# JDBC nedir?

Java JDBC (Java DataBase Connectivity) MySQL, Oracle, MS SQL Server gibi veritabanlarına bağlanmak veri çekme, listeleme, ekleme, silme, güncelleme gibi işlemleri yapmak için kullanılan pakettir.

JDBC API kullanımı için gerekli olan sınıflar java.sql paketinde yer alır.

JDBC yapısı veritabanından bağımsız olduğundan SQL destekleyen tüm ilişkisel veritabanı ile birlikte çalışır.

**JDBC kullanımı :**

JDBC API kullanımı veritabanı sürücünün yüklenmesi, veritabanı bağlantısı, SQL sorgusunun gönderilmesi ve sonuçların alınması adımlarından oluşur.

```
package com.example.jdbc.utils;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Properties;

public final class JdbcUtils {
    private static Connection connection = null;
    private static Statement statement = null;
    private static PreparedStatement preparedStatement = null;

    private JdbcUtils() {
```

```
    }

    public static Connection getConnection() {

        try {
            LogUtils.info("Db properties okunuyor...");
            Properties properties = DbConfigUtil.readDbProperties();
            LogUtils.info("Db properties okundu.");
            LogUtils.info("Db baðlantýsý kuruluyor...");
            connection =
DriverManager.getConnection(properties.getProperty("db.url"),
                    properties.getProperty("db.user"),
properties.getProperty("db.password"));
            LogUtils.info("Db baðlantýsý kuruldu.");
        } catch (SQLException e) {
            LogUtils.severe("Db baðlantýsý kurulamadý!!!");
            e.printStackTrace();
        }
        return connection;
    }

    public static Statement getStatement() {
        try {
            connection.commit();
            connection = getConnection();
            statement = connection.createStatement();
        } catch (SQLException e) {
            try {
                connection.rollback();
            } catch (SQLException e1) {
```

```
                    e1.printStackTrace();
                }
                e.printStackTrace();
            }
            return statement;
        }

        public static PreparedStatement getPreparedStatement(String sql) {
            String columnNames[] = new String[] { "id" };
            try {
                connection = getConnection();
                preparedStatement =
connection.prepareStatement(sql,columnNames);
            } catch (SQLException e) {
                e.printStackTrace();
            }
            return preparedStatement;
        }

        public static void close() {
            closeConnection();
            closeStatement();
            closePreparedStatement();
        }

        public static void closeConnectionAndStatement() {
            closeConnection();
            closeStatement();
        }
```

```
public static void closeConnectionAndPreparedStatement() {
        closeConnection();
        closePreparedStatement();
}

public static void closeConnection() {
        try {
                connection.close();
        } catch (SQLException e) {
                e.printStackTrace();
        }
}
public static void closeStatement() {
        try {
                statement.close();
        } catch (SQLException e) {
                e.printStackTrace();
        }
}
public static void closePreparedStatement() {
        try {
                preparedStatement.close();
        } catch (SQLException e) {
                e.printStackTrace();
        }
}
```

```
        public static void rollback() {
                if(connection != null) {
                        try {
                                connection.rollback();
                        } catch (SQLException e) {
                                e.printStackTrace();
                        }
                }
        }
}
```

**DbConfigUtil Dosyasında :**

```
db.url=jdbc:mysql://localhost:3306/user_information
db.user=root
db.password=12345
```

**DbConfugUtil Classında :**

```
package com.example.jdbc.utils;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;

public final class DbConfigUtil {
```

```
    private static final String PROPERTIES_NAME = "src\\db.properties";

    private DbConfigUtil() {

    }

    public static Properties readDbProperties() {
        Properties properties = new Properties();
        try (InputStream inputStream = new FileInputStream(PROPERTIES_NAME)){
            properties.load(inputStream);
        } catch (IOException | RuntimeException e) {
            LogUtils.severe("Properties bilgileri alýnamadý!!!");
            e.printStackTrace();
        }

        return properties;
    }

}
```

## JdbcTemplate

Spring JdbcTemplate öğreticisi, Spring'in JdbcTemplate'ini kullanarak verilerle nasıl çalışılacağını gösterir.

JdbcTemplateJDBC ile programlamayı basitleştirmek için bir araçtır.

Spring Jdbc Template ile Veritabanı Bağlantısı Yapmak

**Spring Jdbc Template ile veritabanı işlemleri yapmak için öncelikle**

- Veritabanı için kullanılacak bağlantı sürücüsü
- Veritabanı yolu
- Veritabanı kullanıcı adı
- Veritabanı şifresi

bilgilerine sahip olmak gerekmektedir

Veritabanı işlemlerini Spring Jdbc Template'in yönetmesini istiyoruz. Bunun için ApplicationContext.xml dosyasına veritabanı bilgilerini girerek veritabanı bağlantı işlemini Spring'e vereceğiz.

```
<bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
    <property name="url" value="jdbc:mysql://localhost:3306/testdb"/>
    <property name="username" value="root"/>
    <property name="password" value="1234"/>
</bean>
```

Jdbc Template veritabanından verilerin alınmasına yardımcı olan Row Mapper'ı oluşturmadan önce bir veritabanı modelini oluşturmalıyız.

**İlk önce veritabanımızda tablomuzu oluşturalım.**

```
CREATE TABLE Kimlik(
  ID   INT NOT NULL,
  ADI VARCHAR(20) NOT NULL,
  SOYADI VARCHAR(20) NOT NULL
);
```

**Şimdi model oluşturalım.**

```
public class Kimlik {

    private int id;
    private String adi;
    private String soyadi;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getAdi() {
        return adi;
    }

    public void setAdi(String adi) {
        this.adi = adi;
    }

    public String getSoyadi() {
        return soyadi;
    }

    public void setSoyadi(String soyadi) {
```

```
        this.soyadi = soyadi;
    }
}
```

**Verilerimizi Row Mapper şablonu kullanarak alacağız.**

```
import org.springframework.jdbc.core.RowMapper;
import java.sql.ResultSet;
import java.sql.SQLException;

/**
 * Created by: burakkutbay on 25.08.2016.
 * Site: www.burakkutbay.com
 */
public class KimlikRowMapper implements RowMapper<Kimlik> {

    public Kimlik mapRow(ResultSet rs, int i) throws SQLException {
        Kimlik kimlik = new Kimlik();
        kimlik.setId(rs.getInt(1));
        kimlik.setAdi(rs.getString(2));
        kimlik.setSoyadi(rs.getString(3));
        return kimlik;
    }
}
```

Bu yazıda Spring Jdbc Template veritabanından verilerin alınmasına yardımcı olan Row Mapper'dan bahsedeceğim. Row Mapper'ı oluşturmadan önce bir veritabanı hazırlayalım. Modelimizi oluşturalım. Bundan sonraki yazılarımızda bu model üzerinden giceğiz.

İlk önce veritabanımızda tablomuzu oluşturalım.

```
CREATE TABLE Kimlik(
  ID   INT NOT NULL,
  ADI VARCHAR(20) NOT NULL,
  SOYADI VARCHAR(20) NOT NULL
);
```

**Şimdi model oluşturalım.**

```
public class Kimlik {

   private int id;
   private String adi;
   private String soyadi;

   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getAdi() {
      return adi;
   }

   public void setAdi(String adi) {
      this.adi = adi;
```

```
    }

    public String getSoyadi() {
        return soyadi;
    }

    public void setSoyadi(String soyadi) {
        this.soyadi = soyadi;
    }
}
```

Modelimiz hazır. Bu model üzerinden değişkenlerimize veritabanından verileri getireceğiz.

Spring Jdbc Template Row Mapper Kullanımı

Verilerimizi Row Mapper şablonu kullanarak alacağız.

```
import org.springframework.jdbc.core.RowMapper;
import java.sql.ResultSet;
import java.sql.SQLException;

public class KimlikRowMapper implements RowMapper<Kimlik> {

    public Kimlik mapRow(ResultSet rs, int i) throws SQLException {
        Kimlik kimlik = new Kimlik();
        kimlik.setId(rs.getInt(1));
        kimlik.setAdi(rs.getString(2));
        kimlik.setSoyadi(rs.getString(3));
        return kimlik;
    }
}
```

Spring Jdbc Template ile birlikte Row Mapper kullanarak verilerimizi çekeceğiz. Row Mapper'imizi oluşturduk. Sonraki dersimizde verilerimize CRUD (Create, Read, Update, Delete) işlemlerini uygulayabilmek için Data Access Object (DAO) dediğimiz bir interface yapacağız.

```
import java.util.List;
import javax.sql.DataSource;

public interface KimlikDAO {

        public void create(Integer id, String adi, String soyadi);

        public void delete(Integer id);

        public void update(Integer id, String adi);

      public Kimlik getKimlik(Integer id);

        public List<Kimlik> listKimliks();
}
```

İnterface sınıfımızı oluşturduk. İnterface sınıf yapmamızın nedeni bir şablon yaratarak her seferinde hangi veri olursa olsun aynı işlemleri yaptığımız için her yerde kullanabilmeyi sağlamayı amaçlıyoruz

```
public class Sorgular {

   public static final String create = "insert into Kimlik (id,adi, age) values (?,?, ?)";
   public static final String getKimlik = "select * from Kimlik where id = ?";
   public static final String listKimlik = "select * from Kimlik";
   public static final String delete = "delete from Kimlik where id = ?";
```

```
   public static final String update = "update Kimlik set adi = ? where id = ?";



}
```

Sorgularımızı yazdık. Şimdi CRUD işlemleri için

```
public class kimlikTemplate implements KimlikDAO {

        private JdbcTemplate jdbcTemplateObject;



        public void create(Integer id,String adi, String soyadi) {

                jdbcTemplateObject.create(Sorgular.create,id ,name, age);
                System.out.println("Kayıt İşlemi Gerçekleşmiştir");
        }

        public Kimlik getKimlik(Integer id) {

                Kimlik kimlik = jdbcTemplateObject.queryForObject(Sorgular.getKimlik, new
Object[] { id }, new KimlikRowMapper());
                return kimlik;
        }

        public List<Kimlik> listKimliks() {

                List<Kimlik> kimliklist = jdbcTemplateObject.query(Sorgular.listKimlik, new
KimlikRowMapper());
                return kimliklist;
```

```
    }

    public void delete(Integer id) {

        jdbcTemplateObject.update(Sorgular.delete, id);
        System.out.println("Kayıt Silinmiştir.");

    }

    public void update(Integer id, String adi) {

        jdbcTemplateObject.update(SQL, adi, id);
        System.out.println("Kayıt Güncellendi");

    }

}
```

## Hibernate

Hibernate Java geliştiriciler için geliştirilmiş bir ORM kütüphanesidir.

Nesne yönelimli modellere göre veritabanı ile olan ilişkiyi sağlayarak, veritabanı üzerinde yapılan işlemleri kolaylaştırmakla birlikte kurulan yapıyı da sağlamlaştırmaktadır.

Hibernate bir nesne/ilişkisel eşleme (Object/Relational Mapping) aracıdır. Burada nesne/ilişkisel eşleme terimi nesne modelindeki veri tanımlarının ilişkisel veri modeline eşleme (mapping) tekniğini ifade etmektedir.

Kullanacağımız entity(varlık) sınıfı ve tablo yapısı:

```
import javax.persistence.Column;
import javax.persistence.Entity;
```

```
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="Product")

public class Product {

 @Id
 @GeneratedValue(strategy=GenerationType.AUTO)
 @Column(name="id")
 private int id;
 @Column(name="categoryid")
 private int categoryId;

 @Column(name="name")
 private String name;

 @Column(name="label")
 private String details;

 @Column(name="price")
 private String price;

 @Column(name="url")
 private String url;
```

```
@Column(name="icon")
private String image;

public int getId() {
return id;
}
public void setId(int id) {
this.id = id;
}
public int getCategoryId() {
return categoryId;
}
public void setCategoryId(int categoryId) {
this.categoryId = categoryId;
}
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public String getDetails() {
return details;
}
public void setDetails(String details) {
this.details = details;
}
public String getPrice() {
return price;
```

```
 }
 public void setPrice(String price) {
 this.price = price;
 }
 public String getUrl() {
 return url;
 }
 public void setUrl(String url) {
 this.url = url;
 }
 public String getImage() {
 return image;
 }
 public void setImage(String image) {
 this.image = image;
 }
}
```

**Silme ve yeni kayıt ekleme metotları:**

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

import com.mesutpiskin.modal.Product;

public class CrudServicess {

 // Product tipinde verilen objeyi veritabanına ekler
```

```
public void InsertProduct(Product product) {

// Bir oturum başlatacağız oturumdan önce ise hibernate config dosyamızı
// belirteceğiz
@SuppressWarnings("deprecation")
SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();
Session session = sessionFactory.openSession();

// Transactionlar
System.out.println("Transaction başlatıldı.");
session.beginTransaction();

// aldığımız nesne kaydedilmesi için gönderildi
session.save(product);
session.getTransaction().commit();

System.out.println("Transaction tamamlandı.");
System.out.println("Veri kaydedildi.");

session.close();
sessionFactory.close();

}

// id değeri verilen satırı veritabanından siler
public void DeleteProduct(int productId) {
Product productObject;
// Bir oturum başlatacağız oturumdan önce ise hibernate config dosyamızı
// belirteceğiz
```

```
@SuppressWarnings("deprecation")
SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();
Session session = sessionFactory.openSession();

// Transactionlar
System.out.println("Transaction başlatıldı.");
session.beginTransaction();

productObject = (Product) session.load(Product.class, productId);
session.delete(productObject);
session.getTransaction().commit();

System.out.println("Transaction tamamlandı.");
System.out.println("VEri silindii.");

session.close();
sessionFactory.close();

}

}
```

Select işlemi için kullanacağımız sınıf ve metot:

```
import java.util.List;


import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

//Tablo üzerinde select sorgusu yürütür ve tablo verilerini bir list olarak döndürür
public class ListingServicess {

	public List<?> GetListFromTable() {
		@SuppressWarnings("deprecation")
		SessionFactory sessionFactory = new
Configuration().configure().buildSessionFactory();
		Session session = sessionFactory.openSession();

		Query query = session.createQuery("FROM Product");
		List<?> results = query.list();

		return results;
	}
```

```
// Tablo üzerinde select sorgusunu where koşulu ile yürütür ve id değerini
	// verdiğimiz satırı döndürür
	public List<?> GetListById(int id) {


		@SuppressWarnings("deprecation")
		SessionFactory sessionFactory = new
Configuration().configure().buildSessionFactory();
		Session session = sessionFactory.openSession();
		// Hibernate query language sorgusu
		String hql = "FROM Product P WHERE P.id = :id";
		Query query = session.createQuery(hql);
		// id parametresine değerini atıyoruz
		query.setParameter("id", id);
		List<?> results = query.list();
		return results;
	}


}
```

```
import java.util.List;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;


//Tablo üzerinde select sorgusu yürütür ve tablo verilerini bir list olarak döndürür
public class ListingServicess {

 public List<?> GetListFromTable() {
 @SuppressWarnings("deprecation")
 SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();
 Session session = sessionFactory.openSession();

Query query = session.createQuery("FROM Product");
 List<?> results = query.list();
 return results;
 }
 // Tablo üzerinde select sorgusunu where koşulu ile yürütür ve id değerini
 // verdiğimiz satırı döndürür
 public List<?> GetListById(int id) {

 @SuppressWarnings("deprecation")
 SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();
 Session session = sessionFactory.openSession();
```

```
 // Hibernate query language sorgusu
 String hql = "FROM Product P WHERE P.id = :id";
 Query query = session.createQuery(hql);
 // id parametresine değerini atıyoruz
 query.setParameter("id", id);
 List<?> results = query.list();
 return results;
 }
}
```

Bu sınıfları ve metotları kullanabilmek için bir test sınıfı yazalım:

```
import java.util.List;


import com.mesutpiskin.modal.Product;

public class Test {

 public static void main(String[] args) {

 //Parametreli select sorgusu metodumuzu çağıracağız
 //GetListFromTable metodunu çağırsak bize tüm verileri getirecekti yapı olarak aynı metotlar
```

```
ListingServicess listingservices=new ListingServicess();
//bize bir list döndürecek ve id olarak da 9 veriyoruz id si 9 olan veriyi döndürecektir
List<?> liste=listingservices.GetListById(9);
for (Object object : liste) {
//bu veriyi Product entity sınıfımız tipine cast ediyoruz
Product prodcut=new Product();
prodcut=(Product)object;
//sadece adını çağırıyoruz
System.out.println("ADI:"+prodcut.getName());

 }
 }
}
```