JDBC Nedir?

Java JDBC (**J**ava **D**ata**B**ase **C**onnectivity) 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.

Örneğin; MySQL veritabanında yer alan öğrenci listesi Oracle veritabanına taşınıp oradan alınmak istendiğinde sadece bağlantı cümlesinin değiştirilmesi yeterli olacaktır.

JDBC kullanımı

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

Veritabanı bağlantısı için öncelikle kullanılacak olan veritabanı sistemine ait bağlantı sürücüsünün projeye eklenmesi gerekir.

Bu işlem komut yorumlayıcısına jar dosyasının dahil edilmesi, IDE arayüzünde libraries bölümüne eklenmesi veya Maven gibi paket yöneticilerinin kullanımı ile yapılabilir.

JDBC İle CRUD Operasyonları

Ekleme:

```
public class JavaJDBC {


   public static void main(String[] args) {
      String url = "jdbc:mysql://localhost:3306/kisi";
      String user = "root";
      String password = "";


      try {
         Class.forName("com.mysql.jdbc.Driver");


         Connection connection = DriverManager.getConnection(url, user, password);


         Statement statement = connection.createStatement();


         String sql = "INSERT INTO "
               + "kisiler(kisi_adi, kisi_soyadi, kisi_eposta) "
               + "VALUES('Yusuf', 'SEZER', 'yusufsezer@mail.com')";


         System.out.println(statement.executeUpdate(sql) + " kayıt eklendi.");


         statement.close();
         connection.close();


      } catch (ClassNotFoundException | SQLException ex) {
         System.err.println(ex);
      }
   }
}
```

Silme:

```
// Diğer komutlar
String sql = "DELETE FROM kisiler WHERE kisi_eposta = 'yusufsezer@mail.com'";
System.out.println(statement.executeUpdate(sql) + " kayıt silindi.");
// Diğer komutlar
```

Güncelleme:

```
// Diğer komutlar
String sql = "UPDATE kisiler "
      + "SET kisi_adi = 'Yusuf Sefa', kisi_soyadi = 'SEZER' "
      + "WHERE kisi_eposta = 'yusufsezer@mail.com'";

System.out.println(statement.executeUpdate(sql) + " kayıt güncellendi.");
// Diğer komutlar
```

Listeleme:

```
// Diğer komutlar
ResultSet resultSet = statement.executeQuery("SELECT * FROM kisiler");
int sira;
String adi;
String soyadi;
String eposta;
while (resultSet.next()) {
   sira = resultSet.getInt(1);
   adi = resultSet.getString(2);
   soyadi = resultSet.getString(3);
   eposta = resultSet.getString(4);
   System.out.println(sira + " " + adi + " " + soyadi + ", " + eposta);}    // Diğer komutlar
```

JDBC TEMPLATE NEDİR

Yaptığımız yazılımlarda veriler ile iletişime geçmek için normal şartlar altında Java Database Connectivity kullanarak hallediyoruz. JDBC ile işlemlerimizi gerçekleştirirken bir bağlantı bilgileri tanımlayarak her veritabanı işlemlerinde bağlantı açmak, iş bittiğinde tekrar kapatmak gibi zorunda olduğumuz bu işlemleri defalarca tekrar etmek yapılmak istenen asıl işten koparıyor.

Spring Jdbc Template tam bu noktada devreye giriyor. Sizi asıl yapmanız gereken işe yoğunlaştırırken, veritabanı işlemleri için önceden yaptığımız tekrarlı işlemleri en aza indirip “otomatik” olarak yapılmasını sağlayabiliyor.

JDBC Template İle CRUD Operasyonları

```
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 = ?";

}
```

```
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");

}

}
```

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

	kimlikTemplate kTemplate = (kimlikTemplate) context.getBean("kimlikTemplate");



	//Kayıt Ekleme

	kTemplate.create(1,"Burak", "KUTBAY");

	//Kayıt Listeleme

	List<Kimlik> kimliks = kTemplate.listKimliks();

	for (Kimlik record : kimliks) {

		System.out.print("Id : " + record.getId());

		System.out.print(", Adı : " + record.getAdi());

		System.out.println(", Soyad : " + record.getSoyadi());

	}

	//Kayıt Güncelleme

	kTemplate.update(3, "Mehmet");

	//Kayıt Silme

	kTemplate.delete(3);



	}


}
```

HIBERNATE NEDİR?

Yazılım alanında en fazla yaptığımız işlemlerin başında veritabanı operasyonları geliyor. Verilerin saklanması ve saklanan verilerin gerektiğinde geri çağrılması, düzenlenmesi ve silinmesi gibi işlemleri sıklıkla yapıyoruz. Kimi zaman farklı veritabanları için farklı kodlara ve konfigürasyonlara ihtiyaç duyuyoruz. Bu kadar sık kullanılmasının yanında bir yazılım projesinde veritabanı çok değişiklik gösterebiliyor.

Yazılımın felsefesinde birtakım işleri otomatize etmek ve değişikliklere dirençsiz, süreklilik sahibi projeler ortaya koymak varken biz yazılımcılar da işleri kolaylaştıracak ve proje değişikliklerinde bize çok zahmet çıkarmayacak sistemlere ihtiyaç duyuyoruz. İşte bu alanda karşımıza ORM (Object Relational Mapping) yapısı çıkıyor.

ORM'i veritabanının uygulamadan soyutlandığı ve tabloların uygulama tarafından sınıflarla ifade edildiği bir yapı olarak düşünebiliriz. ORM sayesinde veritabanı operasyonlarını çok daha hızlı ve zahmetsizce yapıyoruz. Hibernate ise Java için geliştirilmiş bir ORM çözümüdür kısaca.

Hibernate İle CRUD Operasyonları

@Entity //ilk anotasyonumuz Entity. Bu anotasyon ile Hibernate'e bu classın orm için kullanılacağını belirtmiş oluyoruz.

@Table(name = "employees") // Table anotasyonu içerisinde ilişkilendireceğimiz tablonun ismini yazıyoruz

@AllArgsConstructor

@NoArgsConstructor

public class Employee {

@Id // Altındaki değişkenin bir id olduğunu yani veriyi ayırt edici bir sütun olduğunu belirtiyoruz

@Column(name="emp_no") // Sütunumuzun adını yazıyoruz

private int emp_no;

@Column(name="first_name") // Sütunumuzun adını yazıyoruz

private String first_name;

@Column(name="last_name") // Sütunumuzun adını yazıyoruz

private String last_name;

@Column(name = "gender") // Sütunumuzun adını yazıyoruz

private String gender;

//Constructor

public Employee(int emp_no, String first_name, String last_name, String gender) {

this.emp_no = emp_no;

this.first_name = first_name;

this.last_name = last_name;

this.gender = gender;

}

Session Oluşturmak

Hibernate ile session oluşturmak için SessionFactory classını kullanıyoruz.

```
public class Main {

	public static void main(String[] args) {

		SessionFactory factory = new Configuration()

				.configure("hibernate.conf.xml") //En başta oluşturduğumuz Configuration dosyası

				.addAnnotatedClass(Employee.class)

				.buildSessionFactory();

		Session session = factory.getCurrentSession(); //SessionFactory ile session oluşturuyoruz.

		try {

			session.beginTransaction();

			///Sorgularımızı burada yazıyoruz...

			session.getTransaction().commit();

		}finally {

			factory.close();

		}

	}}
```

Tablodan Tüm Verileri Çekmek

```
public class Main {


	public static void main(String[] args) {
		SessionFactory factory = new Configuration()
				.configure("hibernate.cfg.xml") //En başta oluşturduğumuz Configuration dosyası
				.addAnnotatedClass(Employee.class)
				.buildSessionFactory();


		Session session = factory.getCurrentSession(); //SessionFactory ile session oluşturuyoruz.


		try {
			session.beginTransaction();
			// Sorgumuzu yazıyoruz... getResultList metodu bize List türünde değer döndürür.
			List<Employee> employees = session.createQuery("from Employee").getResultList();
			session.getTransaction().commit();
			for (Employee e : employees) {
				//Listemizdeki Employee nesnelerinin first_name sütununu konsolda yazdırıyoruz
				System.out.println(e.getFirst_name());
			}
		}finally {
			factory.close();
		}
	}}
```

Koşullu Sorgularla Veri Çekmek

```
try {
	session.beginTransaction();
	// Employee e dedikten sonra tablo adı olarak e değişkenini kullanabiliriz. Kullanım şekli ise aşağıdaki gibi
	List<Employee> employees = session.createQuery("from Employee e where e.gender = 'M'").getResultList();
	session.getTransaction().commit();
	for (Employee e : employees) {
		//Listemizdeki Employee nesnelerinin first_name ve gender sütununu yazıyoruz
		System.out.println("Name: " + e.getFirst_name());
		System.out.println("Gender: " + e.getGender());
	}
}finally {
	factory.close();
}
```

Insert İşlemi

```
try {
	session.beginTransaction();

	Employee employee = new Employee(); // Employee sınıfından bir nesne oluşturuyoruz
	employee.setFirst_name("Süleyman"); // Set metodları ile bilgileri giriyoruz
	employee.setLast_name("Özarslan");
	employee.setGender("M");
	session.save(employee); // Save ile insert ediyoruz

	session.getTransaction().commit();
}finally {
	factory.close();}
```

Update İşlemi

```
try {
        session.beginTransaction();
        int guncellemek_istedigimiz_verinin_idsi = 0;
        // Güncellemek istediğimiz veriyi Idsi ile alıyoruz
        Employee e = session.get(Employee.class,guncellemek_istedigimiz_verinin_idsi);
        e.setFirst_name("Ahmet"); // İstediğimiz değişiklikleri yapıyoruz
        session.save(e); // Update ediyoruz
        session.getTransaction().commit();
        }finally {
                factory.close();
        }
```

Delete İşlemi

```
try {
        session.beginTransaction();
        int silmek_istedigimiz_verinin_idsi = 0;
        Employee e = session.get(Employee.class,silmek_istedigimiz_verinin_idsi);
        session.delete(e);
        session.getTransaction().commit();

       }
       finally {
                factory.close();
             }
```