|  |
| --- |
| **JCBC (Java Database Connectivity) (Java Veritabanı Bağlantısı)** |
| Java dilinde yazılmış programların, ilişkisel veritabanları ile iletişim kurabilmesi için kullanılan bir Java API sidir. JDBC, Java dilinde yazılmış sınıf ve arayüzler içermektedir. Veritabanı uygulamalarının Sadece Java ile gerçekleştirilmesini mümkün kılar. |
| **JDBC, Java ortamında SQL veritabanına erişim için bir araçtır.**  JDBC API’si, ilgili veritabanının JDBC Sürücüleri yardımıyla,Veritabanı ile bağlantı kurar,SQL ifadelerini gönderir ve Sonuçları işler. |
| **JDBC’DE TANIMLI SINIFLAR (CLASSES)**  • Driver = **getConnection**  • Bağlantı = **createStatement**  • Sql ifadesi çalıştırma = **execute, executeQuery, executeBatch, executeUpdate**  • Sonuçları alma = **next, getString, getInt, getDate, getMetaData**  • ResultSetMetadata = **getColumnCount, getColumnName, getColumnType** |
| **JDBC ADIMLARI**  1. Kullanılacak Veritabanı için doğru sürücüyü ekle,  2. Veritabanı ile iletişimi başlat,  3. SQL ifadeleri oluştur ve çalıştır (Select, insert/update/delete)  4. Gelen sonuçları işle, kaydet vb.,  5. Veritabanı bağlantısını bitir. |
| **1. DRIVER (SÜRÜCÜ) EKLEME**  • Driver’ı eklemek için aşağıdaki metot kullanılır.= **Class.forName(“driver class”)**  • Driver, seçilen veritabanına göre değişmektedir.=**oracle.jdbc.driver.OracleDriver**  **mysql.jdbc.Driver**  **org.postgresql.Driver**  • ÖRNEK: Class.forName("oracle.jdbc.driver.OracleDriver"); |
| **2. VERİTABANI BAĞLANTISI OLUŞTURMA**  • Veritabanı ile bağlantı oluşturabilmek için **getConnection()** metodu kullanılır.  • Bu metot, bir **Connection** nesnesi döndürür.  • **DriverManager.getConnection(url, user, pwd)**  • ÖRNEK1: "jdbc:oracle:thin:@localhost:1521/orcl", "hr", "oracle"  • ÖRNEK2: "jdbc:oracle:thin:@localhost:1521/ORCLCDB.localdomain", "murat", "1234"  • ÖRNEK=Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521/orcl", "hr", "oracle"); |
| **3. SQL İFADESİ OLUŞTURMA VE ÇALIŞTIRMA**  A) Bir SQL ifade oluşturmak için createStatement() metodu kullanılır.  **Statement st = con.createStatement();**  • **st,** SQL komutlarını yazabilmemize imkan tanıyan bir Statement nesnesidir.  B) SQL ifadelerini çalıştırmak için ise aşağıdaki Metotlar kullanılabilir.  • executeQuery() => SELECT ifadeleri için  • executeUpdate()=> INSERT, UPDATE, DELETE ifadeleri  • SELECT ifadesinin sonuç kümesi (satırlar/sutunlar) bir ResultSet nesnesi olarak döner.  ÖRNEK: ResultSet rs = st.executeQuery(“SELECT \* FROM personel”); |
| **4. SONUÇLARI İŞLE**  • ResultSet olarak alınan SQL sonuçlarının her bir satırına erişmek için bir döngü kullanılabilir.  • Döngü içerisinde dolaşmak için rs.next() metodu kullanılmalıdır.  **ResultSet rs = st.executeQuery("Select personel\_isim FROM personel");**  **while(rs.next()) { System.out.println("Personel Adi:" + rs.getString("personel\_isim")); }**  • Bir sütundaki değerleri almak için  String personelId = rs.getString(“personel\_id”);  Double maas = rs.getDouble(“maas”); |
| **5. BAĞLANTIYI BİTİR**  • ResultSet nesnesi kapat.= **rs.close();**  • Statement nesnesini kapat= **st.close();**  • Bağlantıyı kapat= **con.close();** |
| **import java.sql.\*; // JDBC metotlari icin kutuphane**  **public class JdbcQuery01 {**  **public static void main(String[] args) throws ClassNotFoundException, SQLException {**  **//1) Veritabani icin uygun Driver'ı ekle**  **Class.forName("oracle.jdbc.driver.OracleDriver");**  **//2) Veritabani baglantisi olustur**  **Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521/ORCLCDB.localdomain", "meryem", "1234");**  **//3) SQL komutlari icin bir Statement nesnesi olustur.**  **Statement st = con.createStatement();**  **//4)Sorgu ifadesini calistir. ( Personel tablosundaki id=101 olan personelin ismini sorgula)**  **ResultSet rs1 = st.executeQuery("select isim from personel where id=101");**  **//5)Sonuclari isle**  **while(rs1.next()) {**  **System.out.println("Personel Adi: " + rs1.getString("isim"));**  **}**  **//6) Olusturulan nesneleri bellekten kaldir.**  **st.close();**  **con.close();**  **rs1.close();**  **}**  **}** |

|  |
| --- |
| **Cucumber frameworkte JDBC Java DataBase Connectivity** |
| Java kodları JDBC API de JDBC Driver Manager kullanarak farklı databaselere gerekli olan ön ayarlamaları yapar. Biz bu şekilde test otomasyonu yaparız  Bir kayıt oluşturup bu kayıdın database de de var mı diye kaydının kontrol testi yapılır  Oluşturulacak,silinecek,işlem yapılacak test datalarının bilgileri tester’a verilir. |
| **Database Bağlanmak için Adımlar** Not=Biz testlerimizi javada yapiyoruz  1 ilgili kütüphaneyi POM dosyamıza ekleyelim (com.microsoft.sqlserver) dependency yi kurmamız gerekir |
| **2** Java klasoru altına **dbStepdefinitions** package ekleyelim ve altina **DbStepdefinitions** classi ekleyelim  package dbStepdefinitions; import io.cucumber.java.en.Given; import org.junit.Assert; import java.sql.\*; import java.util.ArrayList; import java.util.List; public class DbStepdefinitions {  String url="jdbc:sqlserver://184.168.194.58:1433;databaseName=qaconcorthotel; user=Ahmet\_User;password=Ahmet123!";  String username="Ahmet\_User";  String password="Ahmet123!";  Connection connection;  Statement statement;  ResultSet resultSet;  @Given("kullanici CHQA database'ine baglanir")  public void kullanici\_chqa\_database\_ine\_baglanir() throws SQLException {  connection=DriverManager.*getConnection*(url,username,password);  statement=connection.createStatement(ResultSet.*TYPE\_SCROLL\_INSENSITIVE*,   ResultSet.*CONCUR\_READ\_ONLY*); }  @Given("kullanici {string} tablosundaki {string} verilerini alir")  public void kullanici\_tablosundaki\_verilerini\_alir(String table, String field) throws SQLException {  *// SELECT Price FROM tHOTELROOM* String readQuery= "SELECT "+field+" FROM "+table;  resultSet=statement.executeQuery(readQuery); }  @Given("kullanici {string} sutunundaki verileri okur ve istedigi islemleri yapar")  public void kullanici\_sutunundaki\_verileri\_okur\_ve\_istedigi\_islemleri\_yapar(String field) throws SQLException {  resultSet.first();*// resultset iterator gibi calisir*  System.*out*.println(resultSet.getObject(field).toString());  *//resultSet.next() bir sonraki objeye gecer, sonraki eleman varsa true, yoksa false doner* System.*out*.println(resultSet.next());  System.*out*.println(resultSet.getObject(field).toString()); *//son objeye gider ve resultSet.next() methodunu calistirirsak bir sonraki eleman olamayacagi icin bize false doner* resultSet.last();  System.*out*.println(resultSet.next());*//false /\*tum listeyi yazdirmak istersek while loop ile birlikte resultSet.next() kullanabiliriz. cunku resultSet.next() bir sonraki obje var oldugu muddetce bize true dondurecek ve while loop calismaya devam edecek son obje ulastigimizda resultSet.next() false donecek ve while loop bitecek. Ancak biz 40. satirda son objeye gittigimiz icin while loop u calistirirsak hic bir sey yazdirmaz while loop calistirmadan once ilk objeye gitmemiz gerekir resultSet.first();//birinci objeye gittik while (resultSet.next()){//usteki satir birinci objeye goturdu ama resultSet.last(); sonraki objeye gecirdi  //dolayisiyla liste 2.fiyattan basladi  System.out.println(resultSet.getObject(field).toString()); }\*/ /\*bende sonuna kadar gitmiyor onun icin yoruma aldik  resultSet.absolute(0);  while (resultSet.next()){  System.out.println( resultSet.getObject(field).toString());}\*//varsa ilk 100 fiyati double olarak bir liste seklinde kaydedelim* int sayac=1;  resultSet.absolute(0);  List<Double>ilkYuzSayi=new ArrayList<>();  while (sayac<=100 && resultSet.next()){  ilkYuzSayi.add(resultSet.getDouble(field));  sayac++; }  System.*out*.println(ilkYuzSayi);  System.*out*.println(ilkYuzSayi.size());  *//7. fiyatin double olarak 620.0 oldugunu test edin* resultSet.absolute(7);  double yedinciSayi=resultSet.getDouble(field);  Assert.*assertTrue*(yedinciSayi==620);  System.*out*.println(yedinciSayi);}} |
| **3.** Runners paketi altinda **DbRunner** class’i olusturalim glue olarak yeni olusturdugumuz dbStepdefinitions paketini yazalim  package runners; import io.cucumber.junit.Cucumber; import io.cucumber.junit.CucumberOptions; import org.junit.runner.RunWith; @RunWith(Cucumber.class) @CucumberOptions (  plugin={"html:target\\cucumber-reports.html",  "json:target/json-reports/cucumber.json",  "junit:target/xml-report/cucumber.xml"},  features="src/test/resources/features",  glue="dbStepdefinitions",  tags="@db",  dryRun= false) public class DbRunner {} |
| **4.** features dosyasi altinda **US1012\_DbRead.feature** olusturalim  Feature: US1012 Kulklanici concort Hotel database'ini test eder  @db  Scenario: TC18 Concort Hotel Database Read Test  Given kullanici CHQA database'ine baglanir  And kullanici "tHOTELROOM" tablosundaki "Price" verilerini alir  And kullanici "Price" sutunundaki verileri okur ve istedigi islemleri yapar |
| **5-** Database baglanmak icin stepdefinition class’inda connection olusturmamiz gerekli. Ilk olarak instance olarak gerekli dediskenleri ve objeleri olusturalim  String url="jdbc:sqlserver://184.168.194.58:1433;databaseName=qaconcorthotel; user=Ahmet\_User;password=Ahmet123!"; String username="Ahmet\_User"; String password="Ahmet123!"; Connection connection; Statement statement; ResultSet resultSet; |
| **6-** Veritabanina baglanmak icin stepdefinition icinde connection’a deger atayalim  connection=DriverManager.*getConnection*(url,username,password); |
| **7-** QUERY leri calistirmak ve verileri almak icin statement’a deger atayalim  statement=connection.createStatement(ResultSet.*TYPE\_***Factory** *\_INSENSITIVE*, ResultSet.*CONCUR\_READ\_ONLY*); |
| **8-** olusturdugumuz ve deger atadigimiz statement objesini kullanarak sorgumuzu yapalim (resultset itaratör gibi davranır)  resultSet=statement.executeQuery("SELECT "+string2+" FROM dbo."+string);  veya  *// SELECT Price FROM tHOTELROOM* String readQuery= "SELECT "+field+" FROM "+table; resultSet=statement.executeQuery(readQuery); |
| **9-** 8.adimda name sutunundaki tum veriler resultSet’e eklendi. resultSet methodlarini kullanarak bu verilere ulasabiliriz  resultSet.getRow() => Hangi rowdasin  resultSet.first() => Ilk rowa git  resultSet.last => son row a git  resultSet.absolute(5) => 5inci row a git  resultSet.getObject("BookName") =>BookName row daki objeyi ver  resultSet.getString("BookName") =>BookName row daki degeri ver  resultSet.first();*// resultset iterator gibi calisir* System.*out*.println(resultSet.getObject(field).toString());  *//resultSet.next() bir sonraki objeye gecer, sonraki eleman varsa true, yoksa false doner* System.*out*.println(resultSet.next()); System.*out*.println(resultSet.getObject(field).toString());  *//son objeye gider ve resultSet.next() methodunu calistirirsak bir sonraki eleman olamayacagi icin bize false doner* resultSet.last(); System.*out*.println(resultSet.next());*//false*  *//tum listeyi yazdirmak istersek while loop ile birlikte resultSet.next() kullanabiliriz cunku resultSet.next() bir sonraki obje var oldugu muddetce bize true dondurecek ve while loop calismaya devam edecek son obje ulastigimizda resultSet.next() false donecek ve while loop bitecek. Ancak biz 40. satirda son objeye gittigimiz icin while loop u calistirirsak hic bir sey yazdirmaz while loop calistirmadan once ilk objeye gitmemiz gerekir* resultSet.first();*//birinci objeye gittik* while (resultSet.next()){*//usteki satir birinci objeye goturdu ama resultSet.last(); sonraki objeye gecirdi  //dolayisiyla liste 2.fiyattan basladi* System.*out*.println(resultSet.getObject(field).toString());  */\*bende sonuna kadar gitmiyor onun icin yoruma aldik resultSet.absolute(0);  while (resultSet.next()){  System.out.println( resultSet.getObject(field).toString()); }\*/  //varsa ilk 100 fiyati double olarak bir liste seklinde kaydedelim* int sayac=1;  resultSet.absolute(0);  List<Double>ilkYuzSayi=new ArrayList<>();  while (sayac<=100 && resultSet.next()){  ilkYuzSayi.add(resultSet.getDouble(field));  sayac++;}  System.*out*.println(ilkYuzSayi);  System.*out*.println(ilkYuzSayi.size());  *//7. fiyatin double olarak 620.0 oldugunu test edin* resultSet.absolute(7);  double yedinciSayi=resultSet.getDouble(field);  Assert.*assertTrue*(yedinciSayi==620);  System.*out*.println(yedinciSayi);}} |
| SCHEMA   * Bu bizim schema * Table’lar primary key ile birbirlerine baglidir * Isimler net anlasilabilir olmayabilir * Database tester’lar schema’yi database’i anlamak, query’leri ona gore yazmak icin kullanır |
| DBUTILS ILE BAGLANTI ve OKUMA   1. Database baglantisi icin   DBUtils.getConnection();  2) Verileri almak icin  String query= "SELECT "+field+" FROM "+tablo;  DBUtils.executeQuery(query);  3) Verileri okumak icin  DBUtils.getResultset().first();  Object ilk=DBUtils.getResultset().getObject(field);  System.out.println(ilk.toString()); |

