JDBC (Java DataBase Connectivity)

JDBC, Java ile bir veritabanına (Mysql, Oracle, PostGreSQL gibi) erişmek ve işlemler yapmak için program ve veritabanı arasında köprü görevi görür.

JDBC, herhangi bir veritabanına bağlanarak; SQL komutları ile verilere ulaşabileceğimiz bir yapıdır. Kullanılan veritabanına göre bağlantı çeşidini değiştirmeniz yeterli olur.

DataBase Drivers

MySQL: com.mysql.jdbc.Driver

Oracle SQL: oracle.cj.jdbc.driver.OracleDriver

PostgreSQL: org.postgresql.Driver

Microsoft SQL: com.microsoft.sqlserver.jdbc.SQLServerDriver

SQLite: org.sqlite.JDBC

JDBC (Java DataBase Connectivity)

Driver: Bu arayüz veritabanı sunucusu ile olan iletişimi idare eder. Genellikle bu nesneyi yöneten DriverManager nesnesi üzerinden erişim yapılır.

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

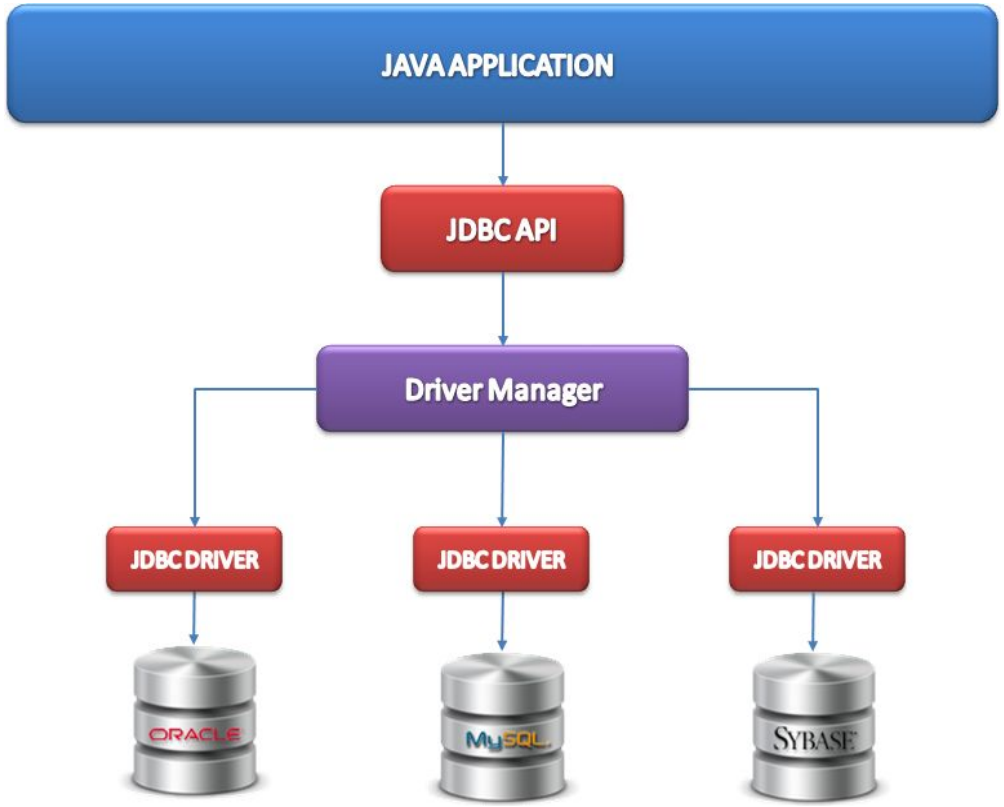
Connection: Bu arayüz, bir veritabanı ile iletişim kurmak için tüm yöntemleri içerir. Connection nesnesi iletişim bağlamını temsil eder, yani veritabanıyla yapılan tüm iletişim yalnızca bağlantı nesnesi aracılığıyla yapılır.

DriverManager: Bu sınıf, veritabanı sürücülerinin bir listesini yönetir. Java uygulamasından gelen bağlantı istekleri ile uygun veritabanı sürücüsünü eşleştirir ve bağlantı oluşturur.

```
Connection con = DriverManager.getConnection( );
```

Statement: Bu arayüz ile SQL ifadeleri veritabanına iletilir ve çalıştırılır.

```
Statement st = con.createStatement();
```

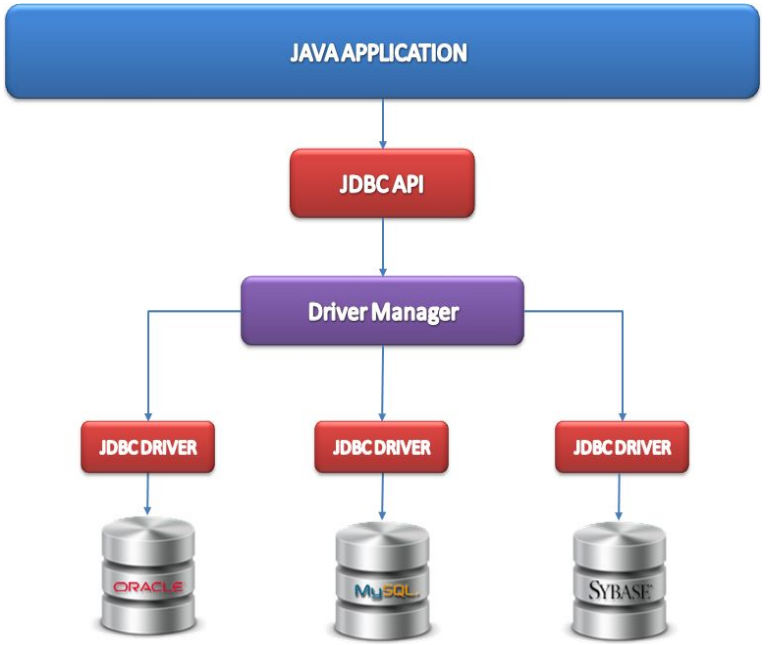


JDBC (Java DataBase Connectivity)

ResultSet: JDBC kullanarak veri çekme işlemi sonrasında veri listelemek için ResultSet sınıfı kullanılır. SQL sorgusu çalıştırıldıktan sonra veritabanından alınan verileri saklar. Verilerin arasında gitmemizi sağlar.

```
ResultSet veri = st.executeQuery("select * from ogrenciler");
```

Veriler üzerinde dolaşmak için next, first, last, previous, absolute gibi metotlara sahiptir.



SQL Exception: Bu sınıf, bir veritabanı uygulamasında meydana gelen hataları yönetir.

```
public static void main(String[] args) throws SQLException
```

JDBC (Java DataBase Connectivity)

ResultSetMetaData: Bu arayüzü kullanarak, ResultSet hakkında daha fazla bilgi alabiliriz. Her ResultSet nesnesi, bir ResultSetMetaData nesnesiyle ilişkilendirilir. Bu nesne, sütunun veri türü, sütun adı, sütun sayısı, tablo adı, şema adı gibi sütun özelliklerinin ayrıntılarına sahip olacaktır. ResultSet'in, getMetaData () yöntemini kullanarak ResultSetMetaData nesnesini alabiliriz.

```
ResultSet rs = ps.executeQuery();
ResultSetMetaData rsmd = rs.getMetaData();
```

JDBC (Java DataBase Connectivity)

PreparedStatement: Yazdığımız herhangi bir SQL sorgusunu Statement durumunda çalıştırdığımızda; bu sorgu her çalıştırıldığında veri tabanının belleğinde bu sorgunun bir örneği saklanır. Bu sorgunun binlerce kere çalıştırıldığını düşünürsek bu durum veritabanı performansını düşürür veya bağlantı kopmaları yaşanabilir.

Bu durumda PreparedStatement kullanmak faydalı olabilir. Herhangi bir SQL sorgusunu PreparedStatement durumunda çalıştırdığımızda; veri tabanında bu sorgusunun sadece bir kere örneği saklanır ve bin kere de çalıştırsak bu sorgunun veri tabanının belleğinde sadece bir örneği tutulur. Böylece PreparedStatement daha performanslı olur.

JDBC (Java DataBase Connectivity)

boolean execute(String SQL): Bir ResultSet nesnesi alınabiliyorsa, true boolean değerini döndürür; aksi halde false döndürür. SQL DDL deyimlerini yürütmek için veya gerçekten dinamik SQL kullanmanız gerektiğinde bu yöntemi kullanın.

int executeUpdate(String SQL): SQL ifadesinin yürütülmesinden etkilenen satır sayısını döndürür. Birkaç satırın etkilenmesini beklediğiniz SQL deyimlerini yürütmek için bu yöntemi kullanın – örneğin, bir INSERT, UPDATE veya DELETE ifadesi.

ResultSet executeQuery(String SQL): Bir ResultSet nesnesi döndürür. Bir SELECT deyiminde olduğu gibi, bir sonuç kümesi almak için bu yöntemi kullanın.