Subscribe to DeepL Pro to translate larger documents.
Visit www.DeepL.com/pro for more information.

Ünite I - Java Veritabanı Bağlantısı (JDBC)

- JDBC'ye Genel Bakış
- Bağlantı Sınıfı
- Meta Veri işlevi
- SQLException
- SQLUyarı
- Açıklama
- Sonuç Seti
- Diğer JDBC Sınıfları

JDBC'ye Genel Bakış

- JDBC API, özellikle İlişkisel Veritabanında depolanan veriler olmak üzere her türlü tablo verisine erişebilen bir Java API'sidir.
- JDBC, Windows, Mac OS ve UNIX'in çeşitli sürümleri gibi çeşitli platformlarda Java ile çalışır.
- JDBC, Java programlama dili ile çok çeşitli veritabanları arasında veritabanından bağımsız bağlantı için standart bir Java API'si olan Java Veritabanı Bağlantısı anlamına gelir.
- JDBC kütüphanesi, veritabanı kullanımıyla yaygın olarak ilişkilendirilen aşağıda belirtilen görevlerin her biri için API'ler içerir.
    - Bir veritabanına bağlantı kurma.
    - SQL veya MySQL deyimleri oluşturma.
    - Veritabanında SQL veya MySQL sorgularının çalıştırılması.
    - Ortaya çıkan kayıtların görüntülenmesi ve değiştirilmesi.

JDBC Uygulamaları

- Temel olarak JDBC, altta yatan bir veritabanına taşınabilir erişim sağlayan eksiksiz bir arayüz seti sağlayan bir spesifikasyondur. Java, aşağıdakiler gibi farklı türde yürütülebilir dosyalar yazmak için kullanılabilir,
    - Java Uygulamaları
    - Java Uygulamaları
    - Java Servletleri
    - Java Sunucu Sayfaları (JSP'ler)
    - Kurumsal JavaBeans (EJB'ler).
- Bu farklı yürütülebilir dosyaların tümü bir veritabanına erişmek için bir JDBC sürücüsü kullanabilir ve
  depolanan verilerden yararlanın.
- JDBC, ODBC ile aynı özellikleri sağlayarak Java programlarının veritabanından bağımsız kod içermesine olanak tanır.

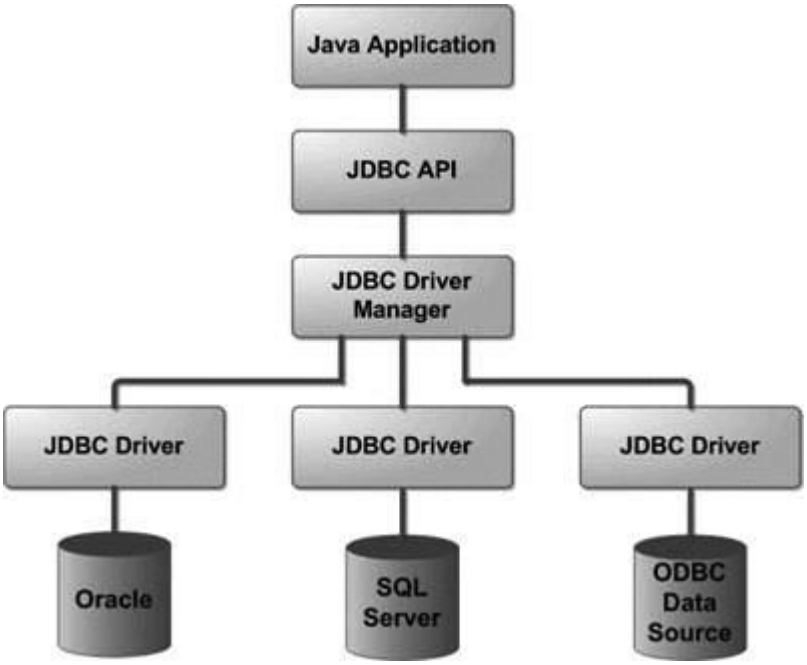
JDBC 4.0 Paketleri

- java.sql ve javax.sql, JDBC 4.0 için birincil paketlerdir.
- Veri kaynaklarınızla etkileşim için ana sınıfları sunar.
- Bu paketlerdeki yeni özellikler aşağıdaki değişiklikleri içermektedir,
    - Otomatik veritabanı sürücüsü yüklemesi.
    - İstisna işleme iyileştirmeleri.
    - Geliştirilmiş BLOB/CLOB işlevselliği.

- Bağlantı ve ifade arayüzü geliştirmeleri.
- Ulusal karakter seti desteği.
- SQL ROWID erişimi.
- SQL 2003 XML veri türü desteği.

JDBC Mimarisi

- JDBC API, veritabanı erişimi için hem iki katmanlı hem de üç katmanlı işleme modellerini destekler ancak genel olarak JDBC Mimarisi iki katmandan oluşur -
    - **JDBC API:** Bu, uygulama-JDBC Yöneticisi bağlantısını sağlar.
    - **JDBC Sürücü API'si:** Bu, JDBC Yöneticiden Sürücüye Bağlantıyı destekler.
- JDBC API, heterojen veritabanlarına şeffaf bağlantı sağlamak için bir sürücü yöneticisi ve veritabanına özgü sürücüler kullanır.
- JDBC sürücü yöneticisi, her veri kaynağına erişmek için doğru sürücünün kullanılmasını sağlar. Sürücü yöneticisi, birden fazla heterojen veritabanına bağlı birden fazla eşzamanlı sürücüyü destekleyebilir.
- Aşağıda, JDBC sürücüleri ve Java uygulamasına göre sürücü yöneticisinin konumunu gösteren mimari diyagram yer almaktadır



JDBC Bileşenleri

JDBC API aşağıdaki arayüzleri ve sınıfları sağlar -

- **DriverManager:** Bu sınıf veritabanı sürücülerinin bir listesini yönetir. İletişim alt protokolünü kullanarak java uygulamasından gelen bağlantı isteklerini uygun veritabanı sürücüsü ile eşleştirir. JDBC altında belirli bir alt protokolü tanıyan ilk sürücü, bir veritabanı Bağlantısı kurmak için kullanılacaktır.
- **Sürücü:** Bu arayüz veritabanı sunucusuyla iletişimi yönetir. Driver nesneleriyle çok nadiren doğrudan etkileşim kurarsınız. Bunun yerine, bu türdeki nesneleri yöneten DriverManager nesnelerini kullanırsınız. Ayrıca Driver nesneleriyle çalışma ile ilgili ayrıntıları da soyutlar.

- **Bağlantı:** Bu arayüz, bir veritabanıyla iletişim kurmak için tüm yöntemleri içerir. Bağlantı nesnesi iletişim bağlamını temsil eder, yani veritabanı ile tüm iletişim yalnızca bağlantı nesnesi aracılığıyla yapılır.
- **İfade:** SQL deyimlerini veritabanına göndermek için bu arayüzden oluşturulan nesneleri kullanırsınız. Bazı türetilmiş arayüzler, saklı yordamları çalıştırmanın yanı sıra parametreleri de kabul eder.
- **ResultSet:** Bu nesneler, Statement nesnelerini kullanarak bir SQL sorgusu yürüttükten sonra bir veritabanından alınan verileri tutar. Verileri arasında gezinmenizi sağlamak için bir yineleyici görevi görür.
- **SQLException:** Bu sınıf, bir veritabanı uygulamasında meydana gelen tüm hataları ele alır.

JDBC Sürücüsü

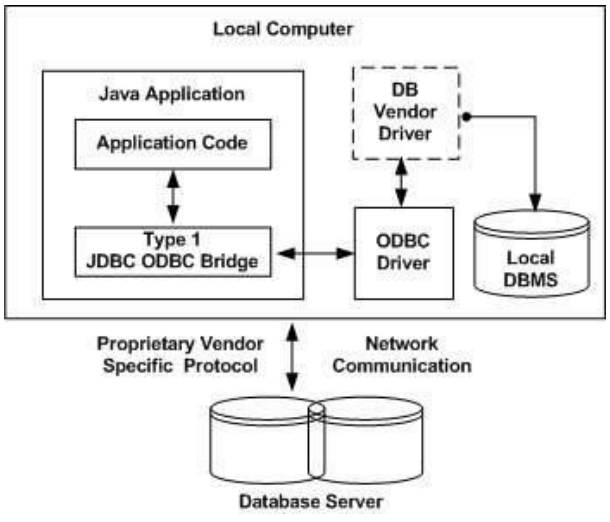
- JDBC sürücüleri, veritabanı sunucunuzla etkileşim kurmak için JDBC API'de tanımlanan arayüzleri uygular.
- Örneğin, JDBC sürücülerini kullanarak veritabanı bağlantıları açabilir ve SQL veya veritabanı komutları gönderip sonuçları Java ile alarak veritabanı ile etkileşime geçebilirsiniz.
- JDK ile birlikte gelen *Java.sql* paketi, davranışları tanımlanmış çeşitli sınıflar içerir ve bunların gerçek uygulamaları üçüncü taraf sürücülerde yapılır. Üçüncü taraf satıcılar kendi veritabanı sürücülerinde *java.sql.Driver* arayüzünü uygularlar.

JDBC Sürücü Türleri

- JDBC sürücü uygulamaları, Java'nın çalıştığı çok çeşitli işletim sistemleri ve donanım platformları nedeniyle farklılık gösterir.
- Sun, uygulama türlerini dört kategoriye ayırmıştır:
    - Tip 1: JDBC-ODBC Köprü Sürücüsü
    - Tip 2: JDBC-Native API
    - Tip 3: JDBC-Net saf Java
    - Tip 4: %100 Saf Java

Tip 1: JDBC-ODBC Köprü Sürücüsü

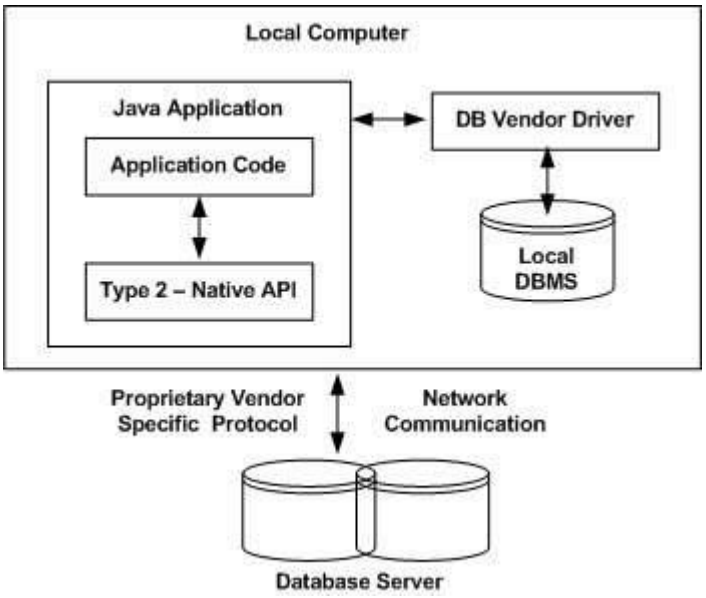
- Tip 1 sürücüde, her bir istemci makinede yüklü ODBC sürücülerine erişmek için bir JDBC köprüsü kullanılır.
- ODBC kullanımı, sisteminizde hedef veritabanını temsil eden bir Veri Kaynağı Adı (DSN) yapılandırılmasını gerektirir.



- Java ilk çıktığında, çoğu veritabanı yalnızca ODBC erişimini desteklediği için bu yararlı bir sürücüydü, ancak şimdi bu tür bir sürücü yalnızca deneysel kullanım için veya başka bir alternatif mevcut olmadığında önerilmektedir.
- JDK 1.2 ile birlikte gelen JDBC-ODBC Köprüsü bu tür sürücülere iyi bir örnektir.

Tip 2: JDBC-Native API

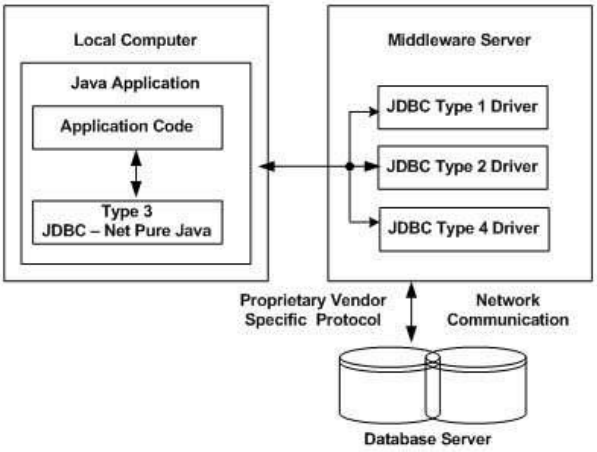
- Tip 2 sürücüde, JDBC API çağrıları veritabanına özgü yerel C/C++ API çağrılarına dönüştürülür.
- Bu sürücüler genellikle veritabanı satıcıları tarafından sağlanır ve JDBC-ODBC Köprüsü ile aynı şekilde kullanılır. Satıcıya özgü sürücü her istemci makineye yüklenmelidir.
- Veritabanını değiştirirsek, bir veritabanına özgü olduğu ve artık çoğunlukla kullanılmadıkları için yerel API'yi değiştirmemiz gerekir, ancak ODBC'nin ek yükünü ortadan kaldırdığı için bir Tip 2 sürücüsü ile biraz hız artışı elde edebilirsiniz.



- Oracle Çağrı Arayüzü (OCI) sürücüsü Tip 2 sürücüye bir örnektir.

Tip 3: JDBC-Net saf Java

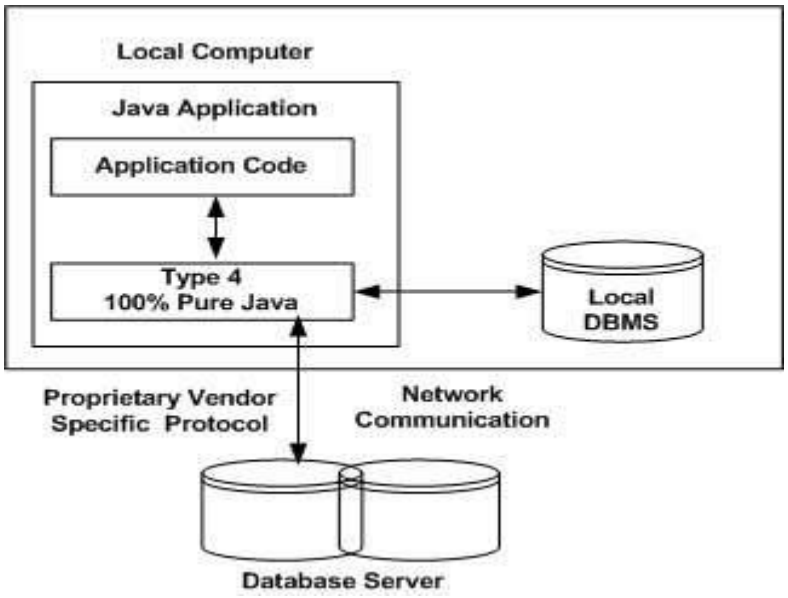
- Tip 3 sürücüde, veritabanlarına erişmek için üç katmanlı bir yaklaşım kullanılır. JDBC istemcileri, bir ara katman uygulama sunucusuyla iletişim kurmak için standart ağ soketlerini kullanır.



- Soket bilgileri daha sonra ara katman uygulama sunucusu tarafından DBMS'nin gerektirdiği çağrı formatına çevrilir ve veritabanı sunucusuna iletilir.
- Bu tür bir sürücü son derece esnektir, çünkü istemciye herhangi bir kod yüklenmesini gerektirmez ve tek bir sürücü aslında birden fazla veritabanına erişim sağlayabilir.
- Uygulama sunucusunu bir JDBC "proxy" olarak düşünebilirsiniz, yani istemci uygulaması için çağrılar yapar. Sonuç olarak, bu sürücü türünü etkili bir şekilde kullanmak için uygulama sunucusunun yapılandırması hakkında biraz bilgi sahibi olmanız gerekir.
- Uygulama sunucunuz veritabanı ile iletişim kurmak için Tip 1, 2 veya 4 sürücü kullanabilir, nüansları anlamak yardımcı olacaktır.

Tip 4: %100 Saf Java

- Tip 4 sürücüde, saf Java tabanlı bir sürücü soket bağlantısı aracılığıyla doğrudan satıcının veritabanıyla iletişim kurar. Bu, veritabanı için mevcut olan en yüksek performanslı sürücüdür ve genellikle satıcının kendisi tarafından sağlanır.
- Bu tür bir sürücü son derece esnektir, istemciye veya sunucuya özel bir yazılım yüklemenize gerek yoktur. Ayrıca, bu sürücüler dinamik olarak indirilebilir.



- MySQL'in Connector/J sürücüsü bir Tip 4 sürücüsüdür. Ağ protokollerinin tescilli doğası nedeniyle, veritabanı satıcıları genellikle tip 4 sürücüler sağlar.

JDBC Uygulaması Oluşturma

Bir JDBC uygulamasının oluşturulmasında aşağıdaki altı adım yer alır:

- **Paketleri içe aktarın:** Veritabanı programlaması için gereken JDBC sınıflarını içeren paketleri dahil etmenizi gerektirir. Çoğu zaman *import java.sql.** kullanmak yeterli olacaktır.
- **JDBC sürücüsünü kaydedin:** Veritabanı ile bir iletişim kanalı açabilmeniz için bir sürücüyü başlatmanızı gerektirir.
- **Bir bağlantı açın:** Veritabanı ile fiziksel bir bağlantıyı temsil eden bir Connection nesnesi oluşturmak için *DriverManager.getConnection()* yönteminin kullanılmasını gerektirir.
- **Bir sorguyu yürütür:** Bir SQL deyimi oluşturmak ve veritabanına göndermek için Statement türünde bir nesne kullanılmasını gerektirir.

- **Özüt Veri itibaren Sonuç Set:** Gerekli o sen kullanım sonuç kümesinden verileri almak için uygun *ResultSet.getXXX()* yöntemini kullanın.
- **Ortamı temizleyin:** JVM'nin çöp toplamasına güvenmek yerine tüm veritabanı kaynaklarının açıkça kapatılmasını gerektirir.

JDBC Bağlantı Sınıfı

JDBC bağlantısı kurmak için gerekli programlama oldukça basittir. İşte bu basit dört adım:

- **JDBC Paketlerini İçe Aktarın:** Java kodunuzda gerekli sınıfları içe aktarmak için Java programınıza içe aktarma deyimleri ekleyin.
- **JDBC Sürücüsünü Kaydedin:** Bu adım, JVM'nin JDBC isteklerinizi yerine getirebilmesi için istenen sürücü uygulamasını belleğe yüklemesine neden olur.
- **Veritabanı URL Formülasyonu:** Bu, bağlanmak istediğiniz veritabanına işaret eden düzgün biçimlendirilmiş bir adres oluşturmak içindir.
- **Bağlantı Nesnesi Oluşturun:** Son olarak, *DriverManager* nesnesinin *getConnection( )* yöntemini kullanarak gerçek veritabanı bağlantısı kurabilirsiniz.

JDBC Paketlerini İçe Aktarma

- **Import** deyimleri Java derleyicisine kodunuzda referans verdiğiniz sınıfları nerede bulacağını söyler ve kaynak kodunuzun en başına yerleştirilir.
- SQL tablolarındaki verileri seçmenize, eklemenize, güncellemenize ve silmenize olanak tanıyan standart JDBC paketini kullanmak için kaynak kodunuza aşağıdaki *içe aktarmaları* ekleyin

```
import java.sql.* ; // standart JDBC programları için
import java.math.* ; // BigDecimal ve BigInteger desteği için
```

JDBC Sürücüsünü Kaydetme

- Sürücüyü kullanmadan önce programınıza kaydetmeniz gerekir. Sürücünün kaydedilmesi, Oracle sürücüsünün sınıf dosyasının belleğe yüklenmesi ve böylece JDBC arabirimlerinin bir uygulaması olarak kullanılabilmesi işlemidir.
- Bu kaydı programınızda yalnızca bir kez yapmanız gerekir. Bir sürücüyü iki yoldan biriyle kaydedebilirsiniz.

1. Yaklaşım I - Class.forName()

- Bir sürücüyü kaydetmek için en yaygın yaklaşım, Java'nın **Class.forName()** yöntemini kullanarak sürücünün sınıf dosyasını belleğe dinamik olarak yüklemek ve otomatik olarak kaydetmektir. Bu yöntem tercih edilir çünkü sürücü kaydını yapılandırılabilir ve taşınabilir hale getirmenize olanak tanır.
- Aşağıdaki örnek Oracle sürücüsünü kaydetmek için Class.forName( ) işlevini kullanır -

```
dene {
  Class.forName("oracle.jdbc.driver.OracleDriver");
}
catch(ClassNotFoundException ex) { System.out.println("Hata:
  sürücü sınıfı yüklenemiyor!"); System.exit(1);
}
```

- Uyumlu olmayan JVM'lerde çalışmak için **getInstance()** yöntemini kullanabilirsiniz, ancak bu durumda aşağıdaki gibi iki ekstra İstisna için kod yazmanız gerekir

```
dene {
  Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
}
catch(ClassNotFoundException ex) { System.out.println("Hata:
  sürücü sınıfı yüklenemiyor!"); System.exit(1);
catch(IllegalAccessException ex) {
  System.out.println("Hata: yükleme sırasında erişim sorunu!");
  System.exit(2);
catch(InstantiationException ex) { System.out.println("Error:
  unable to instantiate driver!"); System.exit(3);
}
```

2. Yaklaşım II - DriverManager.registerDriver()

- Bu ikinci yaklaşım sen olabilir kullanım içinKAYIT a Sürücü, o için kullanım statik **DriverManager.registerDriver()** yöntemi.
- Microsoft tarafından sağlanan gibi JDK uyumlu olmayan bir JVM kullanıyorsanız *registerDriver()* yöntemini kullanmalısınız.
- Aşağıdaki örnek, Oracle sürücüsünü kaydetmek için registerDriver() işlevini kullanır

```
dene {
  Driver myDriver = new oracle.jdbc.driver.OracleDriver();
  DriverManager.registerDriver( myDriver );
}
catch(ClassNotFoundException ex) { System.out.println("Hata:
  sürücü sınıfı yüklenemiyor!"); System.exit(1);
}
```

Veritabanı URL Formülasyonu

- Sonra yüklü ve Sürücü, sen olabilir kurmak a bağlantı **DriverManager.getConnection()** yöntemini kullanarak.
- Aşağıda aşırı yüklenmiş üç DriverManager.getConnection() yöntemi listelenmiştir:
  - getConnection(String url)
  - getConnection(String url, Özellikler prop)
  - getConnection(String url, String user, String password)
- Burada her form bir veritabanı **URL'si** gerektirir. Veritabanı URL'si, veritabanınıza işaret eden bir adrestir.
  veritabanı.
- Bir veritabanı URL'sinin formüle edilmesi, bağlantı kurma ile ilgili sorunların çoğunun meydana geldiği yerdir.
- Aşağıdaki tabloda popüler JDBC sürücü adları ve veritabanı URL'si listelenmektedir.

RDBMS	**JDBC sürücü adı**	**URL biçimi**
MySQL	com.mysql.jdbc.Sürücü	**jdbc:mysql://hostname/** databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	**jdbc:oracle:thin:@hostname**:port Number:databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	**jdbc:db2**:hostname:port Numara/veritabanıAdı

Sybase	com.sybase.jdbc.SybDriver	**jdbc:sybase:Tds:**hostname: port Numara/veritabanıAdı

- URL biçiminde vurgulanan tüm kısım statiktir ve yalnızca kalan kısmı veritabanı kurulumunuza göre değiştirmeniz gerekir.

Bağlantı Nesnesi Oluşturma

- Bir bağlantı nesnesi oluşturmak için **DriverManager.getConnection()** yönteminin üç biçimi vardır.

1. Kullanıcı adı ve parola ile bir Veritabanı URL'si kullanma

- getConnection() işlevinin en yaygın kullanılan biçimi, bir veritabanı URL'si, bir *kullanıcı adı* ve bir *parola* iletmenizi gerektirir:
- Oracle'ın **thin** sürücüsünü kullandığınızı varsayarsak, URL'nin veritabanı kısmı için bir host:port:databaseName değeri belirleyeceksiniz.
- TCP/IP adresi 192.0.0.1 olan ve ana bilgisayar adı amrood olan bir ana bilgisayarınız varsa ve Oracle dinleyiciniz 1521 numaralı bağlantı noktasını dinleyecek şekilde yapılandırılmışsa ve veritabanınızın adı EMP ise, tam veritabanı URL'si şu şekilde olacaktır:

```
jdbc:oracle:thin:@amrood:1521:EMP
```

- Şimdi aşağıdaki gibi bir **Bağlantı** nesnesi elde etmek için getConnection() yöntemini uygun kullanıcı adı ve parola ile çağırın:

```
String URL = "jdbc:oracle:thin:@amrood:1521:EMP"; String
USER = "username";
String PASS = "password"
Bağlantı conn = DriverManager.getConnection(URL, USER, PASS);
```

2. Yalnızca Veritabanı URL'si Kullanma

- DriverManager.getConnection( ) yönteminin ikinci bir biçimi yalnızca bir veritabanı URL'si gerektirir:

```
DriverManager.getConnection(String url);
```

- Ancak bu durumda, veritabanı URL'si kullanıcı adı ve parolayı içerir ve aşağıdaki genel biçime sahiptir

```
jdbc:oracle:driver:username/password@database
```

- Dolayısıyla, yukarıdaki bağlantı aşağıdaki şekilde oluşturulabilir:

```
String URL = "jdbc:oracle:thin:username/password@amrood:1521:EMP"; Bağlantı conn
= DriverManager.getConnection(URL);
```

3. Veritabanı URL'si ve Özellikler Nesnesi Kullanma

- DriverManager.getConnection( ) yönteminin üçüncü bir biçimi, bir veritabanı URL'si ve bir Özellikler nesnesi -

```
DriverManager.getConnection(String url, Properties info);
```

- Özellikler nesnesi bir anahtar sözcük-değer çiftleri kümesi tutar. getConnection() yöntemine yapılan bir çağrı sırasında sürücü özelliklerini sürücüye iletmek için kullanılır.
- Önceki örneklerde yapılan bağlantının aynısını yapmak için aşağıdaki kodu kullanın -

```
import java.util.*;

String URL = "jdbc:oracle:thin:@amrood:1521:EMP"; Özellikler
info = new Özellikler( );
info.put( "kullanıcı", "kullanıcı adı" );
info.put( "şifre", "şifre" );

Bağlantı conn = DriverManager.getConnection(URL, info);
```

JDBC Bağlantılarını Kapatma

- JDBC programınızın sonunda, her bir veritabanı oturumunu sonlandırmak için veritabanına olan tüm bağlantıları açıkça kapatmanız gerekir. Ancak unutursanız, Java'nın çöp toplayıcısı eski nesneleri temizlediğinde bağlantıyı kapatacaktır.
- Özellikle veritabanı programlamada çöp toplamaya güvenmek çok kötü bir programlama uygulamasıdır. Bağlantıyı her zaman bağlantı nesnesiyle ilişkili close() yöntemiyle kapatmayı alışkanlık haline getirmelisiniz.
- Bir bağlantının kapatıldığından emin olmak için, kodunuzda bir 'finally' bloğu sağlayabilirsiniz. Bir *finally* bloğu, bir istisna oluşup oluşmadığına bakılmaksızın her zaman yürütülür.
- Yukarıda açılan bağlantıyı kapatmak için close() metodunu aşağıdaki gibi çağırmalısınız:

```
conn.close();
```

- Bir bağlantıyı açıkça kapatmak DBMS kaynaklarını korur ve bu da veritabanı yöneticinizi mutlu eder.

Metadata

- Genel olarak, veriler hakkındaki veriler meta veri olarak bilinir.
  - JDBC'de iki tür Meta veri vardır:
    - DatabaseMetaData
    - ResultsetMetaData

DatabaseMetaData

- DatabaseMetaData arayüzü, bağlandığınız veritabanı hakkında veritabanı adı, veritabanı sürücüsü sürümü, maksimum sütun uzunluğu vb. gibi bilgileri almak için yöntemler sağlar.
- Aşağıda DatabaseMetaData sınıfının bazı yöntemleri verilmiştir.

Yöntem	Açıklama
getDriverName()	Geçerli JDBC sürücüsünün adını alır
getDriverVersion()	Geçerli JDBC sürümünü alır
	sürücü
getUserName()	Kullanıcı adını alır.
getDatabaseProductName()	Geçerli veritabanının adını alır.

getDatabaseProductVersion()	Geçerli veritabanının sürümünü alır.
getNumericFunctions()	Bu veritabanıyla kullanılabilen sayısal işlevlerin listesini alır.
getStringFunctions()	Sayısal fonksiyonların listesini alır bu veritabanı ile kullanılabilir.
getSystemFunctions()	Bu veritabanı ile kullanılabilen sistem fonksiyonlarının listesini alır.
getTimeDateFunctions()	Bu veritabanıyla kullanılabilen saat ve tarih işlevlerinin listesini alır.
getURL()	Geçerli veritabanı için URL'yi alır.
supportsSavepoints()	Doğrular hava durumu ve güncel veritabanı kaydetme noktalarını destekler
supportsStoredProcedures()	Doğrular hava durumu ve güncel veritabanı saklı yordamları destekler.
supportsTransactions()	Doğrular hava durumu ve güncel veritabanı işlemleri destekler.

ResultsetMetaData

- ResultSetMetaData, elde edilen ResultSet nesnesi hakkında aşağıdaki gibi bilgiler sağlar sütun sayısı, sütunların adları, sütunların veri tipleri, tablonun adı vb...
- Aşağıda ResultSetMetaData sınıfının bazı metotları verilmiştir.

Yöntem	Açıklama
getColumnCount()	Geçerli ResultSet nesnesindeki sütun sayısını alır.
getColumnLabel()	Kullanım için sütunun önerilen adını alır.
getColumnName()	Sütunun adını alır.
getTableName()	Tablonun adını alır.

SQLException

- İstisna işleme, program tanımlı hatalar gibi istisnai durumları kontrollü bir şekilde ele almanızı sağlar.
- Bir istisna koşulu oluştuğunda, bir istisna atılır. Fırlatılan terimi, geçerli program yürütmesinin durduğu ve kontrolün en yakın uygulanabilir yakalama cümlesine yönlendirildiği anlamına gelir. Uygulanabilir bir catch cümlesi yoksa, programın yürütülmesi sona erer.
- JDBC İstisna işleme Java İstisna işlemeye çok benzer ancak JDBC için en sık karşılaşacağınız istisna **java.sql.SQLException'dı**r.

SQLException Yöntemleri

- Bir SQLException hem sürücüde hem de veritabanında meydana gelebilir. Böyle bir istisna oluştuğunda, catch cümlesine SQLException türünde bir nesne aktarılır.
- Aktarılan SQLException nesnesi, istisna hakkında ek bilgi almak için aşağıdaki yöntemlere sahiptir:

Yöntem	Açıklama
getErrorCode( )	ile ilişkili hata numarasını döndürür.
	İstisna.

getMessage( )	Sürücü tarafından işlenen bir hata için JDBC sürücüsünün hata mesajını alır veya Oracle Bir veritabanı hatası için hata numarası ve mesajı.
getSQLState( )	XOPEN SQLstate dizesini döndürür. Bir JDBC sürücü hatası için bu yöntemden hiçbir yararlı bilgi döndürülmez. Bir veritabanı hatası için beş basamaklı XOPEN SQLstate kodu döndürülür. Bu yöntemi null döndürebilir.
getNextException( )	İstisna zincirindeki bir sonraki Exception nesnesini döndürür.
printStackTrace( )	Geçerli istisnayı veya fırlatılabilir dosyayı yazdırır ve standart bir hata akışına geri izleme yapar.
printStackTrace(PrintStream s)	Bu throwable öğesini ve arka izini belirttiğiniz yazdırma akışına yazdırır.
printStackTrace(PrintWriter w)	Bu throwable öğesini ve geri izini belirttiğiniz yazıcıya yazdırır.

- Exception nesnesindeki bilgileri kullanarak bir istisnayı yakalayabilir ve programınıza uygun şekilde devam edebilirsiniz. İşte bir try bloğunun genel biçimi -

```
dene {
  // Riskli kodunuz bu küme parantezlerinin arasına giriyor!!!
}
catch(Exception ex) {
  // İstisna işleme kodunuz bunların arasına gider
  // küme parantezleri, istisna cümlesine benzer
  // bir PL/SQL bloğu içinde.
}
nihayet {
  // Her zaman çalıştırılması gereken kodunuz bunların arasına gider
  // küme parantezleri. Veritabanı bağlantısını kapatmak gibi.
}
```

SQLUyarı

- SQLWarning, veritabanı erişim uyarılarını tutan bir SQLException alt sınıfıdır.
- Uyarılar, istisnaların yaptığı gibi belirli bir uygulamanın yürütülmesini durdurmaz.
- Bir uyarı Connection nesnesi, Statement nesnesi, PreparedStatement ve CallableStatement nesneleri üzerinde veya **getWarnings** yöntemi kullanılarak ResultSet üzerinde alınabilir.
    - SQLWarning warning = stmt.getWarnings();

Örnek kaynak kodu

```
while (uyarı != null)
{
  System.out.println("Mesaj: " + warning.getMessage());
  System.out.println("SQLState: " + warning.getSQLState());
  System.out.println("Satıcı hata kodu: " + warning.getErrorCode()); warning =
  warning.getNextWarning();
}
```

JDBC İfadeleri

- Bir bağlantı elde edildikten sonra veritabanı ile etkileşime geçebiliriz.
- JDBC *Statement, CallableStatement* ve *PreparedStatement* arayüzleri, SQL veya PL/SQL komutları göndermenizi ve veritabanınızdan veri almanızı sağlayan yöntemleri ve özellikleri tanımlar.
- Ayrıca, bir veritabanında kullanılan Java ve SQL veri türleri arasındaki veri türü farklılıklarını kapatmaya yardımcı olan yöntemler de tanımlarlar.
- Aşağıdaki tablo, kullanılacak arayüze karar vermek için her arayüzün amacının bir özetini sunmaktadır.

Arayüzler	**Önerilen Kullanım**
Açıklama	Veritabanınıza genel amaçlı erişim için bunu kullanın. Çalışma zamanında statik SQL deyimleri kullanırken kullanışlıdır. Statement arayüzü parametre kabul edemez.
HazırlanmışBeyan	SQL deyimlerini birçok kez kullanmayı planlıyorsanız bunu kullanın. PreparedStatement arayüzü çalışma zamanında giriş parametrelerini kabul eder.
CallableStatement	Veritabanı saklı yordamlarına erişmek istediğinizde bunu kullanın. CallableStatement arayüzü çalışma zamanı giriş parametrelerini de kabul edebilir.

Deyim Nesneleri Deyim

Nesnesi Oluşturma

Bir SQL deyimini çalıştırmak için Statement nesnesini kullanmadan önce, aşağıdaki örnekte olduğu gibi Connection nesnesinin createStatement( ) yöntemini kullanarak bir tane oluşturmanız gerekir:

```
Statement stmt = null; try
{
  stmt = conn.createStatement( );
  . . .
}
catch (SQLException e) {
  . . .
}
nihayet {
  . . .
}
```

Bir Statement nesnesi oluşturduktan sonra, bu nesneyi üç execute yönteminden birini kullanarak bir SQL deyimini çalıştırmak için kullanabilirsiniz.

- **boolean execute (String SQL)**: Bir ResultSet nesnesi alınabiliyorsa true boolean değerini döndürür; aksi takdirde false döndürür. SQL DDL ifadelerini yürütmek için veya gerçekten dinamik SQL kullanmanız gerektiğinde bu yöntemi kullanın.
- **int executeUpdate (String SQL)**: SQL deyiminin yürütülmesinden etkilenen satır sayısını döndürür. Bu yöntemi, etkilenen satır sayısını almayı beklediğiniz SQL deyimlerini (örneğin, bir INSERT, UPDATE veya DELETE deyimi) yürütmek için kullanın.
- **ResultSet executeQuery (String SQL)**: Bir ResultSet nesnesi döndürür. SELECT deyiminde olduğu gibi bir sonuç kümesi almayı beklediğinizde bu yöntemi kullanın.

Kapanış Bildirimi Nesnesi

Veritabanı kaynaklarını korumak için bir Connection nesnesini kapattığınız gibi, aynı nedenle Statement nesnesini de kapatmalısınız.
close() yöntemine yapılan basit bir çağrı işinizi görecektir. Önce Connection nesnesini kapatırsanız, Statement nesnesi de kapanacaktır. Ancak, düzgün bir temizlik sağlamak için Statement nesnesini her zaman açıkça kapatmalısınız.

```
Statement stmt = null; try
{
  stmt = conn.createStatement( );
  . . .
}
catch (SQLException e) {
  . . .
}
finally { stmt.close();
}
```

PreparedStatement Nesneleri

PreparedStatement arayüzü Statement arayüzünü genişletir, bu da size genel bir Statement nesnesine göre birkaç avantajla birlikte ek işlevsellik sağlar.
Bu ifade size argümanları dinamik olarak sağlama esnekliği verir.

PreparedStatement Nesnesi Oluşturma

```
PreparedStatement pstmt = null; try
{
  String SQL = "Update Employees SET age = ? WHERE id = ?"; pstmt
  = conn.prepareStatement(SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
nihayet {
  . . .
}
```

- JDBC'deki tüm parametreler, parametre işaretçisi olarak bilinen **?** sembolü ile temsil edilir. SQL deyimini çalıştırmadan önce her parametre için değer sağlamanız gerekir.
- **setXXX()** yöntemleri parametrelere değer bağlar; burada **XXX**, giriş parametresine bağlamak istediğiniz değerin Java veri türünü temsil eder. Değerleri sağlamayı unutursanız, bir SQLException alırsınız.
- Her parametre işaretleyicisi sıra pozisyonu ile ifade edilir. İlk işaretçi 1. konumu, sonraki 2. konumu ve benzerlerini temsil eder. Bu yöntem, 0'dan başlayan Java dizi indislerinden farklıdır.

- **Statement nesnesinin** veritabanıyla etkileşime yönelik tüm yöntemleri (a) execute(), (b) executeQuery() ve (c) executeUpdate() PreparedStatement nesnesiyle de çalışır. Ancak, yöntemler parametreleri girebilen SQL deyimlerini kullanacak şekilde değiştirilir.

PreparedStatement Nesnesini Kapatma

- Bir Statement nesnesini kapattığınız gibi, aynı nedenle PreparedStatement nesnesini de kapatmalısınız.
- close() yöntemine yapılan basit bir çağrı işinizi görecektir. Önce Connection nesnesini kapatırsanız, PreparedStatement nesnesi de kapanacaktır. Ancak, düzgün bir temizlik sağlamak için PreparedStatement nesnesini her zaman açıkça kapatmalısınız.

```
PreparedStatement pstmt = null; try
{
  String SQL = "Update Employees SET age = ? WHERE id = ?"; pstmt
  = conn.prepareStatement(SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  pstmt.close();
}
```

CallableStatement Nesneleri

- Connection nesnesi Statement ve PreparedStatement nesnelerini oluşturduğu gibi, bir veritabanı saklı yordamına yapılan bir çağrıyı yürütmek için kullanılacak CallableStatement nesnesini de oluşturur.

CallableStatement Nesnesi Oluşturma

Aşağıdaki Oracle saklı yordamını çalıştırmanız gerektiğini varsayalım:

```
CREATE VEYA REPLACE PROCEDURE getEmpName
  (EMP_ID IN NUMBER, EMP_FIRST OUT VARCHAR) AS
BAŞLANGIÇ
  SELECT first INTO EMP_FIRST
  FROM Çalışanlar
  WHERE ID = EMP_ID;
BITTI;
```

***Not*:** Yukarıdaki saklı yordam Oracle için yazılmıştır, ancak biz MySQL veritabanı ile çalışıyoruz, bu nedenle EMP veritabanında oluşturmak için aynı saklı yordamı MySQL için aşağıdaki gibi yazalım

```
SINIRLAYICI $$

DROP PROCEDURE IF EXISTS `EMP`.`getEmpName` $$
CREATE PROCEDURE `EMP`.`getEmpName`
  (IN EMP_ID INT, OUT EMP_FIRST VARCHAR(255))
BEGIN
```

```
  SELECT first INTO EMP_FIRST
  FROM Çalışanlar WHERE
  ID = EMP_ID;
SON $$

SINIRLAYICI ;
```

- Üç tip parametre mevcuttur: IN, OUT ve INOUT.
- PreparedStatement nesnesi yalnızca IN parametresini kullanır.
- CallableStatement nesnesi her üçünü de kullanabilir. Tanımlar aşağıda verilmiştir:

Parametre	**Açıklama**
İÇİNDE	SQL deyimi oluşturulduğunda değeri bilinmeyen bir parametre. IN parametrelerine setXXX() yöntemleriyle değer bağlarsınız.
DIŞARI	Değeri, döndürdüğü SQL deyimi tarafından sağlanan bir parametre. getXXX() yöntemleriyleOUT parametrelerinden değerleri alırsınız.
GİRİŞ	Hem girdi hem de çıktı değerleri sağlayan bir parametre. Değişkenleri setXXX() yöntemleriyle bağlar ve değerleri getXXX() yöntemleriyle alırsınız.

- Aşağıdaki kod parçacığı, önceki saklı yordama dayalı bir **CallableStatement** nesnesini örneklemek için **Connection.prepareCall()** yönteminin nasıl kullanılacağını gösterir -

```
CallableStatement cstmt = null; try
{
  String SQL = "{call getEmpName (?, ?)}"; cstmt
  = conn.prepareCall (SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
nihayet {
  . . .
}
```

- SQL String değişkeni, parametre yer tutucularıyla birlikte saklı yordamı temsil eder.
- CallableStatement nesnelerini kullanmak, PreparedStatement nesnelerini kullanmaya çok benzer. Deyimi çalıştırmadan önce tüm parametrelere değer bağlamanız gerekir, aksi takdirde bir SQLException alırsınız.
- IN parametreleriniz varsa, bir PreparedStatement nesnesi için geçerli olan aynı kuralları ve teknikleri izleyin; bağladığınız Java veri türüne karşılık gelen setXXX() yöntemini kullanın.
- OUT ve INOUT parametrelerini kullandığınızda, registerOutParameter() adlı ek bir CallableStatement yöntemi kullanmanız gerekir. registerOutParameter() yöntemi, JDBC veri türünü saklı yordamın döndürmesi beklenen veri türüne bağlar.
- Saklı yordamınızı çağırdıktan sonra, uygun getXXX() yöntemiyle OUT parametresinden değeri alırsınız. Bu yöntem, SQL türünden alınan değeri bir Java veri türüne dönüştürür.

CallableStatement Nesnesini Kapatma

- Diğer Statement nesnesini kapattığınız gibi, aynı nedenle CallableStatement nesnesini de kapatmalısınız.
- close() yöntemine yapılan basit bir çağrı işinizi görecektir. Önce Connection nesnesini kapatırsanız, CallableStatement nesnesi de kapanacaktır. Ancak, düzgün bir temizlik sağlamak için CallableStatement nesnesini her zaman açıkça kapatmalısınız.

```
CallableStatement cstmt = null; try
{
  String SQL = "{call getEmpName (?, ?)}"; cstmt
  = conn.prepareCall (SQL);
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  cstmt.close();
}
```

JDBC Resutset

- Bir veritabanı sorgusundan veri okuyan SQL deyimleri, verileri bir sonuç kümesinde döndürür. SELECT deyimi, bir veritabanından satır seçmenin ve bunları bir sonuç kümesinde görüntülemenin standart yoludur. *java.sql.ResultSet* arayüzü, bir veritabanı sorgusunun sonuç kümesini temsil eder.
- Bir ResultSet nesnesi, sonuç kümesindeki geçerli satırı işaret eden bir imleci tutar. "Sonuç kümesi" terimi, bir ResultSet nesnesinde bulunan satır ve sütun verilerini ifade eder.
- ResultSet arayüzünün yöntemleri üç kategoriye ayrılabilir -
    - **Gezinme yöntemleri:** İmleci hareket ettirmek için kullanılır.
    - **Get yöntemleri:** İmleç tarafından işaret edilen geçerli satırın sütunlarındaki verileri görüntülemek için kullanılır.
    - **Güncelleme yöntemleri:** Geçerli satırın sütunlarındaki verileri güncellemek için kullanılır. Güncellemeler daha sonra altta yatan veritabanında da güncellenebilir.
- İmleç, ResultSet'in özelliklerine bağlı olarak hareket edebilir. Bu özellikler, ResultSet'i oluşturan ilgili Statement oluşturulduğunda belirlenir.
- JDBC, istenen ResultSet ile deyimler oluşturmak için aşağıdaki bağlantı yöntemlerini sağlar -
    - **createStatement(int RSType, int RSConcurrency);**
    - **prepareStatement(String SQL, int RSType, int RSConcurrency);**
    - **prepareCall(String sql, int RSType, int RSConcurrency);**
- İlk bağımsız değişken bir ResultSet nesnesinin türünü belirtir ve ikinci bağımsız değişken, bir sonuç kümesinin salt okunur mu yoksa güncellenebilir mi olduğunu belirtmek için kullanılan iki ResultSet sabitinden biridir.

ResultSet Türleri

- Olası RSType aşağıda verilmiştir. Herhangi bir ResultSet türü belirtmezseniz, otomatik olarak TYPE_FORWARD_ONLY olan bir tür alırsınız.

Tip	Açıklama
ResultSet.TYPE_FORWARD_ONLY	İmleç sonuç kümesinde yalnızca ileriye doğru hareket edebilir.
ResultSet.TYPE_SCROLL_INSENSITIVE	İmleç ileri ve geri kaydırılabilir ve sonuç kümesi, sonuç kümesi oluşturulduktan sonra veritabanında başkaları tarafından yapılan değişikliklere duyarlı değildir.
ResultSet.TYPE_SCROLL_SENSITIVE.	İmleç ileri ve geri kaydırılabilir ve sonuç kümesi, sonuç kümesi oluşturulduktan sonra veritabanında başkaları tarafından yapılan değişikliklere duyarlıdır.

ResultSet'in Eşzamanlılığı

- Olası RSConcurrency aşağıda verilmiştir. Herhangi bir Eşzamanlılık türü belirtmezseniz, otomatik olarak CONCUR_READ_ONLY olan bir tür alırsınız.

Eşzamanlılık	Açıklama
ResultSet.CONCUR_READ_ONLY	Salt okunur bir sonuç kümesi oluşturur. Bu varsayılandır
ResultSet.CONCUR_UPDATABLE	Güncellenebilir bir sonuç kümesi oluşturur.

- Yalnızca ileriye dönük, salt okunur bir ResultSet nesnesi oluşturmak için -

```
dene {
  Statement stmt = conn.createStatement(
                   ResultSet.TYPE_FORWARD_ONLY,
                   ResultSet.CONCUR_READ_ONLY);
}
catch(Exception ex) {
  ....
}
nihayet {
  ....
}
```

Sonuç Kümesinde Gezinme

- ResultSet arayüzünde imleci hareket ettirmeyi içeren birkaç yöntem vardır

S. Hayır	Yöntemler & Açıklama
1	**public void beforeFirst() SQLException atar:** İmleci ilk satırın hemen öncesine taşır.
2	**public void afterLast() throws SQLException** İmleci son satırdan hemen sonraya taşır.
3	**public boolean first() throws SQLException** İmleci ilk satıra taşır.
4	**public void last() throws SQLException** İmleci son satıra taşır.
5	**public boolean absolute(int row) throws SQLException** İmleci belirtilen satıra taşır.
6	**public boolean relative(int row) throws SQLException**
	İmleci o anda işaret ettiği yerden verilen sayıda satır ileri veya geri taşır.

7	**public boolean previous() throws SQLException** İmleci bir önceki satıra taşır. Önceki satır sonuç kümesinin dışındaysa bu yöntem false döndürür.
8	**public boolean next() throws SQLException** İmleci bir sonraki satıra taşır. Sonuç kümesinde başka satır yoksa bu yöntem false döndürür.
9	**public int getRow() throws SQLException** İmlecin işaret ettiği satır numarasını döndürür.
10	**public moveToInsertRow() SQLException fırlatır** İmleci, sonuç kümesinde veritabanına yeni bir satır eklemek için kullanılabilecek özel bir satıra taşır. Geçerli imleç konumu hatırlanır.
11	**public moveToCurrentRow() throws SQLException** İmleç şu anda ekleme satırındaysa imleci geçerli satıra geri taşır; aksi takdirde bu yöntem hiçbir şey yapmaz

Sonuç Kümesini Görüntüleme

- ResultSet arayüzü, geçerli satırın verilerini almak için düzinelerce yöntem içerir.
- Olası veri türlerinin her biri için bir get yöntemi vardır ve her get yönteminin iki versiyonlar -
    - Bir sütun adı alan bir sütun.
    - Bir sütun dizini alan bir tane.
- Örneğin, görüntülemek istediğiniz sütun bir int değeri içeriyorsa, ResultSet'in getInt() yöntemlerinden birini kullanmanız gerekir

S. Hayır	Yöntemler & Açıklama
1	**public int getInt(String columnName) throws SQLException** Geçerli satırda columnName adlı sütundaki int değerini döndürür.
2	**public int getInt(int columnIndex) throws SQLException** Belirtilen sütun indeksindeki geçerli satırdaki int değerini döndürür. Sütun indeksi 1'den başlar, yani bir satırın ilk sütunu 1, bir satırın ikinci sütunu 2 ve bu şekilde devam eder.

- Benzer şekilde, ResultSet arayüzünde sekiz Java ilkel türünün her biri için get yöntemlerinin yanı sıra java.lang.String, java.lang.Object ve java.net.URL gibi yaygın türler de vardır.
- SQL veri türleri java.sql.Date, java.sql.Time, java.sql.TimeStamp, java.sql.Clob ve java.sql.Blob almak için yöntemler de vardır. Bu SQL veri türlerini kullanma hakkında daha fazla bilgi için belgelere bakın.

Sonuç Kümesini Güncelleme

- ResultSet arayüzü, bir sonuç kümesinin verilerini güncellemek için bir güncelleme yöntemleri koleksiyonu içerir.
- Get yöntemlerinde olduğu gibi, her veri türü için iki güncelleme yöntemi vardır -
    - Bir sütun adı alan bir sütun.
    - Bir sütun dizini alan bir tane.
- Örneğin, bir sonuç kümesinin geçerli satırındaki bir String sütununu güncellemek için aşağıdaki updateString() yöntemlerinden birini kullanırsınız:

S. Hayır	Yöntemler & Açıklama
1	**public void updateString(int columnIndex, String s) throws SQLException** Belirtilen sütundaki String'i s değeriyle değiştirir.
2	**public void updateString(String columnName, String s) throws SQLException** Sütunun dizin yerine adıyla belirtilmesi dışında önceki yönteme benzer.

- Sekiz ilkel veri türünün yanı sıra String, Object, URL ve java.sql paketindeki SQL veri türleri için güncelleme yöntemleri vardır.
- Sonuç kümesindeki bir satırın güncellenmesi, ResultSet nesnesindeki geçerli satırın sütunlarını değiştirir, ancak altta yatan veritabanında değiştirmez. Veritabanındaki satırda yaptığınız değişiklikleri güncellemek için aşağıdaki yöntemlerden birini çağırmanız gerekir.

S. Hayır	Yöntemler & Açıklama
1	**public void updateRow()** Veritabanındaki ilgili satırı güncelleyerek geçerli satırı günceller.
2	**public void deleteRow()** Geçerli satırı veritabanından siler
3	**public void refreshRow()** Veritabanındaki son değişiklikleri yansıtmak için sonuç kümesindeki verileri yeniler.
4	**public void cancelRowUpdates()** Geçerli satırda yapılan tüm güncellemeleri iptal eder.
5	**public void insertRow()** Veritabanına bir satır ekler. Bu yöntem yalnızca imleç ekleme satırını gösterdiğinde çağrılabilir.

Diğer JDBC Sınıfları
Tarih
Zaman Zaman
Damgası