8.9. Dinaminių užklausų vykdymas

Sudarant programą, kuri išvestų
- vartotojo pasirinktos lentelės ir
- pasirinktų jos stulpelių reikšmes,

negalime aprašyti bazinių kintamųjų, nes nežinome:
- stulpelių skaičiaus;
- stulpelių tipų.

=> **negalime parašyti** `FETCH` sakinio.

Tuomet sakiniuose `PREPARE` ir `FETCH` reikia naudoti SQL apibrėžimo sritį

SQLDA (SQL *Description Area*)

SQLDA – tai kintamo ilgio struktūra, susidedanti iš kelių dalių:
- **pastoviosios** dalies, kuri yra struktūros pradžioje,
- **kintamosios** dalies – struktūrų ***SQLVAR* masyvo**.

```
struct sqlda
{
  char   sqldaid[8]; /* Eye catcher = 'SQLDA   '      */
  long   sqldabc;    /* SQLDA size = 16+44*SQLN       */
  short  sqln;       /*Number of SQLVAR elements      */
  short  sqld;       /* # of columns or host vars.    */
  struct sqlvar sqlvar[1]; /* first SQLVAR element    */
}
```


Kiekvieną SQL sakinio parametrą (stulpelį) atitinka 1 ***SQLVAR*** struktūra.

Masyvo elementų skaičius yra įsimenamas pastovioje dalyje.

Struktūroje ***SQLVAR*** yra laukai, atitinkantys:
- stulpelio duomenų tipą ir ilgį,
- kintamąjį – indikatorių,
- nuoroda į reikiamo ilgio duomenų sritį stulpelio reikšmei patalpinti.

Struktūra SQLVAR yra išsamus stulpelio aprašas (deskriptorius):

```
struct sqlvar          /* Variable Description        */
{
  short   sqltype;     /* Variable data type          */
  short   sqllen;      /* Variable data length        */
  char   *sqldata;     /* Pointer to variable data value*/
  short  *sqlind;      /* Pointer to Null indicator   */
  struct  sqlname sqlname;     /* Variable name       */
}
```


SQL sakiniu `DESCRIBE`

`EXEC SQL DESCRIBE` <SQL sakinio vardas> `INTO` :<sqlda tipo kintamasis>

Galima „sužinoti“ SQL sakinyje – simbolių eilutėje esančių parametrų (stulpelių) skaičių.

Žinant stulpelių skaičių, galima dinamiškai (*malloc, new*) išskirti atmintį SQLVAR struktūrų masyvui.

Paruošus atminties sritį stulpelių aprašams, sakiniu DESCRIBE galima dar kartą kreiptis į DBVS, kad užpildytų stulpelių aprašų sritį (stulpelių vardus,...)

Žinant stulpelių aprašus, programavimo kalbos priemonėmis galima išskirti atmintį užklausos rezultato eilutės visų stulpelių reikšmėms.


```
EXEC SQL INCLUDE SQLDA;/*Įtraukti apibrėžimo sritį*/
EXEC SQL BEGIN DECLARE SECTION;
  char sqlStmt[32000] ;       /*Masymas SELECT'ui */
EXEC SQL END DECLARE SECTION;

  struct sqlda sqlda ;  /*Kintamasis-apibrėžimo sritis */
/* masyve sqlStmt  suformuojamas SELECT'as          */
EXEC SQL PREPARE stmt FROM :sqlStmt ;
EXEC SQL DECLARE curs CURSOR FOR  stmt ;
memset( &sqlda, 0, sizeof(sqlda) ) ; /* "Išvalyti" sritį */
/* Prašome sistemos užpildyti stulpelių kiekį: */
EXEC SQL DESCRIBE stmt INTO :sqlda ;
```



```
/*Išskiriame atmintį stulpelių aprašams-SQLVAR masyvui*/
/*Prašome detalios informacijos apie kiekvieną stulpelį: */
EXEC SQL DESCRIBE stmt INTO :sqlda ;
/* Išskiriame atmintį kiekvieno stulpelio reikšmei. */
/* Nuorodas į reikšmių sritis talpiname SQLVAR laukuose*/
EXEC SQL OPEN  curs; /*Atidaryti užklausos žymenį */
EXEC SQL FETCH  curs USING DESCRIPTOR :sqlda;
/*Duomenų apdorojimas, ir kt. eilučių skaitymas */
```


C funkcija su 1 parametru – simbolių eilute – užklausa.

```
void SelectUsingDescribe(char *selectStmt)  {
  EXEC SQL BEGIN DECLARE SECTION;
    char *strStmt;  /* - kintamasis SELECT sakiniui */
  EXEC SQL END DECLARE SECTION;
  EXEC SQL WHENEVER SQLERROR  GOTO error;
  EXEC SQL WHENEVER NOT FOUND GOTO end;
  struct sqlda *pSqlda = NULL;
  int nofColumns ;
  strStmt = selectStmt ;
  EXEC SQL PREPARE stmt FROM :strStmt;
  /*Ruošiame atminti stulpelių skaičiui*/
  SqldaInit(&pSqlda, 1);
```

```
/* Sužinome stulpelių skaičių: */
EXEC SQL DESCRIBE stmt INTO :*pSqlda;
nofColumns = (int) pSqlda->sqld;
free(pSqlda);
/* Ruošiame atmintį duomenims apie stulpelius: */
SqldaInit(&pSqlda, nofColumns);
/* Sužinome visų stulpelių aprašus */
EXEC SQL DESCRIBE stmt INTO :*pSqlda;
EXEC SQL DECLARE curs CURSOR FOR stmt;
EXEC SQL OPEN curs;
/* Ruošiame atmintį visų stulpelių reikšmėms */
RowDataMemoryAlloc(pSqlda);
```



```
while( 0 == SQLCODE ) {
  EXEC SQL FETCH curs
           USING DESCRIPTOR :*pSqlda;
   RowDataDisplay(pSqlda);
}
EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL WHENEVER NOT FOUND CONTINUE;
error:
  printf("SQL klaida: %ld\n", SQLCODE) ;
end:
  EXEC SQL CLOSE curs ;
}
```



```
void SqldaInit(struct sqlda **ppSqlda,
               int nofColumns)
{
  int memSize =
      offsetof(struct sqlda, sqlvar) +
      nofColumns*sizeof(struct sqlvar);
  *ppSqlda = (struct sqlda *) malloc(memSize);
  memset(*ppSqlda, '\0', memSize ) ;
  memcpy((*ppSqlda)->sqldaid, "SQLDA   ", 8);
  (*ppSqlda)->sqldabc = memSize ;
  (*ppSqlda)->sqln = nofColumns;
  (*ppSqlda)->sqld = 0;
}
```



```
void RowDataMemoryAlloc (struct sqlda
*pSqlda) {
  short iCol;
  unsigned int memSize;
 for(iCol = 0; iCol < pSqlda->sqld; iCol++) {
   /* Išskiriame atmintį indikatoriui */
   pSqlda->sqlvar[iCol].sqlind = (short *)
   malloc(sizeof(short));
   memset(pSqlda->sqlvar[iCol].sqlind, '\0',
          sizeof(short));
```



```
/* Išskiriame atmintį reikšmei */
  switch (pSqlda->sqlvar[iCol].sqltype) {
    case SQL_TYP_FLOAT:
    case SQL_TYP_SMALL:
     memSize=pSqlda->sqlvar[iCol].sqllen;
    case SQL_TYP_DATE:
    case SQL_TYP_TIME:
    case SQL_TYP_CHAR:
    case SQL_TYP_VARCHAR:
     memSize=pSqlda->sqlvar[iCol].sqllen+1;
     break;
```



```
      /*Išskiriame atmintį sudėtingesnių tipų reikšmėms*/
       case SQL_TYP_DECIMAL:
        ...
      }
      pSqlda->sqlvar[iCol].sqldata =
                (char *) malloc(memSize);
      memset(pSqlda->sqlvar[iCol].sqldata, '\0',
             memSize);
  }
}
```



```
void RowDataDisplay(struct sqlda *pSqlda)
{
  short iCol;
  for( iCol = 0; iCol < pSqlda->sqld; iCol++ )
       CellDataDisplay(&pSqlda->sqlvar[iCol]) ;
  printf("\n");   /* - eilutės pabaiga */
}
```



```
void CellDataDisplay( struct sqlvar *pSqlvar )
{
 printf("%s:", pSqlvar->sqlname.data); /*vardas*/
 if(pSqlvar->sqlind < 0)
   printf("-");     /* - NULL reikšmė */
 else {
   switch (pSqlvar->sqltype) {
     case SQL_TYP_DATE:
     case SQL_TYP_TIME:
     case SQL_TYP_CHAR:
     case SQL_TYP_VARCHAR:
      printf("%s", pSqlvar->sqldata);
      break;
```

```
      case SQL_TYP_SMALL:
       printf("%d",*((short *)pSqlvar->sqldata));
       break;
      case SQL_TYP_FLOAT:
       printf("%f",*((double *)pSqlvar->sqldata));
       break;
       /* Kitų tipų reikšmių išvedimas */
        . . .
    }
  }
}
```



```
void RowDataMemoryFree(struct sqlda *pSqlda)
{
  short  iCol;
  for( iCol = 0; 0 != pSqlda && iCol< pSqlda->sqld;
                                        iCol++ ) {
   if( 0 != pSqlda->sqlvar[iCol].sqldata)
     free(pSqlda->sqlvar[iCol].sqldata);
   if( 0 != pSqlda->sqlvar[iCol].sqlind)
     free(pSqlda->sqlvar[iCol].sqlind);
  }
  if( 0 != pSqlda )
     free(pSqlda);
}
```


8.10 Sąsaja JDBC

Taikomųjų programų sąsaja pateikia programuotojams funkcijas-paprogrames SQL sakiniams atlikti.

JDBC (*Java Database Connectivity*) taikoma **JAVA** programavimo kalbos programose.

```
import java.sql.*;
```

Programos nereikia prekompiliuoti.

Visi SQL sakiniai, kuriais kreipiamasi į DBVS paruošiami programos vykdymo metu - dinamiškai

=> programų sąsaja mažiau efektyvi už statinius programų SQL sakinius. Bet .. **patogi**.

Programų sąsaja ypač **gerai tinka dinamiškiems SQL** sakiniams vykdyti.

8.10.1. Ryšys su DB

Ryšys tarp programos ir duomenų bazės nustatomas **2 etapais**:

- Įkeliama konkrečiai DBVS būdinga tvarkyklė, užtikrinti tolimesnių JDBC paslaugų nepriklausomumą nuo DBVS ypatumų.
- **PostgreSQL** tvarkyklę galima aktyvuoti taip

```
Class.forName("org.postgresql.Driver");
```

- IBM **DB2**:

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


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

```
Connection con =
      DriverManager.getConnection(
      "jdbc:postgresql://pgsql.mif/Darbai",
       username, password ) ;
```

`getConnection` parametrai:

- DB URL (*Unified Resource Location*), kurį sudaro:
  protokolas (jdbc),
  DBVS (postgresql, db2,..) ir
  nuoroda į DB (//pgsql.mif/*Darbai*).
- *username* - sistemos vartotojo vardas
- *password* - sistemos vartotojo slaptažodis.

8.10.2. Paprastų SQL vykdymas

SQL sakinių vykdymui JDBC yra numatyta objektų klasė `Statement`.

Sukuriant šios klasės objektą, ryšys su konkrečia DB jau turi būti nustatytas.

Turint `Connection` klasės objektą *con*, klasės `Statement` objektą galima sukurti taip

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

Konkretų klasės `Statement` objektą galima naudoti daugeliui SQL sakinių vykdyti.

Paprasčiausiai vykdomi DDL ir duomenų atnaujinimo sakiniai (`INSERT`, `DELETE`, `UPDATE`):

klasės `Statement` objektui

kviečiamas metodas `executeUpdate`

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

String str = "UPDATE Vykdytojai " +
             "SET Kategorija = Kategorija + 1";
stmt.executeUpdate(str);
```


Kad tą patį SQL sakinį būtų galima efektyviai atlikti keletą kartų

- iš pradžių jis paruošiamas,
- vykdamas daug kartų su reikiamomis parametrų reikšmėmis:

```
PreparedStatement stmt =
con.prepareStatement("UPDATE Vykdytojai" +
            "SET Kategorija = ? WHERE Pavarde = ?" ) ;
stmt.setInt(1, 5);
stmt.setString(2, "Jonaitis");
stmt.executeUpdate(); // - Jonaičiui nauja kategorija
```

stmt.**setInt**(1, 4);
stmt.**setString**(2, "Petraitis");
stmt.**executeUpdate**(); // - Petraičiui nauja kategorija

Metodais **setInt** ir **setString** yra priskiriamos reikšmės parametrams.

8.10.3. Transakcijos

Transakcijos yra valdomos **Connection** objekto metodais.

JDBC numatyta, kad po kiekvieno SQL sakinio **automatiškai** užbaigiama **transakcija**.

Kad užtikrinti kelių SQL sakinių transakcijas, reikia atšaukti numatytąjį transakcijų valdymą.

Automatinis transakcijų valdymas išjungiamas ir įjungiamas metodu **setAutoCommit**, kurį kviečiant reikia nurodyti vieną parametrą - **Boolean** tipo reikšmę.

COMMIT sakinį atitinka **commit** metodas

ROLLBACK – **rollback**.

8.10.4. Klaidų apdorojimas

Klaidoms, atsirandančias programos vykdymo metu, perteikti yra specialios klasės:
SQLException ir **SQLWarning**.

Klaidos apdorojamos objektinei kalbai įprastu būdu – „**gaudant**“ klaidų įvykius:

```
try { // atšaukiame automatini pakeitimų įtvirtinimą:
  con.setAutoCommit(false);
  stmt.executeUpdate(<SQL sakinys 1>);
  ...
  stmt.executeUpdate(<SQL sakinys n>);
  con.commit() ;
  con.setAutoCommit(true) ;
```



```
}
catch(SQLException ex) {
  System.err.println("SQLException: " +
                                ex.getMessage()) ;
  con.rollback() ;
  con.setAutoCommit(true) ;
}
```


8.10.5. Užklausų apdorojimas

Užklausos vykdomos kviečiant klasės **Statement** objektui metodą **executeQuery**.

Metodas rezultate grąžina – klasės **ResultSet** objektą.

Klasė užtikrina rezultato eilučių peržiūrą metodu **next**.

Kviečiant **next**, vidinis objekto žymuo kiekvieną kartą paslenkamas vis prie kitos eilutės.

Metodais, atitinkančiais rezultato stulpelių tipus, gaunamos eilutės reikšmes.

Visų darbuotojų vardai ir jų kategorijos:

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


Metoduose **getString** ir **getInt** naudojamus stulpelių vardus (*Pavardė* ir *Kategorija*) galima pakeisti stulpelių eilės numeriais:

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

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

Patogiam užklausos rezultato perrinkimui yra numatyta pakankamai daug metodų.
Jų vardai atspindi prasmę:

getRow
isFirst
isBeforeFirst
isLast
isAfterLast
absolute
previous
relative

ir kt.

Pvz.,

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