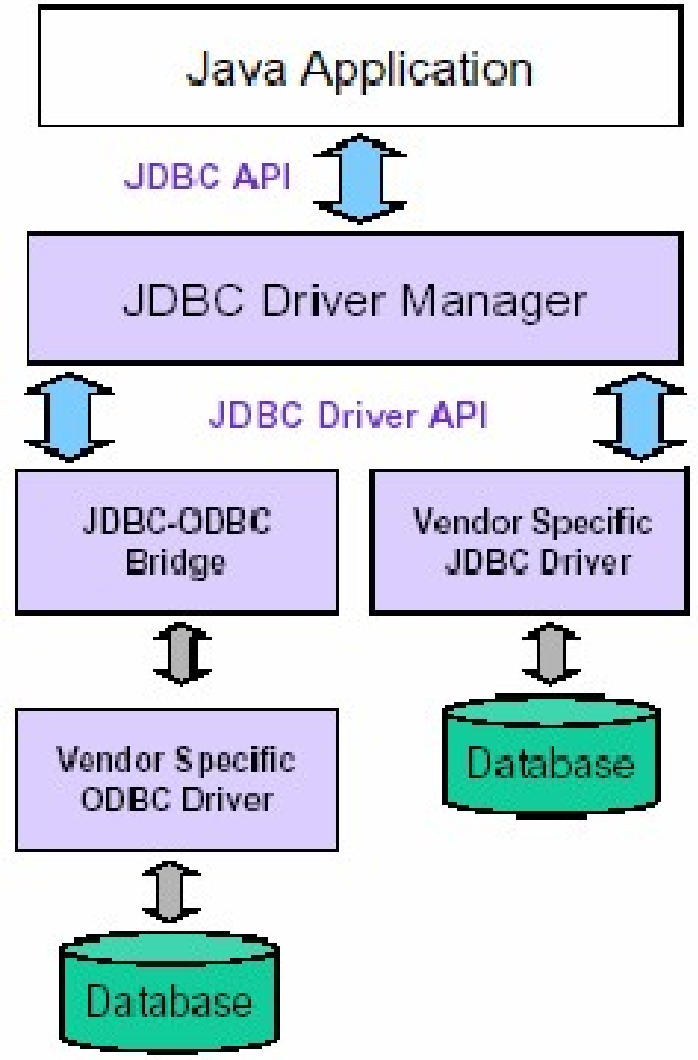
JDBC –

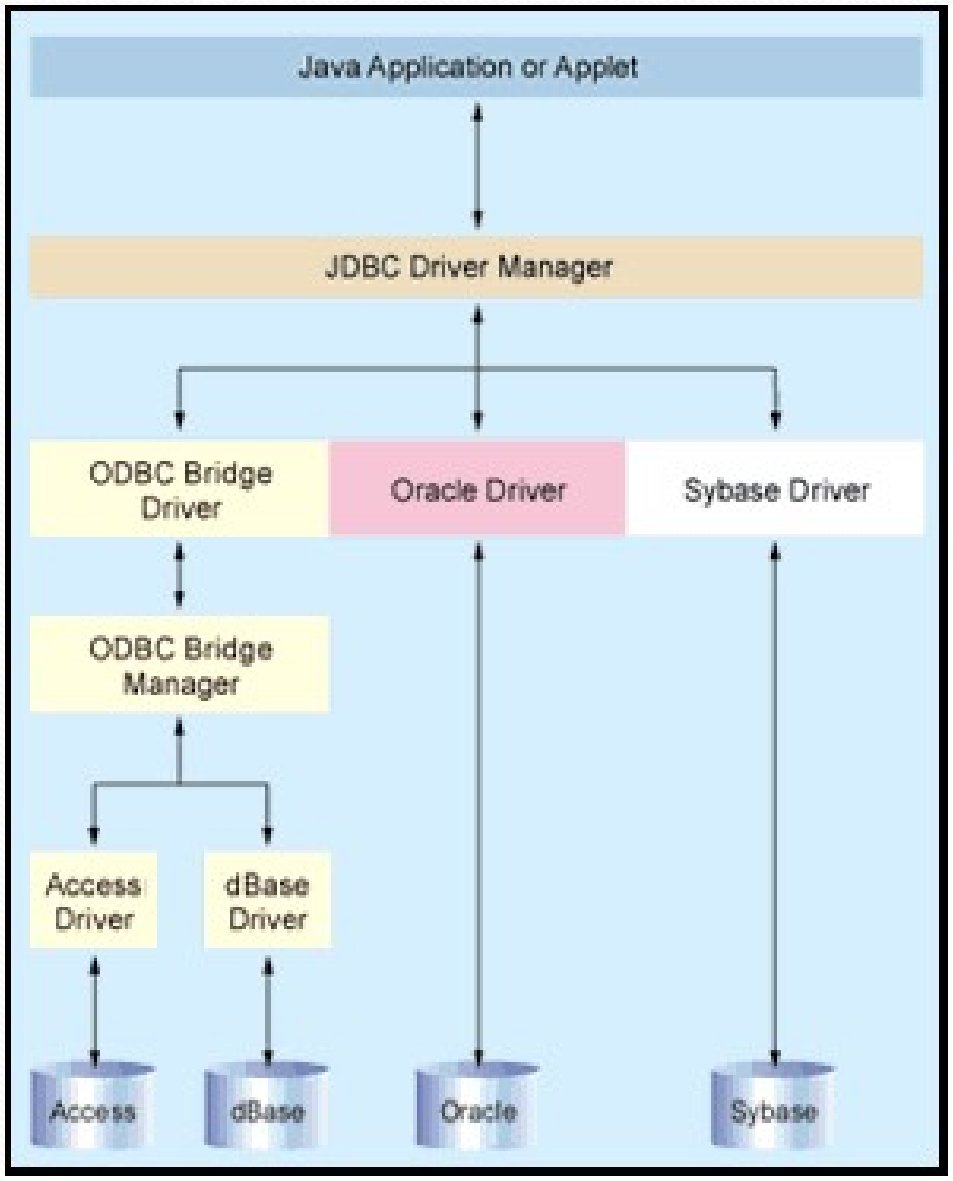
Java DataBase Connectivity

What is JDBC?

- "An API that lets you access virtually any tabular data source from the Java programming language"
  - JDBC Data Access API – JDBC Technology Homepage
  - What's an API?
    - See J2SE documentation
  - What's a tabular data source?
- "… access virtually any data source, from relational databases to spreadsheets and flat files."
  - JDBC Documentation
- We'll focus on accessing Oracle databases

General Architecture




Java Application or Applet
JDBC Driver Manager
ODBC Bridge Driver
Oracle Driver
Sybase Driver
ODBC Bridge Manager
Access Driver
dBase Driver
Access
dBase
Oracle
Sybase

Type – 1 (Bridge driver)

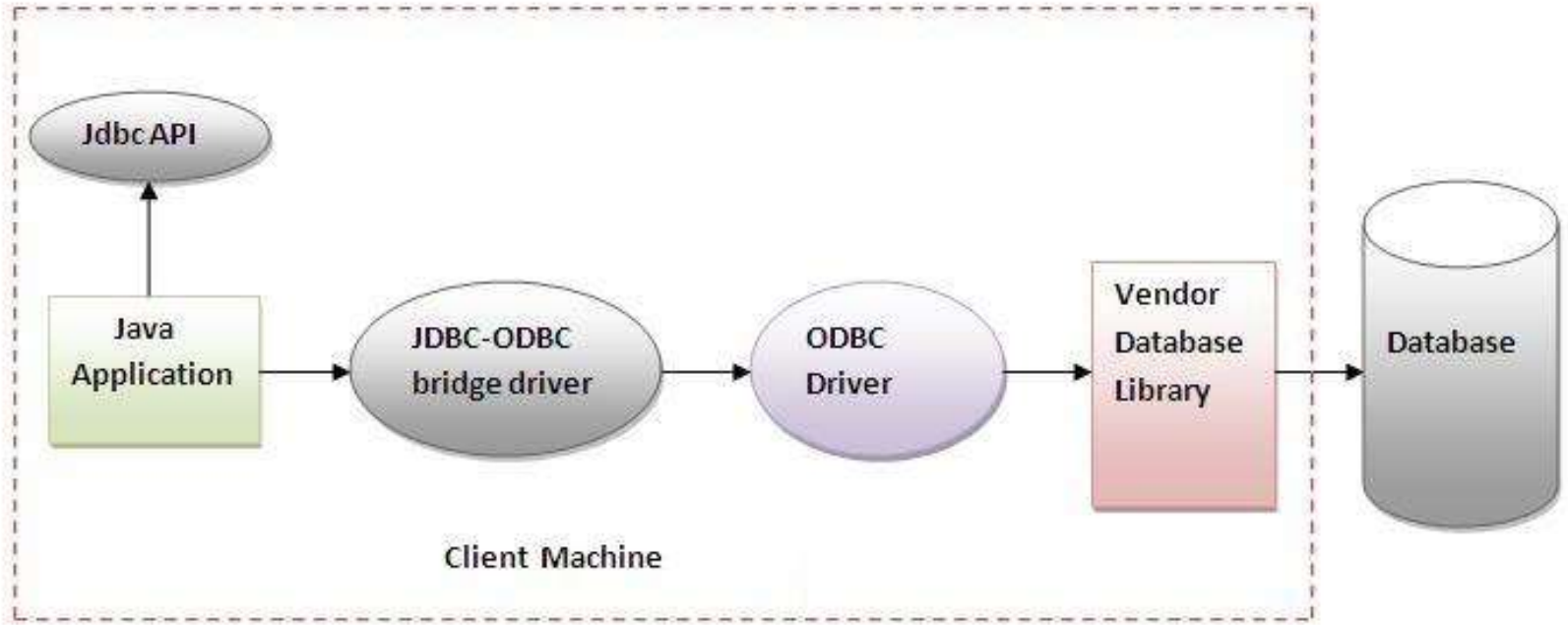


Figure- JDBC-ODBC Bridge Driver

Type – 2 (Native API driver)

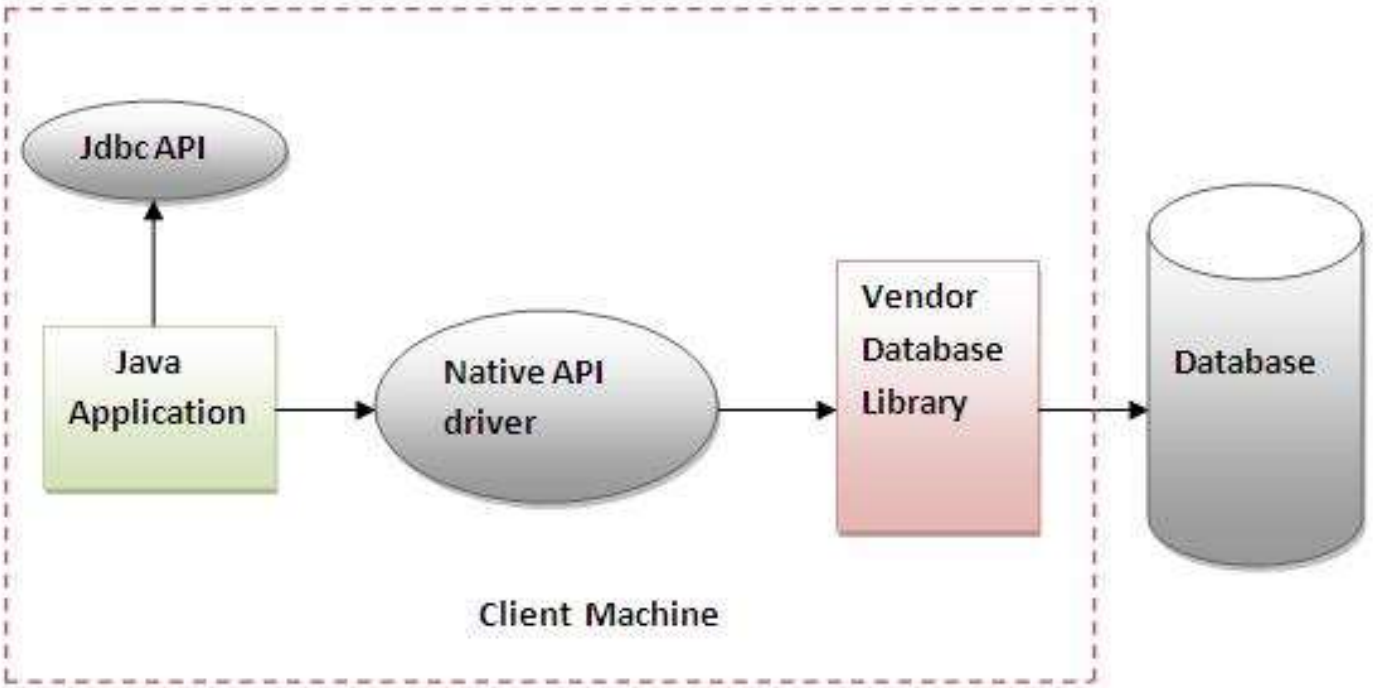


Figure- Native API Driver

Type – 3 (Network protocol)

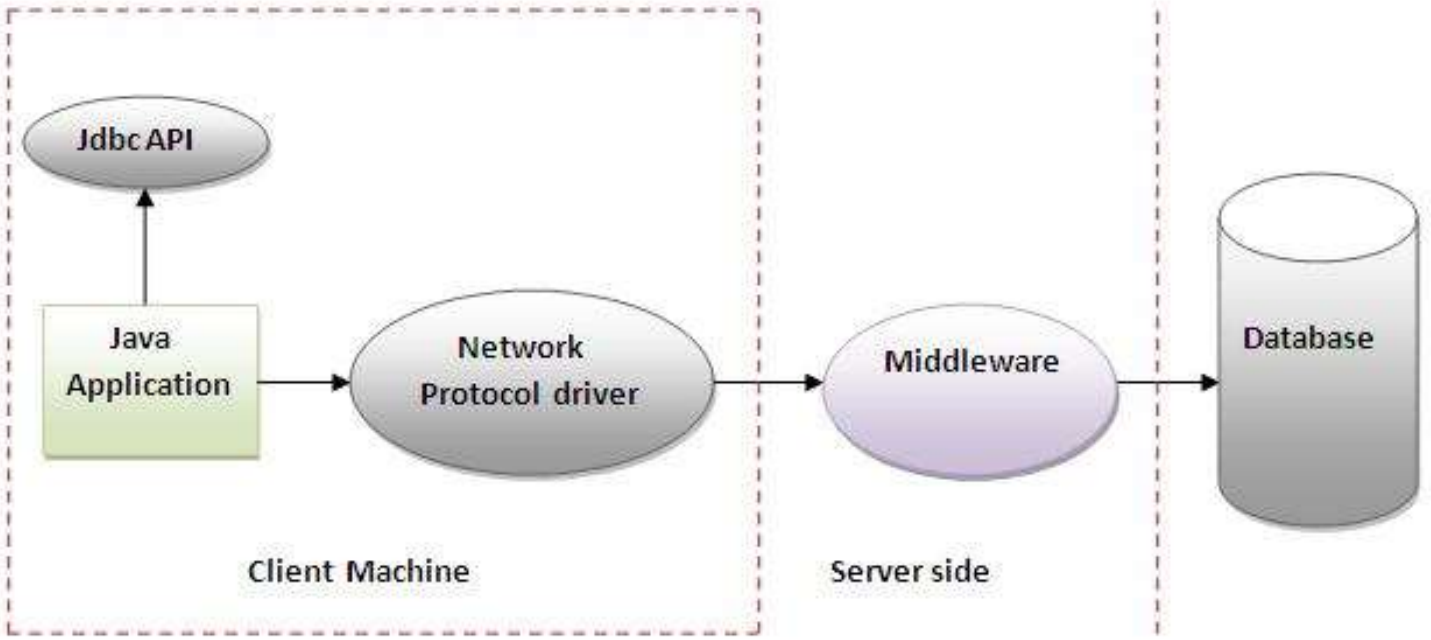


Figure- Network Protocol Driver

Type – 4 (Thin driver)

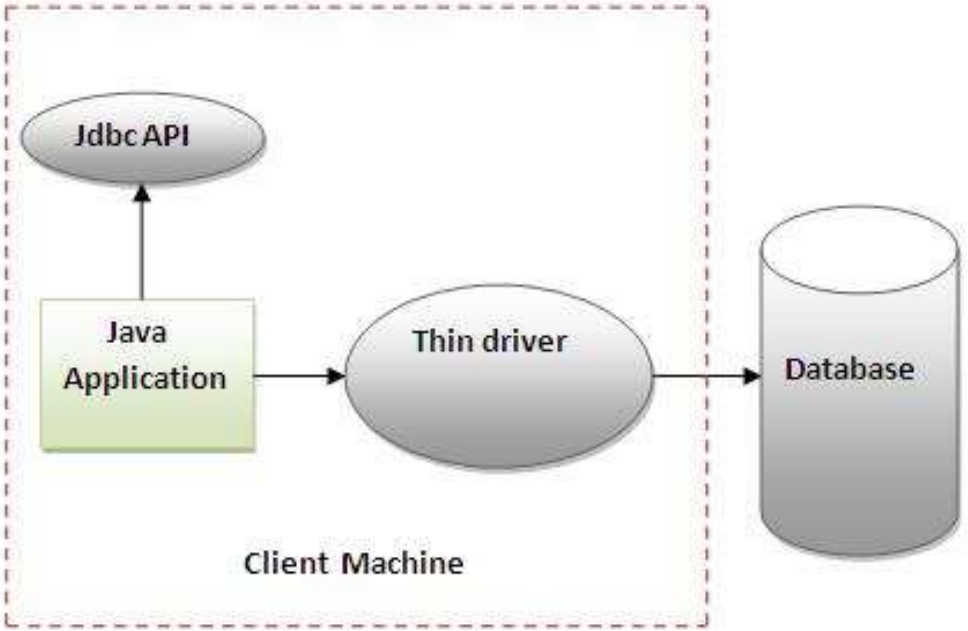


Figure- Thin Driver

Basic steps to use a database in Java

- 1.Establish a **connection**
- 2.Create JDBC **Statements**
- 3.Execute **SQL** Statements
- 4.GET **ResultSet**
- 5.**Close** connections

1. Establish a connection

- **import java.sql.*;**
- **Load the vendor specific driver**
  - Class.forName("oracle.jdbc.driver.OracleDriver");
    - What do you think this statement does, and how?
    - Dynamically loads a driver class, for Oracle database
- **Make the connection**
  - Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", “hr”, “hr”);
    - What do you think this statement does?
    - Establishes connection to database by obtaining a *Connection* object

2. Create JDBC statement(s)

- Statement stmt = con.createStatement() ;
- Creates a Statement object for sending SQL statements to the database

Executing SQL Statements

- String createLehigh = "Create table Lehigh " +
  "(SSN Integer not null, Name VARCHAR(32), " +
  "Marks Integer)";
  stmt.**executeUpdate**(createLehigh);
  //What does this statement do?

- String insertLehigh = "Insert into Lehigh values“ + "(123456789,abc,100)";
  stmt.**executeUpdate**(insertLehigh);

Get ResultSet

```
String queryLehigh = "select * from Lehigh";

ResultSet rs = Stmt.executeQuery(queryLehigh);
//What does this statement do?

while (rs.next()) {
   int ssn = rs.getInt("SSN");
   String name = rs.getString("NAME");
   int marks = rs.getInt("MARKS");
}
```

Close connection

- stmt.close();
- con.close();

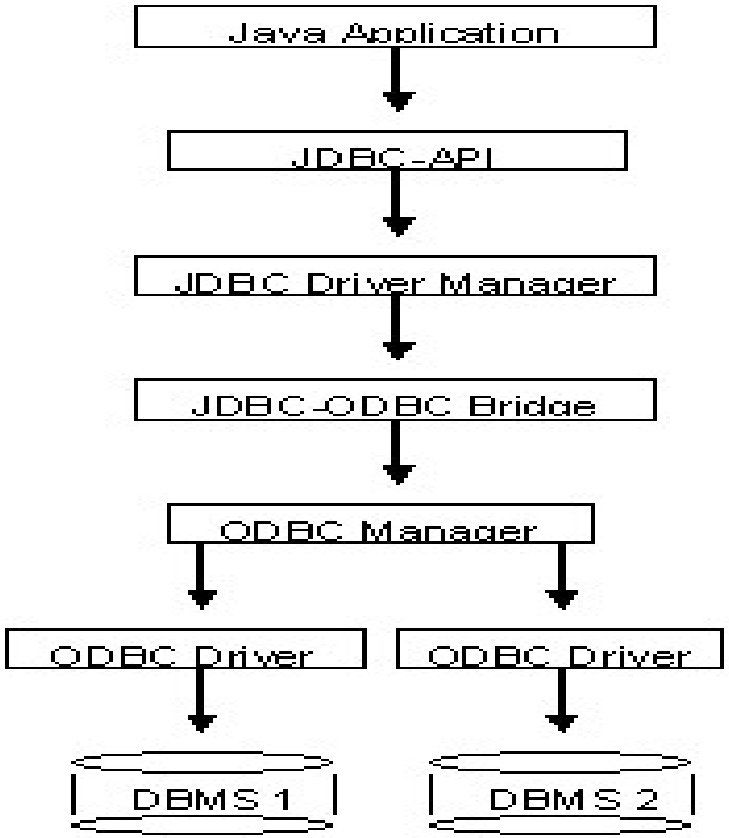
Transactions and JDBC

- JDBC allows SQL statements to be grouped together into a single transaction
- Transaction control is performed by the Connection object, default mode is auto-commit, I.e., each sql statement is treated as a transaction
- We can turn off the auto-commit mode with con.setAutoCommit(false);
- And turn it back on with con.setAutoCommit(true);
- Once auto-commit is off, no SQL statement will be committed until an explicit is invoked con.commit();
- At this point all changes done by the SQL statements will be made permanent in the database.

Handling Errors with Exceptions

- Programs should recover and leave the database in a consistent state.
- If a statement in the try block throws an exception or warning, it can be caught in one of the corresponding catch statements
- How might a finally {…} block be helpful here?
- E.g., you could rollback your transaction in a catch { …} block or close database connection and free database related resources in finally {…} block

Another way to access database (JDBC-ODBC)



What's a bit different about this architecture?

Why add yet another layer?

Sample program

```
import java.sql.*;
class Test  {
   public static void main(String[] args)  {
      try {
         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //dynamic loading of driver
         String filename = "c:/db1.mdb"; //Location of an Access database
         String database = "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=";
         database+= filename.trim() + ";DriverID=22;READONLY=true}"; //add on to end
         Connection con = DriverManager.getConnection( database ,"","");
         Statement s = con.createStatement();
         s.execute("create table TEST12345 ( firstcolumn integer )");
         s.execute("insert into TEST12345 values(1)");
         s.execute("select firstcolumn from TEST12345");
```

Sample program(cont)

```
        ResultSet rs = s.getResultSet();
        if (rs != null) // if rs == null, then there is no ResultSet to view
        while ( rs.next() ) // this will step through our data row-by-row
        {   /* the next line will get the first column in our current row's ResultSet
           as a String ( getString( columnNumber) ) and output it to the screen */
           System.out.println("Data from column_name: " + rs.getString(1) );
        }
        s.close(); // close Statement to let the database know we're done with it
        con.close(); //close connection
      }
     catch (Exception err) { System.out.println("ERROR: " + err);  }
   }
}
```

Mapping types JDBC - Java

JDBC Type	Java Type
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT DOUBLE	double
BINARY VARBINARY LONGVARBINARY	byte[]
CHAR VARCHAR LONGVARCHAR	String

JDBC Type	Java Type
NUMERIC DECIMAL	BigDecimal
DATE	java.sql.Date
TIME TIMESTAMP	java.sql.Timestamp
CLOB	Clob*
BLOB	Blob*
ARRAY	Array*
DISTINCT	mapping of underlying type
STRUCT	Struct*
REF	Ref*
JAVA_OBJECT	underlying Java class

*SQL3 data type supported in JDBC 2.0

JDBC 2 – Scrollable Result Set

```
…
Statement stmt =
con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                ResultSet.CONCUR_READ_ONLY);

String query = "select students from class where type='not sleeping' ";
ResultSet rs = stmt.executeQuery( query );

rs.previous();  / / go back in the RS (not possible in JDBC 1…)
rs.relative(-5); / / go 5 records back
rs.relative(7); / / go 7 records forward
rs.absolute(100); / / go to 100th record
…
```

JDBC 2 – Updateable ResultSet

```
…
Statement stmt =
con.createStatement(ResultSet.TYPE_FORWARD_ONLY,
                                ResultSet.CONCUR_UPDATABLE);
String query = " select students, grade from class
                    where type='really listening this presentation☺' ";
ResultSet rs = stmt.executeQuery( query );
…
while ( rs.next() )
{
   int grade = rs.getInt("grade");
   rs.updateInt("grade", grade+10);
   rs.updateRow();
}
```

Metadata from DB

- A Connection's database is able to provide schema information describing its tables, its supported SQL grammar, its stored procedures the capabilities of this connection, and so on
  - What is a stored procedure?
  - Group of SQL statements that form a logical unit and perform a particular task

This information is made available through a DatabaseMetaData object.

Metadata from DB - example

```
…
Connection con = …. ;

DatabaseMetaData dbmd = con.getMetaData();

String catalog = null;
String schema = null;
String table = "sys%";
String[ ] types = null;

ResultSet rs =
   dbmd.getTables(catalog , schema , table , types );
…
```

JDBC – Metadata from RS

```
public static void printRS(ResultSet rs) throws SQLException
{
   ResultSetMetaData md = rs.getMetaData();
   // get number of columns
   int nCols = md.getColumnCount();
   // print column names
   for(int i=1; i < nCols; ++i)
         System.out.print( md.getColumnName( i)+",");
    / / output resultset
   while ( rs.next() )
   {     for(int i=1; i < nCols; ++i)
               System.out.print( rs.getString( i)+",");
         System.out.println( rs.getString(nCols) );
   }
}
```

Thank You