Java ile Veritabanı Programlama

Amaç

Kurumsal ya da kişisel uğraş olarak geliştirdiğimiz uygulamaların büyük bir bölümünde **kalıcı bilgi** gereksinimi vardır. Bu bilgi belki müşteri, belki harcama belki de arkadaşlarımızın iletişim bilgileri olabilir.
Bilgilerin çeşitli veritabanlarında istediğimiz biçimde saklanması ve gerektiğinde bu bilgilere aynı biçimde, hızlı, doğru ve düzgün olarak erişim, sağlam bir altyapı ile mümkündür. Java bu sağlam altyapıyı size kendiliğinden ve kolaylıkla **JDBC** arayüzü ile sağlamaktadır. JDBC arayüzü, java uygulaması ile veritabanı işlemleri yapılmasına olanak sağlayan **ilk** ve **en yaygın java teknolojisi**dir.

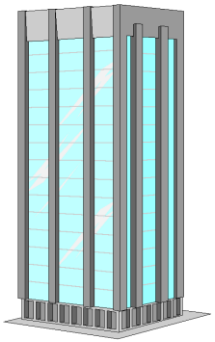
JDBC uygulama arayüzü ile yapabilecekleriniz **hayalleriniz** ve **veritabanınızın yetenekleri** ile sınırlıdır.

Amaç - 2

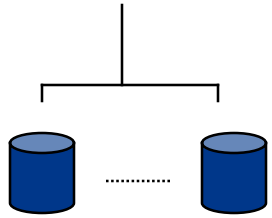
Teller

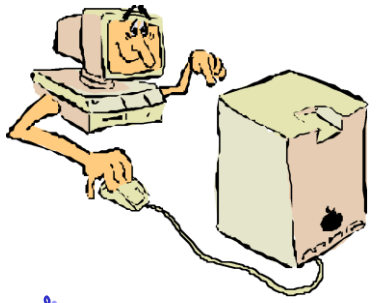
Telefon Bankacılığı

ATM



Banka





İnternet Bankacılığı

Giriş - 1

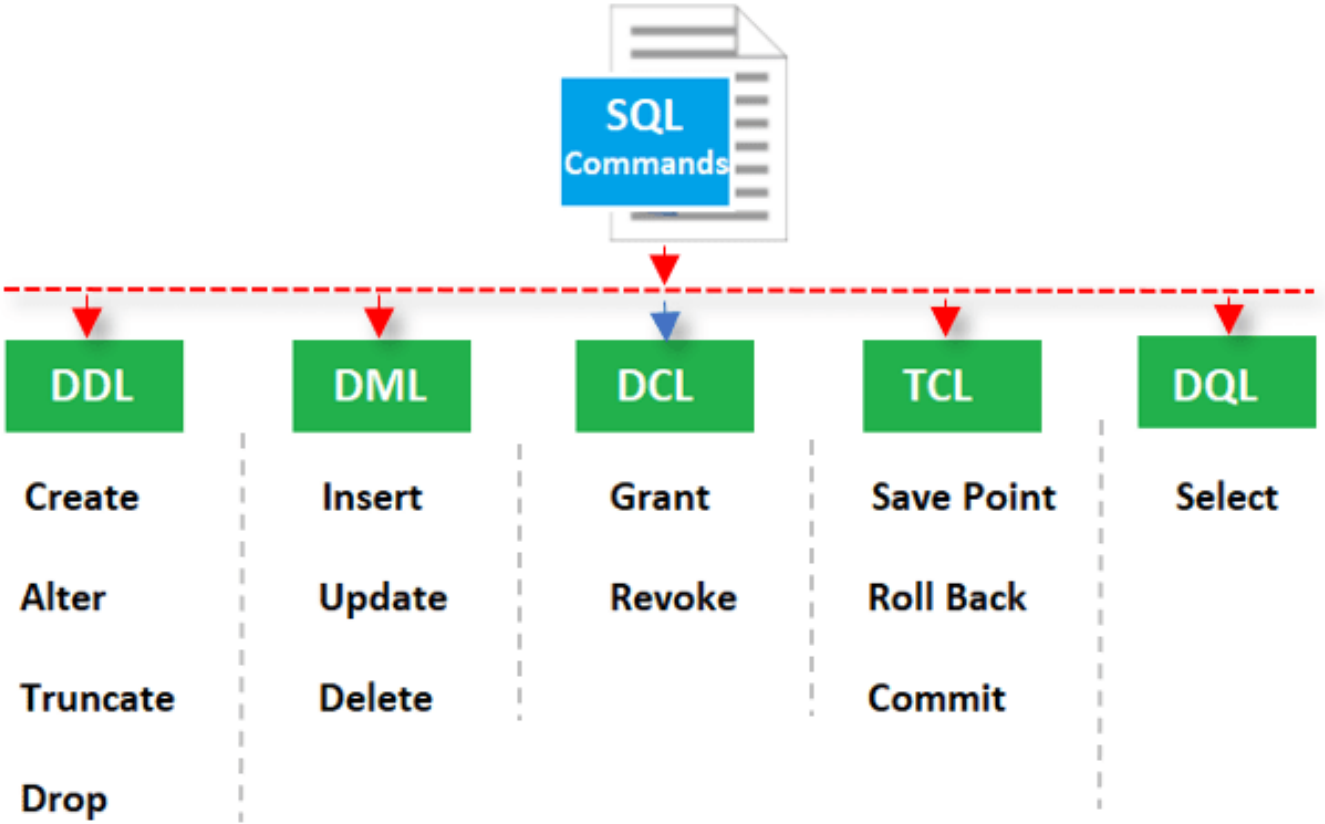
- VTYS
  - ...
  - **SQL** : veritabanı dili
  - ...
- Java
  - ...
  - **JDBC** : veritabanına erişim ve veritabanı işlemleri için kullanılır
  - ...

Giriş - 2

- **SQL** - Structured Query Language
- 70 li yıllarda IBM tarafından ilk defa tanımlandı
- İlk ticari uyarlama Oracle 1979 yılında geliştirdi
- 1992 yılında ANSI/ISO standartları geldi, SQL92 ya da SQL2
- 1999 yılında yeni standartlar eklendi, SQL99 ya da SQL3
- 2003, 2006, 2008, 2011, 2016, 2019
- …

Giriş - 3

Beş kategori; DDL, DML, DCL, TCL ve DQL



Giriş - 4

- SQL ifadeleri uygulama programında iki şekilde yer alır
  - Call Level Interface (**CLI**): Uygulama programı tamamen programlama dili ile geliştirilmiştir
    - SQL ifadeleri metin değişkenlerde tutulur ve metodlara parametre olarak gönderilir

    Örnek : **JDBC**
  - Statement Level Interface (**SLI**): Uygulama programı programlama dili ve SQL ifadelerinden oluşur

    Örnek : **SQLJ**

Giriş - 5

Java VTYS erişim seçenekleri

- JDBC
- SQLJ
- J2EE Entity Beans
- Object-Relational Mapping Araçları
  - Hibernate
  - JDO
  - ...

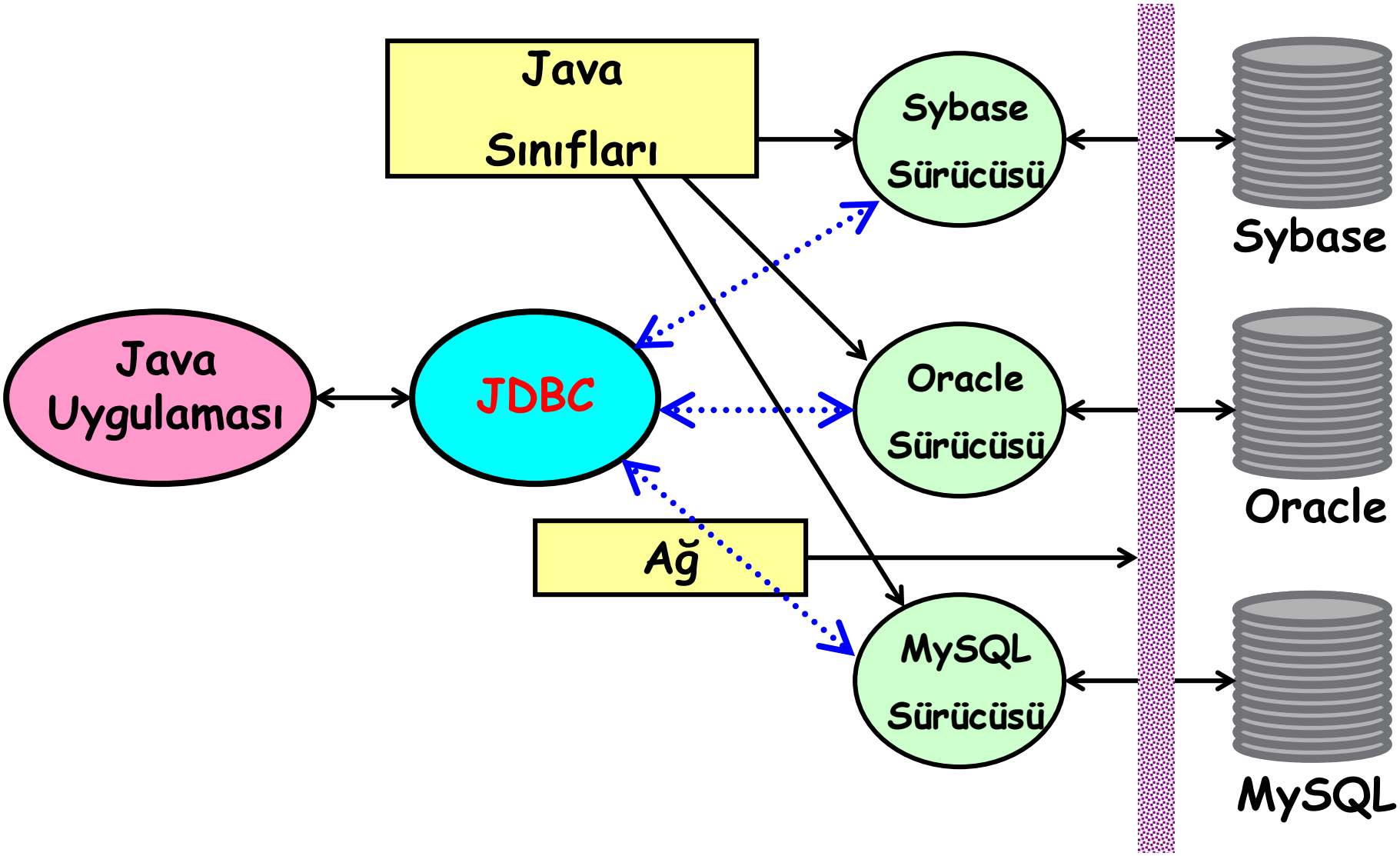
JDBC - 1

- JDBC, java uygulamalarından veritabanına erişmek ve veritabanı işlemleri için kullanılır
- Veri, veritabanından java nesnesine ya da java nesnesinden veritabanına taşınabilir
  - veritabanları gelişmiş indeks yapısına sahiptir ve arama yetenekleri güçlüdür
  - nesneler daha esnek yapıdadır ve hesap işlerine daha uygundur

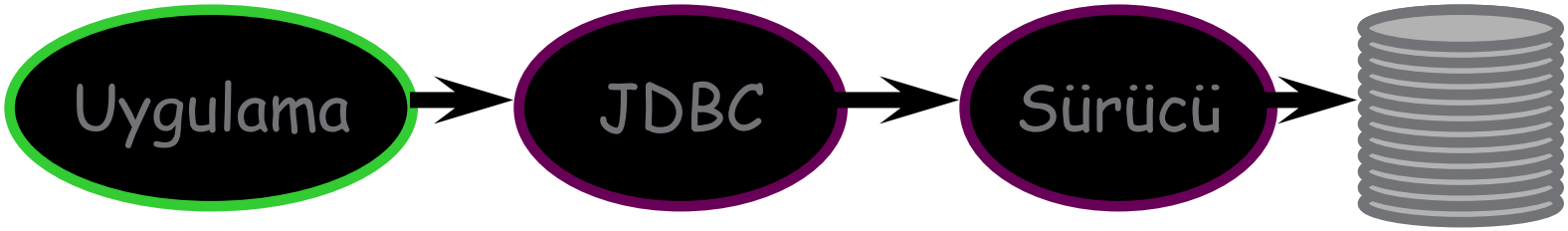
JDBC - 2

- **JDBC** — Java Database Connectivity
- **JDBC 1.0**, JDK 1.1, 1996
- **JDBC 2.0**, JDK 1.2, 1998
- **JDBC 3.0**, JDK 1.4, JSR 54, 2001
- **JDBC 4.0**, Java SE 6, JSR 221, 2006
- **JDBC 4.1**, Java SE 7, JSR 221, 2011
- **JDBC 4.2**, Java SE 8, JSR 221, 2014
- **JDBC 4.3**, Java SE 9, JSR 221, 2017

JDBC Mimarisi - 1



JDBC Mimarisi - 2



- Java kodu JDBC kütüphanesini çağırır
- JDBC sürücüyü yükler
- Sürücü ait olduğu veritabanı ile konuşur
- Birden fazla farklı veritabanı erişimi için farklı sürücüler gerekir
- Ideal: VTYS değişse bile uygulama etkilenmemeli

JDBC Sürücü Tipleri - 1

- **Tip 1 (JDBC-ODBC Bridge Technology)**

  Windows platformunda popüler olan ve Microsoft tarafından geliştirilmiş olan ODBC (Open Database Connectivity) standart SQL işlemleri için kullanılan bir API'dir. Bu tip sürücüler, gelen JDBC istemlerini ODBC komutlarına çevirirler ve geri gelen cevapları tekrar JDBC ve Java nesnelerine çevirerek istemciye sunarlar.

- **Tip 2 (JNI drivers for C/C++ connection libraries)**

  Bu tip sürücüler bir kısmı Java, diğer kısmı veritabanına ve platforma özel teknolojiler kullanırlar. 1. tip sürücülerde olduğu gibi bu tip sürücülerde de her istemciye platforma bağlı bir kurulum yapmak gerekir.

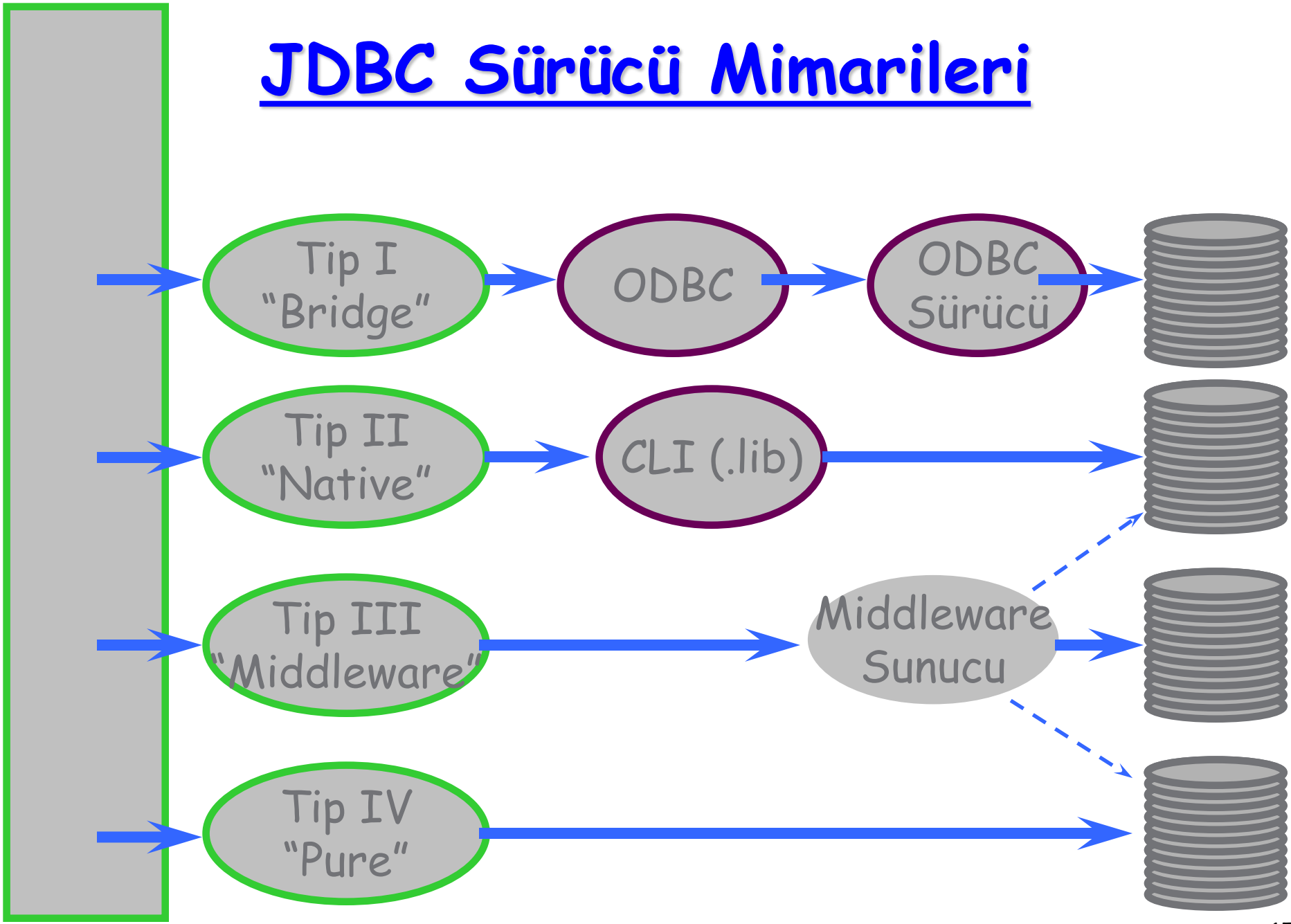
JDBC Sürücü Tipleri - 2

- **Tip 3 (Socket-level Middleware Translator)**

  En esnek mimariye sahip olan 3. tip JDBC sürücüleri, ortakat (middleware) yazılımlarından destek alırlar. Bu tip sürücüler birden fazla istemciyi birden fazla veritabanı sunucusuna bağlayabilirler. Genellikle, J2EE uygulama sunucuları, popüler veritabanları için özel, yüksek performanslı sürücüleri bulundururlar.
- **Tip 4 (Pure Java-DBMS driver)**

  2. Tip sürücülere alternatif olarak geliştirilmişlerdir. En büyük avantajları 100% Java yazılımları olup, istemci katında fazladan bir kurulum yapmayı gerektirmemeleridir. Bu tür sürücüler, doğrudan veritabanına açılan Socketler aracılığıyla, veritabanına özel protokolleri kullanarak veri iletişimi sağlarlar.

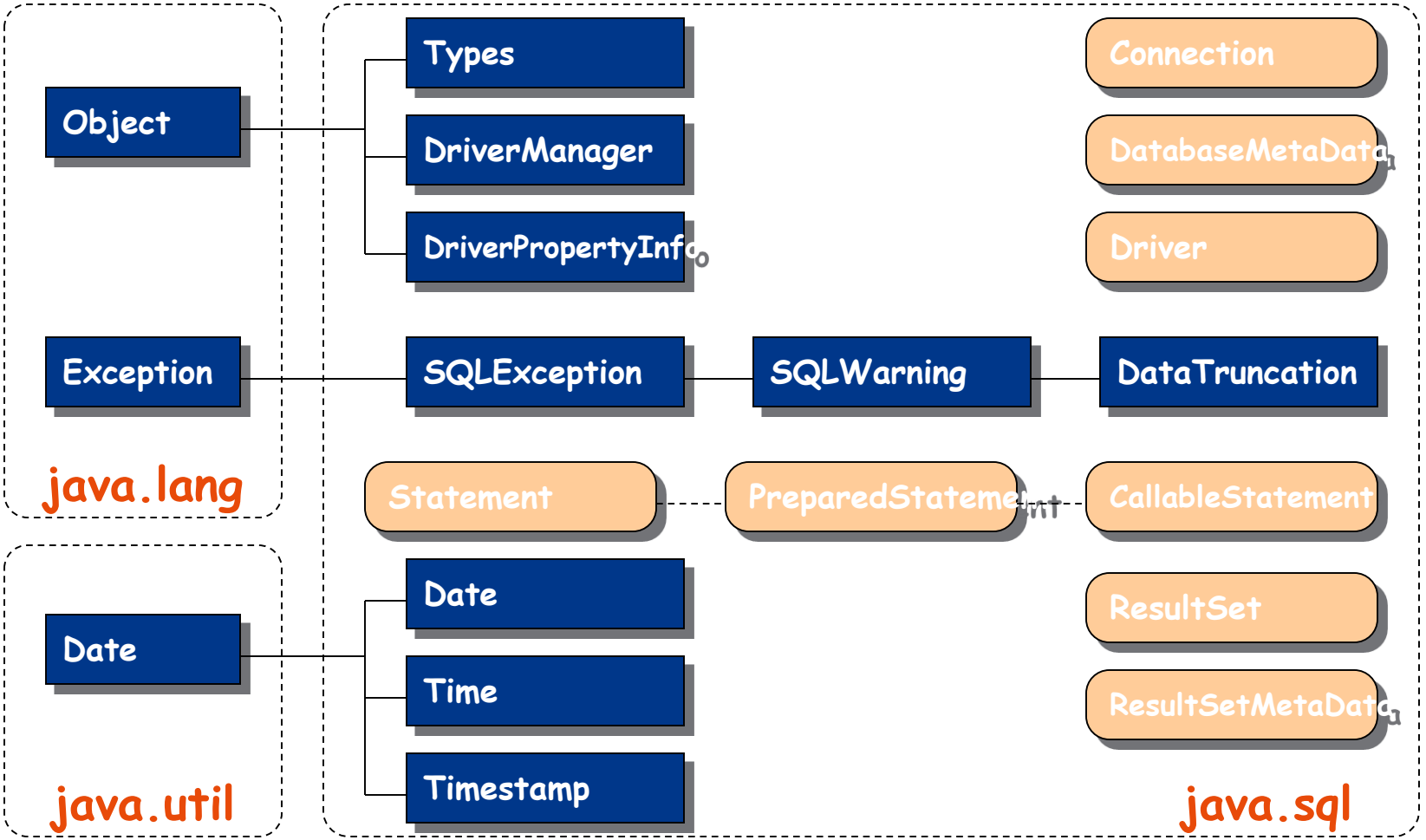
JDBC Sürücü Mimarileri



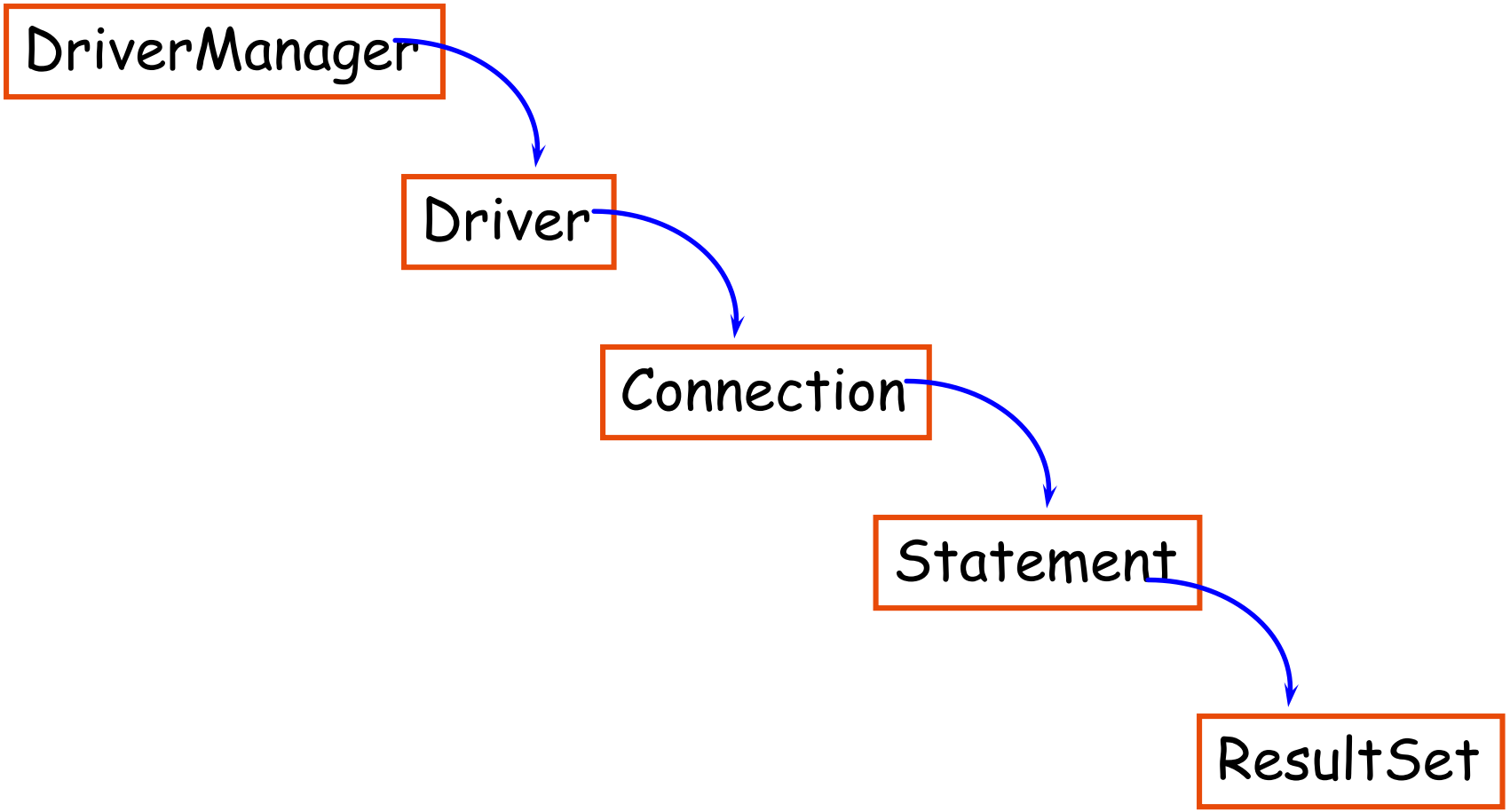
JDBC Sınıf ve Interfaceleri - 1

- **Interfaceler**
  - CallableStatement
  - Connection
  - DatabaseMetaData
  - Driver
  - PreparedStatement
  - ResultSet
  - ResultSetMetaData
  - Statement
- **Sınıflar**
  - Date
  - DriverManager
  - DriverPropertyInfo
  - Time
  - Timestamp
  - Types

JDBC Sınıf ve Interfaceleri - 2



JDBC Sınıf Kullanımı



Yedi Basamak

- Sürücü yüklenir
- URL tanımlanır
- Bağlantı kurulur
- Statement nesnesi stmt elde edilir
- Statement yardımıyla sorgu çalıştırılır
- Sonuçlar işlenir
- Bağlantı kapatılır

Sürücü Yükleme

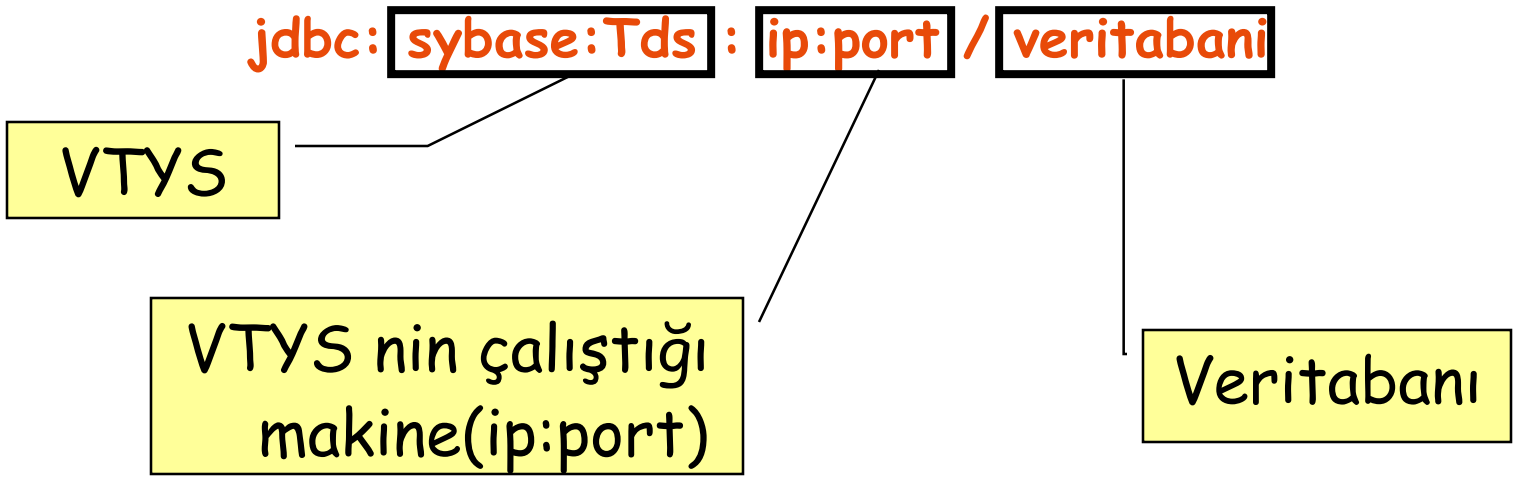
- Sürücü nesnesi oluşturulur ve DriverManager sınıfının registerDriver metodu ile sürücü kaydedilir

```
Driver driver = new com.sybase.jdbc3.jdbc.SybDriver();
DriverManager.registerDriver(driver);
```

URL

Her sürücünün kendine özgü bir subprotocol tanımı vardır

URL yapısı: jdbc:subprotocol:source



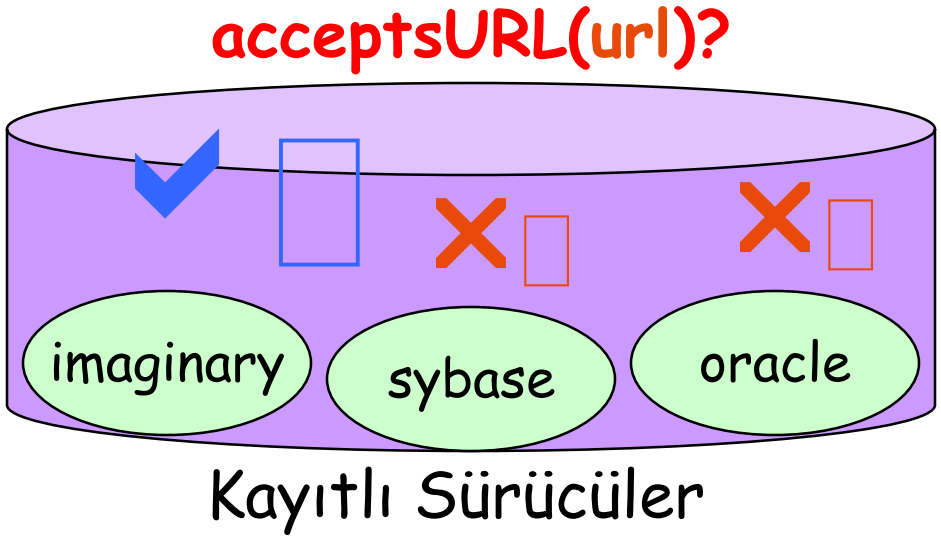
Örnek: jdbc:sybase:Tds:localhost:5001/VT

Veritabanı Bağlantısı - 1

- Her bir veritabanı bir URL ile tanımlanır
- DriverManager URL tanımını kullanarak ilgili veritabanı ile konuşabilmek için uygun sürücüyü arar
- DriverManager uygun sürücüyü bulana kadar kayıtlı tüm sürücüleri dener

Veritabanı Bağlantısı - 2

```
Connection conn = DriverManager.
        getConnection(String url, String user, String password);
```



Veritabanı ile Etkileşim

- Etkileşim için Statement kullanılır
  - Veri elde edilir
  - Veri güncellenir
- Üç farklı interface var

  Statement, PreparedStatement, CallableStatement
- Bunlardan direk nesne oluşturulamaz
- Connection nesnesi ile Statement nesnesi oluşturulur

Statement

```
String queryStr =
        "SELECT * FROM Tb_Ziyaretci " +
        "WHERE Lower(Ad) = 'ad'";

Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(queryStr);
```

- Metod executeQuery sorgu sonucunu ResultSet nesnesi olarak çevirir
  - Az sonra detaylı işlenecek …

Statement ile İşlemler

```
String deleteStr =
        "DELETE FROM Tb_Ziyaretci " +
        "WHERE Lower(Ad) = 'ad'";

Statement stmt = con.createStatement();
int delnum = stmt.executeUpdate(deleteStr);
```

- executeUpdate veri manipulasyonu için kullanılır: insert, delete, update, create table vb. (sorgudan farklı!)
- executeUpdate kayıt sayısını döndürür
- 0 ise güncellenen kayıt yok ya da create table vb. bir ifade çalışmış
- -1 ise hata var

PreparedStatement

- PreparedStatement uygulama içinde birçok kez çalıştırılacak sorgular için kullanılır
- VTYS tarafından bir kere derlenir
- Kolon değerleri derleme sonrası atanır
- Değerler yerine, '?' kullanılır
- Şöyleki bir PreparedStatement çalıştırılırken '?' yerine gerçek değerler atanır

PreparedStatement ile İşlemler - 1

```
String queryStr =
        "SELECT * FROM Tb_Ziyaretci " +
        "WHERE Ad = ?";
```

```
PreparedStatement pstmt =
        con.prepareStatement(queryStr);
```

```
pstmt.setString(1, "ad");
```

```
ResultSet rs = pstmt.executeQuery();
```

PreparedStatement ile İşlemler - 2

```
String deleteStr =
        "DELETE FROM Tb_Ziyaretci " +
        "WHERE Ad = ?";
```

```
PreparedStatement pstmt =
        con.prepareStatement(deleteStr);
```

```
pstmt.setString(1, "ad");
```

```
int delnum = pstmt.executeUpdate();
```

Statement & PreparedStatement - 1
DİKKAT!

- Aşağıdakiler aynı mı? Ne yaparlar?

```
String val = "abc";
PreparedStatement pstmt =
        con.prepareStatement("select * from R where
A=?");
pstmt.setString(1, val);
ResultSet rs =  pstmt.executeQuery();
```

```
String val = "'abc'";
Statement stmt =  con.createStatement( );
ResultSet rs =  stmt.executeQuery("select * from R
where A=" + val);
```

Statement & PreparedStatement - 2 DİKKAT!

- Aşağıdaki çalışır mı?

```
PreparedStatement pstmt =
        con.prepareStatement("select * from ?");

pstmt.setString(1, myFavoriteTableString);
```

- Hayır!!! '?' sadece kolon değeri için kullanılır
- NEDEN?

execute()

```
Statement genericStmt = conn.createStatement();
if (genericStmt.execute(SQLString)) {
ResultSet rs = genericStmt.getResultSet();
else { int updated = genericStmt.getUpdateCount();
}
```

- Sorgu bilinmiyorsa (prosedür çağırma, select, update vb...) ya da birden fazla ResultSet varsa execute kullanılır

Java Tipleri ile SQL Tipleri Eşleştirilmesi

SQL Tip	Java Tip
CHAR, VARCHAR, LONGVARCHAR	String
NUMERIC, DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT, DOUBLE	double
BINARY, VARBINARY, LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Zaman Kavramı

- Zamanlar SQL de standart değil
- Javada zamanlar ile ilgili üç sınıf var
- java.sql.Date
  - yıl, ay, gün
- java.sql.Time
  - saat, dakika, saniye
- java.sql.Timestamp
  - yıl, ay, gün, saat, dakika, saniye, nanosaniye
  - yaygın olarak kullanılır

ResultSet

- Statement ifadesinin çalıştırılması sonucunda oluşan tablo verilere erişim olanağı sağlar
- Bir Statement için birim zamanda bir ResultSet vardır
- Veriler sırayla işlenir
  - ResultSet geçerli veri satırını gösteren bir işaretçiye sahiptir
  - **next** metodu ile bir sonraki satıra geçilir

Önemli ResultSet Metodları - 1

- boolean next()
  - bir sonraki satır aktive olur
  - ilk çalıştırılmada ilk satır aktive olur
  - kayıt yoksa false çevrilir
- void close()
  - ResultSet kapatılır
  - Statement tekrar kullanılabilir
  - Statement metodları otomatik olarak çağırır

Önemli ResultSet Metodları - 2

- Type getType(int columnIndex)
  - istenilen kolon bilgisi belirtilen tipe çevrilir
  - indeks 1 ile başlar (0 değil)
- Type getType(String columnName)
  - aynı, fakat kolon adı kullanılır
  - daha az etkin
- int findColumn(String columnName)
  - belirtilen kolon adının kolon indeksini çevirir

getXXX Metodları - 1

- String getString(int columnIndex)
- boolean getBoolean(int columnIndex)
- byte getByte(int columnIndex)
- short getShort(int columnIndex)
- int getInt(int columnIndex)
- long getLong(int columnIndex)
- float getFloat(int columnIndex)
- double getDouble(int columnIndex)
- Date getDate(int columnIndex)
- Time getTime(int columnIndex)
- Timestamp getTimestamp(int columnIndex)

getXXX Metodları - 2

- String getString(String columnName)
- boolean getBoolean(String columnName)
- byte getByte(String columnName)
- short getShort(String columnName)
- int getInt(String columnName)
- long getLong(String columnName)
- float getFloat(String columnName)
- double getDouble(String columnName)
- Date getDate(String columnName)
- Time getTime(String columnName)
- Timestamp getTimestamp(String columnName)

ResultSet Örnek

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

```
ResultSet rs = stmt.
executeQuery("select Ad,Soyad from Tb_Ziyaretci");
```

```
// Sonucu yaz
while(rs.next()) {
   System.out.print(rs.getString(1) +" ");
   System.out.println(rs. getString("Soyad"));
}
```

Null Değer - 1

- SQLde Null boş anlamındadır
- 0 veya "" değil
- JDBC işlemlerinde kolon değerinin null olup olmadığı wasNull() metodu ile sorgulanır
- Örnek, getInt(kolon) kolon değeri null ya da 0 olsa bile 0 değeri çevirir

Null Değer - 2

- PreparedStatement kullanarak Null değer kaydetmek için:
  - İlkel tipler için setNull(index, sqlType) metodu kullanılır (örnek: INTEGER, REAL);
  - Nesne tipler için setXXX(index, null) metodu kullanılır (e.g. STRING, DATE).

Aman Dikkat

```
rs.close()

stmt.close();

pstmt.close();

cstmt.close();

conn.close();
```

- Connection, Statement, PreparedStatement ve ResultSetleri hep kapatalım!!!

Close Tavsiye

```
if (rs!=null) rs.close();
if (stmt!=null) stmt.close();
if (pstmt!=null) pstmt.close();
if (cstmt!=null) cstmt.close();
if (conn!=null) conn.close();
```

Exception

- Bir exception nesnesi birden fazla exception nesnesi içerebilir

```
catch (SQLException e) {
    while (e != null) {
        System.out.println(e.getSQLState());
        System.out.println(e.getMessage());
        System.out.println(e.getErrorCode());
        e = e.getNextException();
    }
}
```

Warning

- Nadir olarak kullanılır
- Connection, Statement, ResultSet

```
ResultSet rs = stmt.executeQuery(SQLString);
SQLWarning w = stmt.getWarnings();
while (w != null) {
  System.out.println("Mesaj: " + w.getMessage());
  System.out.println("SQLDurum: " + w.getSQLState());
  System.out.println("Hata Kod: " + w.getErrorCode());
  w = w.getNextWarning();
}
```

MetaData

- Connection

  DatabaseMetaData getMetaData()
- PreparedStatement

  ParameterMetaData getParameterMetaData()
- ResultSet

  ResultSetMetaData getMetaData()

ResultSetMetaData

ResultSetMetaData nesnesi ile ResultSet nesnesinin kolon özellikleri ile ilgili bilgiler elde edilir

Örnek:

```
ResultSetMetaData rsmd = rs.getMetaData();
int numcols = rsmd.getColumnCount();

for (int i = 1 ; i <= numcols; i++) {
        System.out.print(rsmd.getColumnLabel(i)+"  ");
}
```

Transaction

- Transaction = birden fazla ifadenin hepsinin çalışması ya da bu ifadelerden herhangi birinde hata olursa hiçbirinin çalışmaması
- Eğer bir hata çıkarsa, sistem önceki başarılı işlemlerinin tamamını geri alır
- Böylelikle veri tutarlılığı sağlanır
- COMMIT = transaction tamamlanır
- ROLLBACK = transaction geri sarar

Transaction Senaryo

- Bir hesaptan (13) başka bir hesaba (72) havale yapalım:

```
PreparedStatement pstmt =
      con.prepareStatement("update BankAccount
                            set amount = amount + ?
                            where accountId = ?");
pstmt.setInt(1,-100);
pstmt.setInt(2, 13);
pstmt.executeUpdate();
pstmt.setInt(1, 100);
pstmt.setInt(2, 72);
pstmt.executeUpdate();
```

Çalışmazsa ne olur?

Transaction Yönetimi

- Transactionlar kendiliğinden açılıp, kapanmaz
- Connection nesnesinin AutoCommit modu vardır
- Eğer AutoCommit true ise, tüm ifadeler otomatik olarak işlenir
- Eğer AutoCommit false ise, her ifade transactiona eklenir
- Varsayılan: true

AutoCommit

setAutoCommit(boolean val)

- Eğer AutoCommit false ise, Connection.commit() ya da Connection.rollback() çalıştırılmalıdır
- LOB işlemlerde veri çekerken, genelde AutoCommit false yapılır
- Not: Transactionda DDL ifadeleri dikkate alınmayabilir ya da commit çalışır. Bu farklı davranışlar VTYS ile ilgilidir

Transaction Örnek

```
con.setAutoCommit(false);
try {
    PreparedStatement pstmt =
        con.prepareStatement("update BankAccount
                              set amount = amount + ?
                              where accountId = ?");
    pstmt.setInt(1,-100); pstmt.setInt(2, 13);
    pstmt.executeUpdate();
    pstmt.setInt(1, 100); pstmt.setInt(2, 72);
    pstmt.executeUpdate();
    con.commit();
catch (Exception e) {
    con.rollback();
}
```

Teşekkürler...