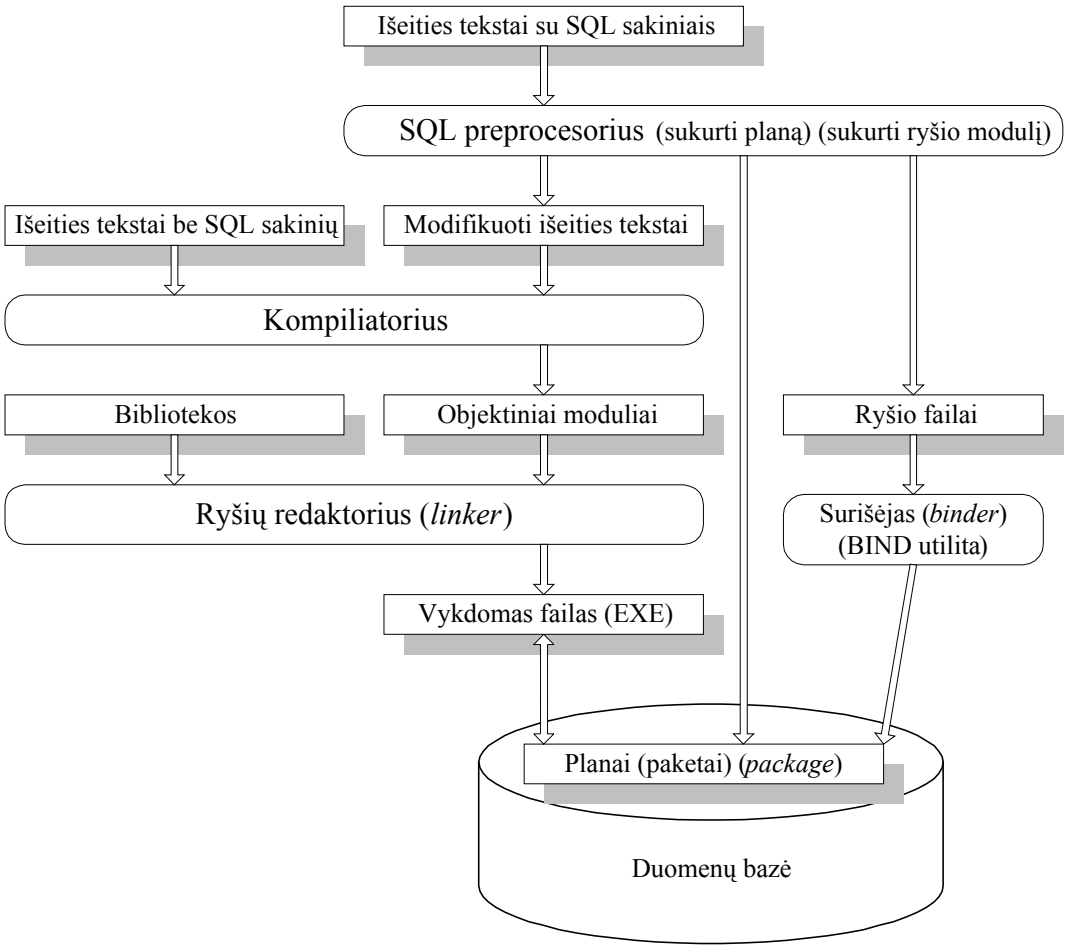
8.4. Programos paruošimo schema

Programa, sudaryta konkrečia programavimo kalba ir kurioje yra pavartoti SQL sakiniai, negali būti kompiliuojama programavimo kalbos kompiliatoriumi. Bendra daugiaetapio išeities teksto apdorojimo schema yra pavaizduota 8.2 pav. Tiksliau, paveiksle yra pavaizduotas programos apdorojimas sistemoje IBM DB2. Daugumoje komercinių DBVS programos apdorojamos labai panašiai. Trumpai aptarsime visus programos apdorojimo etapus.

1. Programos išeities tekstas yra apdorojamas SQL preprocesoriumi - atskira DBVS procedūra (tarnybine programa). Kiekviena programavimo kalba turi savo ypatybių, todėl reikalingas savitas SQL preprocesorius. Komercinės DBVS neleidžia naudoti bet kurią programavimo kalbą. Daugelyje jų vartojamos tik populiariausios programavimo kalbos: C, FORTRAN, COBOL, PL/1 ir kai kurias kitas.
2. Preprocesorius programos išeities tekstą pertvarko taip, kad jis atitiktų programavimo kalbos sintaksės reikalavimus ir galėtų būti kompiliuojamas programavimo kalbos kompiliatoriumi. Preprocesorius išeities tekste pašalina visus SQL sakinius, pakeisdamas juos vidinių (tiesiogiai neprieinamų vartotojams) funkcijų kreipiniais, kurie programos vykdymo metu palaiko ryšį tarp programos ir DBVS. Preprocesorius, apdorodamas programoje esančius SQL sakinius, gali sudaryti ir jų vykdymo planus, patalpindamas juos duomenų bazės sisteminiame kataloge. Tačiau, plano sudarymas preprocesavimo metu gali būti nepriimtinas, nes paruoštą programą gali reikėti vykdyti kitoje terpėje. Sudarant programą, dažnai naudojama testinė DB. Ta DB, kuriai programa yra skirta, gali būti kitur. Preprocesorius gali sudaryti ryšio su DB modulį (bylą) (angl. *Database Request Module* - DBRM). Į ryšio modulį preprocesorius sudeda visus duomenis, kurių reikia vykdymo planui sudaryti. Paprasčiausiu atveju, tai - visi programos SQL sakiniai. Kviečiant preprocesorių, parametrais yra nurodoma, koks turi būti jo rezultatas.
3. Programos tekstas, kuris jau apdorotas preprocesoriumi, toliau yra kompiliuojamas programavimo kalbos kompiliatoriumi. Po kompiliavimo gaunamas objektinis modulis. Šiame etape duomenų bazės nereikia.
4. Ryšių redaktorius visus objektinius modulius “apjungia” į vieną vykdomąją bylą. Šiame etape nustatomas ryšys tarp programoje pavartotų funkcijų kvietimų ir funkcijų

mašininių kodų, esančių bibliotekose. Vidiniai DBVS funkcijų kreipiniai, kuriuos sudaro preprocesorius, yra susiejami su DBVS bibliotekomis.

5. Jei vykdymo planas nebuvo sudarytas preprocesuojant programą, tai jis sudaromas specialia procedūra (tarnybine programa-surišėju, angl. *binder*), pagal preprocesoriaus sudarytą ryšio su DB modulį. Taip yra gaunamas jungtinis visų ryšio modulyje esančių SQL sakinių vykdymo planas. Šis planas yra įsimenamas DB sisteminiame kataloge. Jungtiniam planui paprastai suteikiamas pavadinimas, atitinkantis išeities bylos pavadinimą. Sistemoje DB2 jungtinis planas, atitinkantis vieną išeities bylą, dar vadinamas paketu (angl. *package*).



8.2 pav. Programos paruošimo schema

Suprantama, tarp programos paruošimo etapų ir anksčiau pateiktų SQL sakinio vykdymo etapų yra atitinkamybė. SQL preprocesorius visuomet atlieka sintaksinę analizę (pirmas SQL sakinio vykdymo etapas). Procedūra, kuri pagal ryšio modulį sudaro vykdymo planą, atlieka semantinę analizę, ieško optimalaus vykdymo plano ir sudaro jį (antras, trečias ir ketvirtas etapai). Visus šiuos darbus gali atlikti preprocesorius. Tik penktasis etapas – plano vykdymas atliekamas vykdant programą. Taip pradinė programa su SQL sakiniais suskaidoma į dvi dalis:

- **vykdomoji programa** - dvejetainis vykdomasis (mašininis) kodas, kuris saugomas byloje, kaip ir bet kuri kita programa, neturinti kreipinių į DBVS;
- **vykdymo planas** – dvejetainis (mašininis) kodas, esantis DB sisteminiame kataloge.

Toks programos paruošimo suskaidymas etapais ir pradinės programos suskaidymas į dvi dalis turi keletą svarbių privalumų:

- SQL sakinių vartojimas kartu su programavimo kalbos sakiniais leidžia patogiai spręsti duomenų, esančių duomenų bazėse, apdorojimo uždavinius.

- Preprocesoriaus panaudojimas leidžia didelę dalį darbų, susijusių su SQL sakinių vykdymu (sakinių analizė ir vykdymo optimizacija), atlikti programos paruošimo metu.
- DB ryšio modulio išskyrimas padaro programą pernešama. Programa gali būti sudaroma ir testuojama vienoje kompiuterinėje sistemoje, o vykdoma kitoje. Pernešant programą, yra pernešama vykdomoji byla ir ryšio modulis. Prieš vykdant programą naujojoje sistemoje, tereikia konkrečia procedūra-surišėju sudaryti vykdymo planą, įkeliant jį į DB.
- SQL sakinių pakeitimas vidinių ("paslėptų") funkcijų kreipiniais, leidžia preprocesuotą programą apdoroti įprastomis programavimo sistemomis, nepriklausomai nuo DBVS. Programuotojas, savo ruožtu, tvarkydamas ryšį su DB, naudojasi tik SQL sakiniais, nesirūpindamas apie kitas, kituose etapuose naudojamas, sudėtingesnes sąsajas.

Vykdant programą, susidedančią iš dviejų dalių: vykdomosios (EXE) bylos ir vykdomojo plano, galima išskirti keletą momentų:

- vykdomoji programa pradedama vykdyti įprastai, kaip ir bet kuri kita programa;
- pirmojo kreipinio į DBVS metu DB sisteminiame kataloge randamas vykdymo planas ir jis paruošiamas vykdymui, pakraunant jį į atmintį;
- kiekvienas kreipinys į "paslėptą" funkciją, atitinkančią SQL sakinį, realizuojamas įvykdant konkrečią vykdomojo plano dalį.

8.5. Paprastų statinių SQL sakinių vartojimas

Paprasčiausiai programose naudojami tie SQL sakiniai, kurie nėra susiję su užklausos rezultato, kurį gali sudaryti daug eilučių, apdorojimu. Tokie, pavyzdžiui, yra duomenų atnaujinimo sakiniai. Užrašykime keletą SQL sakinių, pateiktų penktame skyriuje, taip, kaip jie turėtų būti rašomi programoje C kalba:

```
EXEC SQL INSERT INTO Vykdytojai
         VALUES (6, 'Baltakis', 'Informatikas', 2, NULL) ;
EXEC SQL DELETE FROM Vykdytojai WHERE Pavardė = 'Baltakis';
EXEC SQL UPDATE Projektai SET Terminas = Terminas * 1.1
   WHERE Projektai.Nr IN (SELECT Projektas FROM Vykdymas, Vykdytojai
                          WHERE Vykdytojas = Vykdytojai.Nr AND Pavardė = 'Baltakis') ;
```

Tokių sakinių įdiegimas programoje nėra sudėtingas. Tačiau, programos vykdymas dažnai priklauso nuo ankstesnio sakinio rezultato. Vykdant panašius SQL sakinius, programa turėtų atsižvelgti bent į DBVS pranešimą apie sakinio įvykdymo sėkmingumą. Klaidos atveju reikia informuoti apie tai vartotoją.

Klaidų apdorojimas. Vykdant SQL sakinius interaktyviai, DBVS pranešimus apie sakinio įvykdymo sėkmę vaizduoja monitoriaus ekrane. Programoje už klaidų apdorojimą atsako ji pati. Kaip ir įprastose (be SQL sakinių) taikomosiose programose, galima išskirti dvi klaidų rūšis:

- **Kompiliavimo klaidos**. Sintaksines klaidas SQL sakiniuose aptinka SQL preprocesorius. Ištaisius klaidas, SQL preprocesorių reikia iškviesti dar kartą.
- **Vykdymo klaidos**. Kreipinius į neegzistuojančius DB objektus, pvz., stulpelius, dažnai galima aptinkti tik vykdant programą. Tokios klaidos aptinkamos ir sudarant vykdymo planą. Tačiau sudarant vykdymo planą, DB objektas gali egzistuoti, o programos vykdymo metu gali jo jau nebūti. Tik vykdymo metu galima aptikti klaidas apie ryšio su DB praradimą ar resursų trūkumą. Tokias klaidas būtinai reikia apdoroti taikomojoje programoje.

DBVS, įvykdžiusi SQL sakinį, visuomet grąžina programai klaidos kodą. Be to, taikomoji programa, gali prašyti DBVS išsamaus teksto, paaiškinančio klaidą. Klaidos kodas perduodamas specialiu kintamuoju `SQLCODE`. SQL2 standartas numato ir kitą kintamąjį `SQLSTATE`.

Pagal SQL1 standartą, duomenys apie SQL sakinio įvykdymą perduodami programai per **SQL ryšio sritį** (angl. SQL *Communication Area*). Tai duomenų struktūra, kurią užpildo DBVS, patalpindama klaidos kodą ir kitus duomenis, pavyzdžiui, atnaujintų eilučių kiekį. Kad ryšio sritis taptų prieinama programai, pirmuoju SQL sakiniu programoje turi būti programų SQL sakinys `INCLUDE SQLCA`. Programos C kalba pradžioje reikia įterpti eilutę:

```
EXEC SQL INCLUDE SQLCA;
```

Šiuo sakiniu preprocesoriui nurodoma įtraukti SQL ryšio sritį į programą. Šį SQL sakinį preprocesorius pakeičia tokiomis eilutėmis:

```
#include <sqlca.h>
struct sqlca sqlca ;
```

Byloje sqlca.h yra C kalbos struktūros `sqlca` apibrėžimas. Programos tekste patalpinamas ir kintamojo *sqlca* apibrėžimas. SQL kintamasis `SQLCODE` yra struktūros `sqlca` elementas, kuris byloje sqlca.h apibrėžtas taip:

```
#define SQLCODE sqlca.sqlcode
```

Tai reiškia, kad programuotojas programos tekste vietoje SQL kintamojo `SQLCODE` gali naudoti C programos kintamąjį *sqlca.sqlcode*. Sakinys `INCLUDE SQLCA` yra neprasmingas interaktyviame SQL. Jį galima vartoti tik programose.

Priskirdama SQL kintamajam **`SQLCODE` reikšmę**, DBVS praneša programai kaip pavyko įvykdyti SQL sakinį:

- Nulinė `SQLCODE` reikšmė reiškia, kad SQL sakinys įvykdytas sėkmingai.
- Neigiamas skaičius reiškia rimtą klaidą, dėl kurios sakinys nebuvo įvykdytas. Kiekvienai konkrečiai klaidai atitinka savas neigiamas skaičius.
- Teigiamas skaičius reiškia nenormalią situaciją. Taip pranešama, pavyzdžiui, apie įvedamos simbolių eilutės sutrumpinimą ar skaičiaus suapvalinimą. Vienas iš svarbiausių pranešimų yra "nėra duomenų". Taip atsitinka, pavyzdžiaui, kai atnaujinant duomenis, nei viena lentelės eilutė netenkina paieškos sąlygą. Šį pranešimą atitinka skaičius 100.

Programos tekste, po kiekvieno SQL sakinio reikia tikrinti `SQLCODE` reikšmę, pavyzdžiui:

```
EXEC SQL DELETE FROM Vykdytojai WHERE Pavardė = 'Baltakis';
if (0 > SQLCODE)
  printf("Įvyko klaida. Klaidos kodas: %ld\n", SQLCODE) ;
else if (0 == SQLCODE)
 printf("Duomenys pašalinti sėkmingai\n") ;
else if (100 == SQLCODE)
 printf("Šalintinų duomenų nerasta\n") ; .
```

Programoje galima sužinoti ne tik SQL sakinio įvykdymo kodą, bet ir prasminį pranešimą, kuris atitinka kodą. Kaip tai padaryti, galima sužinoti pasitelkus į pagalbą konkrečios DBVS dokumentaciją.

Sakinys `WHENEVER`. Programų SQL sakinys `WHENEVER` palengvina ypatingų situacijų tvarkymą. Tos situacijos apibūdinamos konkrečiomis klaidos kodo reikšmėmis:

```
WHENEVER NOT FOUND | SQLERROR | SQLWARNING
         CONTINUE | GO TO <programos žymė> .
```

Frazė `SQLERROR` atitinka neigiamą `SQLCODE` reikšmę, `SQLWARNING` – teigiamą, o `NOT FOUND` – ypatingą SQL kodą 100. Jei sakinyje yra frazė `GO TO`, tai SQL preprocesorius po kiekvieno SQL sakinio, esančio bet kurioje programos teksto vietoje, bet žemiau sakinio `WHENEVER`, įterpia sąlyginio valdymo perdavimo nurodyta žyme sakinį. Baziniu žodžiu `CONTINUE` nurodoma, kad atitinkamoms aplinkybėms susidarius, valdymas bus perduotamas

tiesiog kitai programos eilutei. Paprastai ši frazė vartojama norint nutraukti ankstesnio sakinio WHENEVER veiksmą, kuriuo buvo nukreipiamas tos situacijos apdorojimas.

Baziniai kintamieji. Pateiktuose pavyzdžiuose nebuvo parametrų. Tarkime, mums reikia iš lentelės *Vykdytojai* pašalinti duomenis apie darbuotoją, kurio numeris yra sužinomas programos vykdymo metu, pvz., jį įveda programos vartotojas. Tokiems uždaviniams spręsti programų SQL yra įvesta bazinio kintamojo sąvoka. **Bazinis kintamasis** tai - programos kintamasis, kuris apibrėžiamas programavimo kalba ir vartojamas SQL sakinyje. Tam, kad nebūtų dviprasmybių atskiriant bazinius kintamuosius nuo DB objektų (pvz., stulpelių) pavadinimų, prieš bazinius kintamuosius dedamas dvitaškis. Bazinius kintamuosius galima vartoti ten, kur ir SQL konstantos.

Baziniai, kaip ir kiti programos kintamieji, turi būti aprašyti. Vietą, kur programoje yra galimas kintamųjų aprašas, nustato programavimo kalbos sintaksė. Bazinių kintamųjų aprašą reikia papildomai išskirti. SQL preprocesorius SQL sakiniuose leidžia vartoti tik tuos kintamuosius, kurie aprašyti bazinių kintamųjų aprašo sekcijoje: tarp programų SQL sakinių BEGIN DECLARE SECTION ir END DECLARE SECTION, pavyzdžiui,

```
EXEC SQL BEGIN DECLARE SECTION;
  long nr ;
EXEC SQL END DECLARE SECTION;
nr = 1;
EXEC SQL DELETE FROM Vykdytojai WHERE Nr = :nr ;
```

Programavimo kalbos duomenų tipai gana dažnai skiriasi nuo duomenų tipų, vartojamų RDBVS. Pvz., C kalboje nėra numatyta specialių duomenų tipų datai ir laikui. Kai kurie duomenų tipai yra visiškai suderinami, pvz., C kalbos tipas long visiškai suderintas su SQL tipu INTEGER, short – su SMALLINT. Prisiminus, kad C kalboje simbolių eilutei išskirtas masyvas turi turėti papildomą baitą eilutės pabaigos požymiui - nuliui, nesunkiai nustatysime atitinkamybę tarp SQL tipo CHAR(n) ir simbolių masyvo char[n+1] C kalboje. Datos ir laiko konstantos SQL sakiniuose vaizduojamos simbolių eilutėmis. Atitinkami baziniai kintamieji C kalba aprašomi simbolių masyvais. Masyvo ilgis turi būti vienetu didesnis negu atitinkamos konstantos ilgis. Pavyzdžiui, datai galima vartoti kintamąjį, aprašytą taip: char[11], o laikui char[9].

Ypatinga situacija susidaro vartojant SQL NULL reikšmę. Daugumoje programavimo kalbų nėra tokios sąvokos. Todėl programose ši problema sprendžiama specialiu **kintamuoju-indikatoriumi**. Programų SQL sakinyje bazinis kintamasis kartu su kintamuoju-indikatoriumi apibrėžia vieną SQL reikšmę:

- Neigiama kintamojo-indikatoriaus reikšmė atitinka bazinio kintamojo reikšmę NULL. Faktinė bazinio kintamojo reikšmė šiuo atveju yra ignoruojama.
- Neneigiama kintamojo-indikatoriaus reikšmė reiškia, kad baziniame kintamajame yra tikroji reikšmė.

Kintamasis-indikatorius turi būti aprašytas kintamųjų aprašymo sekcijoje. C kalboje šis kintamasis turi būti short tipo. Paprastai kintamasis-indikatorius programos tekste rašomas iš karto po bazinio kintamojo. Tačiau, prieš kintamąjį-indikatorių galima papildomai prirašyti bazinį žodį INDICATOR. Šis bazinis žodis – nebūtinas. Jei žinoma, kad SQL reikšmė negali būti NULL, tai kintamasis-indikatorius nereikalingas.

Kintamojo-indikatoriaus negalima vartoti paieškos sąlygoje. Todėl vietoje sintaksiškai neteisingo užrašo:

```
EXEC SQL UPDATE Vykdytojai SET Kategorija = COALESCE(Kategorija, 0) + 1
          WHERE Išsilavinimas = :value :ind ;
```

galima rašyti:

```
if(ind < 0)
  EXEC SQL UPDATE Vykdytojai SET Kategorija = COALESCE(Kategorija, 0) + 1
```

```
            WHERE Išsilavinimas IS NULL ;
  else
    EXEC SQL UPDATE Vykdytojai SET Kategorija = COALESCE(Kategorija, 0) + 1
            WHERE Išsilavinimas = :value ; .
```

Kintamieji-indikatoriai yra labai patogūs kai programoje apdorojamos užklausos rezultato eilutės, kuriose yra galimos NULL reikšmės. Tokį kintamųjų-indikatorių vartojimą aptarsime kitame skyrelyje.

8.6. Statinis užklausų apdorojimas

Pagal anksčiau pateiktą metodiką programoje galima vykdyti praktiškai visus SQL sakinius, išskyrus sakinį SELECT. Šio sakinio ypatybė – rezultatas yra eilučių aibė, kurią gali būti sunku įkelti į programos atmintį. Užklausos rezultato perrinkimui eilutė po eilutės programų SQL yra **eilučių žymens** sąvoka (angl. *cursor*) ir keletas sakinių darbui su žymeniu. Trumpai aptarkime užklausos rezultato apdorojimą programoje.

- Sakiniu DECLARE CURSOR yra apibrėžiama užklausa ir su ja susiejamas žymuo.
- Sakiniu OPEN pradedamas užklausos vykdymas. Rezultato žymuo atidaromas. Jis yra prieš pirmąją užklausos rezultato eilutę.
- Sakiniu FETCH žymuo yra perkeliamas prie artimiausios užklausos rezultato eilutės (jei tokia egzistuoja) ir tos eilutės duomenys priskiriami baziniams kintamiesiems. Kiekvieną kartą vykdant sakinį FETCH žymuo perkeliamas vis prie kitos eilutės, kuri tampa einamąja - pažymėta. Jei užklausos rezultate daugiau eilučių nėra, DBVS grąžina klaidos kodą 100.
- Sakiniu CLOSE nutraukiamas užklausos rezultato perrinkimas, žymens vieta tampa neapibrėžta - žymuo uždaromas.

Sudarysime C funkciją visų vykdytojų pavardėms ir kategorijoms išvesti į ekraną:

```
void Vykdytojai_Kategorijos ()
{
  EXEC SQL INCLUDE SQLCA ;
  /* Bazinių kintamųjų apibrėžimas: */
  EXEC SQL BEGIN DECLARE SECTION;
    char   name [31];                    /* Pavardė       */
    short  category;                     /* Kategorija    */
    short  ind;                          /* Indikatorius  */
  EXEC SQL END DECLARE SECTION;
  /* Klaidų apdorojimo apibrėžimas: */
  EXEC SQL WHENEVER SQLERROR GOTO error;
  EXEC SQL WHENEVER NOT FOUND GOTO end;
  /* Užklausos ir jos rezultato žymens apibrėžimas: */
  EXEC SQL DECLARE curs CURSOR FOR SELECT Pavardė, Kategorija
                                   FROM Vykdytojai ORDER BY Pavardė;
  EXEC SQL CONNECT TO Darbai ;       /* - prisijungti prie DB  */
  EXEC SQL OPEN curs ;               /* - atidaryti žymenį     */
  while ( 1 ) {
    /* Skaitome einamąją užklausos rezultato eilutę: */
    EXEC SQL FETCH curs INTO :name, :category :ind ;
    /* Išvedame perskaitytos užklausos rezultato eilutės duomenis: */
    printf( "Pavardė: %s ", name ) ;
    if ( ind >= 0 )
      printf( "Kategorija: %d\n", category) ;
    else
      printf( "Kategorijos nėra\n") ;
```

```
    } /* ciklo pabaiga*/
    error:
       printf( "SQL klaida: %ld\n", SQLCODE) ;
    end:
       /* Tolimesnių SQL klaidų neapdoroti: */
       EXEC SQL WHENEVER SQLERROR CONTINUE;
       EXEC SQL WHENEVER NOT FOUND CONTINUE;
       EXEC SQL CLOSE curs ;              /* - uždaryti žymenį     */
       EXEC SQL CONNECT RESET ;           /* - atsijungti nuo DB   */
} /* funkcijos Vykdytojai_Kategorijos pabaiga */
```

Pateiksime keletą pastabų apie programų SQL sakinius, skirtus užklausos rezultato perrinkimui.

Žymens susiejimas su užklausa suteikia vartotojui galimybę vienu metu apdoroti keletą užklausų. Kadangi žymuo yra susiejamas su konkrečiu `SELECT` sakiniu, tai kiekvienas duomenų perrinkimo sakinys (`OPEN`, `FETCH`, `CLOSE`), kuriame vartojamas žymuo, nusako veiksmą konkrečios užklausos atžvilgiu. Programoje galima aprašyti keletą žymenų, kuriuos visus galima atidaryti ir su jais dirbti nepriklausomai. Pastebėsime, kad žymens vardas nėra programos kintamasis, tai SQL vardas, kurį naudoja tik SQL preprocesorius.

Užklausoje, kuri apibrėžiama sakiniu `DECLARE CURSOR`, galima vartoti ir bazinius kintamuosius. Užklausoje baziniai kintamieji gali būti ten, kur yra galimos SQL konstantos. Tačiau, `SELECT` frazėje baziniai kintamieji neleidžiami. Baziniai kintamieji, skirti rezultato eilutės reikšmėms priimti, nurodomi sakinyje `FETCH`.

Atidarytas užklausos rezultato žymuo išlieka tokiu, kol jis neuždaromas arba kol nesibaigia tranzakcija. Tranzakcijos pabaigos sakiniais `COMMIT` ir `ROLLBACK` uždaromi visi dar neuždaryti žymenys. Uždarytas žymuo gali būti vėl atidarytas. Atidarant žymenį DBVS visiškai pasiruošia užklausos vykdymui. Atidarant žymenį yra tikrinama, ar visos nurodytos užklausoje lentelės ir jų stulpeliai egzistuoja, ar jie nėra dviprasmiški, ar einamasis vartotojas turi teisę peržiūrėti apibrėžtus užklausa duomenis ir pan. Jei užklausos rezultatas yra tuščia eilučių aibė (nei viena eilutė netenkina sąlygos), tai šis faktas yra "pastebimas" tik vykdant `FETCH` sakinį, kuomet DBVS grąžina klaidos kodą 100.

Standartas SQL1 numato užklausos rezultato perrinkimą tik viena kryptimi ir tik eilutė po eilutės. "Šokinėti" tarp eilučių negalima. 90-jų pradžioje kai kurios komercinės DBVS įvedė **tiesioginio kreipimosi** į užklausos rezultatą sąvoką. Sakinys `FETCH` buvo papildytas galimybėmis `FIRST, LAST, PRIOR, ABSOLUTE` <eilutės numeris>, `RELATIVE` <+|–> <eilučių kiekis>. Kadangi tokios galimybės leidžia labai patogiai apdoroti užklausos rezultatą, tai jos buvo įrašytos į SQL2 standartą.

8.7. Pozicinis duomenų šalinimas ir keitimas

Užklausos rezultato apdorojimas – tai ne tik duomenų peržiūrėjimas, bet ir duomenų šalinimas bei atnaujinimas. Interaktyviai šalinamų ir atnaujinamų eilučių aibė aprašoma paieškos sąlyga. Praktiniuose uždaviniuose dažnai būna labai sunku užrašyti reikiamą paieškos sąlygą. Nuspręsti, ar eilutę reikia apdoroti (pašalinti, pakeisti), kartais yra galima tik po sudėtingų skaičiavimų. Pavyzdžiui, jei vykdytojų, kuriems reikia pakeisti kategoriją, sąrašas yra pateikiamas byloje, tai mums būtinai reikia sudaryti programą.

Programų SQL sakiniai `DELETE` ir `UPDATE` turi pozicines formas. Kad ir koks sudėtingas būtų sprendimo apie duomenų šalinimą ar atnaujinimą priėmimas, pakanka paprastos galimybės nurodyti, kad tai reikia atlikti su einamąja eilute.

Sakinio `DELETE` pozicinė forma yra paprasta:

`DELETE FROM` <lentelės vardas> `WHERE CURRENT OF` <žymens vardas> .

Šiuo sakiniu šalinama einamoji eilutė, t.y. eilutė, kuri šiuo metu yra pažymėta. Po šalinimo žymuo perkeliamas prie kitos eilutės, bet ši netampa einamąja, t.y. žymuo yra prieš kitą eilutę. Kad to kita eilutė taptų einamąja (žinoma, jei tokia egzistuoja), reikia įvykdyti sakinį `FETCH`.

Sakinyje `UPDATE` vietoje paieškos sąlygos taip pat galima nurodyti frazę `WHERE CURRENT OF`. Atnaujinus einamąją eilutę, ši išlieka einamąja. Todėl, jau atnaujintą eilutę, nepakeitus žymens vietos, galima dar kartą atnaujinti ar pašalinti.

Tam, kad bet kurią einamąją eilutę būtų galima keisti, užklausa turi tenkinti gana griežtus reikalavimus:

- frazėje `FROM` yra tik viena lentelė;
- nėra išrūšiavimo – frazės `ORDER BY`;
- nėra vienodų eilučių sutraukimo – frazės `DISTINCT`;
- nėra duomenų grupavimo – frazių `GROUP BY` ir `HAVING`;
- yra pažymėta, kad užklausos rezultatą galima keisti, - frazė `FOR UPDATE`.

Vartojant žymenį keičiamai eilutei nurodyti, kartais komercinės DBVS reikalauja ne tik nurodyti frazę `FOR UPDATE`, bet ir išvardinti stulpelius, kurių reikšmės gali būti keičiamos perrenkant užklausos rezultatą. Tokios žinios yra labai svarbios efektyviai valdyti sistemos resursus. Jei programuotojas žino, kad jis, atidaręs žymenį, nekeis ir nešalins nei vienos eilutės, tai jis gali pabrėžti tai užklausos apibrėžime parašydamas frazę `FOR READ ONLY`. Tokia informacija gali labai pagreitinti užklausos vykdymą bei sumažinti resursų sąnaudas.

Jau apibrėžtą funkciją *Vykdytojai_Kategorijos* pakeiskime, leisdami programos vartotojui kiekvieno vykdytojo atžvilgiu interaktyviai priimti vieną iš sprendimų: nekeisti vykdytojo duomenų, pakeisti jam kategoriją, pašalinti vykdytoją.

```
short Kategorijų_Atnaujinimas()
{
  EXEC SQL INCLUDE SQLCA ;
  EXEC SQL BEGIN DECLARE SECTION;
    char   name [31];                  /* - kintamasis pavardei   */
    short  category, ind;              /* - kintamasis kategorijai */
  EXEC SQL END DECLARE SECTION;
  char   inbuffer[40] ;
  short  ok = 1;
  EXEC SQL WHENEVER SQLERROR   GOTO error;
  EXEC SQL WHENEVER NOT FOUND  GOTO end;
```

```
EXEC SQL DECLARE curs CURSOR FOR SELECT Pavardė, Kategorija
      FROM Vykdytojai FOR UPDATE OF  Kategorija;  /*-galėsime keisti kategoriją */
EXEC SQL CONNECT TO Darbai ;          /* - prisijungti prie DB  */
EXEC SQL OPEN curs ;                  /* - atidaryti  žymenį    */
while ( 1 ) {
   EXEC SQL FETCH curs INTO :name, :category :ind ;
   printf( "Pavardė: %s ", name ) ;
   if ( ind >= 0 )
      printf( "Kategorija: %d\n", category) ;
   else
      printf( "Kategorija: – \n") ;
 action:
   printf( "Pasirinkite veiksmą: Next, Delete, Update, Finish, Cancel\n") ;
   gets(inbuffer) ;
   switch(inbuffer[0]) {
      case 'N':
         break ;
      case 'D':
         EXEC SQL DELETE FROM Vykdytojai WHERE CURRENT OF  curs ;
         break ;
      case 'U':
         printf( "Įveskite naują kategoriją: ") ;
         scanf("%d",&category) ;
         EXEC SQL  UPDATE Vykdytojai SET  Kategorija  = :category
                   WHERE CURRENT OF  curs ;
         break ;
      case 'F':
         goto end;
      case 'C':
         ok = 0 ;
         goto end;
      default:
         goto action;
   }
} /* Ciklo pabaiga*/
error:
   printf( "SQL Klaida: %ld\n", SQLCODE) ;
   ok = 0 ;
end:
   EXEC SQL WHENEVER SQLERROR   CONTINUE;
   EXEC SQL WHENEVER NOT FOUND  CONTINUE;
   EXEC SQL CLOSE curs ;              /* - uždaryti  žymenį     */
   if (ok)
      EXEC SQL COMMIT ;               /* - įtvirtinti pakeitimus */
   else
      EXEC SQL ROLLBACK ;             /* - atšaukti pakeitimus  */
   EXEC SQL CONNECT RESET ;           /* - atsijungti nuo DB    */
return ok ;
}  /* Funkcijos Kategorija_Atnaujinimas pabaiga */
```

Ši funkcija, išvedusi į ekraną eilutę (vykdytojo pavardę ir kategoriją), kviečia vartotoją pasirinkti vieną iš veiksmų, įvedant konkretų simbolį:

- ‘N’ – pereiti prie kitos eilutės;
- ‘D’ – pašalinti šią eilutę;
- ‘U’ – pakeisti šio vykdytojo kategoriją;
- ‘F’ – užbaigti duomenų peržiūrą, įteisinant visus atliktus pakeitimus (jei tokie buvo);
- ‘C’ – užbaigti duomenų peržiūrą, atsisakant visų atliktų pakeitimų (jei tokie buvo).

Įvykus SQL klaidai visi atlikti (jei tokių buvo) pakeitimai atšaukiami (anuliuojami). Sėkmingai apdorojus visas lentelės *Vykdytojai* eilutes, atlikti pakeitimai įtvirtinami (`COMMIT`). Ši funkcija, lyginant ją su ankstesniąja (*Vykdytojai_Kategorijos*), papildomai grąžina rezultate 1, sėkmingai pasibaigus operacijai, ir 0 – nesėkmės atveju.

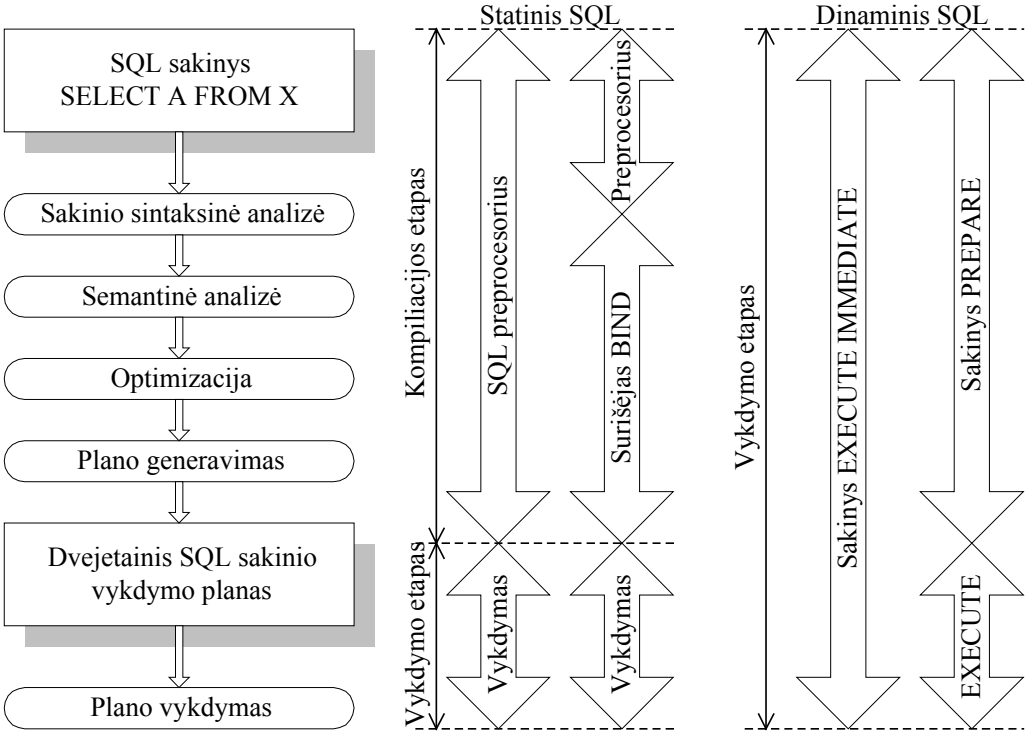
8.8. Dinaminių SQL sakinių vykdymas

Programa, sudaryta panaudojant statinius SQL sakinius, turi griežtai apibrėžtą kreipimosi į DB tvarką. Tokioje programoje, jos sudarymo metu yra nurodomi visi DB objektai, į kuriuos yra kreipiamasi. Tik baziniais kintamaisiais galima siekti lankstumo. Tačiau, baziniais kintamaisiais galima parametrizuoti tik SQL konstantas. DB objektų (pvz., lentelės ir stulpelių vardų) negalima parametrizuoti. Statiniais SQL sakiniais sprendžiami uždaviniai, kai programos sudarymo metu yra žinomi visi SQL sakiniai. Jei konkretūs kreipimosi į DB sakiniai paaiškėja tik programos vykdymo metu, tai tokių uždavinių statiniais sakiniais nepavyksta išspręsti. Tokiems uždaviniams spręsti naudojami dinaminiai SQL sakiniai.

Nors dinaminiai SQL sakiniai sistemoje DB2 buvo nuo pat jos egzistavimo pradžios, standarte SQL1 šių sakinių nėra. Panašius SQL sakinius įdiegus daugelyje komercinių DBVS, sistemos DB2 dinaminiai SQL sakiniai tapo primtu standartu.

Pagrindinė dinaminio SQL idėja yra labai paprasta: kreipimosi į duomenis SQL sakiniai nėra užrašomi programoje, jie sudaromi programos vykdymo metu. Vykdant programą, sudarytas SQL sakinys perduodamas DBVS, kad ši jį išanalizuotų ir įvykdytų.

Prisiminkime, statinių SQL sakinių vykdymo etapus:



8.3 pav. Statinių ir dinaminių SQL sakinių vykdymo etapai

Statinių SQL sakinių vykdymo planai yra paruošiami prieš programos vykdymą: vien SQL preprocesoriaus arba kartu su SQL surišėju (procedūra *BIND*). Vykdant programą yra

realizuojamas iš anksto paruoštas planas. Dinaminis SQL sakinys vartojamas kitomis aplinkybėmis: sakinys "atsiranda" simbolių eilutės pavidalu ir turi pereiti visus SQL sakinio paruošimo etapus vykdant programą.

Kadangi visi dinaminio sakinio vykdymo etapai yra atliekami vykdant programą, tai dinaminis sakinys yra mažiau efektyvus negu statinis. Vykdant programą, vienas ir tas pats SQL sakinys gali būti vykdomas vieną ar keletą kartų. Siekiant efektyvumo, dinamiškai suformuotas SQL sakinys galima vykdyti dvejopai:

- suformuotas SQL sakinys SQL sakiniu `EXECUTE IMMEDIATE` yra kompiliuojamas, sudaromas jo vykdymo planas ir betarpiškai įvykdomas;
- suformuotas SQL sakinys SQL sakiniu `PREPARE` yra kompiliuojamas, paruošiamas jo vykdymo planas ir jam suteikiamas vardas. Vėliau šį sakinį programoje galima daug kartų vykdyti SQL sakiniu `EXECUTE`. Sakinys išlieka paruoštu iki tranzakcijos pabaigos.

Paprastesnis yra pirmasis būdas. Jau pateikėme programą C kalba vykdytojui pašalinti statiniu SQL sakiniu. Sudarykime programą tam pačiam veiksmui atlikti dinaminiu SQL sakiniu:

```
EXEC SQL BEGIN DECLARE SECTION;
  char sqlStmt[256] ;
EXEC SQL END DECLARE SECTION;
long nr ;
printf("Įveskite šalinamo vykdytojo Nr: ");
scanf("%ld", &nr);
sprintf( sqlStmt,  "DELETE FROM Vykdytojai WHERE Nr = %ld", nr ) ;
EXEC SQL EXECUTE IMMEDIATE FROM :sqlStmt ;
```

Sakiniu `EXECUTE IMMEDIATE` galima įvykdyti daugelį SQL manipuliavimo duomenimis (DML) sakinių, tarp jų `INSERT, UPDATE, DELETE, COMMIT, ROLLBACK`. Daugumą duomenų apibrėžimo sakinių (DDL), pavyzdžiui, `CREATE` ir `DROP` taip pat galima įvykdyti su `EXECUTE IMMEDIATE`. Pateiksime programą bet kuriam iš šių SQL sakinių įvykdyti:

```
printf("Įveskite DML sakinį:");
gets(sqlStmt);
EXEC SQL EXECUTE IMMEDIATE FROM :sqlStmt ;
if(SQLCODE < 0)
  printf("SQL klaida: %ld\n", SQLCODE);
else
  printf("Sakinys įvykdytas sėkmingai\n");
```

Tačiau sakiniu `EXECUTE IMMEDIATE` negalima apdoroti užklausų (`SELECT` sakinio). Kitame skyrelyje aptarsime, kaip vykdyti dinamiškai sudarytą `SELECT` sakinį.

8.9. Dviejų etapų dinaminis SQL sakinių vykdymas

Kai kurie dinamiškai sudaryti SQL sakiniai programoje gali būti vykdomi daug kartų. Norint efektyviai naudoti sistemos resursus, sudarytą SQL sakinį (SQL simbolių eilutę) galima vieną kartą paruošti (sukompiluoti simbolių eilutę ir sudaryti sakinio vykdymo planą), o vėliau daug kartų vykdyti jį, nekartojant paruošiamųjų etapų. Tai realizuoja dvietapis dinaminių SQL sakinių vykdymas:

- Sudaroma SQL sakinio simbolių eilutė - kaip ir `EXECUTE IMMEDIATE` atveju. Konstantas sakinyje galima pakeisti **parametro žymeniu** – klaustuku.
- Sakiniu `PREPARE` yra analizuojama SQL simbolių eilutės sintaksė ir semantika, parenkamas ir sudaromas optimalus sakinio vykdymo planas, t.y SQL sakinys yra paruošiamas vykdymui. Paruoštam sakiniui suteikiamas vardas.
- Sakiniu `EXECUTE` paruoštas sakinys yra vykdomas reikiamą kiekį kartų. Kiekvieną kartą nurodoma konkrečios parametrų, atitinkančių parametrų žymenis, reikšmės.

Sudarykime C kalbos sakinius, projektų vykdymo trukmėms atnaujinti:

```
EXEC SQL BEGIN DECLARE SECTION;
   char sqlStmt[256] ;
   short newValue, searchValue ;
EXEC SQL END DECLARE SECTION;
char yes_no[2] ;
strcpy( sqlStmt, "UPDATE Projektai SET Trukmė = ? WHERE Nr = ?") ;
EXEC SQL PREPARE updateStmt FROM :sqlStmt ;
while(1) {
   printf("Įveskite projekto Nr.: ");
   scanf("%d", &searchValue) ;
   printf("Įveskite projekto Nr. %d naują trukmę: ", searchValue);
   scanf("%d", &newValue) ;
   EXEC SQL EXECUTE updateStmt USING :newValue, :searchValue ;
   if(SQLCODE < 0) {
      printf("SQL klaida: %ld\n", SQLCODE);
      break ;
   }
   printf("Ar tęsti (Y/N)?");
   gets(yes_no) ;
   if('N'== yes_no[0])
      break ;
}
```

Šiame pavyzdyje, paruoštam duomenų atnaujinimo sakiniui `UPDATE` yra suteiktas vardas *updateStmt*, kuris nėra programos kintamasis – jo nereikia papildomai aprašyti. Šis vardas reikalingas tik paruoštam sakiniui nurodyti, kai jis vykdomas sakiniu `EXECUTE`. Ši informacija reikalinga tik SQL preprocesoriui. Atitinkamybė tarp sakinio `EXECUTE` frazėje `USING` nurodytų parametrų ir jų žymenų (klaustukų) SQL simbolių eilutėje nustatoma pagal taisyklę: iš kairės į dešinę.

Parametrų žymenis galima vartoti ir paruošiant `SELECT` sakinį vykdymui:

```
EXEC SQL BEGIN DECLARE SECTION;
  char sqlStmt[256] ;
  char buffer[80] ;
  char name[31] ;
EXEC SQL END DECLARE SECTION;
strcpy( sqlStmt, "SELECT Pavardė FROM Vykdytojai ") ;
strcat( sqlStmt, "WHERE Kvalifikacija = ?") ;
EXEC SQL PREPARE s1 FROM :sqlStmt ;
EXEC SQL DECLARE c1 CURSOR FOR s1 ;
do {
  printf( "Įveskite kvalifikaciją: " );
  gets(buffer) ;
  if( 0 == buffer[0]) break ;
  EXEC SQL OPEN c1 USING :buffer ;
  do{
    EXEC SQL FETCH c1 INTO :name ;
    if(SQLCODE != 0) break;
    printf( "Pavardė: %s\n", name );
  } while(1);
  EXEC SQL CLOSE c1;
} while(1);
```

Šiame pavyzdyje, SELECT sakinys yra paruošiamas dinamiškai, tačiau užklausos rezultato stulpeliai yra žinomi iš anksto, programos sudarymo metu. Sakinyje FETCH yra nurodytas tik vienas char[] tipo bazinis kintamasis.

Jei, pavyzdžiui, norime sudaryti programą, kuri išvestų vartotojo nurodytos lentelės visų ar tik vartotojo nurodytų stulpelių reikšmes, tai mes negalime aprašyti bazinių kintamųjų, nes nežinome ne tik jų tipų, bet ir kiekio. Todėl negalime parašyti FETCH sakinį. Tokiomis aplinkybėmis SQL sakiniai PREPARE ir FETCH vartojami kitaip. Juose reikia naudoti SQL apibrėžimo sritį SQLDA (angl. SQL *Description Area*). Tuomet nuskaitomiems iš DB duomenims yra galima išskirti atmintį dinamiškai. *SQLDA* – tai kintamo ilgio duomenų struktūra, susidedanti iš kelių dalių: pastovios, kuri yra stuktūros pradžioje, ir kintamos dalies – struktūrų SQLVAR masyvo. Kintamoje dalyje kiekvienam SQL sakinio parametrui (stulpeliui) atitinka viena SQLVAR tipo struktūra. Masyvo elementų kiekis yra įsimenamas pastovioje dalyje. Struktūroje SQLVAR yra laukai, atitinkantys stulpelio duomenų tipą ir ilgį, taip pat laukas – kintamasis indikatorius, bei nuoroda į reikiamo ilgio duomenų sritį stulpelio reikšmei patalpinti. Struktūra SQLVAR yra išsamus stulpelio aprašas (deskriptorius).

Vykdant programą, SQL sakiniu DESCRIBE galima sužinoti paruošto SQL sakinio parametrų (stulpelių) kiekį. Žinant stulpelių kiekį, programavimo kalbos priemonėmis (pvz., funkcija malloc), galima dinamiškai išskirti atmintį SQLVAR struktūrų masyvui. Paruošus atminties sritį stulpelių aprašams, tuo pačiu sakiniu DESCRIBE galima dar kartą kreiptis į DBVS, kad ši užpildytų stulpelių aprašų sritį (pateiktų stulpelių vardus, jų tipus, ilgius ir pan.). Žinant šią informaciją, vėl programavimo kalbos priemonėmis galima išskirti atmintį dabar jau užklausos rezultato eilutės visų stulpelių reikšmėms. Dinamiškai suformuotą atminties sritį galima panaudoti nuskaitomiems iš DB duomenims.

Apibrėžimo srities ir sakinio DESCRIBE vartojimas yra gana sudėtingas, nes reikalauja gana aukštos programuotojo kvalifikacijos. Kita vertus, uždaviniai, kuriuose iš anksto nėra žinomos įeities ir išeities duomenų savybės, yra gana retai sutinkami. Be to, apibrėžimo srities vartojimas priklauso nuo konkrečios DBVS. Todėl, mes nepateiksime detalios programavimo technikos tokiems uždaviniams spręsti, apsiribosime tik programavimo veiksmų schema (etapais):

```
EXEC SQL INCLUDE SQLDA ;          /*- įtraukti apibrėžimo sritį */
EXEC SQL BEGIN DECLARE SECTION;
  char sqlStmt[32000] ;    /* - masymas ilgam SELECT sakiniui */
EXEC SQL END DECLARE SECTION;
struct sqlda sqlda ;        /*- apibrėžimo srities kintamasis */
/* SELECT sakinys yra sudaromas ir priskiriamas simbolių masyvui sqlStmt. */
EXEC SQL PREPARE stmt FROM :sqlStmt ;
EXEC SQL DECLARE curs CURSOR FOR stmt ;
memset( &sqlda, 0, sizeof(sqlda) ) ;        /* - "išvalyti" apibrėžimo sritį */
/* Prašome sistemos užpildyti stulpelių kiekį: */
EXEC SQL DESCRIBE stmt INTO :sqlda ;
/* Išskiriame atmintį visų stulpelių aprašams - SQLVAR masyvui (praplečiame sqlda). */
/* Prašome sistemos detalios informacijos apie kiekvieną stulpelį: */
EXEC SQL DESCRIBE stmt INTO :sqlda ;
/* Išskiriame atmintį kiekvieno stulpelio reikšmei. */
/* Nuorodą į reikšmės sritį patalpiname SQLVAR struktūros atitinkamame lauke. */
EXEC SQL OPEN curs;     /*-atidaryti užklausos rezultato žymenį */
EXEC SQL FETCH curs USING DESCRIPTOR :sqlda;  /* - skaityti eilutės reikšmes */
/* Perskaitytų duomenų apdorojimas ir kitų eilučių skaitymas. */
/* Visos anksčiau dinamiškai išskirtos atminties srities atlaisvinimas. */
```

Šiame pavyzdyje aprašytas programos kintamasis *sqlda* nors ir yra bazinis kintamasis, tačiau jo nereikia aprašyti bazinių kintamųjų aprašo sekcijoje.

9. Sisteminiai duomenų bazių aspektai

9.1. Duomenų saugumas ir kreipimosi ribojimas

Duomenų saugumas duomenų bazių kontekste reiškia apsaugą nuo neleistino duomenų ieškojimo, keitimo bei šalinimo. Duomenų apsaugai reliacinėje DBVS keliami tokie reikalavimai:

- lentelės duomenys gali būti prieinami vieniems vartotojams, bet neprieinami kitiems;
- vieniems lentelės vartotojams galima susipažinti su duomenimis ir juos keisti, o kitiems - tik susipažinti;
- daliai vartotojų gali būti prieinamos ne visų lentelės stulpelių reikšmės;
- daliai vartotojų lentelės duomenis leidžiama pasiekti tik naudojantis taikomosiomis programomis, bet ne interaktyviai.

Už duomenų apsaugą atsako DBVS. Duomenų apsaugos uždaviniui spręsti RDBVS paprastai yra dvi galimybės:

- virtualios lentelės, kuriomis galima "paslėpti" tam tikras lentelės eilutes ir stulpelius;
- kreipimosi į duomenis ribojimo posistemis, kuris leidžia vartotojams, turintiems atitinkamą teisę, apibrėžti konkrečias kitų vartotojų teises.

Virtualios lentelės "paslepia" duomenis tik sąlygiškai. Taip "paslėpus" duomenis vartotojas nepraranda galimybės sužinoti pradinių lentelių vardus ir pasiekti jose esančius duomenis tiesiogiai, apeinant virtualią lentelę. Kreipimosi draudimą realizuoja specialus DBVS teisių valdymo posistemis. Reliacinėse DBVS vartotojų teisės duomenų atžvilgiu yra nustatomos SQL sakiniais. Duomenų apsaugos valdymą nusako trys pagrindinės sąvokos:

- **DB vartotojai**. Visi veiksmai su duomenimis duomenų bazėje yra atliekami tam tikro vartotojo vardu. DBVS atsisako įvykdyti operaciją, jei vartotojas neturi tam teisės.
- **DB objektai** - tai visi DB elementai, kurių atžvilgiu yra nustatomos teisės (privilegijos). Paprastai teisės yra nustatomos ne tik lentelių ir virtualių lentelių atžvilgiu, bet ir visos DB bei taikomųjų programų atžvilgiu.
- **Privilegijos** – tai vartotojų teisės atlikti vienokius ar kitokius veiksmus su DB objektais. Su kiekvienu vartotoju kiekvieno DB objekto atžvilgiu yra susiejama tam tikra aibė veiksmų, kuriuos vartotojas gali atlikti.

Kiekvienam DB vartotojui yra suteikiamas tapatumo požymis – trumpas vardas, vienareikšmiškai nusakantis vartotoją. DB vartotojo vardas nebūtinai reiškia vieną asmenį. Praktikoje vienas ir tas pats vardas gali žymėti grupę asmenų, pavyzdžiui, su vardu *Informatikas* gali būti susiejami visi darbuotojai, turintys informatiko kvalifikaciją. Todėl, kai kuriose DBVS vietoj vartotojo tapatumo požymio (angl. *user-id*) vartojamas tikslesnis teisių tapatumo požymis (angl. *authorization-id*).

SQL standartas apibrėžia, kad duomenų apsauga realizuojama per vartotojo tapatumo požymį, tačiau neapibrėžia, nei kaip vartotojai užregistruojami, nei kaip tapatumo požymis susiejamas su SQL sakiniais. Daugumoje operacijų sistemų yra savi apsaugos posistemiai, kurie ne tik užregistruoja vartotojus, bet ir stebi, kad sistemą vartotų tik turintys tam teisę vartotojai. Daugeliu atvejų visiškai pakanka, kad DBVS nedubliuotų šių funkcijų ir tik suteiktų tam tikras teises DB objektų atžvilgiu. Operacijų sistema atpažįsta vartotoją (pagal įvestą tapatumo požymį bei, kaip įprasta, slaptažodį) ir perduoda jo vardą DBVS. Kai vartotojas bando atlikti kokią nors operaciją su DB, DBVS tikrina ar šis, einamasis vartotojas, turi tam teisę.

Informacija apie vartotojų teises DB valdymo sistemai yra pranešama SQL sakiniais `GRANT` ir `REVOKE`. Sakiniu `GRANT` yra suteikiamos tam tikros teisės vartotojui ar jų grupei nurodyto DB objekto atžvilgiu, o sakiniu `REVOKE` jos yra atšaukiamos. Šių SQL sakinio bendras pavidalas:

GRANT <privilegijų sąrašas> [ON [<objekto tipas>] <objektų sąrašas>]
TO <vartotojų sąrašas> [WITH GRANT OPTION] ,

REVOKE [GRANT OPTION FOR] <privilegijų sąrašas>
[ON [<objekto tipas>] <objektų sąrašas>] FROM <vartotojų sąrašas>
[CASCADE | RESTRICT],

kur <privilegijų sąrašas> - viena ar kelios privilegijos, atskirtos kableliais, arba bazinis žodis ALL PRIVILEGES, reiškiantis visas įmanomas privilegijas; <objekto tipas> - bazinis žodis, nurodantis vieną iš DB objektų tipų (duomenų bazė, lentelė ir pan.); <objektų sąrašas> - DB objektų, kurių atžvilgiu yra suteikiamos teisės, vardai; <vartotojų sąrašas> - vartotojų tapatumo požymių sąrašas arba bazinis žodis PUBLIC, reiškiantis visus vartotojus. Jei teisės yra suteikiamos lentelei (lentelėms), tai objekto tipą (TABLE) galima nenurodyti.

Apžvelgsime pagrindines privilegijų grupes.

Privilegijos lentelėms. SQL1 standarte lentelių ir jų vaizdų (virtualių lentelių) atžvilgiu yra numatytos keturios privilegijos:

- SELECT – leidžiama susipažinti su nurodytos lentelės duomenimis. Vartotojui, turinčiam šią privilegiją (virtualiai) lentelei, leidžiama vykdyti užklausas, kurių FROM frazėje yra nurodyta ta lentelė;
- INSERT – leidžiama įvesti į (virtualią) lentelę naujus duomenis;
- DELETE – leidžiama šalinti duomenis iš (virtualios) lentelės;
- UPDATE – leidžiama keisti (virtualios) lentelės duomenis. Ši teisė gali būti apribota, leidžiant keisti tik kai kurių lentelės stulpelių reikšmes.

Pavyzdžiui,

GRANT SELECT ON TABLE *Vykdytojai* TO *Jonas* ,
GRANT INSERT ON TABLE *Vykdymas* TO PUBLIC,
GRANT SELECT, UPDATE, DELETE ON TABLE *Vykdytojai* TO *Jonas*,
GRANT ALL PRIVILEGES ON *Vykdytojai, Projektai, Vykdymas* TO *Informatikas,*
GRANT UPDATE (*Valandos*) ON TABLE *Vykdymas* TO PUBLIC,
REVOKE ALL PRIVILEGES ON TABLE *Vykdymas* FROM PUBLIC.

Šiuolaikinėse komercinėse DBVS šis privilegijų sąrašas yra iš esmės papildytas. Paminėsime tik keletą papildomų privilegijų, vartojamų DBVS DB2:

- REFERENCES – leidžiama kurti lentelėms išorinius raktus, imant pagrindu nurodytą sakinyje lentelę;
- ALTER – leidžiama keisti lentelės struktūrą, pavyzdžiui, papildyti ją naujais stulpeliais.

SQL standarte nėra numatytas privilegijų dalijimas atskiroms lentelės eilutėms. Jei to reikia, tai problema nesunkiai sprendžiama naudojantis virtualia lentele. Tarkime, įstaigoje, atitinkančioje DB *Darbai*, eiliniai darbuotojai gali peržiūrėti duomenis tik apie save. Jeigu Jonaičio DB vartotojo vardas yra *Jonaitis*, tai uždavinį jo atžvilgiu galima išspręsti tokiais SQL sakiniais:

CREATE VIEW *Vykdytojas_Jonaitis*
AS SELECT * FROM *Vykdytojai* WHERE *Pavardė* = ‘Jonaitis’,
CREATE VIEW *Vykdo_Jonaitis* AS SELECT * FROM *Vykdymas*
WHERE *Vykdytojas* = (SELECT *Nr* FROM *Vykdytojai* WHERE *Pavardė* = ‘Jonaitis’),
REVOKE ALL PRIVILEGES ON *Vykdytojai, Vykdymas* FROM *Jonaitis*,
GRANT SELECT ON *Jonaitis, Vykdo_Jonaitis* TO *Jonaitis*.

Privilegijos duomenų bazėms. Šiai grupei priklausančių privilegijų sąrašas nėra standartizuotas. Paminėsime tik keletą DBVS DB2 vartojamų privilegijų:

- `CONNECT` – teisė dirbti su DB, tiksliau vykdyti sakinį `CONNECT`;
- `CREATETAB` – leidžiama kurti naujas lenteles;
- `BINDADD` – teisė kurti jungtinius vykdymo planus (vykdyti utilitą `BIND`);
- `DBADM` – DB administratoriaus privilegija, suteikianti teisę atlikti bet kokius veiksmus su DB.

Pavyzdžiui,

```
GRANT CONNECT ON DATABASE Darbai TO PUBLIC,
GRANT DBADM ON DATABASE Darbai TO Informatikas.
```

Privilegijos vykdymo planams. Daugelyje DBVS yra numatyta privilegijos vykdomiesiems planams, kuriuos sukuria SQL preprocesorius tiesiogiai arba kartu su DBVS procedūra `BIND`. DBVS DB2 yra keletas šios grupės privilegijų, tarp kurių dvi pagrindinės yra:

- `BIND` – teisė sukurti planą. Vartotojas, turintis tokią privilegiją, gali vykdyti procedūrą `BIND`;
- `EXECUTE` – teisė vykdyti planą. Vykdant vartotojui SQL sakinį iš programos, teisės įvykdyti operaciją DB objekto atžvilgiu nepakanka - jis dar turi turėti teisę atlikti tai per vykdymo planą.

Pavyzdžiui,

```
GRANT BIND ON PACKAGE Prog1 TO Informatikas,
GRANT EXECUTE ON PACKAGE Prog1 TO PUBLIC.
```

Vartotojas, sukūręs DB objektą, pvz., lentelę, tampa to objekto savininku. Vartotojas, sukūręs objektą, savaime įgyja visas privilegijas jo atžvilgiu ir gali skirstyti jas kitiems vartotojams. Suteikus vartotojui tam tikrą teisę, šis netampa pilnateisu objekto šeimininku – jis negali skirstyti privilegijų to objekto atžvilgiu kitiems vartotojmas. Kartais toks apribojimas nėra patogus. Jei vartotojui suteikiama privilegija su parinktimi `WITH GRANT OPTION`, šis savo nuožiūra galės "perleisti" jam suteiktą privilegiją kitiems vartotojams. Pastebėjus, kad vartotojas "nepateisina lūkesčių", "švaistydamasis" privilegijomis, sakiniu `REVOKE` su fraze `GRANT OPTION FOR` galima atimti iš jo teisę skirstyti privilegijas.

Laikantis standarto SQL2, atimant iš vartotojo privilegijas, reikia nurodyti vieną iš pasirinkčių: `CASCADE` ar `RESTRICT.` Jei privilegija atimama su parinktimi `CASCADE`, tai ji atimama ir iš visų kitų vartotojų, kurie šią privilegiją gavo iš sakinyje `REVOKE` nurodyto vartotojo. Jei privilegijos atėmimo sakinyje `REVOKE` nurodyta `RESTRICT`, tai privilegijos nepavyks atimti, jei sakinyje nurodytas vartotojas jau yra suteikęs privilegiją kam nors kitam.

Suprantama, vartotojui neleidžiama suteikti nei sau, nei kitiems vartotojams privilegijų, aukštesnių negu jam pačiam buvo suteikta.

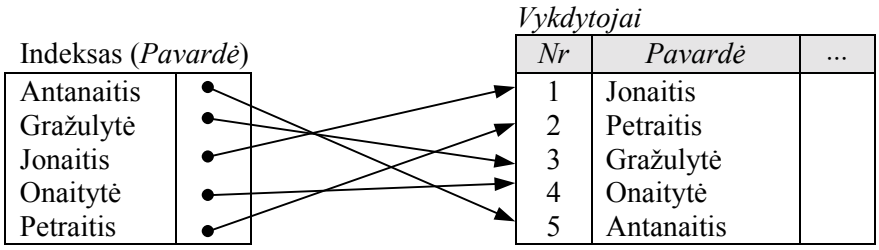
9.2. Indeksai

Lentelės duomenų **indeksas** reliacinėse DBVS yra duomenų paieškos palengvinimo priemonė. Panašiai kaip knygos dalykinė rodyklė palengvina reikiamų puslapių paiešką knygoje, indeksavimas pagreitina informacijos išrinkimą iš duomenų bazės. Duomenų bazėje indeksas atlieka loginių nuorodų į fizinę duomenų vietą vaidmenį.

Tarp indekso ir knygos dalykinės rodyklės yra ir esminių skirtumų. Skaitytojas pats nusprendžia, ar naudotis dalykine rodykle reikiamai informacijai rasti. DB vartotojas tik sprendžia, ar jam sukurti indeksą. Ieškant duomenų, DBVS sprendžia pati, ar naudoti paieškai indeksą. Vartotojas neturi jokių galimybių įtakoti DBVS sprendimo – tai jau yra DBVS dispozicijoje. Indeksas skiriasi nuo dalykinės rodyklės ir tuo, kad DB lentelė gana dažnai turi keletą indeksų, o knygoje paprastai tėra viena dalykinė rodyklė. Panašiai kaip knygoje gali nebūti dalykinės rodyklės, DB lentelė gali neturėti nė vieno indekso.

Indeksas - tai surūšiuota reikšmių ir nuorodų į reikšmes atitinkamas lentelės eilutes aibė. Indeksas apibrėžiamas vienam ar keletui lentelės stulpelių. Indeksas nėra lentelės duomenų dalis - tai atskiras DB objektas. Sukūrus indeksą, DBVS automatiškai sukuria atitinkamą duomenų struktūrą ir stebi, kad indeksas, besikeičiant lentelės duomenims, nuolat atspindėtų jos duomenis ir kad išliktų išrūšiuotas pagal apibrėžtų stulpelių reikšmes.

Indeksai yra labai patogūs todėl, kad duomenų paieška išrūšiuotame masyve yra daug lengvesnė negu neišrūšiuotame. Bet nereikia manyti, kad pakanka saugoti lentelėje išrūšiuotus duomenis. Lentelės duomenys gali būti išrūšiuoti tik pagal vieną kriterijų (tam tikrų stulpelių reikšmes). Jei duomenų masyvas išrūšiuotas pagal vieną kriterijų, tai jis dažnai nėra išrūšiuotas pagal kitą. Kadangi, duomenų paieška lentelėje gali vykti pagal įvairius kriterijus, tai, nors lentelė ir išrūšiuota pagal kurį nors kriterijų, efektyviai paieškai pagal kitus kriterijus reikia papildomų indeksų. Tarkime, lentelė *Vykdytojai* yra išrūšiuota pagal stulpelio *Nr* reikšmes. Sukurkime indeksą dar ir stulpelio *Pavardė* reikšmėms:



DBVS, vykdydama užklausą lentelei *Vykdytojai* pagal nurodytą *Nr*, gali pasinaudoti lentelės savybe, kad jos eilutės yra išrūšiuotos pagal šio stulpelio reikšmes, o vykdant užklausą pagal pavardę gali naudoti indeksą.

Kadangi duomenų indeksas yra fizinė sąvoka, o ne loginė, tai SQL standarte nėra galimybės kurti indeksus. Tačiau daugumoje komercinių DBVS tokia galimybė yra. Kai kuriose iš jų indekso galimybės yra ypač išplėtotos.

Indeksas dar reikalingas ir dėl loginės lentelės rakto vartojimo. Prisiminkime, kad lentelės kiekvienos eilutės rakto reikšmių rinkinys turi būti unikalus lentelėje. Vadinasi, kiekvieną kartą, kai į lentelę įterpiama nauja eilutė, DBVS turi tikrinti ar nebus pažeistas rakto vientisumas. O tai reiškia, kad DBVS turi patikrinti, ar tuo metu lentelėje dar nėra eilutės su nurodyta `INSERT` sakinyje rakto reikšme. Šią procedūrą nesunku realizuoti tik turint išrūšiuotą rakto reikšmių masyvą. Jei lentelė turi tik vieną raktą (pirminį), tai DBVS gali laikyti lentelę išrūšiuotą pagal to rakto reikšmes. Tačiau jei lentelė turi keletą raktų, tai reikalingas papildomas indeksas.

Indeksai yra naudojami:

- siekiant padidinti duomenų paieškos efektyvumą, kadangi paieška išrūšiuotame duomenų masyve yra kur kas lengvesnė negu neišrūšiuotame;
- siekiant užtikrinti atitinkamų stulpelių reikšmių unikalumą lentelėje, kadangi tik išrūšiuotame duomenų masyve galima greitai patikrinti, ar atnaujinant duomenis nebus pažeistas reikšmių unikalumas.

Daugelyje DBVS yra sakinys

`CREATE` [`UNIQUE`] `INDEX` <indekso vardas> `ON` <lentelės vardas>(<stulpelių vardai>).

Sukurkime, pavyzdžiui, keletą indeksų lentelei *Vykdytojai*:

`CREATE UNIQUE INDEX` *IndeksasPavardei* `ON` *Vykdytojai*(*Pavardė*) ,
`CREATE INDEX` *IndeksasKvalifikacijai* `ON` *Vykdytojai*(*Kvalifikacija*).

Šiame pavyzdyje *IndeksasPavardei* ir *IndeksasKvalifikacijai* yra indeksų pavadinimai. Bazinio žodžio `UNIQUE` įtraukimas į indekso *IndeksasPavardei* apibrėžimą garantuoja, kad bus užtikrinama stulpelio *Pavardė* reikšmių unikalumas lentelėje *Vykdytojai*. Tokiu būdu užtikrinamas rakto *Pavardė* vientisumas. Priminsime, kad lentelės raktų vientisumo užtikrinimą jau aptarėme ankstesniame skyriuje.

Daugelis DBVS leidžia apibrėžti **sudėtinius indeksus** – indeksus, kuriuos sudaro keletas lentelės stulpelių. Sudėtiniai indeksai vartojami sudėtinių raktų vientisumui užtikrinti arba kai užklausos yra dažniau vykdomos pagal kelis logiškai susijusius stulpelius, negu pagal kiekvieną iš jų atskirai. Tarus, kad lentelėje *Vykdytojai* yra stulpelis *Vardas*, vietoj indekso *IndeksasPavardei* būtų galima apibrėžti tokį indeksą:

`CREATE UNIQUE INDEX` *IndeksasAsmeniui* `ON` *Vykdytojai*(*Pavardė, Vardas*) .

Indeksas negali būti sukurtas virtualiai lentelei, tačiau vykdant užklausas virtualiai lentelei, DBVS atsižvelgia į bazinių lentelių indeksus.

Užklausose indeksai niekuomet nenurodomi. DBVS pati nusprendžia, ar užklausos vykdymui naudoti indeksus, ir jei taip, tai kokius konkrečiai ir kaip juos panaudoti.

Nors indeksas turi daug gerų savybių, jais negalima piktnaudžiauti, kadangi:

- indeksui reikia kompiuterio atminties (kartai indeksui gali prireikia daugiau vietos atmintyje nei pačios lentelės duomenims);
- indeksui (jo išrūšiavimui) užtikrinti reikalingas procesoriaus laikas. Didelis lentelės indeksų kiekis gali neigiamai atsiliepti lentelės duomenų atnaujinimo laikui.

Sukuriant indeksą nurodomas jo vardas, kuris dar gali būti pavartotas vieninteliu atveju – pašalinant indeksą SQL sakiniu:

`DROP INDEX` <indekso vardas> .

Pašalinus indeksą, jo duomenų struktūra yra sunaikinama ir kompiuterio atmintis yra atlaisvinama.

9.3. Bendro duomenų naudojimo problemos

DB yra bendrai naudojama sistema. Tai reiškia, kad vienu metu gali būti vykdomos kelios transakcijos. Keliems vartotojams (programoms) dirbant tuo pačiu metu, DBVS ne tik turi užtikrinti duomenų atkūrimą, kai atšaukiami transakcijoje padaryti duomenų pakeitimai, bet ir garantuoti, kad vartotojai netrukdytų vienas kitam. Kiekvienas vartotojas turi jaustis taip, tartum jis tik vienas dirbtų su DB. Todėl DBVS derina vartotojų darbą.

Kad suprastume, kaip transakcijos vykdomos kartu ir kokius uždavinius sistemai tenka spręsti, apžvelgsime problemas, kurios kyla, kai transakcijos nederinamos. Tokių problemų yra gana daug, tačiau daugelį jų galima priskirti kuriai nors vienai tipiniai. Aptarsime dvi tipines situacijas, kai dviem transakcijoms dirbant su tais pačiais duomenimis, gali būti gauti neteisingi rezultatai, nors kiekviena iš jų, atskirai paėmus, dirba teisingai.

Vieną iš dviejų kartu veikiančių transakcijų pažymėkime *A*, o kitą - *B*. Bendrai naudojamą konkrečios lentelės eilutę pažymėkime *R*. Fraze `FETCH` *R* žymėsime eilutės *R* nuskaitymą į atmintį, o `UPDATE` *R* – tos eilutės reikšmių atnaujinimą (keitimą). Laiko momentus, kuriais kreipiamasi į DB, žymėsime t_1, t_2, t_3,…. Be to, $\forall$ *i*, *j*, kuriems teisinga nelygybė $i < j$, yra teisinga ir $t_i < t_j$.

Pakeistų duomenų praradimo problema. Būseną, kurioje susidaro ši problema, galima pavaizduoti taip:

Transakcija *A*	Laikas	Transakcija *B*
-	|	-
`FETCH` *R*	t_1	-
-	|	-
-	t_2	`FETCH` *R*
-	|	-
`UPDATE` *R*	t_3	-
-	|	-
-	t_4	`UPDATE` *R*
-	↓	-

Abi transakcijos *A* (laiko momentu t_1) ir *B* (laiko momentu t_2) perskaito eilutės *R* reikšmes, kurias vėliau abi transakcijos pakeičia (atitinkamai, laiko momentais t_3 ir t_4). Laiko momentu

t_4 transakcija A praranda atnaujintus duomenis, kadangi jos pakeistus duomenis pakeičia transakcija B. Transakcija B keičia eilutę R atsižvelgdama tik į jos pačios perskaitytas reikšmes, bet ne į transakcijos A priskirtas naująsias reikšmes. Taip transakcijoje A atliktas duomenų keitimas netenka prasmės.

Ši problema kyla kiekvieną kartą, kai dvi transakcijos nuskaito iš DB tuos pačius duomenis, kuriuos jos abi ir keičia. Nesprendžiant šios problemos, duomenų bazėje atsispindėtų tik paskutinis duomenų keitimas.

Priklausymo nuo neužfiksuoto pakeitimo problema. Ši problema atsiranda, kai nuskaitoma eilutė, kurią anksčiau jau keitė kita transakcija, tačiau tas pakeitimas dar nebuvo galutinai įtvirtintas. Kadangi ta kita transakcija dar nepatvirtino pakeitimo, tai gali atsitikti taip, kad jis ir nebus patvirtintas, tiksliau, bus panaikintas. Pavaizduokime tokią būseną:

Transakcija A	Laikas	Transakcija B
-	|	-
-	t_1	UPDATE R
-	|	-
FETCH R	t_2	-
-	|	-
-	t_3	ROLLBACK
-	↓	-

Laiko momentu t_2 transakcija A nuskaito duomenis, kurie prieš tai (momentu t_1) buvo pakeisti transakcijos B. Nuo momento t_2 transakcijos A tolimesnio darbo teisingumas priklauso nuo to, kaip pasibaigs transakcija B, patvirtindama pakeitimą ar anuliuodama jį. Jei transakcija B atsisako pakeitimo (t_3), tai nuo to momento transakcija A naudoja neteisingus duomenis.

Dar blogiau kai momentu t_2 transakcija A ne skaito duomenis, o keičia juos:

Transakcija A	Laikas	Transakcija B
-	|	-
-	t_1	UPDATE R
-	|	-
UPDATE R	t_2	-
-	|	-
-	t_3	ROLLBACK
-	↓	-

Tokiu atveju, nuo laiko momento t_2 transakcijos A teisingumas priklauso nuo neužfiksuoto pakeitimo, o laiko mometu t_3 ji praranda dar ir savo pakeitimą, nes tuomet transakcija B atkuria eilutės R reikšmes, kurios buvo transakcijos B pradžioje (iki momento t_1).

9.4. Duomenų atribojimas

Bendro duomenų naudojimo problemos sprendžiamos duomenų atribojimu (užrakinimu) (angl. *locking*). Pagrindinė duomenų atribojimo idėja yra labai paprasta - jei transakcijai reikia, kad DB objektas (pvz., lentelės eilutė ar kelios eilutės) nebūtų pakeistas tam tikrą laikotarpį, tai ji užrakina tą objektą. Užrakintas objektas yra atskiriamas nuo kitų transakcijų įtakos, t.y. jis tampa neprieinamas kitoms transakcijoms. Transakcija yra priversta laukti, jei ji nori pasiekti (nuskaityti ar modifikuoti) objektą, kurį užrakino kita transakcija. Todėl, netgi paprasčiausio SQL sakinio, kuriuo norima perskaityti vieną lentelės eilutę, vykdymas gali užtrukti ilgai, jei tą eilutę yra užrakinusi kita transakcija. Kad nesukelti didelių nesklandumų, duomenys užrakinėjami gana lanksčiai.

Komercinėse DBVS paprastai naudojami du duomenų atribojimo užraktai:

- X (angl. *eXclusive lock*) – visiškas užrakinimas;
- S (angl. *Shared lock*) – dalinis užrakinimas.

Kartais X ir S duomenų atribojimo užraktai vadinami atitinkamai **rašymo ir skaitymo užraktais**. Kai kuriose komercinėse DBVS yra daugiau duomenų atribojimo užraktų, tačiau pagrindas yra toks pat.

Duomenų atribojimą pasiaiškinsime lentelės eilutės atveju, nors jis taikomas ir kitiems DB objektams, pvz., užklausos rezultatui bei lentelei. Duomenų atribojimą galima nusakyti tokiomis taisyklėmis:

- Jei transakcija *A* visiškai užrakina (užraktu X) eilutę *R*, tai transakcija *B*, norinti perskaityti ar pakeisti tą eilutę, yra priversta laukti kol transakcija *A* atrakins tą eilutę.
- Jei transakcija *A* iš dalies užrakina (užraktu S) eilutę *R*, tai:
  - jei transakcija *B* nori visiškai užrakinti (užraktu X) tą pačią eilutę *R*, tai *B* priverčiama lauki, kol *A* neatrakins eilutę *R*;
  - jei transakcija *B* nori tik iš dalies užrakinti (užraktu S) eilutę *R*, tai šis noras išpildomas, t.y. ir transakcijai *B* leidžiama užrakinti eilutę *R* užraktu S.

Šias taisykles galima pavaizduoti užraktų suderinamumo matrica:

	X	S	-
X	Ne	Ne	Taip
S	Ne	Taip	Taip
-	Taip	Taip	Taip

Pirmajame iš kairės stulpelyje yra surašyti visi įmanomi transakcijos *A* duomenų atribojimo užraktai. Minusu pažymėtas atribojimo nebuvimas. Viršutinėje eilutėje yra išvardyti duomenų atribojimo užraktai, kuriais kita transakcija *B* nori užrakinti tuos pačius duomenis kaip ir transakcija *A*. Vidiniuose matricos langeliuose pažymėta, ar transakcijos *B* prašymai yra patenkinami (Taip), ar nepatenkinami (Ne). Nesunku suprasti, kodėl ši matrica yra simetriška.

Prašymai užrakinti lentelės eilutę paprastai yra netiesioginiai. Kai transakcija nori perskaityti eilutę (pvz., įvykdyti sakinį FETCH), ji automatiškai "prašo" leidimo užrakinti tą eilutę užraktu S, o kai nori keisti eilutę (įvykdyti sakinį UPDATE) - "prašo" užrakinti eilutę užraktu X. Eilutė yra užrakinama, o SQL sakinys įvykdomas, kai tik transakcija "sulaukia" iš DBVS leidimo.

Transakcija, užrakinusi eilutę užraktu X, paprastai išlaiko šią eilutę užrakintą iki transakcijos pabaigos. Užraktu S užrakinta eilutė gana dažnai atrakinama nepasibaigus transakcijai. Tačiau duomenų atribojimo laikas priklauso ne tik nuo užrakto, bet ir nuo išorinių veiksnių, kuriuos aptarsime vėliau. Bet kuriuo atveju, pačiai transakcijai nereikia rūpintis atrakinimu - tai atliekama automatiškai. Pasibaigus transakcijai visi jos užrakinti objektai atrakinami.

9.5. Bendro duomenų naudojimo problemų sprendimas

Įvedus duomenų atribojimą, prarasto duomenų pakeitimo ir priklausymo nuo neužfiksuoto pakeitimo problemos išnyksta. Dabar **pakeistų duomenų praradimo** padėtyje įvykiai klostysis taip:

Transakcija *A*	Laikas	Transakcija *B*
-		-
FETCH *R* (prašoma ir užrakinama užraktu S)	t_1	-
-		-
-	t_2	FETCH *R* (prašoma ir užrakinama užraktu S)
-		-
UPDATE *R* (prašoma užrakinti užraktu X)	t_3	-
Laukiama		-
Laukiama	t_4	UPDATE *R* (prašoma užrakinti užraktu X)
Laukiama	↓	Laukiama

Atribojant duomenis, vienoje transakcijoje pakeisti duomenys nebus anuliuojami kitoje. Tačiau, spręsdami šią problemą, mes susidūrėme su kita problema - **aklavietė**. Nuo laiko momento t_4, abi transakcijos lauks viena kitos. Kaip sprendžiama aklavietės problema, aptarsime vėliau.

Pažiūrėkime, kaip klostysis įvykiai situacijoje, kuri sukėlė vienos transakcijos **priklausymą nuo neužfiksuoto kitos transakcijos duomenų pakeitimo**. Pirmuoju atveju, kai transakcija *A* perskaito transakcijos *B* pakeistus duomenis, turime:

Transakcija *A*	Laikas	Transakcija *B*
-	|	-
-	t_1	UPDATE *R* (prašoma ir užrakinama užraktu X)
-	|	-
FETCH *R* (prašoma užrakinti užraktu S)	t_2	-
Laukiama	|	-
Laukiama	t_3	ROLLBACK (atrakinama)
Nuskaitoma (užrakinama užraktu S)	t_4	-
-	↓	-

Panašiai kaip ir ankstesnėje situacijoje, momentu t_2 transakcijai *A* neleidžiama perskaityti eilutės *R*. Transakcija *A* yra priversta laukti, kol transakcija *B* atrakins šią eilutę. Momentu t_3 transakcija *B* baigiasi ir transakcija *A* gali pratęsti darbą. Momentu t_4 transakcija *A* perskaito tas eilutės *R* reikšmes, kurios buvo prieš transakcijai *B* keičiant jas (momentu t_1). Abi transakcijos gali tęsti pradėtą darbą.

Antruoju atveju, kai transakcija *A* keičia duomenis, kurie jau buvo pakeisti kitos transakcijos *B*, turėsime labai panašią situaciją:

Transakcija *A*	Laikas	Transakcija *B*
-	|	-
-	t_1	UPDATE *R* (prašoma ir užrakinama užraktu X)
-	|	-
UPDATE *R* (prašoma užrakinti užraktu X)	t_2	-
Laukiama	|	-
Laukiama	t_3	ROLLBACK (atrakinama)
Atnaujinama (užrakinama užraktu X)	t_4	-
-	↓	-

Taigi, duomenų atribojimas, kuris yra gana paprastai tvarkomas, neleidžia susidaryti klaidoms kai tuo pačiu metu su tais pačiais duomenimis dirba kelios transakcijos.

9.6. Aklavietės ir jų likvidavimas

Duomenų atribojimas užtikrina, kad duomenys nebus apdoroti klaidingai dėl to, kad jais tuo pat metu naudojasi kelios transakcijos. Tačiau taip išsprendžiamos ne visos problemos. Duomenų bazės objekto (pvz., lentelės eilutės) užrakinimas gali tapti aklavietės priežastimi. **Aklavietė** - tai būsena, kai dvi ar daugiau transakcijų tuo pačiu metu laukia, kol kita iš jų atrakins duomenis, kad pratęsti darbą.

Ankstesniame skyrelyje mes jau susidūrėme su aklaviete. Būseną, kurioje susidaro aklavietė tarp dviejų transakcijų *A* ir *B*, galima išreikšti taip:

Transakcija *A*	Laikas	Transakcija *B*
-	|	-
R_1 užrakinama S arba X užraktu	t_1	-
-	|	-
-	t_2	R_2 užrakinama S arba X užraktu
-	|	-
Prašoma užrakinti R_2 X užraktu	t_3	-
Laukiama	|	-
Laukiama	t_4	Prašoma užrakinti R_1 X užraktu
Laukiama	↓	Laukiama

Šiame pavyzdyje R_1 ir R_2 žymi du konkrečius DB objektus, nebūtinai eilutes. Tai gali būti ir eilučių grupės ar visos lentelės. Čia į aklavietę patenka dvi transakcijos. Teoriškai į aklavietę tuo pačiu metu gali patekti ir daugiau transakcijų. Tačiau aklavietės, kurios apimtų daugiau negu dvi transakcijas, praktiškai nėra sutinkamos.

Svarbu, kad DBVS aptiktų aklavietes ir padėtų transakcijoms išeiti iš jų. DBVS realizuodama duomenų atribojimą tvarko **transakcijų**, laukiančių kitų transakcijų pabaigos, **žurnalą**. Tam, kad aptikti aklavietę, pakanka rasti ciklą **transakcijų būsenų grafe**, kuriame vaizduojamos transakcijų tarpusavio priklausomybės. Aptikusi aklavietę, DBVS vieną iš transakcijų išrenka **auka** ir, priklausomai nuo konkrečių aplinkybių:

- išrinktą transakciją nutraukia ir atkuria duomenis, buvusius transakcijos pradžioje;
- nutraukia SQL sakinį, kuriam įvykdyti transakcija laukė galimybės užrakinti DB objektą, anksčiau jau užrakintą kitos transakcijos.

Kiekvienu atveju, DBVS praneša vartotojui (programai) apie panaudotas priemones konkrečiu pranešimu apie klaidą. Priklausomai nuo klaidos kodo, vartotojas gali pakartotinai įvykdyti tik paskutinį SQL sakinį arba pakartoti visą transakciją nuo pradžios. Kai kurios komercinės DBVS imasi pačios pakartoti veiksmą.

9.7. Duomenų atribojimo tvarkymas

Jau minėjome, kad paprastai duomenų atribojimas tvarkomas automatiškai. Papildomai daugelyje komercinių DBVS vartotojas pats gali tvarkyti duomenų atribojimą. Dažnai yra dvi duomenų atribojimo tiesioginio tvarkymo priemonės:

- **lentelės užrakinimas**;
- **transakcijos atribojimo lygio** (angl. *isolation level*) priskyrimas transakcijai.

Jei transakcijoje yra daug kartų kreipiamasi į konkrečią lentelę, tai laiko ir kitų resursų sąnaudos, kurios susidaro dėl daugelio nedidelių lentelės sričių užrakinimo ir atrakinimo, gali būti santykinai didelės. Tokiais atvejais yra geriau iš karto užrakinti visą lentelę ir atrakinti ją tik apdorojus visus lentelės duomenis. Visos lentelės užrakinimas, lyginant jį su eilutės užrakinimu, turi keletą privalumų:

- nėra sąnaudų, reikalingų kiekvienai atskirai eilutei užrakinti ir atrakinti;
- nėra pavojaus, kad, pradėjus duomenų apdorojimą, veiksmai užsitęs dėl to, kad dalį eilučių užrakino kita transakcija;
- nėra pavojaus, kad, apdorojus dalį eilučių, transakcija pateks į aklavietę ir reikės vėl pradėti viską iš naujo.

Lentelės užrakinimas turi ir gana didelį trūkumą - visos kitos transakcijos turės laukti lentelės atrakinimo. Todėl, patartina lentelę užrakinti tik programose, nes interaktyvios transakcijos tęsiasi daug ilgiau. Lentelė užrakinama SQL sakiniu:

`LOCK TABLE` <lentelės vardas> `IN` [`SHARE` | `EXCLUSIVE`] `MODE`.

Vykdant šį sakinį su parametru `EXCLUSIVE`, yra prašoma DBVS visiškai užrakinti lentelę, t.y. užrakinti X užraktu. Taip užrakinus lentelę, jokia kita transakcija negalės lentelės duomenų nei perskaityti, nei pakeisti. Visiškas užrakinimas yra naudingas, kai reikia

atnaujinti daugumą ar net visas lentelės eilutes. Parametru SHARE nurodoma užrakinti lentelę S užraktu. Taip atribojus lentelę, kitos transakcijos galės be suvaržymų skaityti lentelės duomenis, bet negalės jų keisti. Šis režimas naudojamas norint sudaryti momentinį lentelės duomenų vaizdą, pavyzdžiui, suformuoti ataskaitą. Įvydžius sakinį

LOCK TABLE *Vykdymas* IN SHARE MODE,

niekas kitas negalės keisti lentelės *Vykdymas* duomenų tol, kol nesibaigs šį sakinį įvykdžiusi transakcija.

9.8. Transakcijų atribojimas

Pagal transakcijos apibrėžimą jokia kita, kartu veikianti transakcija negali daryti įtakos duomenims, su kuriais ši transakcija dirba. Jei transakcijoje yra nuskaitomi bei apdorojami duomenys, ir po to vėl jie nuskaitomi, tai transakcija gali būti tikra, kad perskaityti duomenys bus tapatūs anksčiau perskaitytiems, žinoma, jei tik pati transakcija jų nepakeitė. Pakartotinis nepakeistos eilutės perskaitymas transakcijoje yra pats aukščiausias transakcijos (programos) atribojimo nuo kitų transakcijų lygis.

Visiškas transakcijos atribojimas nuo kitų transakcijų reikalauja daug sąnaudų, nes kiekviena perskaityta eilutė turi būti užrakinta užraktu S. Eilutės išlieka užrakintomis iki transakcijos pabaigos. Dažnai DBVS galėtų duomenų atribojimą (užrakinimą-atrakinimą) tvarkyti daug geriau, jei tik ji žinotų, kas transakcijoje daroma su duomenimis. Todėl su kiekviena transakcija yra susiejamas jos atribojimo lygis. Transakcijos atribojimo lygis išreiškia kompromisą tarp visiško transakcijų atribojimo ir efektyvaus jų darbo.

Transakcijos atribojimo lygis - tai:

- laipsnis, kuriuo lentelių eilutės, perskaitytos ar pakeistos transakcijoje, yra prieinamos kitoms transakcijoms;
- laipsnis, kuriuo kitų transakcijų perskaitytos ar pakeistos eilutės yra prieinamos šioje transakcijoje.

Standarte SQL2 yra apibrėžti keturi transakcijos atribojimo lygiai.

- **Visiško atribojimo** (SERIALIZABLE) lygis. Tai pats griežčiausias transakcijos atribojimo lygis, užtikrinantis, kad:
  - bet kuri transakcijoje perskaityta eilutė nebus pakeista jokios kitos transakcijos kol duotoji transakcija nesibaigs;
  - jokia eilutė, pakeista kitoje transakcijoje nebus perskaityta duotojoje transakcijoje, kol pakeitusi eilutę transakcija nepasibaigs.

  Šis lygis visiškai atriboja transakciją nuo kitų transakcijų poveikio. Jis užtikrina, kad vienoje transakcijoje, pakartotinai perskaitant duomenis (vykdant SELECT sakinį), kiekvieną kartą bus perskaityti tie patys duomenys, jei tik duotoji transakcija pati nekeičia jų. Kitoms transakcijoms nebus leista ne tik keisti ir pašalinti šioje transakcijoje perskaitytas eilutes, bet ir įterpti naujas eilutes, kurios galėtų patekti į perskaitytų duomenų sritį. Šis lygis yra priimamas pagal nutylėjimą, kadangi jis atitinka pagrindinį principą – transakcijos neturi įtakoti viena kitai.
- **Pakartotinio duomenų skaitymo** (REPEATABLE READ) lygis. Šis lygis yra labai panašus į SERIALIZABLE lygį. Transakcijoje nėra prieinami jokie kitų transakcijų neužfiksuoti duomenų pakeitimai. Transakcijai vieną kartą perskaičius duomenis, visos kitos transakcijos negalės jų nei keisti, nei šalinti, kaip ir SERIALIZABLE atveju, bet joms bus leidžiama įterpti naujas eilutes, kurios gali patekti ir į duotosios transakcijos perskaitytų duomenų sritį. Todėl pakartotinai skaitant duomenis toje pačioje transakcijoje gali būti perskaitytos ir papildomos eilutės, įterptos ir užfiksuotos (COMMIT sakiniu) kitų transakcijų. Tokios naujos eilutės, atsirandančios pakartotinai skaitant duomenis vienoje transakcijoje ir įterptos kitos transakcijos, vadinamos “vaiduoklėmis” (angl. *phantom*). Jeigu transakcijoje nenumatyta keletą kartų vykdyti tą pačią užklausą, tai galima vartoti šį lygį nerizikuojant duomenų vientisumu.

- **Užfiksuotų duomenų skaitymo** (`READ COMMITTED`) lygis. Šio lygio transakcijose yra užtikrinama, kad jokia eilutė, pakeista kitoje transakcijoje, nebus prieinama duotojoje transakcijoje, kol pakeitusi eilutę transakcija nepasibaigs. Tačiau užfiksuoti kitų transakcijų rezultatai yra "matomi" šioje transakcijoje. Taip atribotoje transakcijoje, pakartotinai įvykdžius užklausą, gali būti aptinkamos eilutės, kurias pakeitė kitos transakcijos. Tai reiškia, kad transakcijoje perskaitytos eilutės neišlieka užrakintomis iki transakcijos pabaigos. Perskaityta eilutė atrakinama, kai tik užklausos rezultato žymuo pasislenka prie kitos eilutės. Transakcijoje pakeistos ir užraktu `X` užrakintos eilutės išlieka tokiomis iki transakcijos pabaigos. Jei programoje nereikia tas pačias eilutes peržiūrėti keletą kartų ir jei nėra skaičiuojami suminiai rodikliai, tai šio atribojimo lygio pakanka, kad užtikrinti duomenų neprieštaringumą. Šio lygio transakcijos vykdomos greičiau negu aukštesnių atribojimo lygių transakcijos.
- **Neužfiksuotų duomenų skaitymo** (`READ UNCOMMITTED`) lygis. Tai pats žemiausias transakcijos atribojimo lygis. Taip atribotoje transakcijoje perskaitytą lentelės eilutę, kitos transakcijos gali ne tik perskaityti, bet ir keisti. Bet kuri eilutė, pakeista kitoje transakcijoje, gali būti perskaityta šioje transakcijoje, net jei tas pakeitimas neužfiksuotas, t.y. net nepasibaigus eilutę keitusiai transakcijai. Tačiau ir šio lygio transakcijose yra užtikrinama, kad nebus prarandami pakeisti duomenys. Kitose transakcijose pakeistus ir neužfiksuotus duomenis duotojoje transakcijoje galima tik perskaityti, bet ne keisti, kaip ir kitos transakcijos negali keisti šioje transakcijoje pakeistų duomenų kol pakeitimai nebus užfiksuoti. Kadangi šio lygio transakcijoje galima perskaityti neužfiksuotus duomenis, tai jo taikymas yra labai ribotas. Jis yra tinkamas tik tuomet, kai vartotojui yra priimtini ir ne visiškai teisingi duomenys. Jei svarbu tik "pamatyti" duomenis, nesirūpinat jų teisingumu, tai šis atribojimo lygis gali labai pagreitinti rezultato gavimą.

Pateiksime transakcijos atribojimo lygių suvestinę:

Atribojimo lygis	Neužfiksuotų duomenų perskaitymas	Pasikeitimai pakartotiniame perskaityme	Eilutės "vaiduoklės"
`SERIALIZABLE`	Negalimas	Negalimi	Negalimos
`REPEATABLE READ`	Negalimas	Negalimi	Galimos
`READ COMMITTED`	Negalimas	Galimi	Galimos
`READ UNCOMMITTED`	Galimas	Galimi	Galimos

Atribojimo lygį galima nustatyti pačioje transakcijoje SQL sakiniu:

`SET TRANSACTION ISOLATION LEVEL` <atribojimo lygis> [<apdorojimo rūšis>] .

Sakinyje nurodoma vienas iš keturių atribojimo lygių bei duomenų apdorojimo rūšis: duomenų skaitymas (`READ ONLY`) ar ir duomenų keitimas (`READ WRITE`). DBVS naudoja šiuos transakcijos požymius savo veiksmams optimizuoti. Pagal nutylėjimą, yra priimamas `SERIALIZABLE` lygis. Jei nurodytas `READ UNCOMMITTED` lygis, tai, pagal nutylėjimą, priimama `READ ONLY` veiksmų rūšis ir keisti duomenis transakcijoje neleidžiama. Visais kitais atvejais, pagal nutylėjimą, priimamas `READ WRITE` duomenų apdorojimas. Jei šis SQL sakinys yra naudojamas, tai jis turi būti pirmuoju transakcijoje.

Transakcijos atribojimo lygis pradėtas vartoti dar prieš standartizuojant šią sąvoką. Todėl lygių pavadinimuose atsirado painiava. Sistemoje DB2, pavyzdžiui, `REPEATABLE READ` vadinamas lygis, kuris standarte vadinamas `SERIALIZABLE`. Bet to, šioje sistemoje atribojimo lygis yra ne transakcijos požymis, bet transakcijų grupės, tiksliau, jungtinio vykdomojo plano požymis, kuris priskiriamas visoms plano transakcijoms. Atribojimo lygis vykdomajam planui yra priskiriamas jį sukūrus, t.y. SQL preprocesoriui apdorojus vartotojo sudarytą programinį modulį arba procedūra `BIND` pagal ryšio failą. Vėliau atribojimo lygį galima keisti.

Esant keliems transakcijos atribojimo lygiams neišvengiamai kyla klausimas: kaip jį pasirinkti? Sunku pateikti tikslias taisykles, leidžiančias formaliai išspręsti šią problemą.

Atribojimo lygio parinkimas yra gana atsakingas uždavinys. Parinkus per griežtą atribojimo lygį, dėl duomenų užrakinimo gali nepagrįstai sumažėti sistemos efektyvumas bei padidėti tikimybė patekti į aklavietę. Parinkus per žemą transakcijos atribjimo lygį, gali atsirasti šalutiniai poveikai (pvz., perskaitomi neužfiksuoti duomenys), kurie daro įtaką neteisingam programos rezultatui ar net pažeidžia duomenų neprieštaringumą.

Pateiksime paprastas euristikas, galinčias padėti pasirinkti reikiamą izoliavimo lygį:

Duomenų apdorojimo rūšis	Aukštas duomenų stabilumas reikalingas	Aukštas duomenų stabilumas nebūtinas
Skaitymo ir rašymo transakcijos	`REPEATABLE READ`	`READ COMMITTED`
Tik skaitymo transakcijos	`SERIALIZABLE`	`READ UNCOMMITTED`

10. Objektinės technologijos reliacinėse DB

10.1. Įvadas

Reliacinės DB paplitę daug plačiau negu visų kitų rūšių duomenų bazės kartu. Taip yra ir Lietuvoje, ir visame pasaulyje. "Grėsmę" tokiam "viešpatavimui" pastaraisiais metais kelia tik objektinės technologijos (angl. *object oriented technology*). Pavyzdžiui, objektinės programavimo kalbos (C++ ir Java) tapo vienomis populiariausių programavimo kalbų, gal net pačiomis populiariausiomis.

Grupė didžiųjų pasaulio informacinių technologijų firmų 90-jų pradžioje ėmė diegti objektines technologijas duomenų bazių sistemose. Buvo prognozuojama, kad objektinės DBVS (ODBVS) taps tokiomis pat populiariomis, kaip ir objektinės programavimo kalbos. Tačiau prognozės, bent jau iki šiol, nepasitvirtino. Nors ODBVS įdiegimų skaičius pasaulyje santykinai auga greičiau negu RDBVS įdiegimų skaičius, tačiau absoliutūs augimo tempai yra ryškiai RDBVS naudai. Jeigu panaši tendencija išliks ir ateityje, tai RDBVS vyraus dar daugelį metų.

Tai, kad RDBVS ir toliau dominuoja, didelę įtaką padarė pačių RDBVS vystymasis. Komercinių RDBVS kūrėjai pripažįsta objektinių technologijų privalumus ir energingai diegia jas reliacinėse DB valdymo sistemose, taip išplėsdami sistemų galimybes. Dauguma šiuolaikinių RDBVS pradėta vadinti objektinėmis-reliacinėmis DBVS (ORDBVS).

Šiame skyriuje trumpai apžvelgsime objektinių DB privalumus bei trūkumus ir susipažinsime su pagrindinėmis objektinių-reliacinių DBVS savybėmis.

10.2. Objektinės duomenų bazės

Skirtingai nuo reliacinio duomenų modelio, kuris grindžiamas griežtais matematiniais apibrėžimais, objektinės duomenų bazės (ODB) tokio griežto teorinio pagrindo neturi. Kadangi pagrindinės objektinių DB sąvokos yra labai artimos objektinio programavimo sąvokoms, tai jų detaliai nenagrinėsime.

- **Objektai**. Visi ODB duomenys yra vaizduojami objektais. Reliacinėms duomenų bazėms būdingas duomenų tvarkymas lentelių eilutėmis objektinėse DB yra pakeistas objektų tvarkymu.
- **Klasės**. Paprastas reliacinių DB duomenų tipas ODB yra pakeistas hierachine klasės sąvoka. Kiekvienas objektas priklauso konkrečiai objektų klasei.
- **Paveldimumas**. Objektai paveldi savo klasės ir visų aukštesnių klasių savybes.
- **Atributai**. Atributai – tai klasės objektų charakteristikos. Atributo sąvoka yra labai artima stulpelio sąvokai reliacinėse DB.
- **Pranešimai ir metodai**. Objektai "bendrauja" tarpusavyje pranešimais. Objektas, gavęs pranešimą, įvykdo atitinkamą metodą – vidinį paprogramį, kuris realizuoja pranešimo apdorojimą. Objekto elgesys aprašomas metodų aibe.
- **Inkapsuliacija**. Vidinė objekto struktūra ir duomenys yra paslėpti. Informaciją apie objekto būseną galima gauti tik metodais. Objekto būsena keičiama taip pat metodais.
- **Objektų tapatumo požymiai**. Objektai yra atskiriami vienas nuo kito pagal tapatumo požymį (angl. *object ID - OID*). Jie naudojami sąryšiuose tarp objektų. Tapatumo požymį objektui priskiria DBVS. Paprasčiausių objektų, pvz., skaičių, tapatumas nustatomas pagal jo reikšmę. Sudėtingesnių objektų tapatumui nustatyti naudojami loginiai objektų adresai.

Šių principų dėka ODB puikiai tinka uždaviniams, kuriuose apdorojami labai sudėtingos struktūros duomenys, pvz., kompiuterinio modeliavimo ir struktūriškų dokumentų apdorojimo sistemose. Tokioms sistemoms yra būdingos hierarchinės duomenų struktūros, kurios patogiai vaizduojamos objektais ir jų klasėmis.

Pagrindinis ODB šalininkų argumentas yra taikomajai sričiai būdingos duomenų struktūros tiesioginis vaizdavimas objektais tiek duomenų bazėje, tiek ir taikomosiose

programose. Tai labai svarus argumentas prieš gana nelankstų reliacinėse DB vartojamą lentelių modelį. Realaus pasaulio esybės dažnai daug paprasčiau modeliuojamos lanksčia objektų klase, negu lentele. Be to, kai duomenys apdorojami programomis, sudarytomis objektinio programavimo kalba, nėra duomenų struktūrų suderinamumo sunkumų.

Savo ruožtu, ODB oponentai mano, kad nėra būtinybės pereiti nuo RDB prie ODB. Teigiama, pvz., kad objekto tapatumo požymio sąvoka yra artima nuorodos į duomenis sąvokai, kuri buvo būdinga senesniosioms, ikireliacinėms DB. Kadangi ODB neturi griežto matematinio pagrindo, tai yra ir standartizavimo sunkumų. Be to, reliacinio duomenų modelio ir objektinės programavimo kalbos duomenų struktūrų atitikimo sunkumus galima įveikti įvedant naujas objektines-reliacines sąsajas.

Komercinių RDBVS kūrėjai, atsižvelgdami į objektinių technologijų privalumus ir siekdami išlaikyti savo pozicijas rinkoje, įdiegė nemažai sąvokų, kurios anksčiau buvo būdingos tik ODBVS. Taip RDBVS, išsaugodamos reliacinio modelio privalumus, įgavo naujų bruožų, dėl kurių jos pradėtos vadinti objektinėmis-reliacinėmis DBVS (ORDBVS). Vartotojo sąsajos su DB vaidmenį objektinėse-reliacinėse sistemose atlieka SQL kalba, kurios galimybės labai išsiplėtė. Objektines-reliacines DB aptarsime detaliau.

10.3. Objektinės-reliacinės duomenų bazės

Kad pasinaudoti gerosiomis objektinių DB savybėmis, objektinėse-reliacinėse DB įdiegta gana daug papildymų. Tarp jų:

- **Dideli duomenų objektai.** Tradiciniai reliaciniai duomenų tipai yra nedidelės apimties - iki keleto tūkstančių baitų. Dideliuose duomenų objektuose (angl. *large objects*, *LOB*) galima talpinti didelius tekstų procesoriais paruoštus dokumentus, grafinius vaizdus, garso ir vaizdo įrašus, kitus audiovizualinius duomenis.
- **Vartotojo duomenų tipai**. Tradicinėse reliacinėse sistemose vartojamų duomenų tipų rinkinys yra ribotas. Galimybė vartotojui pačiam apibrėžti naujus duomenų tipus panaikino šį ribotumą.
- **Struktūriniai duomenų tipai**. Reliacinių duomenų tipų reikšmės yra nedalomi atomai. Naujieji struktūriniai duomenų tipai leidžia apjungti kelis vienodų ar skirtingų tipų elementus į vientisą struktūrą, kartu išsaugant ir kiekvieno jų išskirtinumą.
- **Vartotojo funkcijos**. Kad būtų tenkinami pastoviai augantys vartotojų poreikiai, reliacinėse DBVS buvo diegiamos naujos skaliarinės bei agregatinės funkcijos. Problema tapo nebeaktuali įdiegus galimybę vartotojui pačiam apibrėžti reikiamas funkcijas. ORDB sistemose funkcijas galima realizuoti įprastomis, tame tarpe objektinėmis, programavimo kalbomis.
- **Eilučių tapatumo požymiai**. Šiuolaikinės ORDBVS automatiškai suteikia lentelių eilutėms unikalius tapatumo požymius, atliekančius pirminio rakto vaidmenį.
- **Aktyvūs duomenys** – tai duomenys su apibrėžtomis elgesio taisyklėmis. Taisyklių prisilaikymu užtikrinamas duomenų vientisumas. Pagal nustatytas taisykles apdorojamos duomenų įvedimo ir keitimo klaidas ir pan. Duomenų elgesys aprašomas deklaratyviais apribojimais reikšmėms, dalykinėmis taisyklėmis (trigeriais) ir kt.

Tekstų procesoriais paruošti dokumentai, skaitmeniniai garsų įrašai ir pan. kompiuterio atmintyje gali užimti daug kilobaitų ar net megabaitų. Tokių duomenų neįmanoma įvesti į RDB tradicinėmis priemonėmis, nurodant juos betarpiškai SQL sakinyje. ORDBVS įdiegta galimybė nurodyti reikšmę bylos vardu. Daug vietos kompiuterio atmintyje reikalaujančias reikšmes, esančias duomenų bazėje, galima nuskaityti į bylą, o paskui apdoroti specialia programa, pavyzdžiui, tektų procesoriumi. Šiuolaikinėse sistemose didelės apimties reikšmę galima vartoti netgi duomenų paieškos sąlygoje. Daugelyje komercinių sistemų yra simboliniai (angl. *character large object*, `CLOB`) ir dvejetainiai dideli objektai (angl. *binary large object*, `BLOB`). `CLOB` skirti dideliems tekstams, o `BLOB` įvairių taikymų dvejetainiams kodams.

Įvedus į ORDB didelius objektus ir kitas objektines priemones (visų pirma, naujus duomenų tipus ir funkcijas), susidarė galimybė sudarinėti **reliacinius išplėtimus** (angl.

relational extender), skirtus atskiroms DB taikymo sritims. Reliacinio išplėtimo sąvoka nėra standartizuota. Skirtingose komercinėse RODBVS reliaciniai išplėtėjai vadinami skirtingai, pvz., DB2 jie vadinami "*Relational Extenders*", Oracle – "*Data Options*", Informix – "*Data Blades*". Reliacinio išplėtimo sąvoka iš dalies atitinka klasių bibliotekos sąvoką objektinio programavimo sistemose. Supaprastintai reliacinį išplėtimą galima apibūdinti kaip papildomus duomenų tipus ir funkcijas, kurie sprendžia konkrečios taikomosios srities uždavinius.

Išvardinsime gana dažnai vartojamus reliacinius išplėtimus, būdingus objektinei-realiacinei sistemai DB2:

- Grafinis išplėtimas (angl. *image extender*), kuris leidžia vartoti visus populiariausius grafinių duomenų formatus: GIF, JPEG, BMP, TIFF, bei konvertuoti duomenis iš vieno formato į kitą. Darbui su grafiniais vaizdais yra sudarytos tokios pagrindinės galimybės: pateikti nedidelį surasto vaizdo fragmentą pradinei jo peržiūrai, pavaizduoti užklausos rezultatą grafiškai, atlikti kontekstinę vaizdų paiešką pagal nurodytą vaizdo ar jo dalies spalvą ir kontrastą, ieškoti vaizdų, panašių į pateiktąjį pagal spalvą bei faktūrą, pagal nurodytą kontūrą ir pan.
- Vaizdų išplėtimas (angl. *video extender*) leidžia įsiminti ir pateikti vaizdo įrašus populiariuose formatuose: MPEG1, MPEG2, AVI, Quicktime. Sudaryta galimybė automatiškai segmentuoti įrašus pagal scenos pasikeitimus bei surasti scenos atstovą. Funkcijomis galima sužinoti įvairias vaizdo įrašo charakteristikas.
- Garsų išplėtimas (angl. *audio extender*) leidžia įsiminti ir pateikti garsų įrašus visuose populiariuose formatuose: AIFF, MIDI, WAVE. Funkcijomis galima sužinoti daugelį įrašų charakteristikų: įrašo trukmę, garso takelių skaičių ir pan.
- Tekstų išplėtimas (angl. *text extender*), kuris:
  - leidžia išsaugoti duomenų bazėje įvairiais tekstų procesoriais (*Microsoft Word*, *Word Perfect*, *AmiPro*) sudarytus dokumentus. Tokie tekstai gali tapti stulpelio reikšme konkrečioje lentelės eilutėje;
  - sukuria specializuotus indeksus kontekstinei (pagal žodžius, jų dalis ir frazes) paieškai dideliuose tekstuose;
  - atsižvelgia į tai, kuria kalba (anglų, vokiečių, prancūzų ir kt.) yra sudarytas dokumentas;
  - ieškant duomenų bazėje dokumentų, atsižvelgia į žodžių sinonimus bei formas.

Sudarysime užklausą, panaudodami joje tekstų išplėtimo funkciją. Tarkime, DB *Darbai* lentelėje *Vykdytojai* yra stulpelis *CV*, kurio reikšmės yra sudarytos tekstų procesoriumi, pvz., *Microsoft Word*. Šio stulpelio reikšmė – darbuotojų gyvenimo aprašas (autobiografija). Sąrašą darbuotojų, kurių autobiografijoje yra paminėtas žodis "stažuotė", bei bent vienas iš žodžių "Vokietija" ar "Prancūzija", ir tekste nėra žodžio "JAV", gausime įvykdę užklausą:

```
SELECT Nr, Pavardė FROM Vykdytojai
WHERE  CONTAINS( CV,
          '("stažuotė" & ("Vokietija" | "Prancūzija") & NOT "JAV" )' ) > 0 .
```

Funkcija `CONTAINS` yra dviejų argumentų: pirmasis - didelis tekstas (simbolinis didelių objektas, `CLOB`), o antrasis – paprasta simbolių eilutė, kurioje tarp dvigubų kabučių rašomi prasmingi žodžiai, kurie jungiami loginėmis operacijomis. Ši simbolių eilutė tekstų išplėtimo funkcijai `CONTAINS` yra sąlyga kontekstinei dokumentų paieškai atlikti. Apdorojant šią simbolių eilutę yra analizuojama jos struktūra.

Jau minėjome, kad objektinėse-reliacinėse DB, lyginant jas su reliacinėmis DB, yra įdiegta daug naujų sąvokų. Plačiau aptarsime tik dvi iš jų: vartotojo duomenų tipus bei funkcijos.

10.4. Naujų duomenų tipų apibrėžimas

Naudodamas jau esamus duomenų tipus vartotojas gali apibrėžti naujus (angl. *user-defined type*). Tam SQL kalba yra papildyta sakiniu `CREATE DISTINCT TYPE`.

Tarkime, DB *Darbai* lentelėje *Projektai* mums reikia saugoti ir projekto vertę pinigine išraiška. Projekto vertė – tai sutartinė suma, kuri patenka į įstaigos sąskaitą bazine užsienio valiuta ir/arba į sąskaitą litais. Kadangi suma nėra konvertuojama pinigų pervedimo ir duomenų įvedimo metu, be to, santykis tarp lito ir bazinės valiutos gali keistis, todėl patogu lentelę papildyti ne vienu, o dviem naujais stulpeliais *Vertė_Valiuta* ir *Vertė_Lt*. Abu šie stulpeliai galėtų būti `DECIMAL` tipo. Tačiau tuomet užklausose galima rašyti, pavyzdžiui, tokią sąlygą *Vertė_Valiuta* < *Vertė_Lt*, kuri nors ir yra sintaksiškai teisinga, bet joje lyginamos dvi logiškai nesulyginamos (netiesiogiai sulyginamos) reikšmės. Todėl kiekvienai valiutai apibrėžkime atskirą duomenų tipą:

```
CREATE DISTINCT TYPE Valiuta  AS DECIMAL(15, 2) WITH COMPARISONS,
CREATE DISTINCT TYPE Litai    AS DECIMAL(15, 2) WITH COMPARISONS,
```

čia *Valiuta* ir *Litai* yra naujų duomenų tipų vardai, o frazė `WITH COMPARISONS` nurodo, kad apibrėžiamo tipo reikšmes bus galima vartoti palyginimo operacijose: <, >, = ir kt. Naujojo tipo reikšmių negalima lyginti, jei ši frazė nenurodoma. Nurodant tipų vardus, kaip ir kitų DB objektų vardus, galima papildyti juos schemos vardu. Vartotojo apibrėžtus tipus galima vartoti, kaip standartinius, pvz.,

```
ALTER TABLE Projektai ADD Vertė_Valiuta Valiuta;
ALTER TABLE Projektai ADD Vertė_Litais  Litai.
```

Sukūrus naują tipą automatiškai sudaroma funkcija naujojo tipo reikšmėms sudaryti. Tos funkcijos vardas sutampa su tipo vardu, o argumentas yra bazinio tipo. Sudarykime užklausą projektams, kurių vertė užsienio valiuta (valiutinėje sąskaitoje) yra didesnė už 10000:

```
SELECT Pavadinimas FROM Projektai WHERE Vertė_Valiuta > Valiuta(10000) ,
```

čia reiškinio *Valiuta*(10000) tipas - *Valiuta*, o jo reikšmė yra 10000. Sakinys:

```
SELECT Pavadinimas FROM Projektai WHERE Vertė_Valiuta > 10000 ,
```

yra sintaksiškai neteisingas. Naujai apibrėžtiems tipams SQL nustato griežtą tipų kontrolę. Sintaksiškai neteisingas ir toks sakinys:

```
SELECT Pavadinimas FROM Projektai WHERE Vertė_Valiuta < Vertė_Litais .
```

Norint palyginti dvi skirtingų tipų reikšmes, reikia atlikti tipų suvienodinimą, pvz., vieną iš reikšmių paversti kito tipo reikšme. Apibrėžiant naują tipą, kartu sudaroma ir funkcija naujojo tipo reikšmei paversti bazinio tipo reikšme, pvz.,

```
SELECT Pavadinimas FROM Projektai
WHERE DECIMAL(Vertė_Valiuta) < DECIMAL(Vertė_Litais).
```

Šios užklausos sintaksė yra teisinga, tačiau logiškai sakinys nėra teisingas, kadangi lyginami skirtingų dimensijų dydžiai. Panašiam palyginimui reikalingas valiutos konvertavimas. Tam užklausoje galima įvesti daugiklį, pvz.,

```
SELECT Pavadinimas FROM Projektai
WHERE DECIMAL(Vertė_Valiuta) * 4 < DECIMAL(Vertė_Litais).
```

Tačiau, jei kursas gali kisti, tai užklausą reikia parametrizuoti. Paprasčiausias būdas įtraukti parametrą, kurį galima vartoti ir interaktyviame SQL, yra sudaryti skaliarinę funkciją.

10.5. Naujų funkcijų apibrėžimas

Sukuriant naują duomenų tipą, jokios kitos funkcijos, išskyrus reikšmių pervedimo tarp bazinio ir naujojo tipo funkcijas, automatiškai nėra sudaromos. Reiškinys *Valiuta*(10) + *Valiuta*(20) yra sintaksiškai neteisingas, jei duomenų tipui *Valiuta* nėra apibrėžta sudėties operacija.

Naujos funkcijos apibrėžiamos sakiniu `CREATE FUNCTION`. Sakinyje nurodoma apibrėžiamos funkcijos vardas, argumentų kiekis bei jų tipai, rezultato tipas ir funkcijos reikšmės apskaičiavimo būdas. Paprasčiausiai funkciją galima apibrėžti betarpiškai per atitinkamą bazinio tipo funkciją. Pvz.:

```
CREATE FUNCTION "+"(Valiuta, Valiuta) RETURNS Valiuta
   SOURCE "+"(DECIMAL(15, 2), DECIMAL(15, 2));
CREATE FUNCTION "*"(Valiuta, DECIMAL(10, 5)) RETURNS Valiuta
   SOURCE "*"(DECIMAL(15, 2), DECIMAL(10, 5)).
```

Apibrėžus šias dvi funkcijas, galima atlikti sudėties ir daugybos operacijas tarp dviejų *Valiuta* tipo reikšmių. Funkcijų apibrėžimuose nurodyta, kad jų reikšmės apskaičiuojamos panaudojant bazines operacijas, vartojamas su bazinio tipo `DECIMAL` reikšmėmis. Kadangi kvadratiniai piniginiai vienetai (pvz., litai × litai) yra neprasmingi, todėl daugybos operaciją apibrėžėme piniginių vienetų daugyba iš bedimensinio koeficiento. Naujai apibrėžtas sudėties ir daugybos operacijas, kaip ir standartines, galima rašyti tiek prefiksine, tiek ir sufiksine formomis.

Panašiai galime apibrėžti ne tik skaliarines, bet ir agregatines (stulpelių) funkcijas, pvz.:

```
CREATE FUNCTION Sum_Valiuta(Valiuta)
   RETURNS Valiuta SOURCE SUM(DECIMAL(15, 2)).
```

Naująsias funkcijas apibrėžėme per standartines funkcijas, praktiškai, tik papildydami standartinių funkcijų apibrėžimo sritį naujų tipų reikšmėmis. Naujųjų funkcijų reikšmės gali būti apskaičiuojamos SQL sakiniais. Apibrėžkime, bazinės valiutos konvertavimo į litus funkciją:

```
CREATE FUNCTION Valiuta_Litai(X Valiuta) RETURNS Litai
   LANGUAGE SQL
   CONTAINS SQL
   RETURN X * 4.0 ,
```

čia *X* yra formalus funkcijos parametras, fraze `LANGUAGE` nurodoma, kad funkcija yra realizuojama SQL reiškiniu, frazėje `RETURN` pateikiamas SQL reiškinys funkcijos reikšmei apskaičiuoti, fraze `CONTAINS SQL` nurodoma, kad SQL reiškinyje nėra vykdoma jokia užklausa. Vietoje `CONTAINS SQL` nurodžius `READS SQL DATA`, SQL reiškinyje galima naudoti ne tik konstantas ir formalius apibrėžiamos funkcijos parametrus, bet ir SQL sakinius. Apibrėžkime skaliarinę funkciją, kuria galima būtų apskaičiuoti, kiek konkretus vykdytojas skiria valandų visiems projektams vykdyti:

```
CREATE FUNCTION Sum_Valandos(Nr INTEGER) RETURNS INTEGER
   LANGUAGE SQL
   NOT DETERMINISTIC
   NO EXTERNAL ACTION
   READS SQL DATA
   RETURN (SELECT SUM(Valandos) FROM Vykdymas WHERE Vykdytojas = Nr),
```

čia fraze `NOT DETERMINISTIC` nurodoma, kad apskaičiuojant funkcijos reikšmę tai pačiai argumento reikšmei galima gauti skirtingus rezultatus, fraze `NO EXTERNAL ACTION` pranešama sistemai, kad vykdant funkciją nepasikeis jokio išorinio objekto, kurio nevaldo DBVS, pvz., failo būsena. Alternatyva frazei `NOT DETERMINISTIC` yra `DETERMINISTIC`, o frazei `NO EXTERNAL ACTION` - `EXTERNAL ACTION`. Tiek frazė

DETERMINISTIC, tiek ir NO EXTERNAL ACTION, leidžia DBVS optimizuoti pakartotinų kreipinių į funkciją atlikimą.

Apibrėžta skaliarinė funkcija *Sum_Valandos* gali labai sutrumpinti kai kuriuos SQL sakinius, pvz.,

SELECT *Pavardė*, *Sum_Valandos*(*Nr*) FROM *Tiekėjai.*

Priminsime, kad užklausos SELECT frazėje esantys reiškiniai skaičiuojami kiekvienai lentelės eilutei. Šioje užklausoje nėra dalinių užklausų. Tačiau, jei prisiminsime, kaip skaičiuojama funkcijos *Sum_Valandos* reikšmė, tai pastebėsime, kad pastaroji užklausa struktūriškai labai panaši į užklausą su koreliuota daline užklausa. Dėl panašių priežasčių negalima piktnaudžiauti tokiomis funkcijomis. Jų vartojimas gali būti paslėpta užklausų neefektyvumo priežastimi.

SQL nėra vienintelė kalba, kuria galima apibrėžti funkcijos reikšmę. Daugelis komercinių DBVS leidžia apibrėžti išorines funkcijas. **Išorine** vadinama **funkcija**, kuri yra realizuota kuria nors bazine programavimo kalba, jos vykdomasis (mašininis) kodas yra dinaminio ryšio bibliotekoje (angl. *Dynamic Link Library*), o duomenų bazėje yra saugomas tik funkcijos sąsajos apibrėžimas. Daugelis DBVS leidžia kreiptis į išorines funkcijas, sudarytas C ir JAVA programavimo kalbomis.

Tarkime, turime litų konvertavimo į valiutą funkciją, realizuotą ne SQL reiškiniu, bet C programavimo kalba. Tuomet mums reikia pranešti duomenų bazei apie funkcijos egzistavimą ir jos sąsają:

```
CREATE FUNCTION Litai_Valiuta(Litai) RETURNS Valiuta
EXTERNAL NAME 'konvertavimas ! litai_valiuta'
   LANGUAGE C
   PARAMETER STYLE DB2SQL
   DETERMINISTIC
   NO SQL
   NO EXTERNAL ACTION,
```

čia fraze EXTERNAL NAME nurodyta bibliotekos vardas (*konvertavimas*) ir joje esančios funkcijos vardas (*litai_ valiuta*), LANGUAGE - bazinė programavimo kalba (C) nuo kurios gali, pvz., priklausyti parametrų perdavimo tvarka, kreipiantis į funkcijos mašininį kodą; PARAMETER STYLE - funkcijos argumentų tipų atitikimo stilius (DB2SQL - funkcijai perduodami ir grąžinami parametrai tenkina DB2 SQL tipų atitikimo programoje reikalavimus, pvz., data vaizduojama dešimties simbolių eilute); NO SQL - funkcijos kūne nėra SQL sakinių (alternatyva – CONTAINS SQL).

DBVS, pasiruošdama vykdyti užklausą su kreipiniu į išorinę funkciją, pvz.,

```
SELECT   Pavadinimas,
         Vertė_ Valiuta + Litai_Valiuta(Vertė_Litais) AS "Bendra vertė valiuta"
FROM Projektai,
```

atlieka sintaksinę sakinio analizę pasitelkusi funkcijos sąsajos apibrėžimą. Pradedant vykdyti užklausos vykdomąjį planą, DBVS kreipiasi į operacijų sistemą "prašydama" įkelti į operatyviąją atmintį reikiamą biblioteką ir kiekvieną kartą, skaičiuodama reiškinio reikšmę, vykdo išorinės funkcijos mašininį kodą.

Išorinių funkcijų sudarymo technika yra labiau susijusi su programavimo technika nei su DB vartojimu, todėl detaliau jos nenagrinėsime.