BAB IX

KONEKSI MARIADB PADA JAVA (Netbeans)

9.1 Bahasan dan Sasaran

9.1.1 Bahasan

- Pada bab kali ini akan membahas tentang koneksi MariaDB dengan bahasa pemrograman java.
- Selain hal itu akan dibahas juga mengenai kode pemrograman untuk manipulasi data.

9.1.2 Sasaran

Mahasiswa memahami dalam penggunaan Database MariaDB dan Bahasa pemrograman Java untuk membuat suatu program aplikasi.

9.2 Materi

9.2.1 Langkah-langkah Koneksi database

Terdapat beberapa langkah yang secara umum harus dilakukan sehingga aplikasi yang berbasis Java dapat berinteraksi dengan database server. Langkah-langkah tersebut sebagai berikut :

1. Impor package java.sql
2. Memanggil Driver JDBC
3. Membangun Koneksi
4. Membuat Statement
5. Melakukan Query
6. Menutup Koneksi

1. Impor package java.sql

Pertama-tama yang harus dilakukan sebelum Anda membuat program JDBC adalah mengimpor package java.sql terbih dahulu, karena di dalam package java.sql tersebut terdapat kelas-kelas yang akan digunakan dalam proses-proses berintekasi dengan database server misalnya kelas DriverMaganer, Connection, dan ResultSet.

Hal ini sangat penting dilakukan karena bagi pemula seringkali lupa untuk mengimpor package yang kelas-kelas yang akan digunakan terdapat di dalamnya, sehingga mengakibatkan kegagalan dalam mengkompile program Java.

Adapun listing untuk mengimpor package java.sql adalah sebagai berikut :

```
Import java.sql.*;
```

Listing ini dituliskan sebelum Anda menulis kelas.

2. Memanggil Driver JDBC

Langkah pertama untuk melakukan koneksi dengan database server adalah dengan memanggil JDBC Driver dari database server yang kita gunakan. Driver adalah library yang digunakan untuk berkomunikasi dengan database server. Driver dari setiap database server berbeda-beda, sehingga Anda harus menyesuaikan Driver JDBC sesuai dengan database server yang Anda gunakan.

Berikut ini adalah listing program untuk memanggil driver JDBC.

Class.forName(namaDriver); atau *Class.forName(namaDriver).newInstance();*

Kedua cara di atas memiliki fungsi yang sama yaitu melakukan registrasi class driver dan melakukan intansiasi. Apabila driver yang dimaksud tidak ditemukan, maka program akan menghasilkan exception berupa **ClassNotFoundException**. Untuk menghasilkan exception apabila driver tidak ditemukan, maka diperlukan penambahan **try-catch**. Adapun cara menambahkan **try-catch** untuk penanganan error apabila driver tidak ditemukan, sebagai berikut :

```
try {
   Class.forName(namaDriver);
} catch (ClassNotFoundException e) {
   ... Penanganan Error ClassNotFoundException
}
```

Contoh listing memanggil driver menggunakan MySQL adalah :

```
try {
   Class.forName("org.mariadb.jdbc.Driver")
} catch (ClassNotFoundException e) {
   System.out.println("Pesan Error : " + e)
}
```

]Berikut ini adalah daftar nama-nama driver dari beberapa database server yang sering digunakan.

Database Server	Nama Driver
JDBC-ODBC	sun.jdbc.odbc.JdbcOdbcDriver
MySQL	com.mysql.jdbc.Driver
MariaDB	org.mariadb.jdbc.Driver
Microsoft SQLServer	com.microsoft.jdbc.sqlserver.SQLServerDriver
Oracle	oracle.jdbc.driver.OracleDriver
IBM DB2	COM.ibm.db2.jdbc.app.DB2Driver

3. Membangun Koneksi

Setelah melakukan pemanggilan terhadap driver JDBC, langkah selanjutnya adalah membangun koneksi dengan menggunakan interface **Connection**. Object Connection yang dibuat untuk membangun koneksi dengan

database server tidak dengan cara membuat object baru dari interface Connection melainkan dari class **DriverManager** dengan menggunakan methode **getConnection()**.

```
Connection koneksi = DriverManager.getConnection(<argumen>);
```

Untuk menangani error yang mungkin terjadi pada proses melakukan koneksi dengan database maka ditambahkan try-catch. Exception yang akan dihasilkan pada proses ini adalah berupa SQLException. Adapun cara penulisan listingnya adalah sebagai berikut :

```
try {
... koneksi database
} catch (SQLException sqle){
... penanganan error koneksi
}
```

Ada beberapa macam argumen yang berbeda dari methode getConnection() yang dipanggil dari DriverManager, yaitu :

- **getConnection(String url)**

  Pada methode diatas hanya memerlukan argumen URL, sedangkan untuk data user dan password sudah diikutkan secara langsung. Adapun penulisan nilai sebagai berikut :

  ```
  jdbc:<DBServer>://[Host][:Port]/<namaDB>?<user=User>&<password=Password>
  ```

  Berikut contoh penggunaan metode ini didalam program :

  ```
  try {
  String url = "jdbc: mariadb://localhost:3306/Dbase? User = adi & password = pas";
  Connection koneksi = DriverManager.getConnection(url);
  System.out.prinln("Proses apabila koneksi sukses");
  } catch (SQLException sqle) {
  System.out.println("Proses apabila koneksi gagal dilakukan");
  }
  ```

- **getConnection(String url, Properties info)**

  Pada methode ini memerlukan **URL** dan sebuah object **Properties**. Sebelum menggunakan methode ini, Anda harus melakukan import package berupa **java.util.***, ini dikarenakan object Properties terdapat pada package tersebut. Object Properties berisikan spesifikasi dari setiap parameter database misalnya user name, password, autocommit, dan sebagainya.

  Berikut ini contoh penggunaan methode ini didalam program :

  ```
  try {
  String url = "jdbc: mariadb://localhost:3306/praktikumdbd";
  Properties prop = new java.util.Properties(); // tidak mengimpor kelas
  prop.put("user","NamaUser");
  prop.put("password","datapassword");
  Connection koneksi = DriverManager.getConnection(url, prop);
  System.out.prinln("Proses apabila koneksi sukses");
  } catch (SQLException sqle) {
  System.out.println("Proses apabila koneksi gagal dilakukan");
  }
  ```

- **getConnection(String url, String user, String password)**

  Pada methode ini memerlukan argumen berupa **URL**, **user name**, dan **password**. Methode ini secara langsung mendefinisikan nilai URL, user name dan password.

  Berikut ini contoh penggunaan methode ini didalam program :

```
try {
String url = "jdbc: mariadb://localhost:3306/ praktikumdbd";
String user = "adi"
String password "ternate"
Connection koneksi = DriverManager.getConnection(url, user,
password);
System.out.prinln("Proses apabila koneksi sukses");
} catch (SQLException sqle) {
System.out.println("Proses apabila koneksi gagal dilakukan");
}
```

Berikut ini adalah daftar penulisan URL dari beberapa database server yang sering digunakan.

Database Server	Nama URL	Contoh penggunaan
JDBC-ODBC	jdbc:odbc:<NamaDatabase>	jdbc:odbc:Dbase
MySQL	jdbc:mysql://<nmHost>:<port>/<nmDB>	jdbc:mysql://localhost:3306/Dbase
MariaDB	jdbc:mariadb://<nmHost>:<port>/<nmDB>	jdbc:mariadb://localhost:3306/Dbase
Microsoft SQLServer	jdbc:microsoft:sqlserver://<nmHost>:<port>; DatabaseName=<namaDatabase>	jdbc:microsoft:sqlserver://localhost:1433; DatabaseName=Dbase
Oracle	jdbc:oracle:thin:@<nmHost>:<port>:<nmDB>	jdbc:oracle:thin:@localhost:1521:Dbase
IBM DB2	jdbc:db2:<NamaDatabase>	jdbc:db2:Dbase

4. Membuat Statement

JDBC API menyediakan interface yang berfungsi untuk melakukan proses pengiriman statement SQL yang terdapat pada package java.sql. Statement yang ada secara umum digunakan terdiri dari berikut :

- **Statement**

  Interface ini dibuat oleh methode **Connection.createStatement()**. Object Statement digunakan untuk pengiriman statement SQL tanpa parameter serta setiap SQL statement yang dieksekusi dikirim secara utuh ke database.

```
Statement stat = Connection.createStatement();
```

- **PreparedStatement**

  Interface ini dibuat oleh methode **Connection.prepareStatement()**. Object PreparedStatement digunakan untuk pengiriman statement SQL dengan atau tanpa parameter. Interface ini memiliki performa lebih baik dibandingkan dengan interface Statement karena dapat menjalankan beberapa proses dalam sekali pengiriman perintah SQL, pengiriman selanjutnya hanya parametered querynya saja.

```
PreparedStatement stat = Connection.prepareStatement();
```

5. Melakukan Query

Setelah kita memiliki object statement, kita dapat menggunakannya untuk melakukan pengiriman perintah SQL dan mengeksekusinya. Methode eksekusi yang digunakan untuk perintah SQL terbagi menjadi dua bagian yaitu untuk perintah **SELECT** methode eksekusi yang digunakan adalah **executeQery()** dengan nilai kembaliannya adalah **ResultSet**, dan untuk perintah **INSERT, UPDATE, DELETE** methode eksekusi yang digunakan adalah **executeUpdate().**

Berikut ini adalah contoh melakukan eksekusi perintah SQL dan mengambil hasilnya (ResultSet) dengan menggunakan perintah SELECT :

```
String sql = "SELECT kode, nama, alamat, kelas FROM dataSiswa";
      ResultSet set = stat.executeQuery(sql);
      while (set.next()) {
        String kode = set.getString("kode");
        String nama = set.getString("nama");
        String alamat = set.getString("alamat");
        String kelas = set.getString("kelas");
      }
```

Berikut ini adalah contoh melakukan eksekusi perintah SQL dengan menggunakan perintah DELETE.

```
String sql = "DELETE FROM data_siswa WHERE kode = "1234";
PreparedStatement stat = konek.prepareStatement(sql);
stat.executeUpdate();
```

6. Menutup Koneksi

Penutupan terhadap koneksi database perlu dilakukan agar sumber daya yang digunakan oleh object Connection dapat digunakan lagi oleh proses atau program yang lain. Sebelum kita menutup koneksi database, kita perlu melepas object Statement dengan kode sebagai berikut :

```
statement.close();
```

Untuk menutup koneksi dengan database server dapat kita lakukan dengan kode sebagai berikut :

```
connection.close();
```

9.2.2 Praktek Langkah-langkah Koneksi database dengan java di Netbeans

Materi kali ini akan sedikit membubuhkan tutorial untuk pengkoneksian dan penyampaian contohnya. Seperti berikut langkah-langkahnya :

1. buatlah project baru pada netbeans
2. pada project tersebut, **klik kanan – properties**
3. pilih **Libraries** pada list Properties

   Library bisa didownload di http://downloads.mariadb.org/client-java/
4. **add Library**
7. add **JAR/Folder**

8. browse file Konektor mariaDB

9. ambil file konektor

10. kemudian **open**

11. Klik **OK**

Setelah selesai maka bisa dilanjutkan membuat kelas java untuk mengkoneksikan database yang telah dibuat dengan java. Untuk mempermudah gambaran kode programnya disini terdapat contoh listing sebagai berikut :

Contoh Listing Program

a. Koneksi

Berikut contoh kelas koneksi :

```
package com.java.myapp;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class MyClass {

  public static void main(String[] args) {

          Connection connect = null;

          try {
                  Class.forName("org.mariadb.jdbc.Driver");
                  connect =
DriverManager.getConnection("jdbc:mariadb://localhost/mydatabase" +
                                  "?user=root&password=root");

                  if(connect != null){
                          System.out.println("Database Connected.");
                  } else {
                          System.out.println("Database Connect Failed.");
                  }

          } catch (Exception e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
          }

          // Close
          try {
                  if(connect != null){
                          connect.close();
                  }
          } catch (SQLException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
          }

  }

}
```

b. Insert Data

Berikut contoh kode program insert data pada tabel asisten yang berdiri sendiri :

```
package com.java.myapp;

import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Connection;
import java.sql.DriverManager;

public class MyClass {
      public static void main(String[] args) {

              Connection connect = null;
              Statement s = null;

              try {
                     Class.forName("org.mariadb.jdbc.Driver");
                     connect =
DriverManager.getConnection("jdbc:mariadb://localhost/mydatabase" +
                                    "?user=root&password=root");

                     s = connect.createStatement();

                     String sql = "INSERT INTO customer " +

      "(CustomerID,Name,Email,CountryCode,Budget,Used) " +
                                    "VALUES
('C005','DBD2019','praktikum@gmail.com'" +
                                    ",'TH','1000000','0') ";
             s.execute(sql);

             System.out.println("Record Inserted Successfully");

              } catch (Exception e) {
                     // TODO Auto-generated catch block
                     e.printStackTrace();
              }

              // Close
              try {
                     if(connect != null){
                            s.close();
                            connect.close();
                     }
              } catch (SQLException e) {
                     // TODO Auto-generated catch block
                     e.printStackTrace();
              }

      }

}
```

c. Update Data

Berikut contoh kode program update data pada tabel asisten yang berdiri sendiri :

```
package com.java.myapp;
```

```
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Connection;
import java.sql.DriverManager;

public class MyClass {

  public static void main(String[] args) {

          Connection connect = null;
          Statement s = null;

          try {
                  Class.forName("org.mariadb.jdbc.Driver");
                  connect =
DriverManager.getConnection("jdbc:mariadb://localhost/mydatabase" +
                                  "?user=root&password=root");

                  s = connect.createStatement();

                  String sql = "UPDATE customer " +
                                  "SET Budget = '8000000' " +
                                  " WHERE CustomerID = 'C005' ";
              s.execute(sql);

              System.out.println("Record Update Successfully");

          } catch (Exception e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
          }

          // Close
          try {
                  if(connect != null){
                          s.close();
                          connect.close();
                  }
          } catch (SQLException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
          }

  }

}
```

d. Hapus Data

Berikut contoh kode program delete data pada tabel asisten berdasarkan idnya yang berdiri sendiri :

```
package com.java.myapp;

import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Connection;
import java.sql.DriverManager;

public class MyClass {
```

```
  public static void main(String[] args) {

          Connection connect = null;
          Statement s = null;

          try {
                  Class.forName("org.mariadb.jdbc.Driver");
                  connect =
DriverManager.getConnection("jdbc:mariadb://localhost/mydatabase" +
                                  "?user=root&password=root");

                  s = connect.createStatement();

                  String sql = "DELETE FROM customer " +
                                  " WHERE CustomerID = 'C005' ";
              s.execute(sql);

              System.out.println("Record Delete Successfully");

          } catch (Exception e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
          }

          // Close
          try {
                  if(connect != null){
                          s.close();
                          connect.close();
                  }
          } catch (SQLException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
          }

  }

}
```