



Java

Data Base

Connectivity

(J D B C)

차례

- ◆ SQL 개요
- ◆ JDBC 개요
- ◆ JDBC 역할
- ◆ Driver TYPE
- ◆ JDBC Programming
 - ◇ 1.Loading Driver
 - ◇ 2.Making the Connection
 - ◇ 3.Submitting Query
 - ◇ 4.Retrieving ResultSet
- ◆ JDBC 2.0 API



JDBC

SQL Statement

- DML(Data Manipulation Language)
 - SELECT
 - INSERT
 - DELETE
 - UPDATE
- DDL(Data Definition Language)
 - CREATE
 - DROP

Java DataBase Connectivity?

- can access DB in java application
- DB vendor independent
- JDBC API
 - Interface to access DB with uniform method
- JDK
 - provide JDBC API spec.
- Application developer
 - use JDBC API to access DB
- DB vendor
 - JDBC API spec.에 맞는 driver 제공

Java DataBase Connectivity?

◆ JDBC(Java Database Connectivity)

- ◇ JDBC란 자바를 이용하여 데이터베이스에 접근하여 각종 SQL문을 수행할 수 있도록 제공하는 API를 말한다.

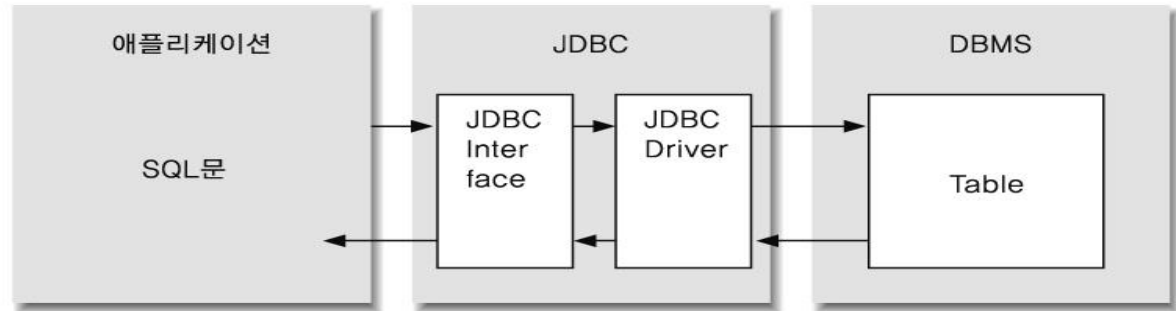
◆ 각종 DBMS를 통합한 라이브러리 필요성

- ◇ 자바가 데이터베이스에 접근하는 프로그램을 시도할때 한 가지 문제점이 있었다.
- ◇ 문제점 : DBMS의 종류가 다양하고, 구조와 특징이 다름.
- ◇ 자바는 모든 DBMS에서 공통적으로 사용할 수 있는 인터페이스와 클래스로 구성하는 JDBC를 개발하게 되었고, 실제 구현은 DBMS의 벤더에게 구현하도록 했다.
- ◇ 각 DBMS의 벤더에서 제공하는 구현 클래스를 JDBC 드라이버라고 한다.
- ◇ JDBC로 코딩하기 위해서는 DBMS를 선택하고, DBMS에서 제공하는 JDBC 드라이버가 반드시 필요하다.

JDBC의 역할

◆ JDBC의 구조와 역할

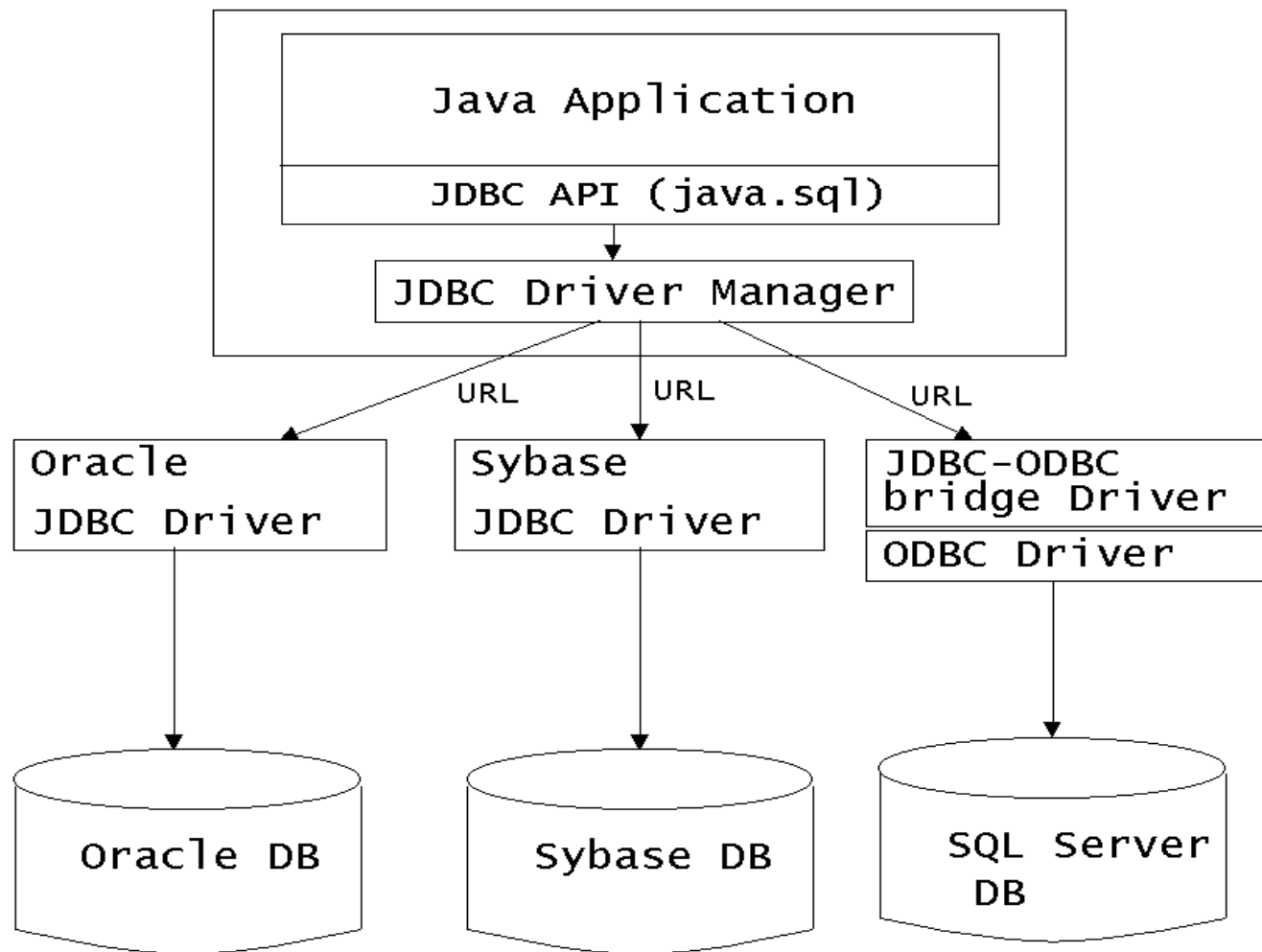
- ◇ JDBC는 크게 JDBC 인터페이스와 JDBC 드라이버로 구성되어 있다



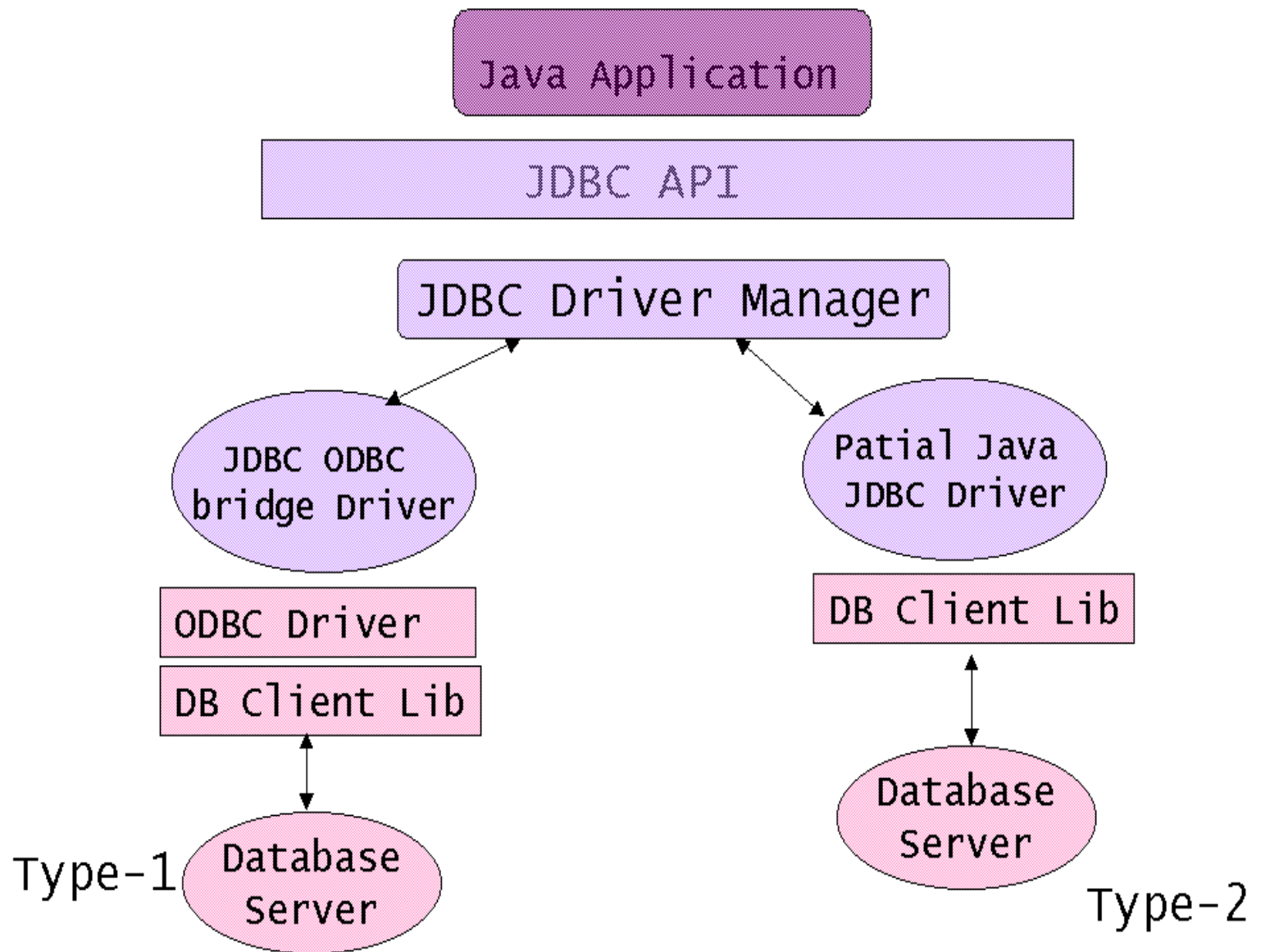
[그림 17-12] 애플리케이션과 JDBC, DBMS의 관계

- ◇ 응용프로그램에서는 SQL문 만들어 JDBC Interface를 통해 전송하면 실제 구현 클래스인 JDBC 드라이버에서 DBMS에 접속을 시도하여 SQL문을 전송하게 된다.
- ◇ JDBC의 역할은 Application과 DBMS의 Bridge 역할을 하게 된다.

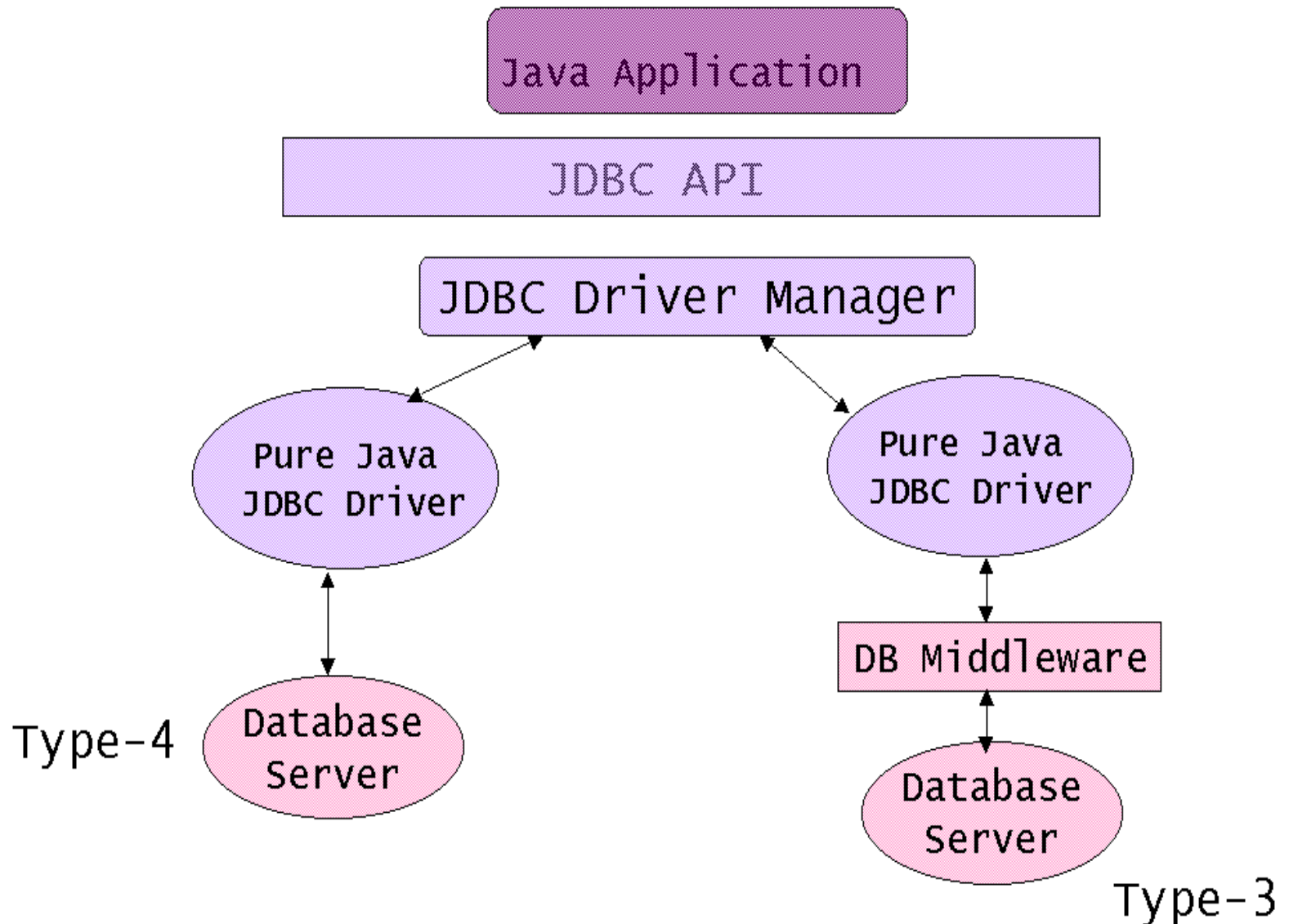
JDBC의 역할



JDBC Driver Type



JDBC Driver Type



Type-1

- ODBC Driver를 연결하여 작동하는 JDBC driver.
- MicroSoft의 표준인 ODBC driver를 연결하고 이를 통하여 DB를 접근하는 방식
- JDBC-ODBC bridge
- Client 컴퓨터에 ODBC driver, JDBC driver가 설치
- 동적으로 driver가 download 되어 사용하는 환경에서는 사용하기 어려움.
- `sun.jdbc.odbc.JdbcOdbcDriver`
- 속도가 느림
- Microsoft DB와의 interface (MS-ACCESS, SQL SERVER...)

TYPE-2

- 일부가 Java 로 되어 있는 JDBC Driver
- client 컴퓨터에 설치되어 있는 local driver가 필요.
- Local driver가 데이터베이스를 만든 Vendor가 제공하는 다른 형태의 driver.
- Vendor의 local driver와 JDBC driver가 Client 컴퓨터에 설치되어 있어야 사용가능.
- Oracle, Informix, DB2 등 DBMS Vendor들이 제공하는 API를 다시 호출.
- Client 컴퓨터에 정적으로 driver를 설치하여 데이터베이스와 연동하는 환경에 적합.
- Applet: 불가능 , Servlet : 가능

TYPE-3

- pure Java JDBC Driver
- **DBMS Vendor**가 표준을 기준으로 기능을 추가 또는 삭제하여 독자적인 형태
- 단점
 - 특정 DBMS에 의존적
- 장점
 - 해당 DBMS만이 제공하는 기능 사용가능
- **Client**가 동적인 환경에서 특정 **DBMS**만이 제공하는 기능들을 사용할 경우 적합
- intranet에서는 적합

TYPE-4

- pure Java JDBC Driver
- JDBC API 표준에 의하여 만들어 졌기 때문에 DBMS의 종류에 상관없이 사용할 수 있다.
- 가장 융통성이 뛰어난 Driver Type
- 동적으로 download되는 Applet 같은 환경에 적합
- 시스템 유지보수가 쉬워진다.
- 사용하기 가장 적합
 - 예외
 - 특정 DBMS 만에 제공하는 기능을 사용할 필요
 - DBMS 자체가 type 4를 지원하지 않는 경우

Getting Started

- Install JDK, JDBC
- Install JDBC Driver
 - vendor's web site
 - JDBC-ODBC bridge Driver는 JDK에서 제공
 - www.technet.oracle.co.kr (classesXX.zip)
- Install DBMS if needed
- Database는 구축되어 있어야 함.

JDBC Programming Task

1. Loading Driver

```
DriverManager.registerDriver(  
new oracle.jdbc.driver.OracleDriver());
```

```
Class.forName( "oracle.jdbc.driver.OracleDriver" );
```

```
new oracle.jdbc.driver.OracleDriver();
```

```
prompt>java  
-D jdbc.drivers=oracle.jdbc.driver.OracleDriver  
SimpleJDBC
```

관련 클래스

- java.sql.Driver

2. Making the Connection

```
URL = jdbc:subprotocol:subname
```

```
String url = "jdbc:oracle:thin:@200.0.0.1:1521:ORCL";  
Connection con = DriverManager.getConnection(url,  
        "scott","tiger");
```

```
String url = "jdbc:ODBC:Stock";  
Connection con = DriverManager.getConnection(url,  
        "scott","tiger");
```

-URL은 DB Vendor마다 다름.

java.sql.Connection

- Making Connection to DB
- Method
 - void createStatement()
 - DatabaseMetaData getMetaData()
 - void close()
 - void setAutoCommit(boolean autoCommit)
 - void commit()
 - void rollback()
 - PreparedStatement prepareStatement(String sql)

3. Submitting Query

```
1 stmt = con.createStatement();  
2 stmt.executeUpdate("DELETE FROM Customer  
    WHERE ssn='111-111'");  
3 ResultSet rs = stmt.executeQuery(  
    "SELECT * FROM Customer");
```

Statement

- executing a static SQL statement
- obtaining the results produced by it
- method
 - boolean execute(String sql)
 - produce multi result set (CREATE, DROP..)
 - ResultSet executeQuery(String sql)
 - produce single result set (ex] SELECT ...)
 - int executeUpdate(String sql)
 - Update,delete,insert 시 사용
 - void close()

4. Retrieving Result

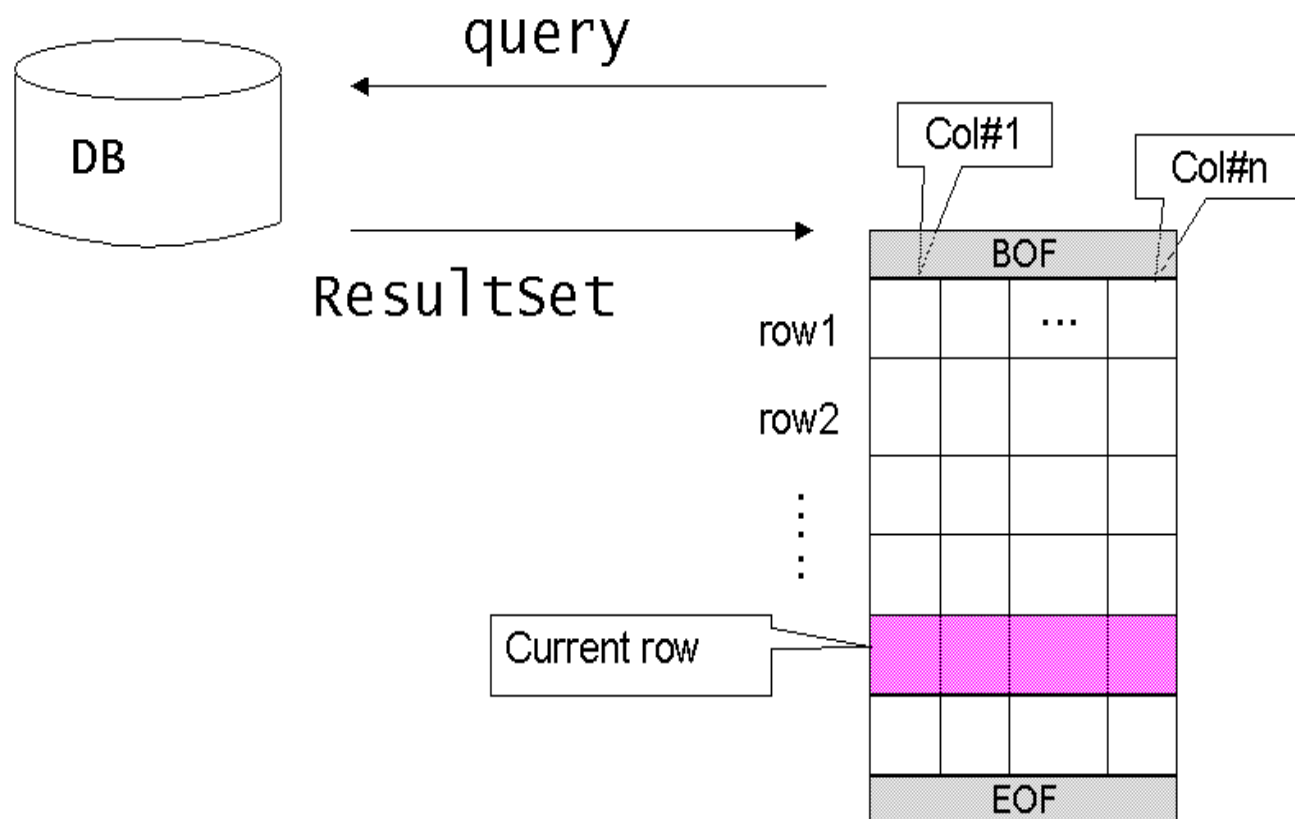
-Set of rows : Result of executing query statement

```
java.sql.ResultSet
```

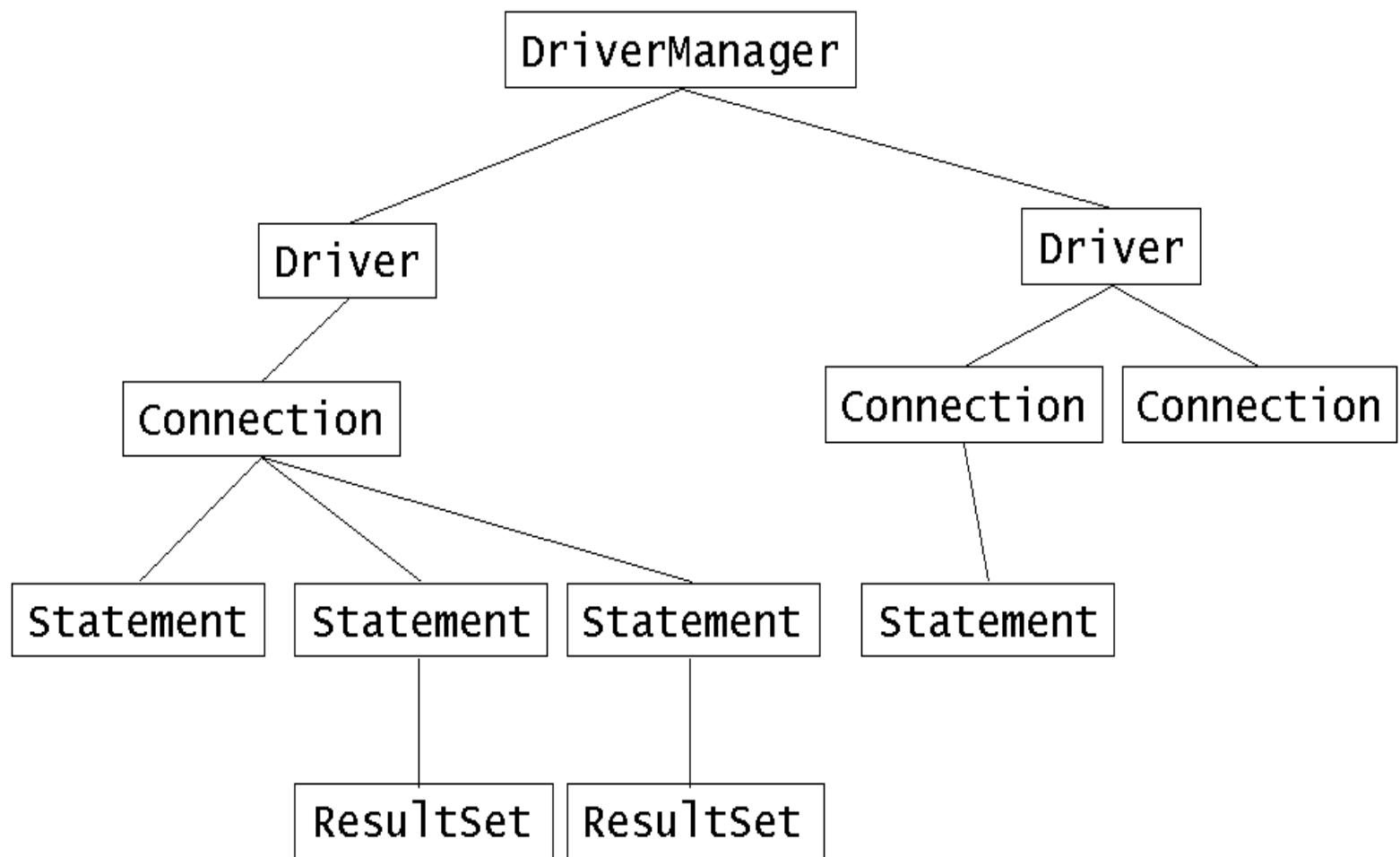
- next()
- getXXX(col#)
- close()

```
ResultSet rs = stmt.executeQuery("Select * FROM Customer");  
while( rs.next())  
{  
    System.out.println( rs.getString(1));  
    System.out.println( rs.getString(2));  
}
```

4. Retrieving Result



JDBC Flow



DatabaseMetaData

- information about the database
- table information(name, primary key...),
DB vendor information, Driver information,
stored procedure,

```
DatabaseMetaData dbmd = con.getMetaData();  
System.out.println(dbmd.getDatabaseProductName());  
  
ResultSet rsMeta = dbmd.getTables(null,null,null,null);  
while( rsMeta.next())  
    System.out.println(rsMeta.getString("TABLE_NAME");
```


ResultSetMetaData

- Query result에 대한 정보
- column 수, type, name, ...

```
ResultSetMetaData rsmd = rs.getMetaData();  
int count = rsmd.getColumnCount();  
for( int i = 1; i <= count ; i++)  
{  
    System.out.println(rsmd.getColumnLable( i ));  
    System.out.println(rsmd.getColumnTypeName(i));  
}
```

PreparedStatement


- 계속 같은 SQL을 반복하여 사용하는 경우
- 미리 statment를 준비
- 변경되는 값은 "?"를 사용하여 표현
- SQL을 실행하기 전에 "?" 값을 setXXX()
함수를 이용하여 치환

```
int ids[3] = { 1, 2, 3 };
String names[3] = { "홍길동", "김삿갓", "성춘향" };
String sql = "INSERT INTO TB_CUST"+
              "(ID, NAME) VALUES ( ?, ? )";
PreparedStatement pstmt = conn.prepareStatement(sql);
for(int i=0, i < ids.length; i++) {
    pstmt.setInt(1,ids[i]);
    pstmt.setString(2,names[i]);
    pstmt.executeUpdate();
}
```

CallableStatement

- DB안에 있는 stored procedure를 호출하기 위한 interface

```
String call = "{ ?= call maxplus(?, ?)}";  
int result = 0;  
  
CallableStatement cstmt = conn.prepareCall(call);  
cstmt.setInt(2,100);  
cstmt.setInt(3,5);  
  
cstmt.registerOutParameter(1, java.sql.Types.INTEGER);  
cstmt.execute();  
System.out.println("result : " + cstmt.getInt(1) );
```



JDBC 2.0 API

Getting Set up

- Setup

- 1. Download JDK 1.2
- 2. Install JDBC driver(support JDBC 2.0)
- 3. Use JDBC 2.0 features

- Feature

- Scroll forward and backward in a result set
- Move to a specific row
- Update db, using method instead of SQL
- Send multiple SQL statement to the DB as a unit or batch

Moving the Cursor

- Move result set's cursor backward & forward
- requirement
 - create scrollable ResultSet object

```
st = con.createStatement(  
    ResultSet.TYPE_SCROLL_SENSITIVE,  
    ResultSet.CONCUR_READ_ONLY );
```

Moving in the Cursor

- Method

- first(), last()
- beforeFirst(), afterLast()
- absolute(number)
 - absolute(4) : 4th row
 - absolute(-4) : (last - 4 + 1)th row
- next(), previous()
- relative(offset) : $\nearrow | \searrow \Rightarrow$ current row
- getRow() : return current row number

Updatable Result Set

- Query를 다시 실행시키지 않고 기존의 ResultSet에서 data를 update 가능
- requirement

```
stmt = con.createStatement(  
    ResultSet.TYPE_SCROLL_SENSITIVE,  
    ResultSet.CONCUR_UPDATABLE);  
ResultSet rs = stmt.executeQuery(  
    "Select * FROM CUSTOMER");
```


Updatable ResultSet

- JDBC 1.0

```
Stmt.executeUpdate("UPDATE CUSTOMER  
SET ADDRESS = 'pusan' WHERE SSN = '111-111'");
```

- JDBC 2.0

```
// previous example ResultSet rs  
rs.absolute( 5 ) // 5th record'ssn = '111-111'  
rs.updateString("ADDRESS", "Pusan");  
rs.updateRow();
```

Insert & Delete

```
stmt = con.createStatement(  
    ResultSet.TYPE_SCROLL_SENSITIVE,  
    ResultSet.CONCUR_UPDATABLE);  
ResultSet rs = stmt.executeQuery(  
    "Select * FROM CUSTOMER");  
rs.moveToInsertRow();  
rs.updateString("SSN", "999-999");  
rs.updateString("CUST_NAME", "kim");  
rs.updateString("ADDRESS", "DEAGU");  
rs.insertRow();
```

```
rs.absolute( 5 ) ;  
rs.deleteRow();
```

Making Batch Updates

- A Set of multiple update statements that is submitted to the database for processing as a batch

```
con.setAutoCommit( false );  
Statement stmt = con.createStatement();  
stmt.addBatch("INSERT INTO COFFEES VALUES  
              ("Moca", 49, 9.99, 0, 0 )");  
stmt.addBatch("INSERT INTO COFFEES VALUES  
              ("hazelnut", 49, 9.99, 0, 0 )");  
stmt.addBatch("INSERT INTO COFFEES VALUES  
              ("Amaretto", 49, 9.99, 0, 0 )");  
int [] updateCounts = stmt.executeBatch();
```



