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.

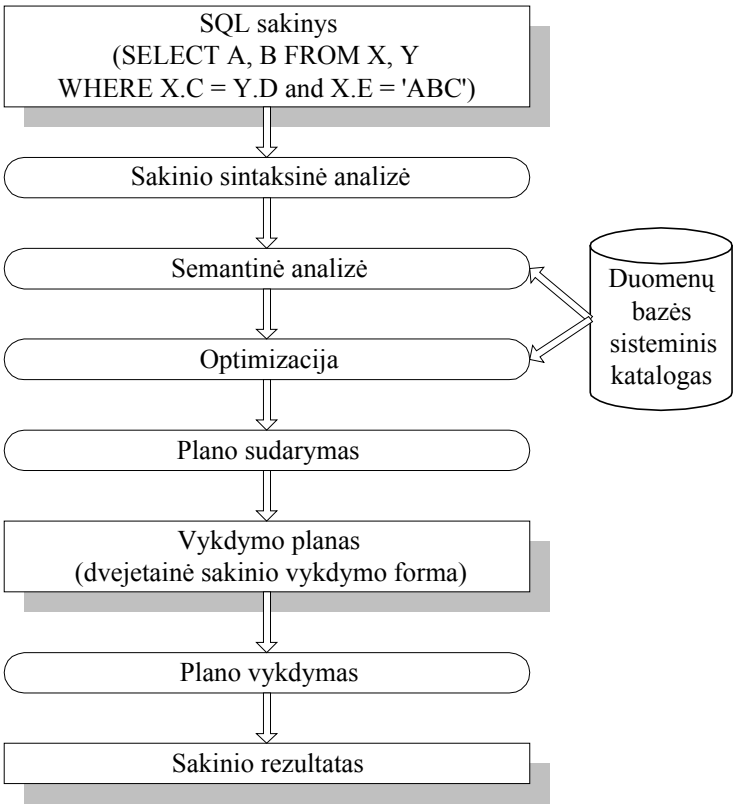
Praktikoje sėkmingai naudojama tiek programų SQL, tiek ir taikomųjų programų sąsaja. Šioje knygelėje aptarsime tik programų SQL.

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.

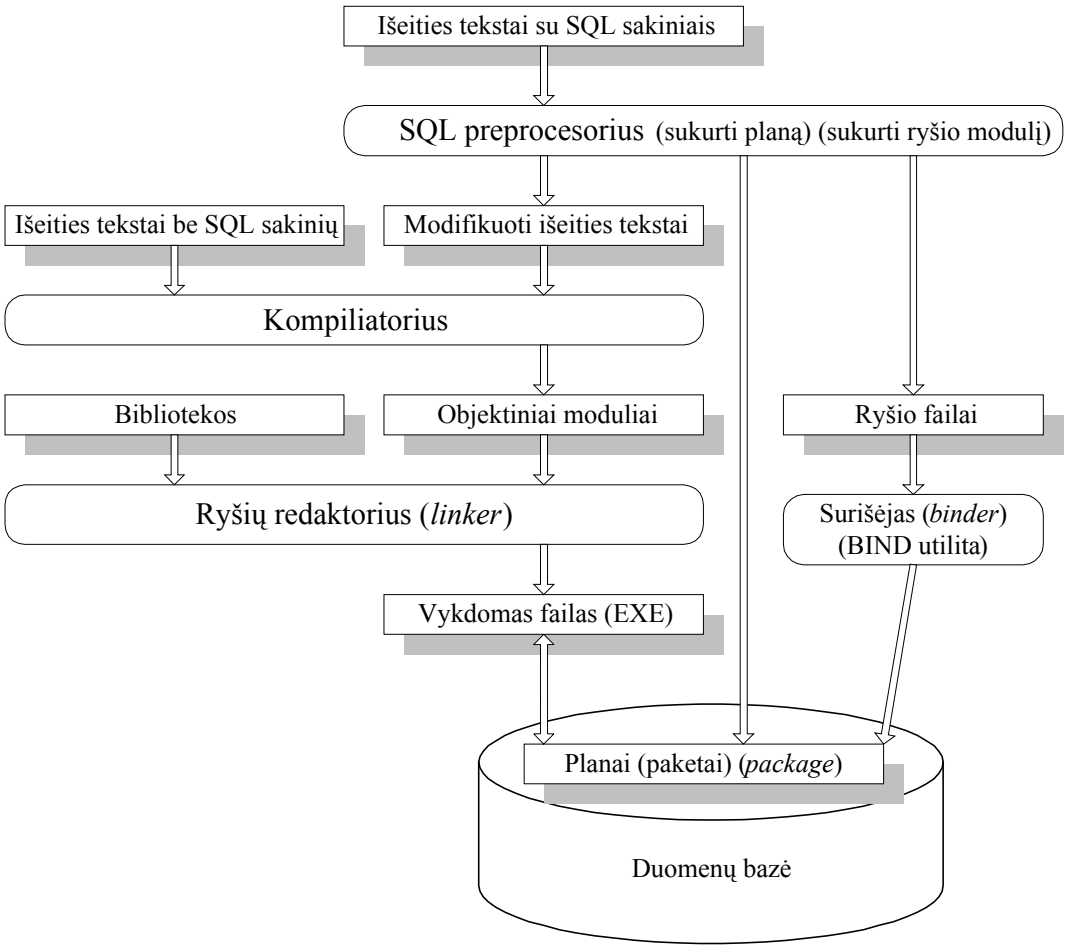
8.4. Programos paruošimo schema

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.

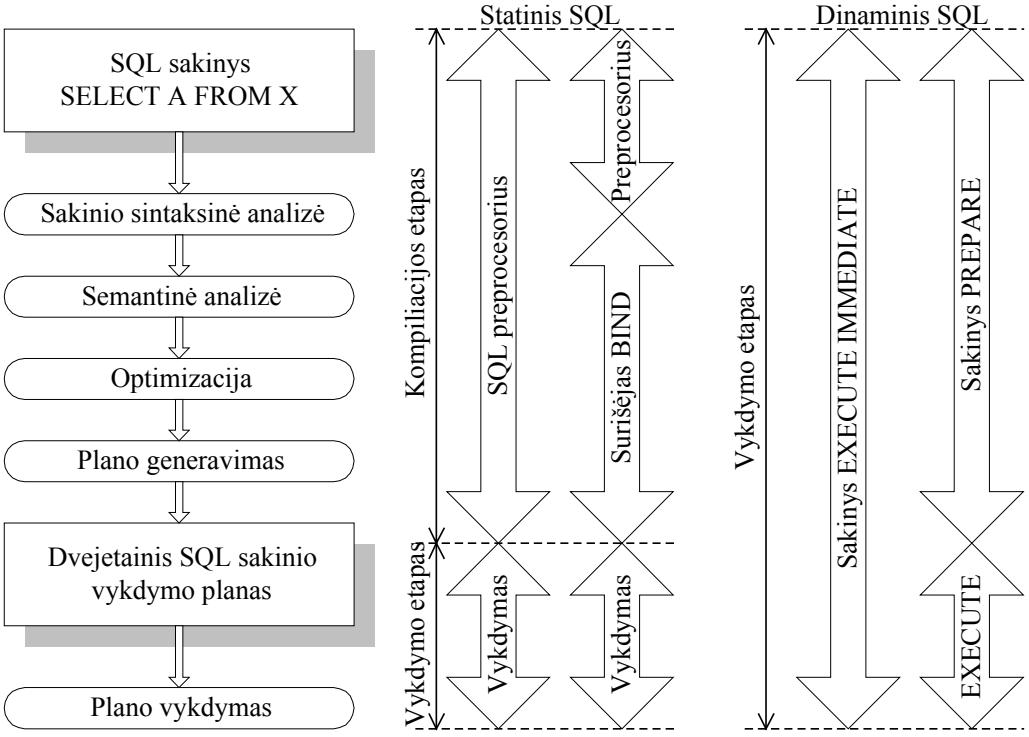
8.8. Dinaminių SQL sakinių vykdymas

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.