JDBC käskude ülevaade

Kui kord ühendus loodud, siis edasised toimetused võiksid juba mõnevõrra lihtsamalt sujuda. Järgnevalt proovitakse läbi mitmed levinumad andmetega ümber käimise võtted.

Kõikide ridade väljastus.

Ehk levinumaiks esimeseks andmebaasiga seotud rakenduse ülesandeks ongi andmete kasutajale näitamine soovitud kujul. Tüüpilisel juhul tuleb luua ühendus, koostada päring, käivitada see ning saabunud andmed sobival kujul kuvada. Järgnevas näites küsitakse eelnevalt loodud inimeste andmete tabelist välja kõikide veergude andmed. Ekraanile aga näidatakse neist eesnime ja sünniaasta väärtused.

```
import java.sql.*;
public class Inimloetelu{
  public static void main(String[] argumendid) throws Exception{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection("jdbc:odbc:poebaas");
    Statement st=cn.createStatement();
    ResultSet rs=st.executeQuery("select * from inimesed");
    while(rs.next()){
      System.out.println(rs.getString("eesnimi")+":"+rs.getInt("synniaasta"));
    }
    cn.close();
  }
}

/*
C:\jaagup\andmed>java Inimloetelu
Juku:1989
Kati:1987
Mati:1983
*/
```

Andmed andmete kohta

Kui vaja rakenduse võimalusi sageli muuta, või kui soovitakse sama andmeväljastuslõiku kasutada mitmesuguste andmete korral, siis aitab päringu vastusega koos tulev metadata ehk andmed andmete kohta. Mõne andmebaasimootori või draiveri puhul võib vastav võimalus puududa või töötada nuditult. Kui aga andmeid kirjeldavate andmete küsimise võimalus olemas, siis saab neid sarnaselt kätte sõltumata kasutatavast andmebaasist.

Nagu alljärgnevast näitest näha, tuleb andmete kirjeldus ResultSet'ist eraldi käsuga välja küsida. Edasi õnnestub juba ResultSetMetaData tüüpi objektist üksikute käskude abil omale soovitavaid andmeid teada saada.

```
    ResultSetMetaData rmd=rs.getMetaData();
    int veergudearv=rmd.getColumnCount();
```

Kui veergude arv ja andmed teada, siis saab koostada koodilõigu, mis juba vastavalt andmed välja küsib ja nendega edasi toimetab. Siin näites trükitakse tulemused vaid ekraanile, kuid sarnaselt võib ette valmistada väljastuse veebilehele või mujalegi. Kes juhtub kasutama andmebaasihaldusvahendeid nagu näiteks

veebipõhine PHP MyAdmin, võib aimata, et taoliste rakenduste koostamisel kuluvad saabuvad andmed andmete kohta väga marjaks ära.

```
import java.sql.*;
public class Inimloetelu2{
  public static void main(String[] argumendid) throws Exception{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection("jdbc:odbc:poebaas");
    Statement st=cn.createStatement();
    ResultSet rs=st.executeQuery("select * from inimesed");
    ResultSetMetaData rmd=rs.getMetaData();
    int veergudearv=rmd.getColumnCount();
    System.out.println("Tabelis on "+veergudearv+" veergu:");
    for(int i=1; i<=veergudearv; i++){
      System.out.println("Nimi:         "+rmd.getColumnName(i));
      System.out.println("Kirjeldus:    "+rmd.getColumnLabel(i));
      System.out.println("Tyyp:         "+rmd.getColumnTypeName(i));
      System.out.println("Java klass:   "+rmd.getColumnClassName(i));
      System.out.println("Isesuurenev:  "+rmd.isAutoIncrement(i));
      System.out.println("Suurim laius: "+rmd.getColumnDisplaySize(i));
      System.out.println();
    }

    while(rs.next()){
       System.out.print(rs.getRow()+". ");
       for(int i=1; i<=veergudearv; i++){
         System.out.print(rs.getObject(i)+"    ");
       }
       System.out.println();
    }
    cn.close();
  }
}
```

Ning programmi töö tulemusena anti ilus selge ülevaade kasutatavast andmetabelist. Nii tulpade kirjeldused ükshaaval, kui pärast kogu tabeli sisu.

```
/*
Tabelis on 3 veergu:
Nimi:         id
Kirjeldus:    id
Tyyp:         COUNTER
Java klass:   java.lang.Integer
Isesuurenev:  true
Suurim laius: 11

Nimi:         eesnimi
Kirjeldus:    eesnimi
Tyyp:         VARCHAR
Java klass:   java.lang.String
Isesuurenev:  false
Suurim laius: 50

Nimi:         synniaasta
Kirjeldus:    synniaasta
Tyyp:         INTEGER
Java klass:   java.lang.Integer
Isesuurenev:  false
Suurim laius: 11

1. 1    Juku    1989
2. 2    Kati    1987
3. 3    Mati    1983

*/
```

Päringu tulemuste hulgas liikumine.

Esimese lähendina võib relatsioonilisest andmebaasist andmete välja küsimine tunduda küllalt selgena. Kõik andmed ja nende vahelised seosed esitatakse baasis

tabelitena ning ka tulemuseks on tabel - sarnane programmeerijale tuttava kahemõõtmelise massiiviga. Et toimingute sisemine keerukus püütakse rakenduse loojate eest võimalikult peita, siis ideaaljuhul polegi vaja muule mõelda kui rea numbrile ja veeru nimele või numbrile, kust andmed enesele välja küsida.

Andmebaasimootorid peavad hakkama saama suurte andmemahtude ning mitmete üheaegsete kasutajatega. Selle toimimise tarvis tuleb arvestada andmebaasiühenduse kasutatavate enesekaitsemehhanismidega. Vaikimisi seadete korral õnnestub päringust andmeid välja meelitada vaid ridu järjest eest tahapoole lugedes ning iga väärtust vaid ühe korra küsides. Sellise lähenemise puhul ei pea sugugi kõik päringu väljastatavad andmed olema korraga baasist välja küsitud. Mällu loetakse ja üle kantakse vaid need, mis parasjagu tarvilikud. Ülejäänud võivad veel oma aega oodata ning uuritud ridade arvelt võib sootuks mälu vabastada. Selline vaikimisi järjest küsimine peab ka kõigi töötavate draiverite puhul ühtviisi leiduma ja toimima. Kui teada ja arvestada taolist ette kirjutatud andmete küsimise järjekorda, siis enamike rakenduste puhul olulist probleemi ei teki. Ekraanile või veebilehele saadetaksegi andmed sageli saabumise järjekorras. Ning kui vaja kokkuvõtteid teha või mõnda väärtust korduvalt kasutada, siis tuleb lihtsalt vajalikud andmed muutujatesse ja massiividesse kirjutada ning edaspidi kasutada kui tavalisi programmeerimise juures tarvilikke andmeid.

On aga mingil põhjusel kindel vajadus mööda andmeid sageli edasi-tagasi liikuda, siis tuleb vajadust juba eelnevalt arvestada. Javakeelsete programmide puhul luuakse kõigepealt ühendus (Connection). Iga ühenduse kaudu võib korraga töötada mitu käsklust (Statement). Ning iga käskluse küljes võib vajadusel olla avatud korraga kuni üks vastuste kogum (ResultSet). Vastuste kogumi omadused määratakse juba käskluse loomisel. Järgnevas näites paistavad konstandid

```
ResultSet.TYPE_SCROLL_INSENSITIVE
ResultSet.CONCUR_READ_ONLY
```

Esimene neist määrab, et saabunud vastustehulka võib soovitud suunas kerida. Olgu siis edaspidi või tagurpidi. Suurema andmehulga korral võib selline nõudmine märgatavalt ressursse nõuda või riistvarale sootuks üle jõu käia. Kuid kui näiteks Swingi vahenditega tabelit luua, siis on päris mugav, kui võib otse ResultSetist saabuvaid andmeid usaldada ning ei pea hakkama veel lisaks oma andmekogumit looma.

Teise parameetri tagamaad on veel mõnevõrra keerulisemad. CONCUR_READ_ONLY tähendab, et andmeid võib päringust vaid lugeda, mis nagu päringu juures võikski loomulik tunduda. Nagu aga hilisematest näidetest paistab, ei pruugi see päringu kasutamise ainuke võimalus olla.

Järgnevas näites käiakse läbi ResultSet'is liikumise tähtsamad käsud. Enamik neist võiks olla otse inglise keelest tõlgitavad. Käsk last palub minna viimasele reale, isLast kontrollib, kas ollakse viimasel real; relative liigub soovitud arvu ridasid jooksvast reast alates, absolute loendab ridasid alates päringu algusest.

```
import java.sql.*;
public class Inimloetelu3{
  public static void main(String[] argumendid) throws Exception{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection("jdbc:odbc:poebaas");
    Statement st=cn.createStatement(
      ResultSet.TYPE_SCROLL_INSENSITIVE,
      ResultSet.CONCUR_READ_ONLY
    );
    ResultSet rs=st.executeQuery("select * from inimesed");
    rs.last();
    System.out.println("Ridu kokku: "+rs.getRow());
    rs.previous();
    System.out.println("Eelviimane eesnimi: "+rs.getString("eesnimi"));
```

```
    rs.absolute(3);
    System.out.println("Kolmas eesnimi: "+rs.getString("eesnimi"));
    rs.relative(2);
    System.out.println("Ylejärgmine eesnimi: "+rs.getString("eesnimi")+
                       ", reanr: "+rs.getRow());
    if(rs.isLast()){
      System.out.println("Tegemist on viimase reaga");
    }
    rs.relative(-1);
    System.out.println("Eelmine eesnimi: "+rs.getString("eesnimi")+
                       ", reanr: "+rs.getRow());
    cn.close();
  }
}



/*
Ridu kokku: 6
Eelviimane eesnimi: Juk's
Kolmas eesnimi: Mati
Ylejõrgmine eesnimi: Juk's, reanr: 5
Eelmine eesnimi: Sass, reanr: 4

____

Nimed:

Juku
Kati
Mati
Sass
Juk's
Jass

*/
```

Päringu mahu piiramine

Rakendusi koostades on vahel raske aimata, kui suurte andmekogustega tuleb tegemist teha. Näited, mis kahe või kümne rea juures ilusti toimivad, ei pruugi tuhandete vastusridade puhul enam sugugi kasutatavad olla. Olgu siis tegemist ekraani nähtava ala ummistumisega, arvuti mälumahu või arvutusvõimsuse lõpuga. Üheks võimaluseks on eelneva päringu abil kontrollida, millises suurusjärgus vastustehulgaga võiks tegemist olla ning edasi juba mitme võimaluse tarbeks kood kokku panna. Lihtsamaks ning mõnigikord kasutatavaks lähendiks aga piisab päringu ridade arvu või tööaja piiramisest. Kui ka ei saa soovitud kohast kõiki andmeid kätte, siis vähemasti jääb rakendus ellu, arvuti ei jookse kokku ning võimalik näiteks täpsustatud parameetritega uus päring kokku panna.

```
import java.sql.*;
public class Inimloetelu4{
  public static void main(String[] argumendid) throws Exception{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection("jdbc:odbc:poebaas");
    Statement st=cn.createStatement();
    st.setMaxRows(2);             //Suurim väljastatavate ridade arv
    //    st.setQueryTimeout(2);
    //Valikuline käsklus, päringu suurim aeg sekundites
    ResultSet rs=st.executeQuery("select eesnimi from inimesed");
    while(rs.next()){
      System.out.println(rs.getString("eesnimi"));
    }
    cn.close();
  }
}

/*
Juku
Kati
*/
```

Lisamine.

Harilik lisamine käib pea kõikide keelte ja vahendite puhul sarnaselt. Ikka tuleb kokku panna SQLi INSERT-lause ning siis käivitada. Java käskluseks nii lisamise kui muutmiste korral on executeUpdate. Vaid päringu puhul oli executeQuery, kus siis tulemuseks väljastati ResultSet.

```
import java.sql.*;
public class Inimlisamine1{
  public static void main(String[] argumendid) throws Exception{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection("jdbc:odbc:poebaas");
    Statement st=cn.createStatement();
    st.executeUpdate(
     "INSERT INTO inimesed (eesnimi, synniaasta) values ('Sass', 1977)"
    );
    cn.close();
  }
}
```

PreparedStatement

Lisamist levinud operatsioonina on püütud mitmel moel paindlikumaks muuta. Kui vaja kümneid kordi sama lauset erinevate andmetega käivitada, siis igakordne masinapoolne SQL-lause analüüs ning enesele sobivaks seadmine võtavad oma aja. Kui aga kord koostada PreparedStatement ning hiljem vaid väärtusi sees vahetada, siis võiks tulemus mõnevõrra kiiremini saabuda.

Teiseks mureks sisestuste juures on erisümbolid. Ehkki mõeldakse välja mitmeid viise andmete sisse jäävate ülakomade ja muude märkide varjestamiseks, kipub ikka turvaauke sisse jääma, kus lihtsalt sisestuse kaudu suudetakse SQL-laused sassi ajada ning rakenduse tööd muuta. Või teistpidi juhtub vahel, et sümboleid varjestatakse mitmekordselt ning hiljem kipuvad varjestuse jäljed väljundisse sisse jääma.

Kui kasutada aga PreparedStatementi, siis varjestusega muresid pole. Sest sisestatavaid andmeid otse SQL-lausesse ei kirjutatagi. Algsesse lausesse pannakse andmete kohale küsimärgid. Ning alles hiljem määratakse, millised andmed selle käivitamise korral küsimärkide asemele paigutatakse.

```
import java.sql.*;
public class Inimlisamine2{
  public static void main(String[] argumendid) throws Exception{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection("jdbc:odbc:poebaas");
    PreparedStatement st=cn.prepareStatement(
     "INSERT INTO inimesed (eesnimi, synniaasta) values (?, ?)"
    );
    st.setString(1, "Juk's");
    st.setInt(2, 1972);
    st.executeUpdate();
    cn.close();
  }
}
```

Päringus lisamine

Lihtsustamaks rakenduste loomist, kus nähtavaid andmeid ka kohe muuta saab, võimaldavad osa andmebaasimootoreid lihtsamate päringute korral ka päringust väljastatud tulemusi muuta või sama tüüpi ridasid algsetesse andmetesse juurde luua. Kui summeeritaks päringus näiteks inimeste arv, siis seda loomulikult muuta ei lubata - pole ju võimalik nõnda lihtsalt olematuid inimesi juurde tekitada. Kui aga päringuks on lihtsalt ühe tabeli esitus või ka lihtsam ühend, siis võib muutmine ja lisamine täiesti õnnestuda. Allpoolses näites küsitakse inimeste andmed ning edaspidiste käsklustega lisatakse üks inimene loetellu juurde.

```
    rs.moveToInsertRow();
    rs.updateString(1, "Jass");
    rs.updateInt(2, 1968);
    rs.insertRow();
```

räägib igaüks enese eest. Nii nagu võib mõnikord vormis andmeid viimasele reale juurde kirjutada, nii lubatakse ka siin programmi abil üks rida päringu poolt väljastatud tabelisse juurde panna, lahtrid väärtustega täita ning siis tulemused paika saata.

```
import java.sql.*;
public class Inimlisamine3{
  public static void main(String[] argumendid) throws Exception{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection("jdbc:odbc:poebaas");
    Statement st=cn.createStatement(
      ResultSet.TYPE_SCROLL_INSENSITIVE,
      ResultSet.CONCUR_UPDATABLE
    );
    ResultSet rs=st.executeQuery(
      "SELECT eesnimi, synniaasta FROM inimesed"
    );
    rs.moveToInsertRow();
    rs.updateString(1, "Jass");
    rs.updateInt(2, 1968);
    rs.insertRow();
    cn.close();
  }
}
```

Transaktsioonid

Vahel pidada olema pool muna halvem kui tühi koor. Et kui töö jäi tervikuna tegemata, siis järgmisel korral teada, et võib kõike otsast alustada. Kui aga miskit poole peale rippuma jäi, võib kergemini juhtuda, et mõni tegevus hiljem kaks korda tehtud saab või sootuks kahe silma vahele jääb. Tüüpiliseks näiteks tuuakse pangaülekannet, kus ühelt kontolt võtmine ning teisele ülekandmine ikka paarikaupa peavad käima. Ning enne lõplikku tulemuste kinnitamist peab veenduma, et mõlemad toimingud õnnestuvad. Selliste seoste loomiseks võib automaatse täitmise peatada käsuga.

```
    cn.setAutoCommit(false);
```

Edasised toimingud jäävad ootele. Kui selgub, et midagi tuli vahele, siis saab algseisu taastada käsuga

```
    cn.rollback();
```

```
import java.sql.*;
public class Inimlisamine4{
  public static void main(String[] argumendid) throws Exception{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
    Connection cn=DriverManager.getConnection("jdbc:odbc:poebaas");
    cn.setAutoCommit(false);
    Statement st=cn.createStatement();
    st.executeUpdate(
      "INSERT INTO inimesed(eesnimi, synniaasta) values('Elmar', 1955)"
    );
    cn.rollback();
    cn.close();
  }
}
```

On aga kõik õnneks läinud, siis kannatab öelda

```
    cn.commit();
```

ning muutused kinnistatakse.

```
import java.sql.*;
public class Inimlisamine5{
  public static void main(String[] argumendid) throws Exception{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection("jdbc:odbc:poebaas");
    cn.setAutoCommit(false);
    Statement st=cn.createStatement();
    st.executeUpdate(
      "INSERT INTO inimesed(eesnimi, synniaasta) values('Mann', 1911)"
    );
    cn.commit();
    cn.close();
  }
}
```

SQL-laused

Järgnevalt vaatame läbi enamlevinud SQL-käsklused. Näited on tehtud MySQLi-nimelise andmebaasi abil, kuid samalaadsed käsklused leiduvad ka teiste andmebaaside juures. Mõnel korral lihtsalt vaja täpne kuju vastavast manuaalist järele vaadata.

Tabeli loomiseks kasutatakse käsklust CREATE TABLE. Tavaks on kirjutada otse SQL-keele käsklused suurte tähtedega, muud väikestega, kuid iseenesest on SQL-andmebaasid tõstutundetud. Tabeli nimeks siin näites teated2. Edasi tulevad sulgudes komadega eraldatult tulpade nimed ja kirjeldused. Järgnevalt on tulpade nimedeks id, teade ja nimi. Tüüpideks vastavalt int, text ja text. NOT NULL esimese välja taga tähendab, et tulbas ei tohi olla tühiväärtusi; auto_increment aga, et andmete lisamisel tabelisse paigutatakse sinna lahtrisse automaatselt leitud unikaalne väärtus. PRIMARY KEY(id) loetelu lõpus näitab, et tulp nimega id on primaarvõtmeks ehk üldjuhul kui viidatakse selle tabeli reale, siis kasutatakse selleks primaarvõtme unikaalset väärtust.

```
CREATE TABLE teated2(
  id int NOT NULL auto_increment,
  teade TEXT,
  nimi TEXT,
  PRIMARY KEY(id)
);
```

Järgmisena andmete lisamise lause, mis peaks sellisel kujul kõikidele SQL-andmebaasidele arusaadav olema. INSERT INTO, millele järgneb tabeli nimi, sulgudes tulpade loetelu kuhu lisatakse, seejärel sõna VALUES ning edasi väärtuste loetelu. Tekstilised väärtused ülakomade vahel. Sõltuvalt andmebaasimootorist on sellel käsklusel mitmeid erikujusid mitme rea andmete korraga sisestamiseks või

tulpade nimede ja väärtuste lähemale kirjutamiseks, kui siintoodu peaks kõige üldisem ja töökindlam olema.

```
INSERT INTO teated2(teade, nimi) VALUES
  ('Kool hakkab kell 10', 'Mati');
```

MySQLi-spetsiifiline kirjeldus tabeli tutvustuse kuvamiseks. Ka teistel andmebaasidel leiab selliseid kirjeldavaid vahendeid, olgu siis tekstipõhiseid või graafilisi.

```
mysql> explain teated2;
+-------+---------+------+-----+---------+----------------+
| Field | Type    | Null | Key | Default | Extra          |
+-------+---------+------+-----+---------+----------------+
| id    | int(11) |      | PRI | NULL    | auto_increment |
| teade | text    | YES  |     | NULL    |                |
| nimi  | text    | YES  |     | NULL    |                |
+-------+---------+------+-----+---------+----------------+
3 rows in set (0.26 sec)
```

Kõige universaalsem pärngulause. Tärn tähendab, et näha soovitakse kõiki ridu. Tärni asemel võiks olla ka soovitavate tulpade loetelu.

```
mysql> select * from teated2;
+----+------------------------------+------+
| id | teade                        | nimi |
+----+------------------------------+------+
|  1 | Kool hakkab kell 10          | Mati |
|  2 | Võta vihikud kaasa           | Kati |
|  3 | Matemaatika vihik on kadunud | Mati |
|  4 | Otsi riiuli tagant           | Kati |
|  5 | Mina toon palli              | Siim |
|  6 | Mina ka                      | Mati |
|  7 | Jätke mu ilus kleit valgeks  | Kati |
+----+------------------------------+------+
7 rows in set (0.38 sec)
```

Kui tahta näha vaid erinevaid väärtusi, siis selle juures aitab käsklus distinct. Ehkki Mati on saatnud tunduvalt rohkem kui ühe teate, siis siin kuvatakse iga nimi ikkagi ainult ühe korra.

```
mysql> select distinct nimi from teated2;
+------+
| nimi |
+------+
| Mati |
| Kati |
| Siim |
+------+
3 rows in set (0.34 sec)
```

Kui soovitakse mõne tulba järgi järjestada, siis selleks võib lisada lauseosa order by ning soovitud tulba nime.

```
mysql> select * from teated2 order by nimi;
+----+------------------------------+------+
| id | teade                        | nimi |
+----+------------------------------+------+
|  2 | Võta vihikud kaasa           | Kati |
|  4 | Otsi riiuli tagant           | Kati |
|  7 | Jätke mu ilus kleit valgeks  | Kati |
|  1 | Kool hakkab kell 10          | Mati |
|  3 | Matemaatika vihik on kadunud | Mati |
|  6 | Mina ka                      | Mati |
|  5 | Mina toon palli              | Siim |
+----+------------------------------+------+
7 rows in set (0.07 sec)
```

Soovides vastava tulba järgi tagurpidises järjestuses tulemust näha, tuleb lisada võtmesõna desc. Soovides rõhutada päripidist järjestust, võib kirjutada sõna asc, kuid see kehtib ka vaikimisi.

```
mysql> select * from teated2 order by nimi desc;
+----+------------------------------+------+
| id | teade                        | nimi |
+----+------------------------------+------+
|  5 | Mina toon palli              | Siim |
|  1 | Kool hakkab kell 10          | Mati |
|  3 | Matemaatika vihik on kadunud | Mati |
|  6 | Mina ka                      | Mati |
|  2 | Võta vihikud kaasa           | Kati |
|  4 | Otsi riiuli tagant           | Kati |
|  7 | Jätke mu ilus kleit valgeks  | Kati |
+----+------------------------------+------+
7 rows in set (0.03 sec)
```

MySQL võimaldab juba SQL-lauses vastuste arvu piirata. Praegusel juhul seati suurimaks väljastatavate vastuste arvuks neli. Mõne andmebaasimootori korral on vastavaks piiravaks käskluseks top.

```
mysql> select * from teated2 limit 4;
+----+------------------------------+------+
| id | teade                        | nimi |
+----+------------------------------+------+
|  1 | Kool hakkab kell 10          | Mati |
|  2 | Võta vihikud kaasa           | Kati |
|  3 | Matemaatika vihik on kadunud | Mati |
|  4 | Otsi riiuli tagant           | Kati |
+----+------------------------------+------+
4 rows in set (0.14 sec)
```

Siin antakse teada, et soovitakse näha teateid alates kolmandast kaks tükki. Selline piirang on näiteks mugav lehtede loomisel, kus kõiki andmeid ei soovita korraga ühele lehele paigutada, vaid vastavalt kasutaja soovile näidatakse järgmisi lehekülgi.

```
mysql> select * from teated2 limit 3, 2;
+----+--------------------+------+
| id | teade              | nimi |
+----+--------------------+------+
|  4 | Otsi riiuli tagant | Kati |
|  5 | Mina toon palli    | Siim |
+----+--------------------+------+
2 rows in set (0.00 sec)
```

Tahtes konkreetse väärtuse järgi piirata väljastatavaid ridu, tuleb selline piirang kirjutada where-lausesse. Kui tingimusi on rohkem, siis ühendamiseks sobivad sõnad AND ning OR. Võrdlemiseks märgid < ja > nagu muudelgi puhkudel.

```
mysql> select * from teated2 where nimi='Kati';
+----+------------------------------+------+
| id | teade                        | nimi |
+----+------------------------------+------+
|  2 | Võta vihikud kaasa           | Kati |
|  4 | Otsi riiuli tagant           | Kati |
|  7 | Jätke mu ilus kleit valgeks  | Kati |
+----+------------------------------+------+
3 rows in set (0.00 sec)
```

Loendamiseks sobib käsklus count. Sõnapaar "as nr" avaldise count(*) taga tähendab, et loenduse tulemus väljastatakse tulbana, mil nimeks nr.

```
mysql> select count(*) as nr from teated2 where nimi='Kati';
+----+
```

```
| nr |
+----+
|  3 |
+----+
1 row in set (0.32 sec)
```

Ühe tabeliga seotud lihtsamad rakendused enamasti eeltoodud päringutega piirduvadki. Loendava statistika puhul aga teeb järgnev vahend elu mõnevõrra mugavamaks. Lisand "group by" võimaldab väljundisse jätta näidatud tulba väärtustest vaid erinevad. Samas mitme rea andmeid arvestavad funktsioonid nagu count, sum ja avg töötavad siis eraldi iga sellise grupi kohta ning nõnda võibki leida ühe käsuga iga inimese teadete arvu või kokku kulutatud summa.

```
mysql> select nimi, count(*) as kogus from teated2 group by nimi;
+------+-------+
| nimi | kogus |
+------+-------+
| Kati |     3 |
| Mati |     3 |
| Siim |     1 |
+------+-------+
3 rows in set (0.01 sec)
```

Nii nagu harilikel päringutel saab piiranguid seada WHERE-lausega nii gruppide jaoks on piiranguid tähistav sõna HAVING.

```
mysql> select nimi, count(*) as kogus from teated2 group by nimi having kogus>1;

+------+-------+
| nimi | kogus |
+------+-------+
| Kati |     3 |
| Mati |     3 |
+------+-------+
2 rows in set (0.01 sec)
```

Ja saidki ühe tabeliga katsetused ühele poole.

```
mysql>
```

Kaks tabelit

Andmebaaside puhul peetakse tähtsaks iga sisestatud väärtust hoida vaid ühe eksemplarina. Et iga uue kauba ostmisel ei peaks uuesti kirja panema kliendi aadressi või konto numbrit. Kui on vaja sisestust kontrollida, siis kasutatakse selleks mõnd kontrollsummat või muud piirajat, kuid ideaaljuhul samu andmeid andmebaasis mitmes kohas ei hoita. Sama lugu nagu koodilõikude puhul: mis kord tehtud, seda uuesti kirjutada pole hea.

Tabelid seotakse omavahel üldjuhul täisarvudega. Kui allpool loodi tabelid toitude ja jooksjate tarvis ning jooksjate tabelis olev tulp lemmiktoidu_id näitab toidutabeli vastava ID-numbriga toidule, siis toitude tabeli ID-tulpa nimetatakse primaarvõtmeks ning tulba lemmiktoidu_id väärtusi võõrvõtmeks.

```
CREATE TABLE jooksjad (ID int NOT NULL AUTO_INCREMENT, eesnimi varchar(30),
lemmiktoidu_id int, PRIMARY KEY(ID));

mysql> CREATE TABLE toidud(ID int NOT NULL AUTO_INCREMENT, nimetus varchar(30),
PRIMARY KEY(ID));
```

Tabelitesse mõned väärtused, et oleks pärast mille peal katsetada. Esimesele toidule pannakse automaatselt järjekorranumbriks 1

```
mysql> insert into toidud(nimetus) values ('Hernesupp');
Query OK, 1 row affected (0.10 sec)
```

Ka jooksja id-number pannakse automaatselt. Juku lemmiktoidu number tuleb aga määrata.

```
mysql> INSERT INTO jooksjad(eesnimi, lemmiktoidu_id) values ('Juku', 1);
Query OK, 1 row affected (0.00 sec)
```

Väljatrükk näitamaks, milliste andmetega edaspidi katsetatakse. Matile on jäetud lemmiktoit määramata ning selle välja väärtuseks on NULL.

```
mysql> select * from jooksjad;
+----+---------+----------------+
| ID | eesnimi | lemmiktoidu_id |
+----+---------+----------------+
|  1 | Juku    |              1 |
|  2 | Kati    |              1 |
|  3 | Mati    |           NULL |
+----+---------+----------------+
3 rows in set (0.01 sec)

mysql> SELECT * FROM toidud;
+----+------------+
| ID | nimetus    |
+----+------------+
|  1 | Hernesupp  |
|  2 | Kapsasupp  |
|  3 | Pannkoogid |
+----+------------+
3 rows in set (0.01 sec)
```

Kõige tavalisem päring paigutamaks ühte tabelisse nii jooksjad kui nende lemmiktoidud. WHERE-osa puudumisel antaks välja kõikvõimalikud kahe tabeli ridade omavahelised kombinatsioonid. Praegusel juhul 3*3 ehk üheksa rida. WHERE seab aga piirangu ning välja näidatakse vaid kaks - need, kus jooksja lemmiktoidu_id vastab toitude tabelis leiduvale ID-veeru väärtusele. Et Mati juures olevat NULL-väärtust toitude tabeli ID-väärtuste hulgas pole, siis jääb Mati ka nimekirja kuvamata.

```
mysql> SELECT * FROM jooksjad, toidud WHERE jooksjad.lemmiktoidu_id=toidud.ID;

+----+---------+----------------+----+-----------+
| ID | eesnimi | lemmiktoidu_id | ID | nimetus   |
+----+---------+----------------+----+-----------+
|  1 | Juku    |              1 |  1 | Hernesupp |
|  2 | Kati    |              1 |  1 | Hernesupp |
+----+---------+----------------+----+-----------+
2 rows in set (0.06 sec)
```

Kui soovitakse kõiki ühe tabeli väärtusi näha ning paremale poole lisada mittetühjadena vaid need väärtused, mida võõrvõtme kaudu võimalik leida on, siis aitab tabeleid ühendada LEFT JOIN. ON-lauseosas tuleb siis määrata, millised tulbad omavahel seotud on. Suuremate rakenduste korral võib nõnda kokku ühendada tunduvalt rohkem kui kaks tabelit. Juhul, kui näiteks soovitakse uurida, millised sobivas vanuses inimesed töötavad ettevõttes, mille leidub filiaal ka Pärnus.

```
mysql> SELECT * FROM jooksjad LEFT JOIN toidud ON toidud.id=jooksjad.lemmiktoidu_id;
+----+---------+----------------+------+-----------+
| ID | eesnimi | lemmiktoidu_id | ID   | nimetus   |
+----+---------+----------------+------+-----------+
```

```
|  1 | Juku    |              1 |    1 | Hernesupp |
|  2 | Kati    |              1 |    1 | Hernesupp |
|  3 | Mati    |           NULL | NULL | NULL      |
+----+---------+----------------+------+-----------+
3 rows in set (0.00 sec)
```

Nii nagu failinimede küsimisel aitasid elu hõlpsamaks teha tärn ning küsimärk, nii saab MySQLi puhul kasutada LIKE-võrdluses protsenti ning alljoont.

```
mysql> select * from jooksjad where eesnimi like 'J%';
+----+---------+----------------+
| ID | eesnimi | lemmiktoidu_id |
+----+---------+----------------+
|  1 | Juku    |              1 |
+----+---------+----------------+
1 row in set (0.01 sec)



mysql> select * from jooksjad where eesnimi like 'J_ku';
+----+---------+----------------+
| ID | eesnimi | lemmiktoidu_id |
+----+---------+----------------+
|  1 | Juku    |              1 |
+----+---------+----------------+
1 row in set (0.00 sec)

mysql> select * from jooksjad where eesnimi like 'J_k';
Empty set (0.01 sec)
```

SQL-keelest võib leida ka komplekti muudeski keeltes kasutatavaid funktsioone. Olgu siis arvutamise või tekstitöötluse tarbeks.

```
mysql> select eesnimi, length(eesnimi) from jooksjad;
+---------+------------------+
| eesnimi | length(eesnimi) |
+---------+------------------+
| Juku    |                4 |
| Kati    |                4 |
| Mati    |                4 |
+---------+------------------+
3 rows in set (0.08 sec)

mysql> select eesnimi, left(eesnimi, 1) from jooksjad;
+---------+-------------------+
| eesnimi | left(eesnimi, 1) |
+---------+-------------------+
| Juku    | J                 |
| Kati    | K                 |
| Mati    | M                 |
+---------+-------------------+
3 rows in set (0.07 sec)
```

Lauluandmetega rakendus

Järgnevalt kinnistatakse eelpool kirja pandud tarkused väikese programmi abil. Andmed on esiotsa ühes ja pärast kolmes tabelis. Ning väljund saadetakse nii tekstiekraanile kui veebilehtedele.

Alustame võimalikult lihtsast väljamõeldud olukorrast, kus soovitakse meeles pidada laulude pealkirju ning laulude esitajaid. Viiekümnest tähest kummagi välja salvestamisel võiks piisata. Nagu tavaks, lisatakse igale tabelireale ka võtmeväli, et oleks hiljem kindlasti võimalik kontreetsetele ridadele viidata.

```
CREATE TABLE laulud (
  id int(11) NOT NULL auto_increment,
  pealkiri varchar(50) default NULL,
  esitaja varchar(50) default NULL,
```

```
  PRIMARY KEY  (id)
)
```

Mõned väljamõeldud andmed sisse

```
INSERT INTO laulud VALUES (1,'Valged Roosid','Joala');
INSERT INTO laulud VALUES (2,'Kuldkannike','Joala');
INSERT INTO laulud VALUES (3,'Mererannal','Linna');
INSERT INTO laulud VALUES (4,'Kungla Rahvas','Veskimaja');
INSERT INTO laulud VALUES (5,'Koolisellid','Tammik');
```

ning võibki tulemust imetleda.

```
mysql> select * from laulud;
+----+---------------+-----------+
| id | pealkiri      | esitaja   |
+----+---------------+-----------+
|  1 | Valged Roosid | Joala     |
|  2 | Kuldkannike   | Joala     |
|  3 | Mererannal    | Linna     |
|  4 | Kungla Rahvas | Veskimaja |
|  5 | Koolisellid   | Tammik    |
+----+---------------+-----------+
5 rows in set (0.30 sec)
```

Eeltoodud näidete põhjal saab andmeid väljastava käsureaprogrammi kokku küllalt lihtsalt – juhul kui tarvilikud ühendused on valmis seatud.

```
import java.sql.*;

public class Laulud1{
 public static void main(String argumendid[]) throws Exception{
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection(
            "jdbc:odbc:esimene", "", "");
  Statement st=cn.createStatement();
  String lause="SELECT pealkiri, esitaja FROM laulud";
  ResultSet rs=st.executeQuery(lause);
  while(rs.next()){
    System.out.println(rs.getString("pealkiri")+
                              " "+rs.getString("esitaja"));
  }
  cn.close();
 }
}
```

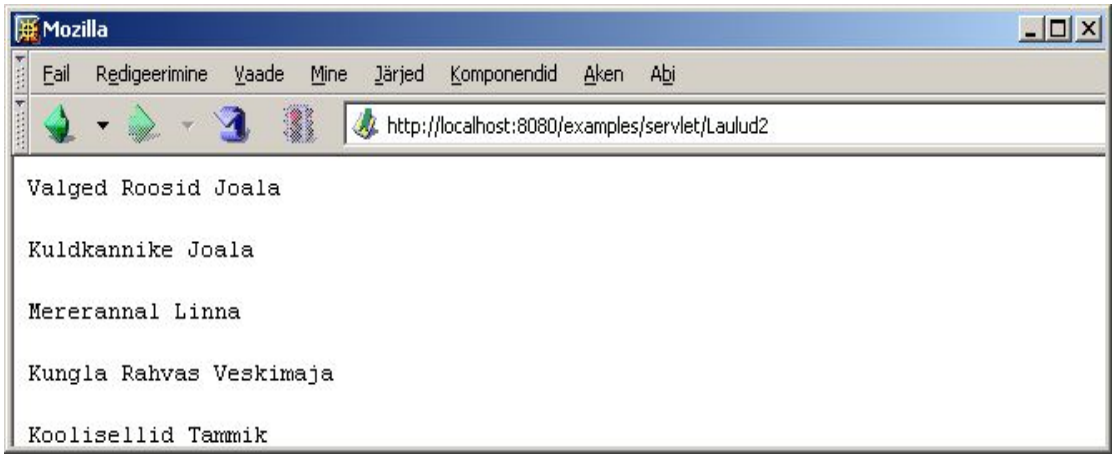
Kui programm tööle panna, võib ka väljundit näha.

```
C:\temp>Java Laulud1
Valged Roosid Joala
Kuldkannike Joala
Mererannal Linna
Kungla Rahvas Veskimaja
Koolisellid Tammik
```

Kui andmeid rohkem või kliendid üle võrgu kaugemal, siis muudab servletiväljund andmed kergemini kättesaadavaks. Kui veebiserver jookseb avalikult kättesaadavas masinas, siis piisab kasutajal teada vaid rakenduse aadressi ning võibki omale vajaliku teabe ekraanilt ammutada.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class Laulud2 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
    {
      vastus.setContentType("text/plain");
```

```
      PrintWriter valja = vastus.getWriter();
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
        Statement st=cn.createStatement();
        String lause="SELECT pealkiri, esitaja FROM laulud";
        ResultSet rs=st.executeQuery(lause);
        while(rs.next()){
          valja.println(rs.getString("pealkiri")+
                    " "+rs.getString("esitaja")+"\n");
        }
        cn.close();
      }catch(Exception viga){
         viga.printStackTrace(valja);
      }
    }
}
```



Standardile vastav lehekülg.

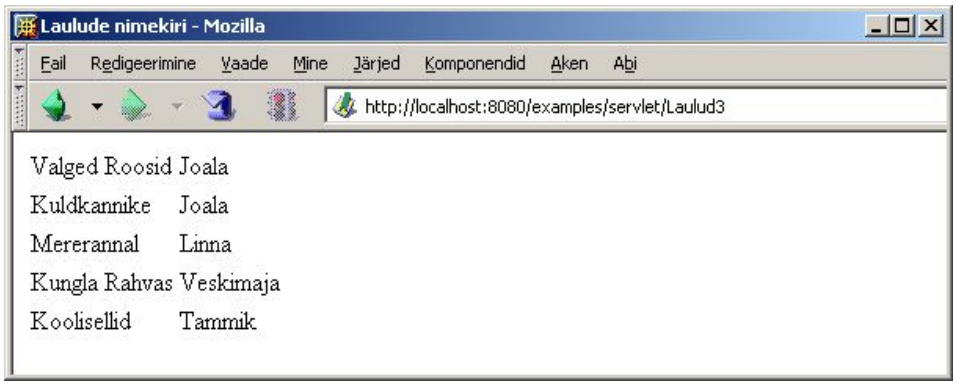
HTMLi keel on aastate jooksul arenenud. Algsest kümnekonna käsuga tekstiillustreerimisvahendist kasvas välja sajakonna käsuga kujundusvahend. Seiluritootjad on omalt poolt võimalusi lisanud ning aegapidi on neid ka standardisse võetud. W3-konsortsium on taoline firmade ja muude asutuste ühendus, mille kaudu lepitakse kokku veebiga seotud standardeid. Nõnda on rohkem lootust, et kusagil koostatud leheküljed või muud failid ka mujal kasutatavad on.

Standard pannakse kirja kas tekstilise kirjeldusena, tabelina, XML-failide puhul ka DTD või Schema abil. Programmeerija loeb kirjeldust ja püüab tulemuse selle järgi sättida, kuid iga inimene tahab ja vajab tagasisidet, et kas tema koostatu ka loodetud ootustele vastab. Kui Pascali, C või Java koodi kirjutada, siis teatab kompilaator süntaksivead julgesti välja. Veebilehte avades aga on seilur tagasihoidlikum ning väikesed näpuvead jäävad enamasti märkamata. Mõnikord võib aimamine nii ilusti välja tulla, et lehte vaadates ei leia mingit märki HTML-koodi trükiveast. Teises seiluris võib aga aimamise algoritm muud moodi käituda ning lehe sisu võib imelikult paista või sootuks märkamatuks jääda. Taoliste viperuste vältimiseks saab kasutada HTMLi validaatorit - programmi kontrollimaks HTMLi õigekirja. Siin uuritakse, et elementide nimed oleks õigesti kirjutatud, lõpetamist vajavad elemendid lõpetatud ning et elemendid paikneksid ka üksteise sees lubatud kujul.

Et validaator teaks millise standardi järgi kontrollida, peab vastav rida ka faili alguses kirjas olema. Lisaks on päises nõutud ka pealkirja ja kooditabeli märkimine.

Edasi mõistab validaatorprogramm ülal kirjeldatud standardi järgi lehte kontrollima hakata.

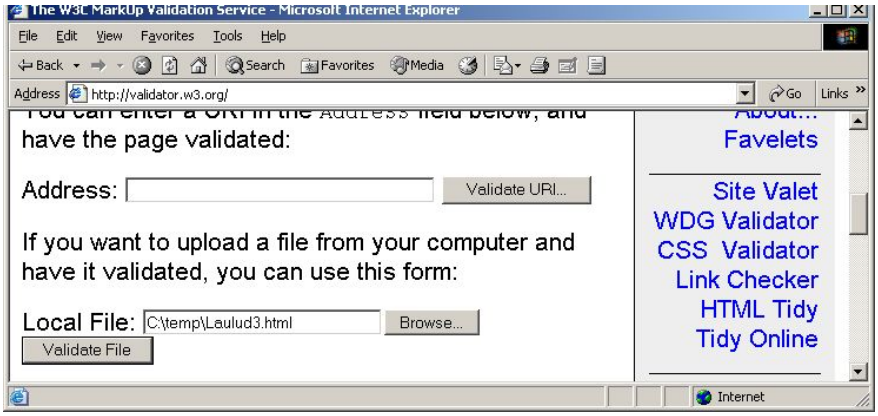
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class Laulud3 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
    {
      vastus.setContentType("text/html");
      PrintWriter valja = vastus.getWriter();
      valja.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\"
"+
          "\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">");
      valja.println("<html><head>");
      valja.println("<meta http-equiv=\"Content-Type\" content=\"text/html;
charset=ISO-8859-1\" />");
      valja.println("<title>Laulude nimekiri</title></head>\n");
      valja.println("<body>");
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
        Statement st=cn.createStatement();
        String lause="SELECT pealkiri, esitaja FROM laulud";
        ResultSet rs=st.executeQuery(lause);
        valja.println("<table>");
        while(rs.next()){
         valja.println("<tr><td>"+rs.getString("pealkiri")+
          "</td><td>"+rs.getString("esitaja")+"</td></tr>");
        }
        valja.println("</table>");
        cn.close();
        valja.println("</body>");
        valja.println("</html>");

      }catch(Exception viga){
         viga.printStackTrace(valja);
      }
    }
}
```



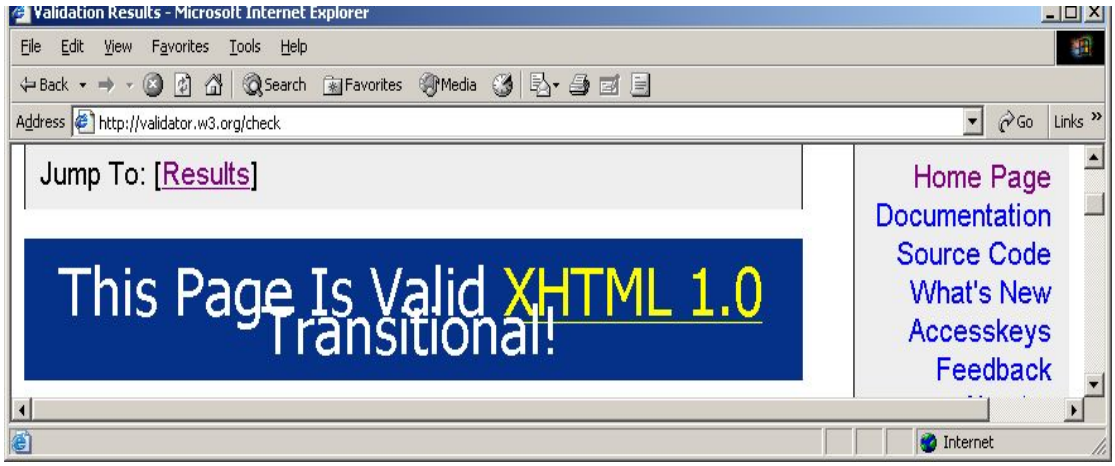

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<title>Laulude nimekiri</title></head>

<body>
<table>
<tr><td>Valged Roosid</td><td>Joala</td></tr>
<tr><td>Kuldkannike</td><td>Joala</td></tr>
<tr><td>Mererannal</td><td>Linna</td></tr>
<tr><td>Kungla Rahvas</td><td>Veskimaja</td></tr>
<tr><td>Koolisellid</td><td>Tammik</td></tr>
```

```
</table>
</body>
</html>
```

Kontrollija leiab aadressilt http://validator.w3.org/ Seal võimalik kontrollimiseks pakkuda nii veebiaadress kui oma kettal leiduv fail. Esimest võimalust on mugav kasutada, kui koostatav leht avalikult veebi kaudu kättesaadav. Kui aga kohalik veebiserver otse avalikku võrku ei paista, siis saab servleti loodud HTML-faili kohalikku masinasse salvestada ning siis võrku üles laadida.



Leht kontrollijasse loetud, antakse sealt vastus. Kui päiserida puudu või vigane või mõni tähtsam element puudu, siis antakse vastav teade. Suudab aga validaator lehe struktuurist aru saada, siis vaadatakse kõik kujunduselemendid ükshaaval üle ning kui kusagil midagi kahtlast, siis antakse teada. Mitmete teineteise sees paiknevate tabelite või taanete korral oleks käsitsi käskude ja nende paiknemise õigsuse kontroll küllaltki vaevaline. Validaator aga leiab kohe üles kui mõni märk vales kohas, üle või puudu. Siis ei jää muud midagi üle, kui veateadet uurida, leida, et mille poolest siis loodud HTML kahtlane on. Koodis püüda koht parandada ning uuesti proovida. Kui pilt tundub väga segane ning kuidagi ei oska veateate asukohta leida, siis aitab jupikaupa uurimine. Et algul tõsta uude tühja faili vaid päise ja tühja body-osaga leht ning kontrollida selle korrektsust. Siis tasapisi lisada/kopeerida sisemisi elemente ja iga sammu järel kehtivust kontrollida. Nii on kohe selge, millise sammu juures raskus tekkis ning võimalik seda sammu lähemalt uurida. Vajadusel osadeks jagada ning uuesti uurida. Mõnikod võib ka juhtuda, et pealtnäha oleks justkui kõik korras, aga sellegipoolest näidatakse, et ühes kindlas kohas on midagi viltu. Sellisel juhul võib põhjuseks olla miski klahvikombinatsiooni tulemusena tekkinud salapärane sümbol, millest siis kustutamise ja uuesti kirjutamise abil võitu saab. Ning kui lõpuks õnnestub validaatorilt välja meelitada teade lehekülje korrektsuse kohta, siis võib tulemusega rahule jääda. Ehkki tasuks korrektsust kontrollida ka mitmesuguste andmete põhjal sama servleti loodud lehtede puhul.



Sortimine

Vähegi pikemate loetelude puhul võib sobiva väärtuse leidmine päris tülikaks osutuda. Find-käsu kõrval on sobivaks vahendiks järjestamine. Andmebaasi põhjal toimivate rakenduste eeliseks on võimalus paari sõna abil määrata väljastatavad andmed soovitavasse järjekorda. Suuremaks nuputamiseks on, et mille põhjal seda järjestust määrata. Lihtsam tundub olema jagada ülesanne kaheks osaks. Ühelt poolt valmistada ette lehekülg, mis väljastaks andmed etteantud parameetritele vastavas järjekorras. Ning teiselt poolt hoolitseda, et kasutajal oleks võimalikult mugav järjestust määrata. Esimeses lähenduses võib andmeid väljastavat lehte vaadelda kui musta kasti. Servletile antakse andmeid ette nagu veebilehtede puhul ikka - aadressirea parameetrite kaudu. Määran, et parameetri nimeks oleks "sorttulp" ning väärtuseks tulba nimi, mille järgi soovitatakse sortida. Seega siis, kui lehe aadressiks oleks

```
http://masinanimi/kataloog/servletinimi?sorttulp=esitaja
```

siis tuleksid andmed lehele järjestatuna tähestikulises järjekorras esitaja järgi. Kui aga

```
http://masinanimi/kataloog/servletinimi?sorttulp=pealkiri
```

siis tuleksid tähestiku algupoolel asuvad pealkirjad eespool nähtavale.

SQL-keeles määratakse sorteerimist määrav tulp lauseosa ORDER BY järgi. Lihtsaim lahendus oleks kasutaja poolt tulnud tulba nimi otse sellesse lausesse paigutada ning tulemus välja näidata. Et aga veebirakenduste puhul soovitakse kõiki veebi poolt tulevaid andmeid umbusaldada, siis saabuvat parameetrit SQL-lausesse ei kirjutata. Esiteks tekiks probleem pahatahtliku sisestuse korral, mis võiks sobivalt seatuna muudele andmetele liiga teha või neid välja näidata. Teiseks mureks on aga, et vigase sisestuse korral tuleks nähtavale süsteemi loodud veateade või sootuks mitte midagi. Viisakam oleks aga kasutajale sõnaliselt teada anda, mis lahti on.

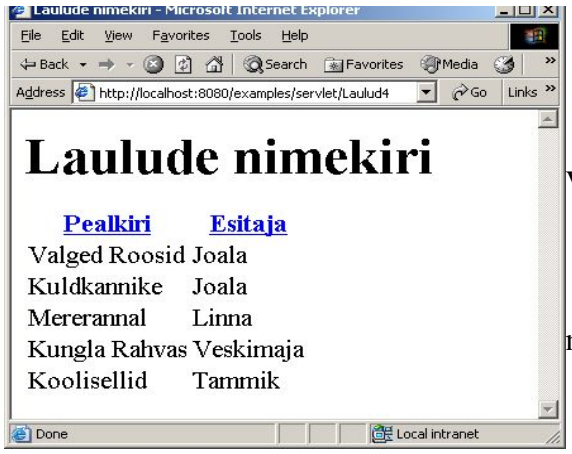
Siin näites on sorteerimist määrava tulba nimi küsitud muutujasse sort. Vaikimisi väärtuseks on "id". On aga parameetriks pealkiri või esitaja, siis määratakse see sorteerimist seadvaks tulbaks. Nõnda pole võimalust, et kasutaja saadetud suvaline sisestus andmeid võiks rikkuda või kahtlase veateate ekraanile manada, vaid igasuguse vigase sisestuse korral näidatakse ekraanile read sorteerituna id järgi. Kui sobiva tulba nimi lauses olemas, siis võib andmed ekraanile näidata nagu eelmiselgi korral.

Teiseks tuleb miskil kasutajale mugavamal ja vastuvõetavamal moel anda võimalus sorteerimisjärjekord valida. Ning tulba nime aadressireal sissetippimine ei pruugi selleks mitte olla. Siinne

```
"<th><a href='"+kysimus.getRequestURI()+"?sorttulp=pealkiri'>Pealkiri</a></th>"+
```

näitab, kuidas võimalik viide nõnda koostada, et see samale lehele näitaks ning sorteerimiseks sobiva tulba nimi andmetena kaasa antaks. HttpServletRequesti käsklus getRequestURI annab tekstina välja jooksva lehe URLi alates serveri juurkataloogist, piisab juurde lisada vaid sobivad parameetrid. Teise võimalusena saaks jooksva kataloogi puhul kirjutada ka vaid failinime. Kui iga tulba pealkirjal taoline viide küljes, jääbki kasutajale võimalus, et igale pealkirjale vajutades sorteeritakse andmed vastava tulba järgi.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class Laulud4 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
    {
      vastus.setContentType("text/html");
      PrintWriter valja = vastus.getWriter();
      valja.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\"
"+
          "\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">");
      valja.println("<html><head>");
      valja.println("<meta http-equiv=\"Content-Type\" content=\"text/html;
charset=ISO-8859-1\" />");
      valja.println("<title>Laulude nimekiri</title></head>\n");
      valja.println("<body><h1>Laulude nimekiri</h1>");
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
        Statement st=cn.createStatement();
        String sort="id";
        String sortTulp=kysimus.getParameter("sorttulp");
        if(sortTulp==null){sortTulp="id";}
        if(sortTulp.equals("pealkiri")){sort="pealkiri";}
        if(sortTulp.equals("esitaja")){sort="esitaja";}
        String lause="SELECT pealkiri, esitaja FROM laulud ORDER BY "+sort;
        ResultSet rs=st.executeQuery(lause);
        valja.println("<table>");
        valja.println("<tr>"+
           "<th><a href='"+kysimus.getRequestURI()+
           "?sorttulp=pealkiri'>Pealkiri</a></th>"+
           "<th><a href='"+kysimus.getRequestURI()+
           "?sorttulp=esitaja'>Esitaja</a></th>"+
           "</tr>");
        while(rs.next()){
         valja.println("<tr><td>"+rs.getString("pealkiri")+
          "</td><td>"+rs.getString("esitaja")+"</td></tr>");
        }
        valja.println("</table>");
        cn.close();
        valja.println("</body>");
        valja.println("</html>");

      }catch(Exception viga){
         viga.printStackTrace(valja);
      }
    }
}
```



Vaikimisi järjestus id järgi ning määratud järjestus pealkirja järgi.

Andmete lisamine

Lihtsa otsinguvahendi puhul võib baasi veebiliides piirduda andmete välja näitamisega. Muutmised-lisamised võetakse ette andmebaasi enese vahenditega, või on selle tarvis hoopis omaette rakendus loodud. Turvalisuse mõttes on veebist vaid vaatamist lubav rakendus lihtsam - pahalastel pole kuigivõrd põhjust ega võimalust andmeid ohtu seada. Ohtudeks jäävad vaid serveri või võrguühenduse võimsust ületav päringute hulk või logifailide abil ketta täiskirjutamise oht. Kui ka veebi kaudu liituval kasutajanimel pole andmebaasis muutmise õigusi, siis võib taolisel rakendusel suhteliselt mureta toimida lasta.

Nõudmiste kasvades aga vaid vaatamiseks mõeldud rakendusest ei piisa. Andmete veebikaudseks lisamiseks on võimalused täiesti olemas, lihtsalt peab arvestama pahatahtlike kasutajate tekitatud segaduste ohuga.

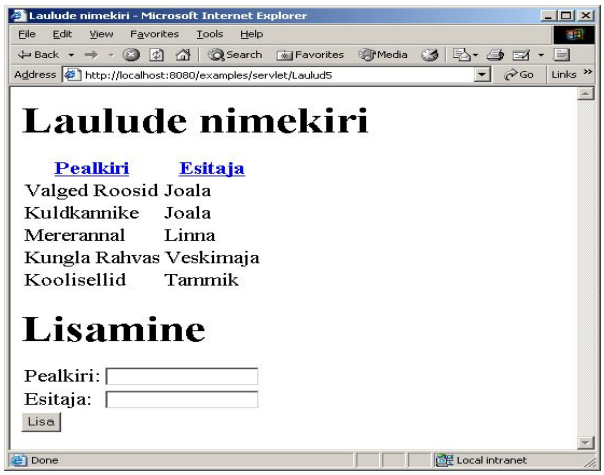
Andmete lisamiseks tuleb need kasutajalt kõigepealt kätte saada. Praegusel juhul on selleks otstarbeks tarvitatud tekstivälju. Graafikakomponendid jällegi vormi

sees. Atribuut action='#' tähendab jällegi, et andmete vastuvõtjaks on sama leht uuel avamisel. Võrreldes muude näidetega, on siin ka submit-nupule nimi antud. Vormi andmete teele saatmisel pannakse kaasa nimega komponentide andmed, subm it-nupu puhul vajutatud nupu andmed. Nõnda on võimalik kontrollida, kas nuppu vajutati. Ning mitme nupu puhul kontrollida, et millist nuppu vajutati.

Siinse lehe avamisel kontrollitakse, kas parameeter "lisamine" pole null. Et kas vajutati lisamisnuppu. Vaid sel juhul koostatakse SQL-lause andmete lisamiseks ning käivitatakse. PreparedStatement'i eelis tavalise käskluse ees on, et kasutaja saadetud erisümbolid ei saa serveris segadust tekitada, vaid need kirjutatakse samasuguse rahuga edasi andmebaasi tabelisse. Muu andmete näitamise osa sarnane kui eespool.

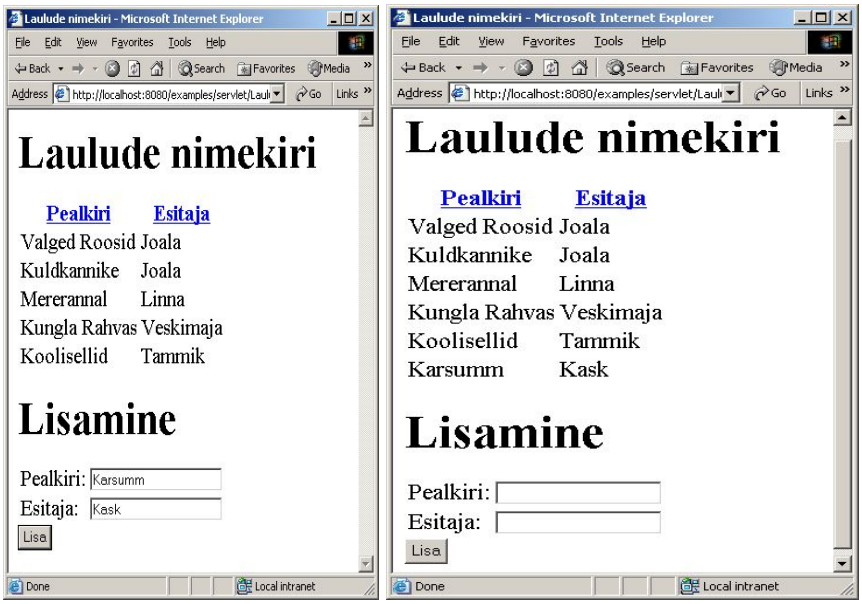
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class Laulud5 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
    {
      vastus.setContentType("text/html");
      PrintWriter valja = vastus.getWriter();
      valja.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\" "+
          "\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">");
      valja.println("<html><head>");
      valja.println("<meta http-equiv=\"Content-Type\" "+
                            "content=\"text/html; charset=ISO-8859-1\" />");
      valja.println("<title>Laulude nimekiri</title></head>\n");
      valja.println("<body><h1>Laulude nimekiri</h1>");
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
        Statement st=cn.createStatement();
        if(kysimus.getParameter("lisamine")!=null){
           PreparedStatement ps=cn.prepareStatement(
             "INSERT INTO laulud (pealkiri, esitaja) VALUES (?, ?)"
           );
           ps.setString(1, kysimus.getParameter("pealkiri"));
           ps.setString(2, kysimus.getParameter("esitaja"));
           ps.executeUpdate();
        }
        String sort="id";
        String sortTulp=kysimus.getParameter("sorttulp");
        if(sortTulp==null){sortTulp="id";}
        if(sortTulp.equals("pealkiri")){sort="pealkiri";}
        if(sortTulp.equals("esitaja")){sort="esitaja";}
        String lause="SELECT pealkiri, esitaja FROM laulud ORDER BY "+sort;
        ResultSet rs=st.executeQuery(lause);
        valja.println("<table>");
        valja.println("<tr>"+
           "<th><a href='"+kysimus.getRequestURI()+
           "?sorttulp=pealkiri'>Pealkiri</a></th>"+
           "<th><a href='"+kysimus.getRequestURI()+
           "?sorttulp=esitaja'>Esitaja</a></th>"+
           "</tr>");
        while(rs.next()){
         valja.println("<tr><td>"+
          Abi.filtreeriHTML(rs.getString("pealkiri"))+
          "</td><td>"+
          Abi.filtreeriHTML(rs.getString("esitaja"))+
          "</td></tr>");
        }
        valja.println("</table>");
        valja.println("<h1>Lisamine</h1>");
        valja.println(
          "<form action='#'><table>"+
          "<tr><td>Pealkiri:</td><td>"+
           "<input type='text' name='pealkiri' /></td></tr>\n"+
          "<tr><td>Esitaja:</td><td>"+
           "<input type='text' name='esitaja' /></td></tr>\n"+
          "</table>"+
          "<input type='submit' name='lisamine' value='Lisa' />"+
          "</form>"
        );
        cn.close();
```

```
            valja.println("</body>");
            valja.println("</html>");

          }catch(Exception viga){
             viga.printStackTrace(valja);
          }
       }
}
```



Lisamisleht

- Algul
- Väärtuste sisestamine
- Pikenenud loetelu



Mitme tabeliga rakendus

Märgatav osa lihtsatest veebirakendustest võibki ühe tabeliga piirduda. Harilikuks andmete lisamiseks ning välja näitamiseks piisab sellest sageli. Kui mõista sobivalt pealkirju valida ning mõnikord ka andmete põhjal tulemusi arvutada, siis võib ühe tabeliga näite edukalt kokku panna nii külalisraamatu, tunniplaani, sünnipäevade hoidla kui muudki. Keerukamate andmete puhul ei pruugi aga kõige ühes tabelis hoidmine kuigi ökonoomne olla.

Andmebaasiteoreetikud soovitavad, et võimaluse korral tuleks andmed sättida nii, et midagi korduvalt ei hoitaks. Andmete korrektsuse jaoks kasutatagu pigem kontrollsummasid, kui mingi väärtus aga kusagil kirjas, siis sama asja poleks

võimaluse korral mõistlik enam kusagile mujale kirjutada. Ehk sama põhimõte nagu koodigi kirjutamise juures: mis kord tehtud, seda püüa ka tulevikus kasutada, mitte sama asja uuesti kirja panema hakata.

Liiatigi võib vaja minna küllalt erinevaid andmeid. Näiteks inimeste ja bussiliinide andmeid oleks teoreetiliselt võimalik hoida samas tabelis, kuid üldjuhul ei tundu see loogiline. Ehkki taolist andmete segamist vahel kasutatakse, ei tohiks see mitte harilike rakenduste puhul tavaks saada. Pigem tulevad kindla semantikata tabelitulbad ette rakendustes, kus programmeerija mõtleb objektide tasandil ning andmete talletamise ja väljalugemise eest otsustab eraldi koostatud vahelüli. Siin aga püüame tabeli andmed mõistetavad hoida ning nende kohale üheselt mõistetava veebiliidese koostada.

Lisaks laulude andmetele koostame tabelid heliplaatide tarbeks: väljadeks plaadi nimi ning väljalaskeaasta.

```
CREATE TABLE plaadid (
  id int(11) NOT NULL auto_increment,
  plaadinimi varchar(30) default NULL,
  aasta int(11) default NULL,
  PRIMARY KEY  (id)
);
```

Ning mõned andmed ka sisse.

```
INSERT INTO plaadid VALUES (1,'Tantsulood',1985);
INSERT INTO plaadid VALUES (2,'Laululood',1988);
```

Samuti loome juurde tabeli, mille abil määratakse, millised lood millise plaadi juurde kuuluvad. Seosetabelisse otseseid tekste ei kirjutatagi. Tabelis on kolm tulpa: id, plaadi_id ning laulu_id. Iga taoline rida märgib üht seost, kus märgitakse, et vastava reanumbriga laul kõlab märgitud numbriga plaadil.

Sellist teise tabeli reale viitava lahtri väärtust nimetatakse võõrvõtmeks (foreign key). MySQL neljandas tavaversioonis veel ei kontrolli võõrvõtmete viidete korrektsust, see jäetakse programmeerija hooleks. Mõne tabeliga ja paarisaja andmereaga rakenduse puhul saab sellise järje pidamisega täiesti hakkama. Kui aga andmete hulk ja keerukus märkimisväärselt kasvab, siis aitab andmebaasi võime viidete korrektsust kontrollida süsteemi töökindlust tagada ning murdunud ja vigastest viidetest hoiduda.

```
mysql> CREATE TABLE laulud_plaadid(id INT NOT NULL auto_increment PRIMARY KEY,
laulu_id INT, FOREIGN KEY lauluvoti (laulu_id) REFERENCES laulud (id), plaadi_id
INT, FOREIGN KEY plaadivoti(plaadi_id) REFERENCES plaadid(id));
```

Kui võõrvõtme loomist käsuna eraldi mitte kirja panna, siis tegelikult on tegemist lihtsa kolme tulbaga tabeliga.

```
CREATE TABLE laulud_plaadid (
  id int(11) NOT NULL auto_increment,
  laulu_id int(11) default NULL,
  plaadi_id int(11) default NULL,
  PRIMARY KEY  (id)
)
```

Järgnevalt näide, kuidas võivad andmed tabelites paikneda. Kõigepealt laulude tabel:

```
mysql> select * from laulud;
+----+---------------+-----------+
| id | pealkiri      | esitaja   |
+----+---------------+-----------+
|  1 | Valged Roosid | Joala     |
|  2 | Kuldkannike   | Joala     |
|  3 | Mererannal    | Linna     |
|  4 | Kungla Rahvas | Veskimaja |
|  5 | Koolisellid   | Tammik    |
|  6 | Karsumm       | Kask      |
+----+---------------+-----------+
6 rows in set (0.03 sec)
```

Siis plaadid:

```
mysql> select * from plaadid;
+----+------------+-------+
| id | plaadinimi | aasta |
+----+------------+-------+
|  1 | Tantsulood |  1985 |
|  2 | Laululood  |  1988 |
+----+------------+-------+
2 rows in set (0.05 sec)
```

Ning edasi seosetabel, milliseid laule millistelt plaatidelt leida võib. Näiteks viimasest reast annab välja lugeda, et seos id-numbriga 3 teatab, et laulu number 4 ehk “Kungla rahvas” võib kuulda plaadilt number 1 ehk “Tantsulood”.

```
mysql> select * from laulud_plaadid;
+----+----------+-----------+
| id | laulu_id | plaadi_id |
+----+----------+-----------+
|  1 |        1 |         1 |
|  6 |        2 |         2 |
|  3 |        4 |         1 |
+----+----------+-----------+
3 rows in set (0.06 sec)
```

Hoolimata sellest, et tabelitesse laiali jagatuna võib andmete paiknemine keerukas tunduda, annab SQL-lausetega väärtused küllalt sobival kujul välja küsida. Tabeleid ühendavaid lauseid annab mitmel kujul kirja panna, üks neist näha järgnevalt; seletus:

Küsimiseseks SELECT-lause nagu ikka. Järgneb näha soovitavate tulpade loetelu. Mõnikord võivad tulpade nimed eraldi tabelitest kattuda. Sellisel juhul tuleb määrata tulba nimi koos tabeli nimega - näiteks kujul plaadid.id . Edasi järgneb FROM ning kasutatavate tabelite loetelu. Edasi tuleb seada, milliste tabelite millised tulbad omavahel seotud on. Kuna praegu tabeli laulud_plaadid laulu_id näitab laulude tabeli id-veerule ning laulud_plaadid plaadi_id näitab plaatide tabeli id-veerule, siis see tuleb siin ka kirja panna.

```
mysql> SELECT pealkiri, plaadinimi FROM laulud, laulud_plaadid, plaadid WHERE
laulud.id=laulud_plaadid.laulu_id AND laulud_plaadid.plaadi_id=plaadid.id;
```

Päringu tulemuseks nagu ikka - andmed tabeli kujul.

```
+---------------+------------+
| pealkiri      | plaadinimi |
+---------------+------------+
| Valged Roosid | Tantsulood |
| Kungla Rahvas | Tantsulood |
| Kuldkannike   | Laululood  |
+---------------+------------+
3 rows in set (0.03 sec)
```

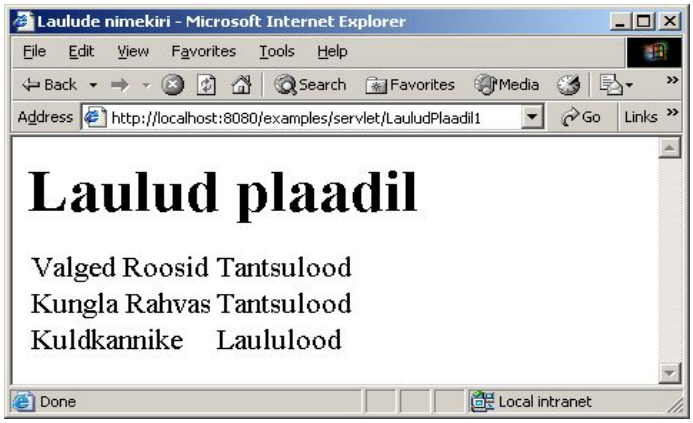
Edasi tuleb vaid servlet päringule ümber kirjutada ning tulemus lehele välja näidata. Väärtusi saab küsida sarnaselt tulba nime järgi nagu ühestki tabelist andmeid korjava päringu korral.

```
       valja.println("<tr><td>"+rs.getString("pealkiri")+
            "</td><td>"+rs.getString("plaadinimi")+"</td></tr>");
```

Teiseks võimaluseks oleks väärtuste küsimine tulba järjekorranumbri järgi. Eriti on sellest kasu juhul, kui tabelite liitmise tulemusena võivad loetellu sattuda samanimelised tulbad. Samuti tasuks väärtusi küsida tulpadega samas järjekorras. Mõne draiveri ja baasi puhul on võimalik väärtusi küsida vaid ühes järjekorras: tulpi vasakult paremale ning ridu ülevalt alla. Kui pole olulist põhjust taolisest järjestusest kõrvale hoidmiseks, siis tasuks seda järgida. Siis tekib vähem probleeme rakenduse teise serverisse või teise baasi külge tööle panekul.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class LauludPlaadil1 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
    {
      vastus.setContentType("text/html");
      PrintWriter valja = vastus.getWriter();
      valja.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\"
"+
          "\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">");
      valja.println("<html><head>");
      valja.println("<meta http-equiv=\"Content-Type\" content=\"text/html;
charset=ISO-8859-1\" />");
      valja.println("<title>Laulude nimekiri</title></head>\n");
      valja.println("<body><h1>Laulud plaadil</h1>");
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
        Statement st=cn.createStatement();
        String lause="SELECT pealkiri, plaadinimi "+
            "FROM laulud, laulud_plaadid, plaadid "+
            "WHERE laulud.id=laulud_plaadid.laulu_id AND "+
            "laulud_plaadid.plaadi_id=plaadid.id";
        valja.println("<table>");
        ResultSet rs=st.executeQuery(lause);
        while(rs.next()){
           valja.println("<tr><td>"+rs.getString("pealkiri")+
            "</td><td>"+rs.getString("plaadinimi")+"</td></tr>");
        }
        valja.println("</table>");
        cn.close();
        valja.println("</body>");
        valja.println("</html>");

      }catch(Exception viga){
         viga.printStackTrace(valja);
      }
   }
}
```

Ning servleti töö tulemuseks on ilus lihtne tabel.



Nõnda nagu laule üldloendisse lisatakse, nii ka tuleb luua võimalus määramiseks, millised laulud millistele plaatidele lindistatud on. Ehkki tehniliselt tuleb vaid laule ja plaate siduvasse tabelisse lisada kaks id-numbrit, on kasutajal mugavam nime järgi valida, milline plaat ja milline pala omavahel ühendatud on.

Et nii laulude kui plaatide nimed juba kirjas, siis pole neid mõistlik enam uuesti sisse kirjutada. Kuni andmeil on kuni mõniteist, siis võiks sobiva väärtuse välja valimiseks sobida rippmenüü. Suuremate mahtude puhul tuleks kasutada järjestamist, raadionuppe või otsinguvahendit. Siinne näide töötab rippmenüüga. Samuti on mõistlik lasta kasutajal valida väärtuste hulgast, tegelikult serverisse salvestamiseks saata aga id-numbrid. Rippmenüül tuleb selleks iga väärtuse (option) juurde kirjutada väärtus (value), mida siis tegelikult serverisse saata soovitakse.

```
          valja.println("  <option value='"+rs.getString("id")+
"'>"+rs.getString("plaadinimi")+"</option>");
```

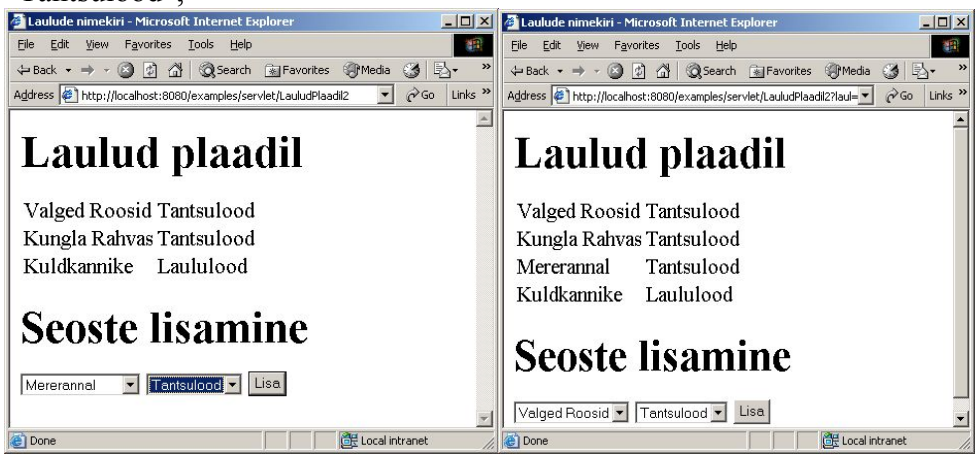
Selline ID-numbrite järgi valimine ja andmete näitamine lihtsustaks ka näiteks rakenduse tõlkimist: kui andmeid oleks vaja mitmes keeles näidata, siis numbrid võiksid samaks jääda, tõlkida oleks tarvis vaid sõnu.

Lisamine näeb laulude tabeliga võrreldes välja küllalt sarnane. Et servleti parameetritena saabuvad vaid ühendatava laulu ja plaadi ID-numbrid ning need numbrid ongi vaja seosetabelisse kirjutada, siis pole muud kui väärtused tabelisse kirjutada. Et andmete lisamine on lehel tulemuste näitamisest eespool, siis võib juba kohe pärast lisamist näha, et valitud laulu ja plaadi paar ka loetellu on ilmunud.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class LauludPlaadil2 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
    {
      vastus.setContentType("text/html");
      PrintWriter valja = vastus.getWriter();
      valja.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\"
"+
          "\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">");
      valja.println("<html><head>");
      valja.println("<meta http-equiv=\"Content-Type\" content=\"text/html;
charset=ISO-8859-1\" />");
      valja.println("<title>Laulude nimekiri</title></head>\n");
      valja.println("<body><h1>Laulud plaadil</h1>");
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
       if(kysimus.getParameter("lisamine")!=null){
```

```
            PreparedStatement ps=cn.prepareStatement(
              "INSERT INTO laulud_plaadid (laulu_id, plaadi_id) VALUES (?, ?)"
            );
            ps.setInt(1, Integer.parseInt(kysimus.getParameter("laul")));
            ps.setInt(2, Integer.parseInt(kysimus.getParameter("plaat")));
            ps.executeUpdate();
          }

          Statement st=cn.createStatement();
          String lause="SELECT pealkiri, plaadinimi "+
              "FROM laulud, laulud_plaadid, plaadid "+
              "WHERE laulud.id=laulud_plaadid.laulu_id AND "+
              "laulud_plaadid.plaadi_id=plaadid.id";
          valja.println("<table>");
          ResultSet rs=st.executeQuery(lause);
          while(rs.next()){
             valja.println("<tr><td>"+rs.getString("pealkiri")+
              "</td><td>"+rs.getString("plaadinimi")+"</td></tr>");
          }
          valja.println("</table>");
          valja.println("<h1>Seoste lisamine</h1>");
          valja.println("<form action='#'>");
          lause="SELECT id, pealkiri FROM laulud";
          rs=st.executeQuery(lause);
          valja.println("<select name='laul'>");
          while(rs.next()){
            valja.println("  <option value='"+rs.getString("id")+
                "'>"+rs.getString("pealkiri")+"</option>");
          }
          valja.println("</select>");

          lause="SELECT id, plaadinimi FROM plaadid";
          rs=st.executeQuery(lause);
          valja.println("<select name='plaat'>");
          while(rs.next()){
            valja.println("  <option value='"+rs.getString("id")+
                "'>"+rs.getString("plaadinimi")+"</option>");
          }
          valja.println("</select>");
          valja.println("<input type='submit' name='lisamine' value='Lisa' />");
          valja.println("</form>");
          cn.close();
          valja.println("</body>");
          valja.println("</html>");

        }catch(Exception viga){
           viga.printStackTrace(valja);
        }
    }
}
```

Järgnevatel piltidel võibki imetleda, kuidas lisati laul “Mererannal” plaadile “Tantsulood”,



Kustutamine

Enamikel asjadel on nii algus kui ots. Ehkki suuremate andmebaaside puhul soovitatakse andmed kustutamise puhul arhiveerida ning arhiveerimisatribuudi abil määrata, et seda rida pole vaja enam arvestada. Nõnda on kergem jälgida baasi arengulugu ning võimalike vigade puhul olukord taastada. Rakendust jälle lihtsam teha ning omal pilt selgem, juhul kui toimimiseks hädavajalikke veerge ja andmeid vähem on.

Arvutikettalt kustudades kipuvad andmed kasutaja jaoks jäädavalt kadunud olema. Kas ja kui täpselt on endist seisu võimalik taastada, sõltub serveri andmete varundamisest ning andmebaasimootori vastavatest võimalustest. Küllalt sageli seetõttu lisatakse pärast kustutatavate andmete märkimist kasutaja tarbeks koht üle kontrollimaks, kas ikka soovitakse kustutada. Siin näites aga on piirdutud lihtsama võimalusega ning lisakontrolli ette ei võeta.

Kustutuskasutajaliidese võimalusi on mitmeid. Lihtsaim on luua arvatavasti lehte, kus iga andmerea taga on viide, millele vajutades vastav rida baasis kustutatakse. Kas siis kustutustoiming tehakse eraldi servleti abil ning siis suunatakse kasutaja loetelulehele tagasi või on kustutusoskused juba loetelulehele sisse ehitatud, see on juba realiseerimise küsimus. Esimese võimaluse eeliseks on, et lehe värskenduse korral ei saadeta kustutusteavet uuesti.

Tehniliselt ligikaudu sama lihtne või keerukas on võimalus lasta kasutajal raadionupuga määrata kustutatav rida. Ikkagi jõuab andmeid vastu võtvale lehele kaasa kustutamist määrav väärtus, enamasti kustutatava rea id-number.

Siin näites aga tehti läbi võimalus, kus kasutaja saab kustutamiseks valida need read, mida ta parajasti soovib. Sellise olukorra lahendus on keeleti erinev. Üheks ning siingi kasutatud võimaluseks on määrata kõigile märkeruutudele sama nimi. Vaid elemendi väärtus on igal puhul erinev.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class LauludPlaadil3 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
    {
      vastus.setContentType("text/html");
      PrintWriter valja = vastus.getWriter();
      valja.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\" "+
          "\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">");
      valja.println("<html><head>");
      valja.println("<meta http-equiv=\"Content-Type\" "+
                           "content=\"text/html; charset=ISO-8859-1\" />");
      valja.println("<title>Laulude nimekiri</title></head>\n");
      valja.println("<body><h1>Laulud plaadil</h1>");
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
       if(kysimus.getParameter("lisamine")!=null){
           PreparedStatement ps=cn.prepareStatement(
             "INSERT INTO laulud_plaadid (laulu_id, plaadi_id) VALUES (?, ?)"
           );
           ps.setInt(1, Integer.parseInt(kysimus.getParameter("laul")));
           ps.setInt(2, Integer.parseInt(kysimus.getParameter("plaat")));
           ps.executeUpdate();
       }

        Statement st=cn.createStatement();
        String lause="SELECT laulud_plaadid.id as seosenr, pealkiri, plaadinimi "+
            "FROM laulud, laulud_plaadid, plaadid "+
            "WHERE laulud.id=laulud_plaadid.laulu_id AND "+
```

```
            "laulud_plaadid.plaadi_id=plaadid.id";
          valja.println("<form action='Kustutus' method='post'><table>");
          ResultSet rs=st.executeQuery(lause);
          while(rs.next()){
             valja.println("<tr>"+
              "<td><input type='checkbox' name='kustutus' value='"+
              +rs.getInt("seosenr")+"' /></td>"+
              "<td>"+rs.getString("pealkiri")+
              "</td><td>"+rs.getString("plaadinimi")+"</td></tr>");
          }
          valja.println("</table><input type='submit' value='Kustuta' /></form>");
          valja.println("<h1>Seoste lisamine</h1>");
          valja.println("<form action='#'>");
          lause="SELECT id, pealkiri FROM laulud";
          rs=st.executeQuery(lause);
          valja.println("<select name='laul'>");
          while(rs.next()){
            valja.println("  <option value='"+rs.getString("id")+
                "'>"+rs.getString("pealkiri")+"</option>");
          }
          valja.println("</select>");

          lause="SELECT id, plaadinimi FROM plaadid";
          rs=st.executeQuery(lause);
          valja.println("<select name='plaat'>");
          while(rs.next()){
            valja.println("  <option value='"+rs.getString("id")+
                "'>"+rs.getString("plaadinimi")+"</option>");
          }
          valja.println("</select>");
          valja.println("<input type='submit' name='lisamine' value='Lisa' />");
          valja.println("</form>");
          valja.println("<br /><a href='Laulud6'>Laulude haldus</a>");
          cn.close();
          valja.println("</body>");
          valja.println("</html>");

        }catch(Exception viga){
           viga.printStackTrace(valja);
        }
    }
}
```

Nõnda koostatuna jõuab serverisse sama nimega nii mitu parameetrit, palju kasutaja märkeruute märkinud on. Mõnikord on taoliste andmete välja lugemine keeruline – ka Java puhul suudab käsklus getParameter väljastada vaid parameetri ühe väärtuse. Õnneks on siin kasutada ka käsklus getParameterValues, mis väljastab küsitud nimega parameetrite väärtused massiivina. Ning siin on näha ka PreparedStatement'i tähtsam kasutusvaldkond. Olukord, kus sarnast lauset tuleb käivitada mitmete andmetega. Kui õnnestub, siis PreparedStatement'i luues harutatakse SQL-tekst lahti masina jaoks kiiremini käsitsetavale kujule (näiteks

kahendpuusse) ning järgmistel kordadel kulub energiat vaid uute andmetega käskluse andmebaasiserveris käivitamiseks.

Kustutuslehelt saadetakse kasutaja jälle algsele lehele tagasi.

```
vastus.sendRedirect("LauludPlaadil3");
```

Sel käsklusel on mõtet olukorras, kus midagi pole veel välja trükitud. Nii saab HTTP päiste kaudu teada anda, et soovitakse hoopis järgmise lehe avamist. Ning seiluri ülesandeks on siis sobivat lehte avama hakata.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class Kustutus extends HttpServlet {
    public void doPost(HttpServletRequest kysimus,
                       HttpServletResponse vastus)
    throws IOException, ServletException
    {
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
        PreparedStatement ps=cn.prepareStatement(
          "DELETE FROM laulud_plaadid WHERE id=?"
        );
        String[] vastused=kysimus.getParameterValues("kustutus");
        if(vastused!=null){
         for(int i=0; i<vastused.length; i++){
          ps.setInt(1, Integer.parseInt(vastused[i]));
          ps.executeUpdate();
         }
        }
        cn.close();
        vastus.sendRedirect("LauludPlaadil3");
      }catch(Exception viga){
         viga.printStackTrace(vastus.getWriter());
      }
   }
}
```

Ja võibki näha, kuidas lugu “Mererannal” on taas plaadilt maha võetud.

Laulude haldus

Et laulude andmed hoitakse eraldi ning hoopis teine tabel osutab, milline laul millistel plaatidel paikneb, siis on mugavam ja töökindlam koostada rakendus nõnda, et ka laulude sisestamine ning nende plaatidele määramine on eraldi toimingud. Kui tahta lõppkasutajale sisestamist võimalikult mugavaks teha, siis võib ju püüda nii laulu sisestamine kui plaadile määramine ühele lehele paigutada. Sellisel juhul aga jääb programmi hooleks hoolitseda, et andmed ka sobivalt tabelitesse saaksid ning kogemata ühe laulu andmeid mitut korda ei sisestataks.

Kui aga kasutajate näol pole tegemist väga arvutikaugete inimestega, siis läheb programmeerija töö märkimisväärselt selgemaks, kui on võimalik ühe tabeli andmete lisamise, kustutamise ja muutmisega korraga tegelda, vajadusel lihtsalt kasutajatele teistest tabelitest lisainfot juurde näidates.

Kui nüüd laule plaadile lisades selgub, et mõni laul puudu või valesti kirja pandud, siis tuleb vastava viite kaudu liikuda lihtsalt laulude halduse lehele. Sinna lauludega tegelevale lehele on võrreldes varasemate näidetega paigutatud ka laulude andmete muutmine. Laulude loetelu lehel on lihtsalt juurde tulnud viide LauluMuutusVormile, kus siis pala andmetega edasi tegeldakse.

```
"<a href='LauluMuutusVorm?id="+rs.getInt("id")
```

Nagu näha, antakse vormile kaasa ka laulu ID-number. Siis on vormil teada, millise lauluga tuleb toimetama asuda.

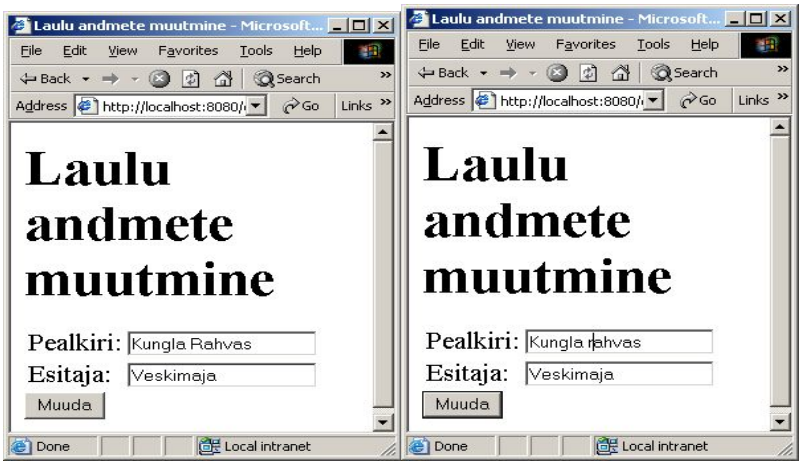
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class Laulud6 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
    {
      vastus.setContentType("text/html");
      PrintWriter valja = vastus.getWriter();
      valja.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\"
"+
          "\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">");
      valja.println("<html><head>");
      valja.println("<meta http-equiv=\"Content-Type\" content=\"text/html;
charset=ISO-8859-1\" />");
      valja.println("<title>Laulude nimekiri</title></head>\n");
      valja.println("<body><h1>Laulude nimekiri</h1>");
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
        Statement st=cn.createStatement();
        if(kysimus.getParameter("lisamine")!=null){
           PreparedStatement ps=cn.prepareStatement(
             "INSERT INTO laulud (pealkiri, esitaja) VALUES (?, ?)"
           );
           ps.setString(1, kysimus.getParameter("pealkiri"));
           ps.setString(2, kysimus.getParameter("esitaja"));
           ps.executeUpdate();
        }
        String sort="id";
        String sortTulp=kysimus.getParameter("sorttulp");
        if(sortTulp==null){sortTulp="id";}
        if(sortTulp.equals("pealkiri")){sort="pealkiri";}
        if(sortTulp.equals("esitaja")){sort="esitaja";}
        String lause="SELECT id, pealkiri, esitaja FROM laulud ORDER BY "+sort;
        ResultSet rs=st.executeQuery(lause);
        valja.println("<table>");
```

```
            valja.println("<tr>"+
               "<th><a href='"+kysimus.getRequestURI()+
               "?sorttulp=pealkiri'>Pealkiri</a></th>"+
               "<th><a href='"+kysimus.getRequestURI()+
               "?sorttulp=esitaja'>Esitaja</a></th>"+
               "<th>Muutmine</th>"+
               "</tr>");
            while(rs.next()){
             valja.println("<tr><td>"+
              Abi.filtreeriHTML(rs.getString("pealkiri"))+
              "</td><td>"+
              Abi.filtreeriHTML(rs.getString("esitaja"))+
              "</td><td>"+
              "<a href='LauluMuutusVorm?id="+rs.getInt("id")+
              "' />muuda</a></td></tr>");
            }
            valja.println("</table>");
            valja.println("<h1>Lisamine</h1>");
            valja.println(
              "<form action='#'><table>"+
              "<tr><td>Pealkiri:</td><td>"+
               "<input type='text' name='pealkiri' /></td></tr>\n"+
              "<tr><td>Esitaja:</td><td>"+
               "<input type='text' name='esitaja' /></td></tr>\n"+
              "</table>"+
              "<input type='submit' name='lisamine' value='Lisa' />"+
              "</form>"
            );
            cn.close();
            valja.println("<br />");
            valja.println("<a href='LauludPlaadil3'>Laulude paiknemine plaatidel</a>");
            valja.println("</body>");
            valja.println("</html>");

          }catch(Exception viga){
             viga.printStackTrace(valja);
          }
       }
}
```

Siin näites on muutmisvormi ülesandeks näidata etteantud id-numbriga laulu andmed tekstiväljadesse, et inimesel oleks võimalik väärtusi parandada. Edasine tegelik muutmistöö baasis usaldatakse järgmisele servletile nimega LauluMuutusLeht. Lisaks pealkirjale ja esitajale tuleb LauluMuutusLehele edasi anda ka muudetava pala ID – muidu poleks ju teada, millise pala väärtused vahetada tuleb. Identifikaatori kaasa andmiseks sobib varjatud väli, sisestuselement parameetriga hidden.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class LauluMuutusVorm extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException{
      vastus.setContentType("text/html");
      PrintWriter valja = vastus.getWriter();
      valja.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML "+
          "1.0 Transitional//EN\" "+
          "\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">");
      valja.println("<html><head>");
      valja.println("<meta http-equiv=\"Content-Type\" "+
                        "content=\"text/html; charset=ISO-8859-1\" />");
      valja.println("<title>Laulu andmete muutmine</title></head>\n");
      valja.println("<body><h1>Laulu andmete muutmine</h1>");
      try{
        String id=kysimus.getParameter("id");
        if(id==null){
          valja.println("Laul määramata</body></html>");
          return;
        }
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
        PreparedStatement st=cn.prepareStatement(
          "SELECT pealkiri, esitaja FROM laulud WHERE id=?");
          st.setInt(1, Integer.parseInt(id));
        ResultSet rs=st.executeQuery();
        if(!rs.next()){
          valja.println("Soovitud laul puudub.</body></html>");
          return;
        }
        valja.println("<form action='LauluMuutusLeht' method='post'>\n"+
          "<table><tr><td>Pealkiri:"+
          "</td><td><input type='text' name='pealkiri' value='"+
          rs.getString("pealkiri")+"' /></td></tr>"+
          "<tr><td>Esitaja:"+
          "</td><td><input type='text' name='esitaja' value='"+
          rs.getString("esitaja")+"' /></td></tr>"+
          "</table><input type='submit' value='Muuda' />"+
          "<input type='hidden' name='id' value='"+id+"'"+
          "</form>"
        );
        cn.close();
      }catch(Exception viga){
         viga.printStackTrace(valja);
      }
    }
}
```

Ning ongi ees leht, kus võimalik ennist sisse kirjutatud väärtused uutega asendada.



Muutmislehele jõuavad nii pealkiri, esitaja kui laulu identifikaator. Lehel paikneva UPDATE-lause ülesandeks on etteantud id-ga laulu parameetrite väärtused asendada. Turvalisuse mõttes ikka PreparedStatementi kasutades.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class LauluMuutusLeht extends HttpServlet {
    public void doPost(HttpServletRequest kysimus,
                       HttpServletResponse vastus)
    throws IOException, ServletException
    {
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
        PreparedStatement ps=cn.prepareStatement(
          "UPDATE laulud SET esitaja=?, pealkiri=? WHERE id=?"
        );
        ps.setString(1, kysimus.getParameter("esitaja"));
        ps.setString(2, kysimus.getParameter("pealkiri"));
        ps.setInt(3, Integer.parseInt(kysimus.getParameter("id")));
        ps.executeUpdate();
        cn.close();
        vastus.sendRedirect("Laulud6");
      }catch(Exception viga){
         viga.printStackTrace(vastus.getWriter());
      }
   }
}
```

Kui andmed muudetud ning loetellu tagasi vaatama minna, siis ka seal on Kungla rahva teine sõna ilusti väikese r-iga. Ning et andmed võetakse ikka samast tabelist, siis muutus paistab ka lehel, kus kirjas laulude paiknemine plaatidel.


Laulude nimekiri - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/examples/servlet/Lau
Go Links
Laulude nimekiri
Pealkiri Esitaja Muutmine
Valged Roosid Joala muuda
Kuldkannike Joala muuda
Mererannal Linna muuda
Kungla rahvas Veskimaja muuda
Koolisellid Tammik muuda
Karsumm Kask muuda
Lisamine
Pealkiri:
Esitaja:
Lisa
Laulude paiknemine plaatidel
Done
Local intranet

Ülesandeid

Söökla menüü

* Loo lehestik sööklas pakutavate toitude lisamiseks ja vaatamiseks. Igal toidul on nimi ja hind.
* Sisestatud toidu andmeid on võimalik ka muuta ning olemasolevaid toite kustutada.
* Loo vahend, mille abil saaks määrata, milliseid toite millisel päeval pakutakse.

Bussiplaan

* Lehele on võimalik salvestada ja sealt vaadata busside väljumise aegu.
* Buss sõidab sihtkohta 2 tundi ja 30 minutit. Vastavalt väljumisaegadele arvutatakse ja sihtkohta jõudmise ajad.
* Lisaks eelmisele on olemasolevaid aegu võimalik kustutada või muuta.

Kirjasõprade otsing sünniaasta järgi

* Inimese kohta saab veebilehtl salvestada nime, sünniaasta ning sünniaastate vahemiku, millises vanuses kirjasõpra ta otsib.
* Iga inimese lehel kuvatakse sisestatutest nende inimeste andmed, kelle sünniaasta jääb soovitud vahemikku.
* Lisaks kuvatakse inimesele need, kelle soovitud vahemikku vaataja ise jääb ning eraldi loeteluna nimed, kelle puhul mõlemad vahemikud sobivad.

Kirjasõprade otsing huviala järgi.

* Loo administraatorile võimalus huvialade sisestamiseks ja vaatamiseks.
* Lisa võimalus isikute andmete sisestamiseks. Iga isik saab valida, millised huvialad talle loetelust sobivad.
* Näita inimese lehel loetelu teistest inimestest, kellega tal vähemasti üks huviala kattub. Vajutades vastava inimese nimele näidatakse loetelu kattuvatest huvialadest.

Autovaruosade otsing

* Loo administraatorile leht automarkide sisestamiseks ning leht osade nimetuste sisestuseks.
* Loo leht, kus saab valida margi ja osa. Samuti, kas soovitakse müüa või osta ning lisada kommentaari (näit. väljalaskeaasta).

* Loo leht, kus müüjale näidatakse kõik vastava osa ostusoovid ning ostjale müügisoovid.

Telefoninumbrite märkmik

* Kasutaja saab veebi kaudu talletada ja vaadata inimeste eesnimesid, perekonnanimesid ja telefoninumbreid.
* Andmeid on võimalik parandada ning perekonnanime järgi otsida.
* Andmeid saab ka kustutada ning rippmenüüst iga numbri juurde valida, kas tegemist on kodu- või töötelefoniga.

Veahaldusvahend

* Loo leht veateadete salvestamiseks ja vaatamiseks.
* Lisa administraatorileht, kus on võimalik olemasolevate parandajate hulgast määrata, kes konkreetse teatega tegelema hakkab.
* Parandaja näeb ainult enesele määratud veateateid. Saab teate juurde märkida, millised probleemid on lahendatud.

Korrapidajate tabel

* Lehel on võimalik sisestada kuupäev ning nimi, kes sel päeval korrapidaja. Tulemusi saab ka näha.
* Võrreldes eelmisega kontrollitakse, et samaks päevaks ei määrataks korraga mitut korrapidajat.
* Salvestatud väärtusi saab muuta ja kustutada vaid vastava rolliga kasutaja.

Linnuvaatlusmärkmik

* Lehel saab salvestada nähtud linnuliigi, vaatlusaja ja isendite arvu. Tuemusi õnnestub vaadata.
* Eraldi vahend on olemasolevate linnuliikide sisestamiseks. Vaatlusandmete sisestamisel valitakse linnuliik rippmenüüst.
* Liikide loendi kõrval näidatakse, mitu korda vastavat liiki on loendatud ning mitut isendit vastavast liigist kokku nähtud.

Taimevaatlusmärkmik

* Lehel sisestatakse vaatleja nimi, taime liik ja vaatluskoht. Tulemusi õnnestub vaadata.

* Vajutades vaatleja nimele, näidatakse kõik andmed, mis see inimene on kirja pannud.
* Lisaks eelmisele näidatakse kõik erinevad vaatlejanimed, nime taga number, mitu erinevat liiki taimi on see inimene kirja pannud (sõltumata vaatluskohast).

Putukate kogu

* Lehel sisestatakse kogutud putuka liik, vaatluspäev/aeg, piirkond, koht, leidja ning määraja nimi. Andmeid õnnestub vaadata.
* Loetelus näidatatakse vaid putukaid, kes on kogutud enne kasutaja sisestatud kellaaega.
* Lisaks eelmisele asub iga rea ees märkeruut. Märgistatud putukad näidatakse eraldi lehele, kus õnnestub igale neist etikett trükkida. Etiketis on andmed tabelina, iga väärtuse ees tunnuse nimetus.

Loodusfotograafi märkmik

* Iga pildistuse kohta talletatakse pildi teema, kaugus, säriaeg ja ava suurus. Andmeid näeb lehel.
* Säriajad valitakse rippmenüüst. Õnnestub otsida kasutaja määratud säriajaga pildistuste andmeid.
* Lisaks eelmisele luuakse statistika, millise säriaja ja ava suuruse kombinatsiooniga kui mitu pilti on tehtud. Igale pildile saab lisada kommentaare.

Ilmavaatlusandmed

* Igal vaatluskorral sisestatakse temperatuur, tuule kiirus (m/s), suund (nt. NNO). Andmeid näidatakse tabelina.
* Eraldi lehele luuakse statistika, mitu korda millist temperatuuri olnud on.
* Lisaks luuakse tabel, mitu korda millisest suunast tuul puhunud on. Mida tugevam on vastavast suunast puhunud tuule keskmine kiirus, seda tumedamalt vastav arv joonistatakse.

Videokahuri kasutusgraafik

Programmi eesmärgiks on pakkuda veebipõhiselt võimalust jälgida ja reserveerida videokahuri kasutusvõimalust.

* Tabelis on kirjas kahuri kasutusaegade algused ja kestused. Tabel kuvatakse.
* Lisaks eelmisele võib kasutaja sisestada aja. Väljastatakse, kas sel ajal on kahur vaba.
* Lisaks eelmisele on registreeritud kasutajatel võimalik kahurit reserveerida.

Raamatukogulaenutus

* Tabelis on raamatute ilmumisandmed. Tabel kuvatakse veebilehele.
* Lisaks eelmisele on loodud kasutajate tabel ning seosetabel, kus on näidatud, milliseks ajaks on raamat kasutajale laenatud. Väljastatakse, kas praegu on soovitud raamat vaba.
* Lisaks eelmisele on võimalik lisada nii raamatuid, kasutajaid kui laenutusi ning otsida sobivat parajasti vabade raamatute hulgast.

Tööaja arvestusgraafik

* Tööle sisenedes ning töölt lahkudes sisestab kasutaja oma koodi. Kood, liikumissuund ning kellaaeg talletatakse tabelisse.
* Lisaks eelmisele on võimalik kontrollida, et tabelis poleks järjest kaht sisenemist või väljumist. Kui andmed korras, väljastatakse inimese tööl viibitud aeg.
* Inimese tööl oldud aeg väljastatakse kuude lõikes. Eraldi näidatakse puhkepäevadel tööl viibitud aeg. Iga kuu kohta leitakse aeg, kus korraga on olnud kõige rohkem inimesi kohal.

Komandeeringuaruanded

* Kasutaja saaab veebi kaudu tabelisse märkida oma nime, reisi sihtpunkti ja reisi kestuse
* Lisaks saab kasutaja märkida, kui suured on tema sõidukulud. Päringulehel on võimalik vaadata nii kogu suurt tabelit kui inimese kaupa, kui palju on ta kokku kulutanud komandeeringusõitudele.
* Eraldi tabelid on nii kasutajate, komandeeringute kui üksikute sõidutsekkide tarvis. Veebilehelt on võimalik lisada komandeeringuid ning sinna kuuluvaid pileteid.

Uksed

Abiprogramm kaardipõhisele ustesüsteemile.
* Tabelites on kirjas väljaantud magnetkaardid, majas paiknevad kaardiga avanevad uksed ning õigused, millise kaardi omanik millist ust avada tohib.
* Sisestades kaardi numbri ja ukse numbri, vastatakse, kas vastava kaardi omanik tohib sellest uksest siseneda.
* Lisaks eelmisele saab sissepääsu õigusi jagada piiratud ajaks. Vale kaardi või ukse numbri sisestamisel antakse veateade.

Laulude andmebaas

* Võimalda tabelisse lisada laulu sõnad, pealkiri ja märksõnad.
* Lisaks eelmisele on võimalik nii sisu, pealkirja kui märksõnade järgi otsida.
* Lisaks eelmisele on registreeritud kasutajatel võimalik laulu andmeid täiendada ja parandada.

Vilistlaste kontaktandmed

* Tabel, milles on vilistlaste nimed, aadressid, telefoninumbrid ja elektronposti aadressid, kuvatakse veebilehele.
* Inimestel on võimalus oma andmeid muuta. Logisse jäetakse kirja, millised olid andmed enne, millisest masinast ning milliseks andmed muudeti.
* Administraatoril on võimalik vaadata valitud kirjega seotud logisid ning määrata, millise sammuni olukord ennistatakse.

Tunniplaan

* Tabelis on kirjas aine nimetus, grupi tähis, ruumi nr, algusaeg ning kestus. Andmed kuvatakse veebilehele. Soovi korral saab vaadata vaid ühe grupi aineid.
* Programm kontrollib andmete korrektsust: hoiatus antakse, kui samal grupil on korraga mitu ainet, samuti kui samas ruumis on korraga mitu ainet.
* Kui sisestatakse grupi tähis ja ruumi nr, siis väljastatakse võimalikud ajavahemikud, mil tund võiks toimuda. Seejärel saab sisestada aine nime, valida aja ja pikkuse ning andmed talletatakse tabelisse.

Jõe vooluhulgad

Igal keskpäeval saadetakse Pikasilla mõõtmispunktist kesktabelisse teade, mitu liitrit sekundis voolab vett Väikesest Emajõest Võrtsjärve.
* Tabel väljastatakse veebilehele, näidates, vooluhulgad nii liitrites sekundi kohta kui kuupmeetrites ööpäeva kohta.
* Lisaks eelmisele näidatakse väärtused lehele tulpdiagrammina.
* Tubad koostatakse viie päeva keskmise kaupa. Kasvava vooluhulgaga piirkondades on tulbad teise värviga eristatud.

Ülesannete kogu

* Veebilehe kaudu salvestatakse tabelisse tulpadena nii ülesanded kui nende pealkirjad. Andmeid on väljastamisel võimalik sortida nii pealkirja kui sisu järgi.
* Eraldi tabeli(te)s on võimalikud märksõnad nii teemade kui raskusastmete kohta. Kasutaja saab valida, millised märksõnad konkreetse ülesande juurde kuuluvad. On võimalik otsida nii pealkirja kui märksõna järgi.
* Kord sisestatud andmeid ja seoseid on võimalik muuta. Samuti saab lisada märksõnu.

Koodinäidete kogu

* Koodinäidete tekstid ning pealkirjad salvestatakse tabelisse. Vaatamisel saab valida, kas näide väljastatakse veebilehele lihttekstina või HTML-kodeeritult.
* Igal näitel on pealkiri ning valikuline selgitustekst. Eraldi saab näite juures määrata, milliste teemade juurde ta kuulub. Samuti saab määrata, millised näited on loodud näitega lähedased. Otsida on võimalik nii teemade, pealkirja kui näites või selgituses leiduva sõna järgi.
* Lisaks eelmisele on teemad puukujulises hierarhias. Näidatakse valitud teemast alanev teemade ning näidete puu.

Failipuu

* Veebis näidatakse välja programmi sees määratud kataloogis paiknevate failinimede ja alamkataloogide loetelu.
* Vajutades failinimele, on võimalik näha selle sisu. Vajutades kataloginimele, näidatakse vastav kataloog oma failide ja alamkataloogide nimedega.
* Andmed on näha puuna, iga kataloogi ees + või - vastavalt sellele, kas sisu näidatakse. Plussile vajutades kataloog avaneb, miinusele vajutades sulgub.

Baasipõhine failipuu

* Andmetabeli kirjeteks on failid (nimed) ning kataloogid. Veergudes paiknevate arvude abil on võimalik tuvastada, kuhu kataloogi miski fail või kataloog kuulub. Trüki puu alates juurkataloogist välja.
* Koosta programm lisamaks kasutaja määratud fail või kataloog (koos alanejatega) määratud kohta andmetabelipuusse.
* Võimalda talletada ka failide sisu baasi, liikuda andmepuus harusid avades ja sulgedes ning näha valitud faili sisu.

Restorani ladu

* Tabelis on kirjas laos leiduvad toiduained, sees olevad kogused ning iga aine kohta alampiir, mis puhul tuleks seda juurde tellida. Laohoidja sisenemisel näidatakse talle loetelu nimetustest, mida oleks vaja tellida.
* Lisaks eelmisele, kui töötaja toob laost kaupa, valib ja märgib ta arvutisse, mida ja kui palju võttis. Töötaja saab jätta laohoidjale soovi, millist toodet tellida. Soovid kaovad laohoidja ees olevast nimekirjast, kui vastav aine saabub lattu või kui laohoidja märgib, et seda pole võimalik muretseda.
* Lisaks eelmisele liiguvad soovid laohoidjale viimase postkasti. Statistikas on võimalik vaadata, milline töötaja milliseid tooteid ja millises koguses võtnud on.

Tähed muusikas

* Laulust on arvutil meeles kuus järjestikust sõna. Väljastatakse nii mitmes sõna, kui suure arvu kasutaja kirjutab. Sobimatu numbri puhul palutakse uuesti valida.
* Lisaks eelmisele jäävad valitud sõnad ekraanile näha, juba avatud sõna valides antakse veateade.
* Lisaks eelmisele on arvutil meeles paljude laulude kuus järjestikust sõna. Administraator saab valida, millised neist tulevad sel korral võistlusse. Vajutades nupule "järgmine", võetakse ette administraatori valitutest juhuslikult uus lugu kuni lugusid jagub.

Kuldvillak

* Arvutil on meeles 25 küsimust jaotatuna võrdselt viie teema vahel. Lehel on näha 5*5 tabel, kus igale küsimusele vastab viide. Vajutades viitele, kuvatakse lehel vastava küsimuse tekst.
* Lisaks eelmisele ei ole võimalik sama küsimust rohkem kui korra valida. Pärast küsimuse aktiivseks muutmist administraatori poolt on võimalik kolmel mängijal see enesele vastamiseks küsida/vajutada. Kärmemale antakse võimalus tekstiväljas vastata, ülejäänutele teatatakse hilinemisest.
* Lisaks eelmisele loetakse iga mängija punkte.

Eurovisiooni hääletus

* Tabelis on kirjas osalevate maade ja laulude nimekiri. Andmed näidatakse veebilehel.
* Iga maa esindaja saab täita tabeli, kus määrab, mitu punkti ta igale teisele andis. Pärast täitmist näidatakse, mitu punkti millisel lool on ning mitmendal kohal sellega ollakse.

* Lisaks eelmisele kontrollitakse, et ükski maa ei saaks hääletada rohkem kui korra. Samuti, et punkte antaks korrektselt, st., et 1. koht 12 punkti, teine 10, kolmas 8, neljas 7 ning edasi
ühe punkti kaupa vähenedes allapoole.

Miljonimäng

* Pärast administraatori vajutust algküsimuse väljakuulutamiseks on võistlejatel võimalus enesele laadida küsimus ja järjestatavad vastusevariandid. Enneaegse päringu peale teatatakse "vara veel".
* Vastuste salvestamisel jäetakse meelde kulunud aeg ning kas vastus on õige. Tulemused väljastatakse veebilehel. Eraldi tuuakse välja kõige kiirem õige vastus.
* Lisaks eelmisele loo vahend põhimängu kajastamiseks. Küsimused ja õiged vastused loetakse andmekandjalt. Peetakse arvestust kroonide ning kasutatud õlekõrte üle. Publik saab pakkuda õiget vastust. 50-50 eemaldab juhuslikult kaks vale vastust.

Kahevõitlus

* Ekraanil on küsimus ning kümme vastusevarianti. Antakse teada, kas valitud variant oli vale või õige.
* Peetakse meeles ja näidatakse, kumma mängija vastamiskord on. Loetakse punkte, vale vastuse korral kukuvad vastava mängija punktid nulli, samuti läheb järg automaatselt üle teisele mängijale.
* Mäng realiseeritakse täielikult: küsimused loetakse sisse, neljas voor kolmekordsete punktidega, lõpus võitjal eraldimäng.

7 vaprat

* Tabelis on kirjas loo pealkirjad, esitajad ning eelmisel korral antud punktid. Andmed kuvatakse veebilehele punktide arvu järjekorras, 2 uut lugu kõige viimasena.
* Inimestel on võimalik valida lugu ning seda hinnata. Lehte vaadates on näha iga loo hääletamiste arv ning keskmine hinne.
* Lisaks eelmisele tehakse analüüs, milliselt IP-lt on millist lugu mitmel korral hinnatud. Kui arv ületab administraatori seatud limiiti, siis sealt tulnud hääli vastava loo kohta lõpptabelis ei arvestata.

Kuldlaul

* Tabelis on kirjas 9 eelmisest saatest jäänud ning 3 uut lugu. Iga loo kohta peetakse meeles, mitmendat korda too tabelis. Tulemused väljastatakse veebilehele.

* Külastajad saavad administraatori määratud ajavahemikus lauludele hääli anda. Hääletuse lõpus näidetakse, palju hääli iga laul saanud, samuti järjestatakse laulud saadud häälte arvu järjekorras.
* Järgmisse saatesse jõuavad 9 enim hääli saanud lugu. Kui mõni laul on suutnud tabelis püsida 12 korda, siis see arvatakse kuldlauluks ning ka tema asemele tuleb administraatoril järgmise korra tarbeks sisestada uus lugu. Aasta lõpus väljastatakse sel aastal tekkinud kuldlaulude loetelu.

Autoregister

* Tabelis on kirjas registris olevate autode number, mark ning väljalaskeaasta. Andmed kuvatakse veebilehele.
* Markide loetelu on omaette tabelis. Samuti on loodud isikute tabel ning seosetabel, milline isik millise autoga seotud on. Väljastatakse sisestatud isikukoodiga seotud autode andmete loetelu.
* Lisaks eelmisele on võimalik veebi kaudu nii isikute kui autode andmeid lisada ja muuta. Samuti on eraldi leht müügitehingu tarbeks, kus auto läheb ühe isiku juurest teise valdusesse.

Õppetooli raamatukogu

Rakenduse eesmärgiks on anda ülevaade õppetoolis kasutatavatest ja laenutatavatest materjalidest.
* Iga inimene saab veebi kaudu sisestada oma riiulil leiduvate raamatute pealkirjad. Andmed kuvatakse veebilehele.
* Inimeste andmete jaoks on eraldi tabel. Üks rida selles tabelis tähendab, et raamat on õppetooli oma, muul juhul on tegemist inimese isikliku raamatuga. Ühe seose järgi on võimalik näha, kelle oma raamat on, teise seose abil õnnestub vaadata, kas ja kellele on raamat laenutatud. Lisamisel on võimalik rippmenüüst valida, kelle oma on raamat.
* Lisaks eelmisele on võimalik laenutus veebi kaudu registreerida. Samuti saab end raamatule järjekorda panna. Peetakse logi, mitu korda ja millal on millist raamatut laenutatud.