JDBC (JAVA DATABASE CONNECTIVITY)

Javada veri tabanlarına bağlanmak için kullanılan bir kütüphanedir. Herhangi bir veri tabanına(Postgre, MSSQL, MYSQL) bağlantı sağlanabilir. Veri ekleme, slime, güncelleme gibi işlemleri statementler ile gerçekleştiririz. Bunları da sql sorguları ile gerçekleştiririz. Kullandığımız veri tabanını değiştirisek sadece Driver Manager’I değiştirmek yeterli olur, java kodlarını değiştirmeye gerek yoktur.

Veri tabanı bağlantısı:

Hangi veri tabanını kullanacaksak o veri tabanının JDBC driver’ını indirmemiz gerekir. İndirdikten sonar proje ana dizini içerisine atmamız gerekir, sürükle bırakla yapabiliriz. Sonrasında project stricter kısmından kütüphane olarak eklememiz gerekir.

public static final String *DB\_URL*="jdbc:mysql://localhost/university?user=root&password=12345";

bağlanıt için bu şekilde get isteği kullanılabilir veya constructor içerisine 3 ayrı değer olarak yazılabilirler.

public static final String *DB\_URL*="jdbc:mysql://localhost/university";  
public static final String *DB\_USER*="root";  
public static final String *DB\_PASSWORD*="12345";

Connection connection= DriverManager.*getConnection*(*DB\_URL*,*DB\_USER*,*DB\_PASSWORD*);

DB’den veri çekme:

Statement ile veri çekme, ekleme, slime gibi işlemleri gerçekleştirebiliriz.

connection= DriverManager.*getConnection*(*DB\_URL*,*DB\_USER*,*DB\_PASSWORD*);  
Statement statement=connection.createStatement();  
ResultSet resultSet= statement.executeQuery(sql);

Buradan sonra bir while döngüsü kullanarak resulset’e gelen verilerin elde edebiliriz.

System.*out*.print("ID: "+resultSet.getInt("student\_id")+" ");  
System.*out*.print("Name: "+resultSet.getString("student\_name")+" ");  
System.*out*.println("Class: "+resultSet.getString("student\_class"));  
System.*out*.println("########################################");

DB’ye veri ekleme:

Statement ile:

public static void addTableWithStatement(Connection connection) throws SQLException {  
 String sql = "INSERT INTO student (student\_name,student\_class) VALUES('ERAY',4)";  
 Statement statement = connection.createStatement();  
 System.*out*.println(statement.executeUpdate(sql));  
 statement.close();  
}

prepared Statement ile:

public static void addTableWithPreparedStatement(Connection connection)throws SQLException{  
 String sql = "INSERT INTO student (student\_name,student\_class) VALUES(?,?)";  
 PreparedStatement preparedStatement=connection.prepareStatement(sql);  
 preparedStatement.setString(1,"Buğra");  
 preparedStatement.setInt(2,4);  
 preparedStatement.executeUpdate();  
 preparedStatement.close();  
}

prepared statement’de normal statement’den farklı olarak, değerleri baştan sorgu içerisinde belirlemeyiz, ? koyduğumuz yerlere daha sonradan metodlar ile değer atayabiliriz, çalıştırdığımızda normal statement gibi değer ekleme işlemini gerçekleştriri.

Transaction yönetimi:

Eğer veri tabanına bir biri ile bağlantılı CRUD işlemleri gerçekleştiriyorsak, herhangi bir adımda hata olması halinde diğer işlemlerin geri alınmasını sağlamak için transaction kullanılır.

connection = DriverManager.*getConnection*(*DB\_URL*, *DB\_USER*, *DB\_PASSWORD*);  
connection.setAutoCommit(false);

AutoCommit false olursa gerkçekleştirdiğimiz işlem, örneğin veri ekleme işlemi yaptığımızı varsayalım, veri tabanına eklenmeyip bellekte tutulur ta ki commit metodu manuel çalıştırılıncaya kadar.

Yaptığımız işlemleri try catch içerisinde yazarsak ve istenmeyen bir durum olduğunda hata fırlatırsak, en sonda commit işlemi kısmına program hiç gitmez ve veri tabanına değer eklenmez, ayrıca catch kısmında eğer bir veritabanına bağlı isek rollback metodu ile yapılan işlemleri geri alabiliriz.