8. SQL sakiniai taikomosiose programose

8.6'. Statinių užklausų apdorojimas

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

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

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

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

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

8.10. Dinaminių užklausų vykdymas

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

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

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

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

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

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

8.11. Sąsaja JDBC

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

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

8.11.1. Ryšys su DB

Ryšys tarp programos ir duomenų bazės nustatomas dviem etapais:

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

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

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

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

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

8.11.2. Paprastų SQL vykdymas

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

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

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

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

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

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

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

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

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

8.11.3. Klaidų apdorojimas

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

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

8.11.4. Užklausų apdorojimas

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

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

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

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

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

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

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