Lecture 05
JDBC Database Access

JDBC- Java Database Connectivity
(5 slots)

References:

- **Java-Tutorials/tutorial-2015/jdbc/index.html**
- **Java Documentation, the java.sql package**

Why should you study this lecture?

- In almost all large applications. Data are organized and stored in databases which are managed by database management systems (DBMS) such as MS Access, MS SQL Server, Oracle, My SQL,…
- Do you want to create Java applications which can connect to DBMSs?
- Database programming is a skill which can not be missed for programmers.

Objectives

- Introduction to databases
- Relational Database Overview
- JDBC and JDBC Drivers
- Steps to develop a JDBC application.
- Demonstrations.

Contents

1- Database and DBMS
2- Relational Database Overview
3- JDBC and JDBC Drivers
4- Steps to develop a JDBC Application
5- A Demonstration

1- Database and DBMS

- **Database** is a collection of related data which are stored in secondary mass storage and are used by some processes concurrently.
- Databases are organized in some ways in order to reduce redundancies.
- **DBMS**: Database management system is a software which manages some databases. It supports ways to users/processes for creating, updating, manipulating on databases and security mechanisms are supported also.
- DBMS libraries (C/C++ codes are usually used) support APIs for user programs to manipulate databases.

2- Relational Database Overview

- Common databases are designed and implemented based on relational algebra (set theory).
- Relational database is one that presents information in tables with rows and columns.
- A table is referred to as a relation in the sense that it is a collection of objects of the same type (rows).
- A Relational Database Management System (RDBMS)- such as MS Access, MS SQL Server, Oracle- handles the way data is stored, maintained, and retrieved.

Table - dbo.Items

itemCode	itemName	supCode	unit	price
E0001	Mouse Proview	MT	block 10	30
E0002	Keyboard Proview	MT	block 10	40
E0003	LCD	MT	1-unit	90
E0004	Main Asus MK1234	HT	1-unit	78
E0005	Main Gigabyte GM34A	HT	1-unit	67

RDBMS: Structure Query Language (SQL)

Data Definition Language (DDL):
CREATE…/ ALTER…/ DROP…

3 languages:

Table - dbo.Items

itemCode	itemName	supCode	unit	price
E0001	Mouse Proview	MT	block 10	30
E0002	Keyboard Proview	MT	block 10	40
E0003	LCD	MT	1-unit	90
E0004	Main Asus MK1234	HT	1-unit	78
E0005	Main Gigabyte GM34A	HT	1-unit	67

Data Manipulating Language (DML):
SELECT…/ INSERT INTO … / UPDATE … / DELETE

Data Control Language (DCL):
GRANT…/ REVOKE … / DENY…

RDBMS: SQL…

- ***Common DML queries***:
  - SELECT columns FROM tables WHERE condition
  - UPDATE table SET column=value,… Where condition
  - DELETE FROM table WHERE condition
  - INSERT INTO table Values (val1, val2,…)
  - INSERT INTO table (col1, col2,…) Values (val1, val2,…)

3-JDBC and JDBC Driver

- The JDBC™ API was designed to keep simple things simple. This means that the JDBC makes everyday database tasks easy. This trail walks you through examples of using JDBC to execute common SQL statements, and perform other objectives common to database applications.
- The JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database.

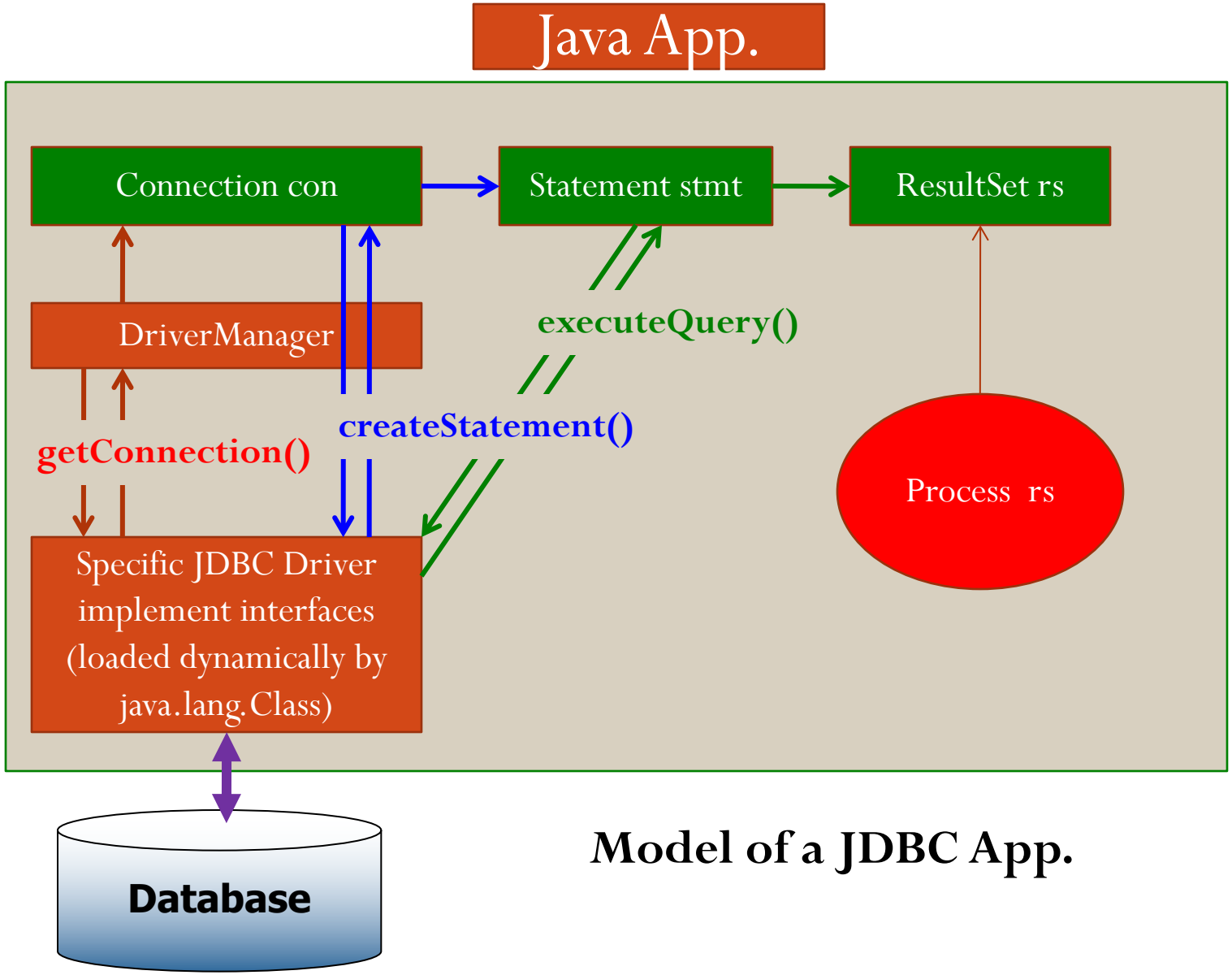
JDBC and JDBC Driver…

- JDBC APIs has 02 parts in the **java.sql** package.

Part	Details	Purposes
JDBC Driver	**DriverManager** class	Java.lang.Class.forName(DriverClass) will dynamically load the concrete driver class, provided by a **specific provider for a specific database**. This class implemented methods declared in JDBC interfaces. The class DriverManager will get a connection to database based on the specific driver class loaded.
JDBC API	Interfaces: **Connection,** **Statement** **ResultSet** **DatabaseMetadata** **ResultSetMetadata** Classes **SQLException**	 For creating a connection to a DBMS For executing SQL statements For storing result data set and achieving columns For getting database metadata For getting resultset metadata

Refer to the java.sql package for more details in Java documentation

JDBC and JDBC Driver…

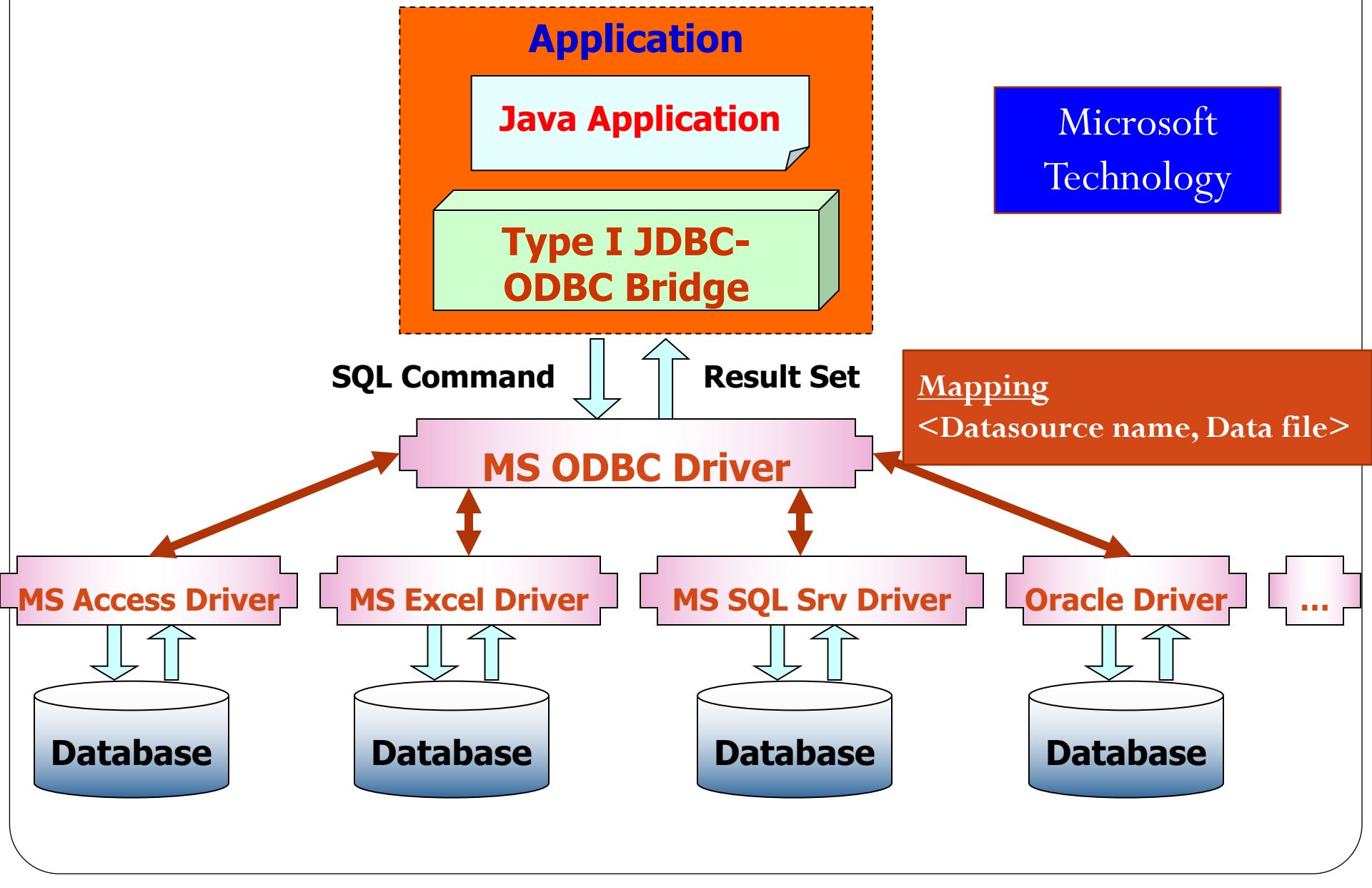


Model of a JDBC App.

JDBC and JDBC Driver…

- DBMS provider/developer will supply a package in which specific classes implementing standard JDBC driver (free).
- Based on characteristics of DBMSs, four types of JDBC drivers are:
  - Type 1: JDBC ODBC
  - Type 2: Native API
  - Type 3: Network Protocol
  - Type 4: Native Protocol
- Type 1 and Type 4 are populated.

Type 1-Driver : JDBC-ODBC Bridge

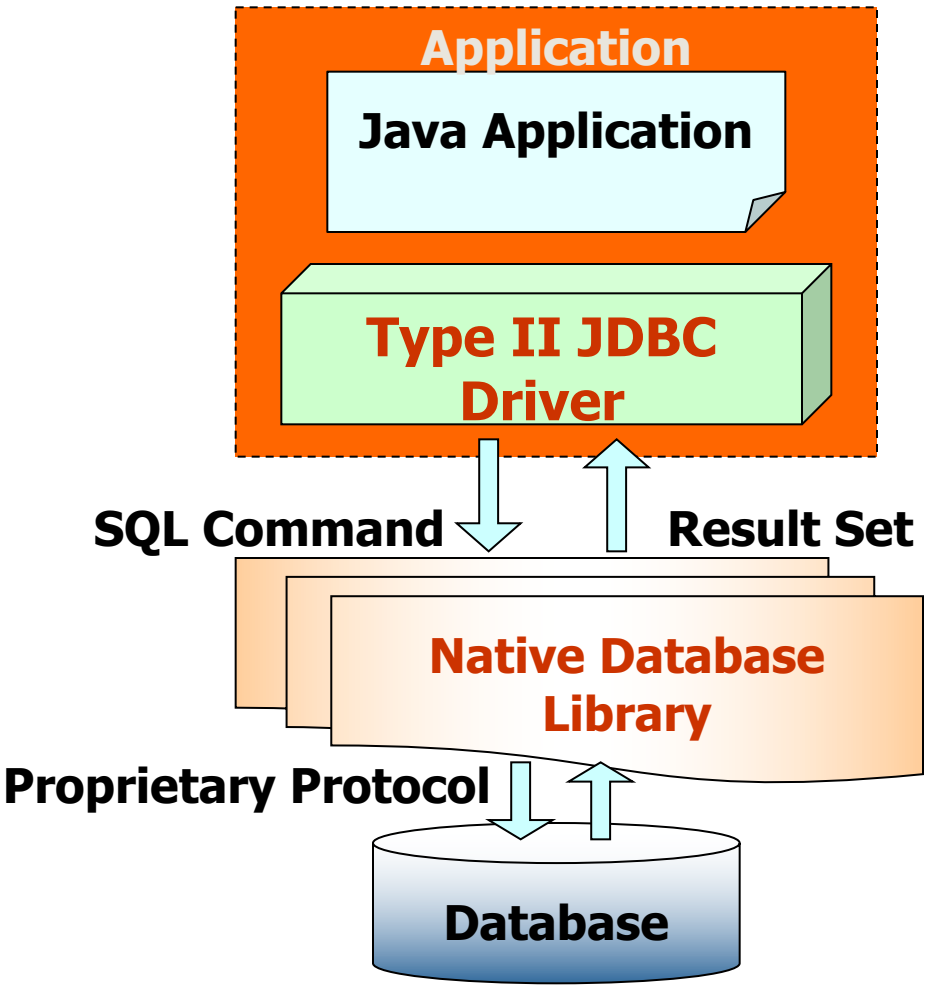


Type 1-Driver : JDBC-ODBC…

- This package is in the JDK as default.
- Translates JDBC APIs to ODBC APIs
- Enables the Java applications to interact with any database supported by Microsoft.
- Provides platform dependence, as JDBC ODBC bridge driver uses ODBC
- **JDBC-ODBC bridge is useful when Java driver is not available for a database but it is supported by Microsoft.**
- Disadvantages
  - Platform dependence (Microsoft)
  - The performance is comparatively slower than other drivers
  - Require the ODBC driver and the client DB to be on the server.
- Usage: DSN is registered to use connecting DB (a data source is declared in Control Panel/ODBC Data sources)

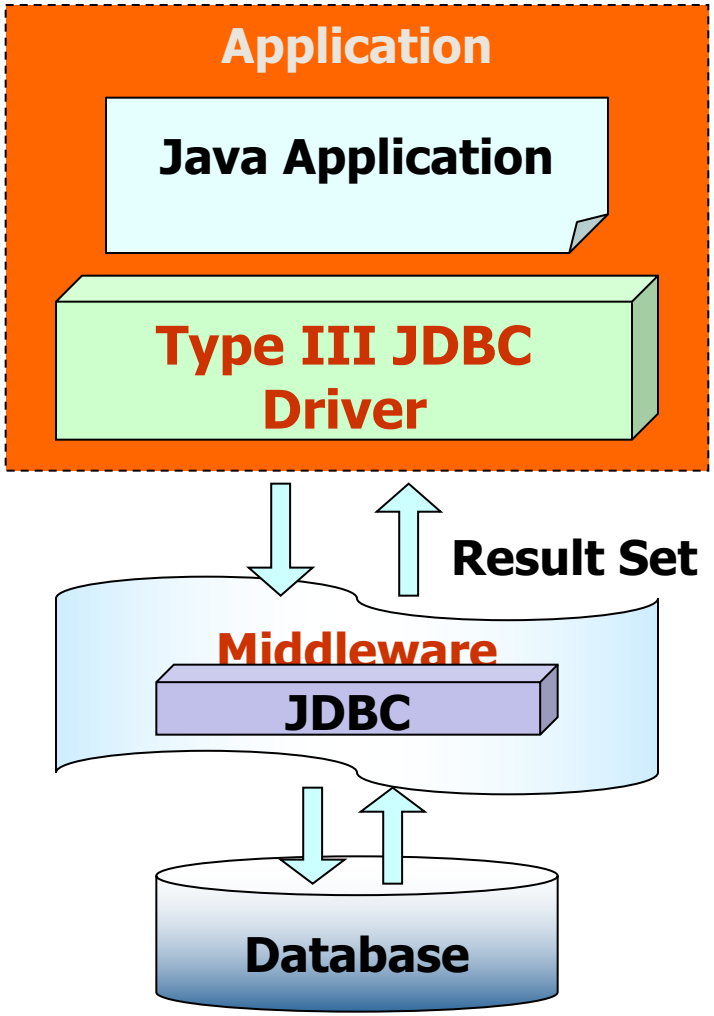
Type 2-Driver: Native API

- Provides access to the database through C/C++ codes.
- Developed using native code libraries
- Native code libraries provide access to the database, and improve the performance
- Java application sends a request for database connectivity as a normal JDBC call to the Native API driver
- Establishes the call, and translates the call to the particular database protocol that is forwarded to the database



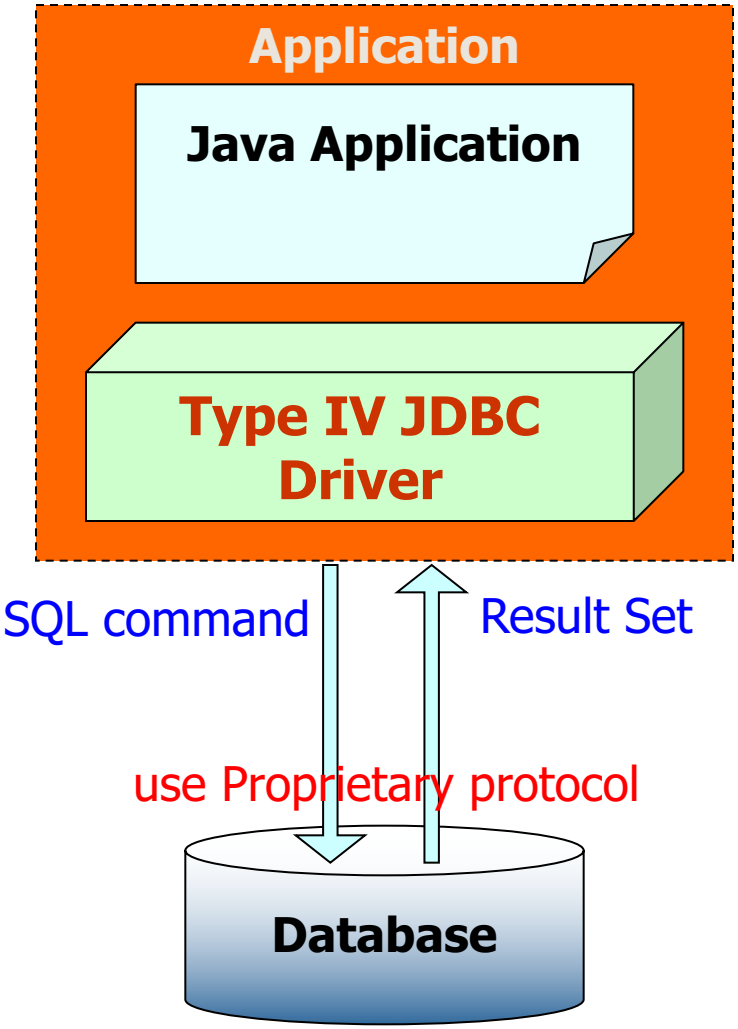
Type 3-Driver: Network Protocol

- Use a pure Java client and communicate with a middleware server using a database-independent protocol.
- The middleware server then communicates the client's requests to the data source
- Manages multiple Java applications connecting to different databases



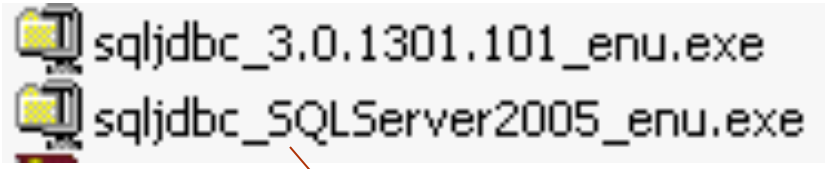
Type 4-Driver: Native Protocol

- Communicates directly with the database using Java sockets
- Improves the performance as translation is not required
- Converts JDBC queries into native calls used by the particular RDBMS
- The driver library is required when it is used and attached with the deployed application (**sqlserver 2000**: mssqlserver.jar, msutil.jar, msbase.jar; **sqlserver 2005**: sqljdbc.jar; **jtds**: jtds.jar …)
- Independent platform



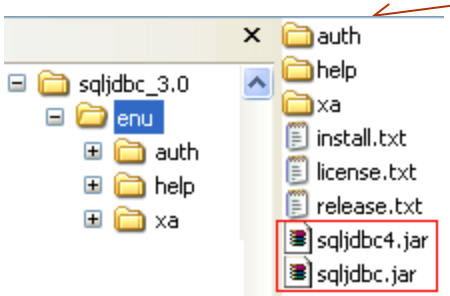
Download Type 4 SQL Server JDBC

Google : Microsoft SQL Server JDBC Driver



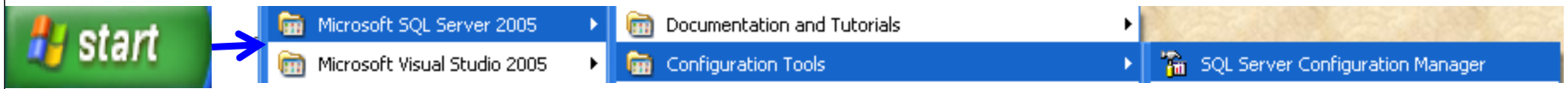
MS SQL Server 2008
MS SQL Server 2005

Setup

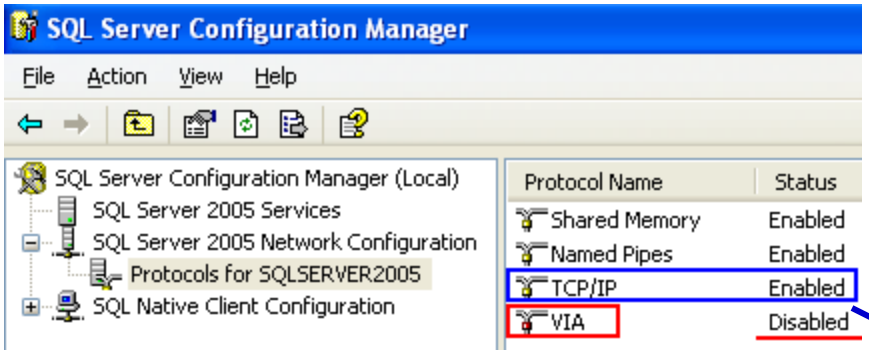


Latest Driver Release:	7.08
Last Update:	Oct 15, 2010
Java Version:	1.4 or higher for JDBC 3.0 1.6 or higher for JDBC 4.0
JDBC API Level:	3.0 / 4.0
Driver Type:	4
Supported DBMS:	MS SQL Server 6.5 - 2008 with all Service Packs (32 bit / 64 bit)
Download Size:	472 KB
Driver Size:	230 KB
Sun Certificate for J2EE 1.3:	Yes

Configure Ports, Protocols for SQL Server

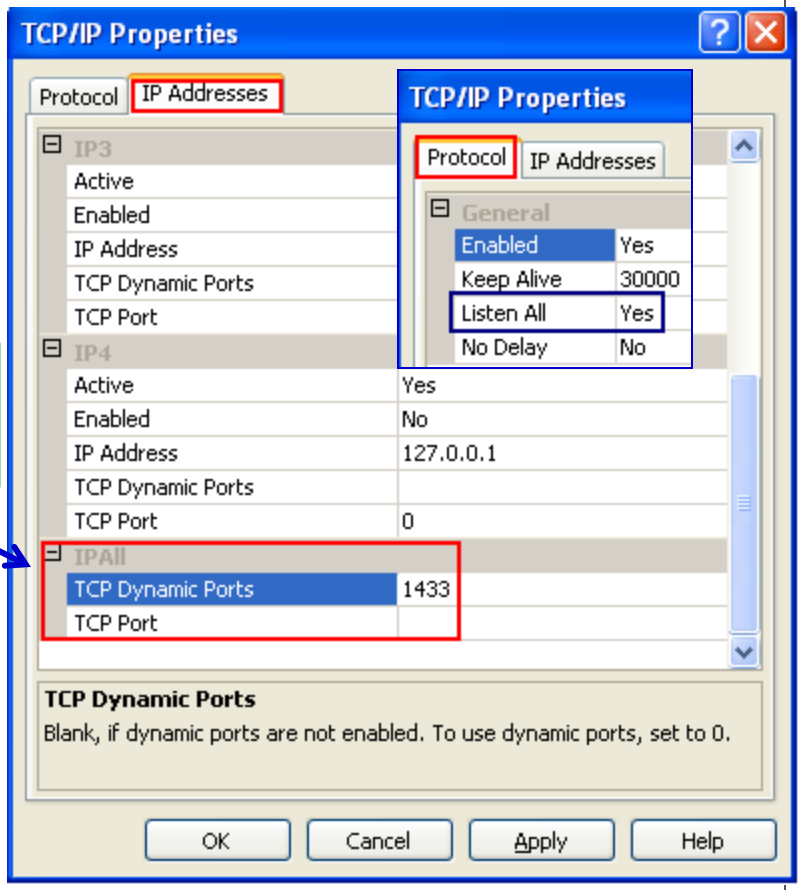


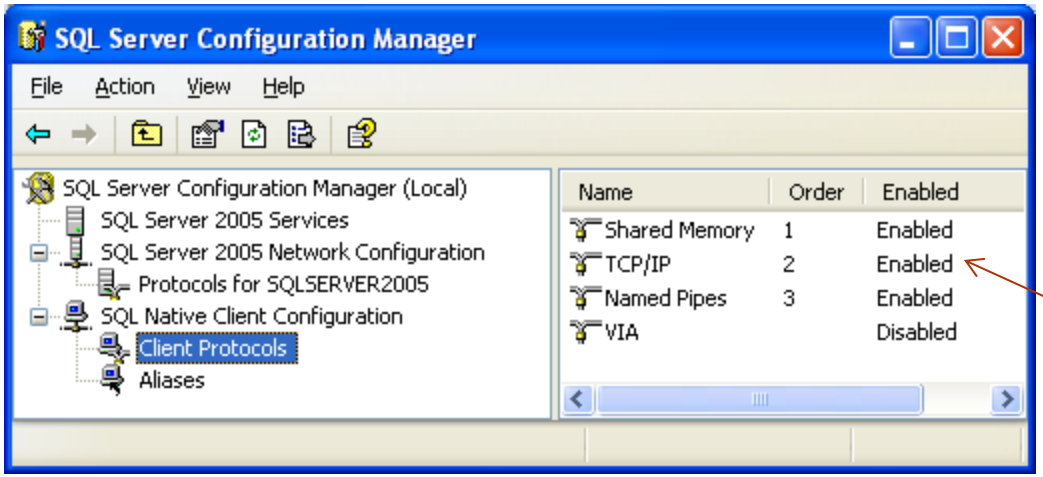
Enable Server protocols and port



Attention: Disable VIA

Right click

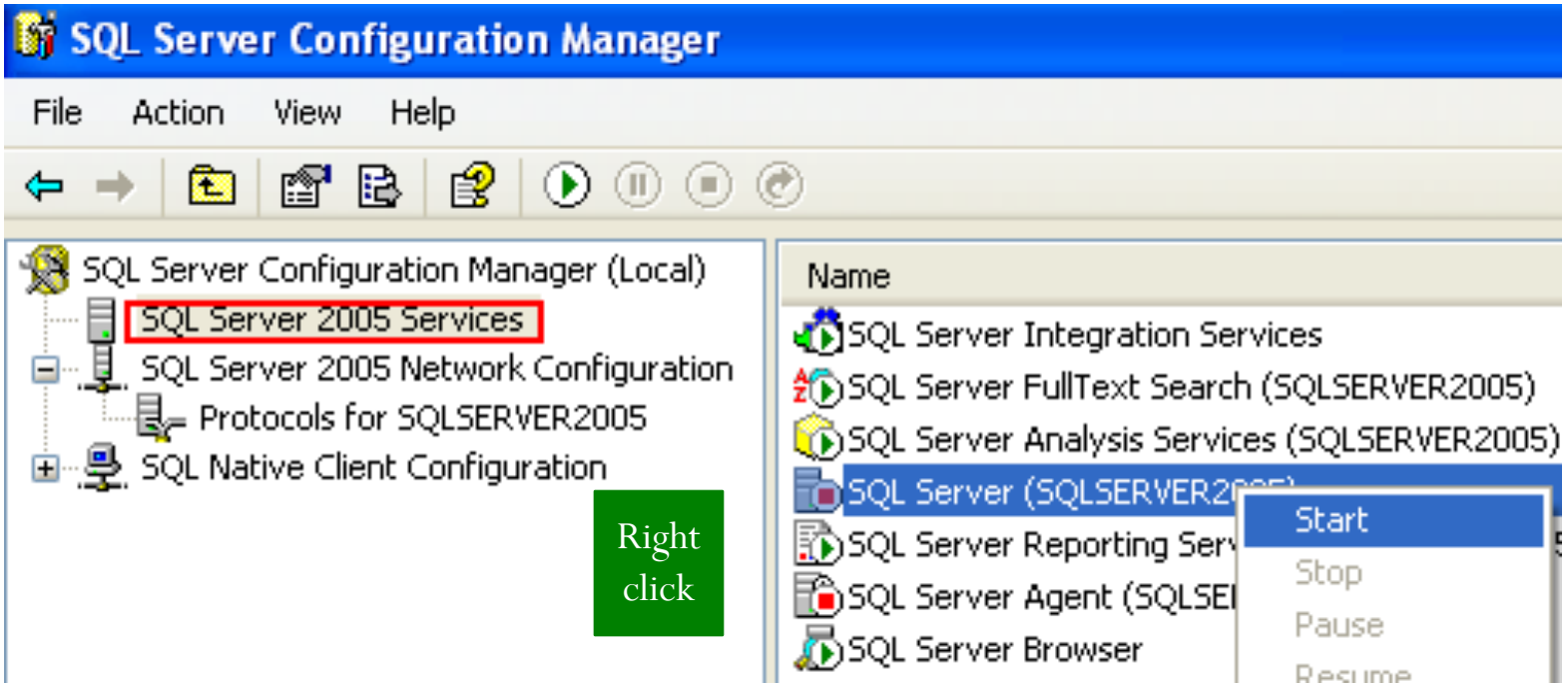




Enable client protocols and port

Configure Ports, Protocols for SQL Server…

Stop then restart SQL Server and SQL Server Agent for settings are affected.



4-Steps to Develop a JDBC Application

Step	Description	Use (java.sql package)	Methods
1	**Load JDBC Driver**	Java.lang.Class	forName(…)
2	**Establish a DB connection**	java.sql.Connection java.sql.DriverManager	DriverManager getConnection(…) → Connection
3	**Create & execute SQL statements**	java.sql.Statement java.sql.PrepareStatement java.sql.CallableStatement	execute(…) executeQuery(…) → SELECT executeUpdate(…) → INSERT/UPDATE/DELETE
4	**Process the results**	java.sql.ResultSet	first(), last(), next(), previous() getXXX(..)
5	**Close**	ResultSet, Statement, Connection	close()

Step 1: Register JDBC Driver
Step 2: Establish a connection to DB

Java

Driver Class

Driver Type 1 with Data Source Name registered in ODBC

```
// Open a connection to database registered a Data source name
Connection openConnection1(){
  String driver="sun.jdbc.odbc.JdbcOdbcDriver"; // Driver Type 1
  String url="jdbc:odbc:KZone"; // DSN of the KidZoneDB database
  String uid="sa", pwd="";
  Connection c = null;
  try {
    Class.forName(driver); // loading driver
    c= DriverManager.getConnection(url,uid,pwd); // connect
  }
  catch (Exception e)
  { JOptionPane.showMessageDialog(this, e);
   // System.exit(0);
  }
  return c;
}
```

Attention to the syntax of URL

Step 1: Register JDBC Driver
Step 2: Establish a connection to DB

Java

Driver type 4 (MS SQL Server)

```
// Open a connection with Type 4 driver of Microsoft
Connection openConnection2()
{ String IP = "127.0.0.1" ; // or "localhost" or "computer name".
  // You CAN NOT use "." for local host with Type 4 driver of Microsoft
  String instanceName="SQLSERVER2005";
  String db = "KidzoneDB";
  String uid="sa";
  String pwd="";
  String port="1433";
  Connection c=null;
  String driver="com.microsoft.sqlserver.jdbc.SQLServerDriver";
  String url = "jdbc:sqlserver://" + IP + "\\" + instanceName + ":" + port  +
               ";databaseName=" + db + ";user=" + uid + ";password=" + pwd;
  try
   { Class.forName(driver);
     c = DriverManager.getConnection(url);
   }
   catch (Exception e)
   { JOptionPane.showMessageDialog(this, e.getMessage());
    System.exit(0);
   }
  return c;
}
```

Driver Class

Attention to the syntax of URL

Step 3: Create &Execute a SQL statement

```
String sql1 = "SELECT  columns FROM table1, table2, …  WHERE condition";
String sql2 = "UPDATE table SET column = value, … WHERE  condition";
String sql3 = "INSERT INTO table VALUES ( val1, val2, … )" ;
String sql4 = "INSERT INTO table  (col1, col2, col3) VALUES ( val1, val2, val3)" ;
String sql5 =  "UPDATE table SET col1 = ?, col2=?  WHERE  condition";
```

```
// Connection con was created
Statement stmt= con.createStatement();
ResultSet rs= stmt.executeQuery(sql1);
int numOfInfectedRows = stmt.executeUpdate(sql2);
int numOfInfectedRows = stmt.executeUpdate(sql3);
int numOfInfectedRows = stmt.executeUpdate(sql4);

PreparedStatement pStmt = con.preparedStatement(sql5);
pStmt.setXXX (index, val);  // from 1
int numOfInfectedRows = pStmt.executeUpdate(); // no argument
```

Step 4: Process the results

Move the current row:

boolean next(), previous(), first(), last()

Default: Result set moves forward only.

Get data in columns of the current row:

TYPE getTYPE (int columnIndex) // begin from 1

TYPE getTYPE (String columnLabel)

SELECT desc AS description FROM T_employee

➔ **Column name: desc**

➔ **Column Label: description**

At a time, resultset maintains a current position. When the resultset is initialized, the position is the BOF position. An exception is thrown when the current position is out of its scope.

Step 5: Close the connection

Opening Order:	Connection	→ Statement	→ ResultSet
Closing Order:	ResultSet	→ Statement	→ Connection

Attention!!!

At a time, a connection can be bound with ONLY ONE result set.

An exception will be thrown if we try binding a connection with another result set.

EX:

```
String sql1 ="SELECT…";
String sql2 ="SELECT…";
ResultSet rs1= stmt.executeQuery(sql1);
ResultSet rs2= stmt.executeQuery(sql2); ➔ EXCEPTION
```

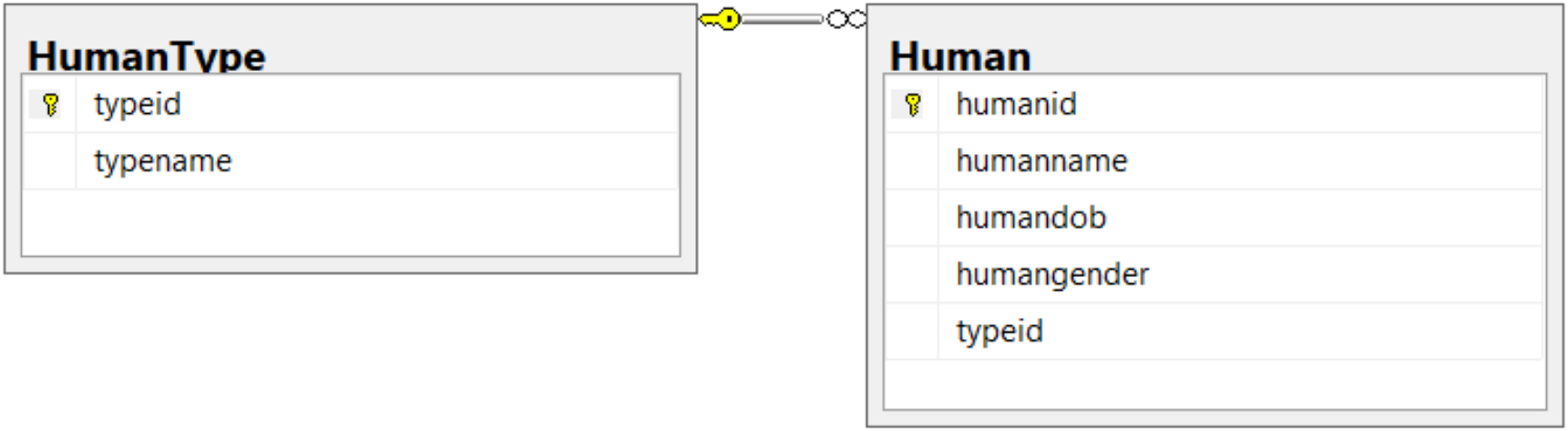
➔You should close the rs1 before trying get the rs2 result set

➔Solution: Transfer data in the rs1 to ArrayList (or Vector) then close rs1 before get new data to rs2.

Demonstrations

(Demo 1) Create database

- Use MS Access or MS SQL Server 2008
- Database name: Human
- Tables and Relationship:



You can download this database file from CMS.

(Demo 1) Create database…

Initial data:

DESKTOP-VCU483E\...- dbo.HumanType

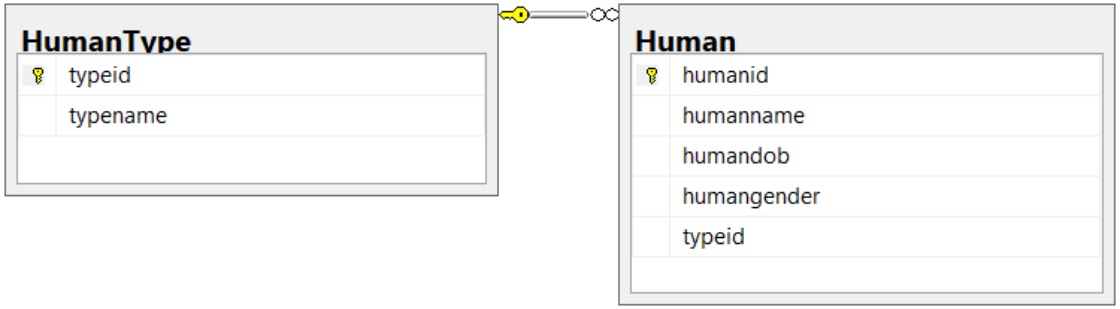
	typeid	typename
	1	student
	2	teacher
	3	worker

DESKTOP-VCU483E...uman - dbo.Human

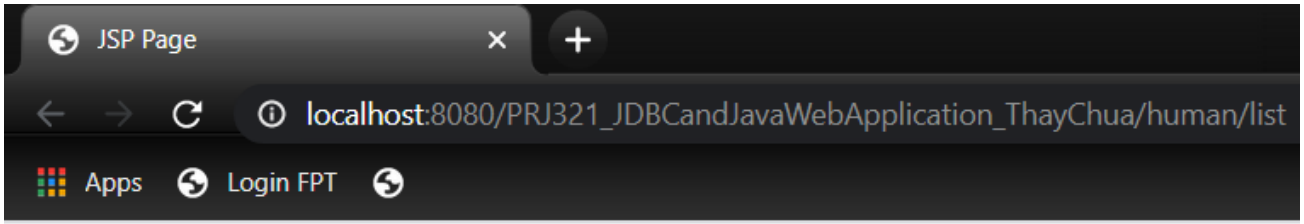
	humanid	humanname	humandob	humangen...	typeid
	1	Steve	2020-09-09	True	1
	2	Elon	2019-09-09	False	2
	3	Musk	2020-09-09	True	3
	4	Melon	2019-08-09	True	1
	5	Kais	2020-07-08	False	3

(Demo 3) Develop the program for managing items using MS Sql Server JDBC

- Database:



- Program GUI

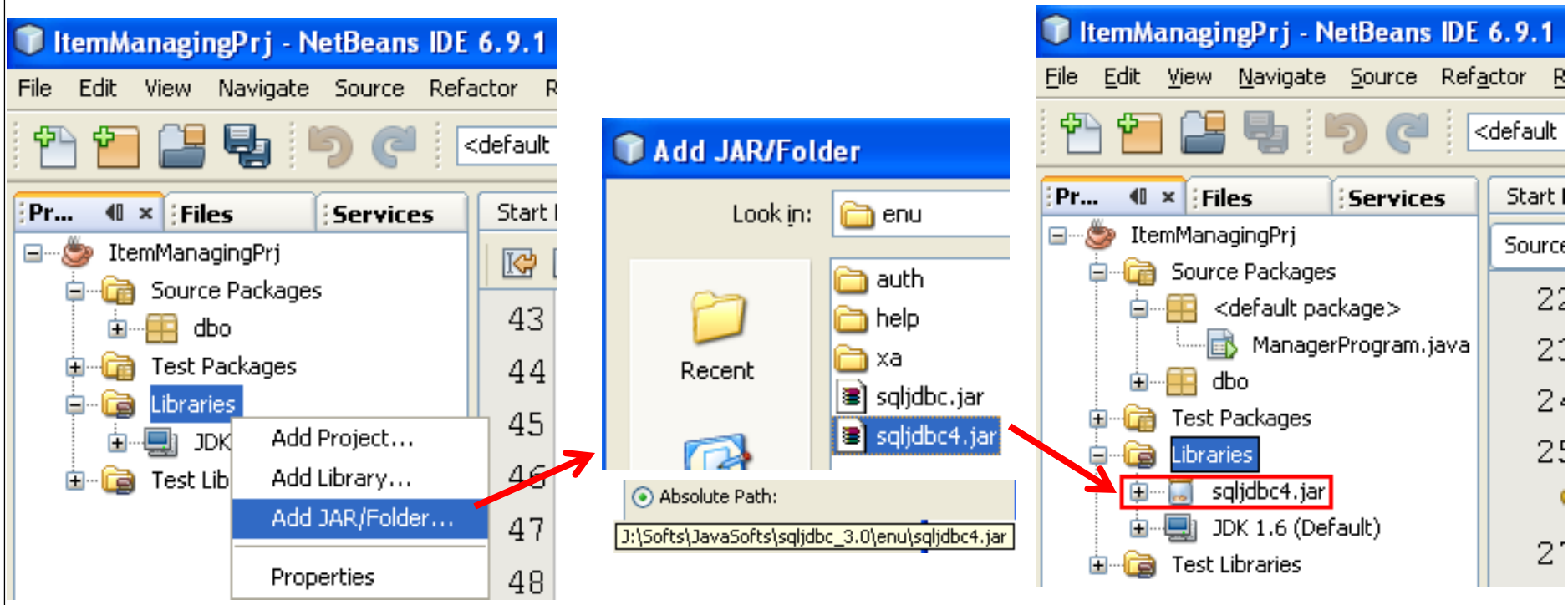


Name	DOB	Gender	Type	
Steve	2020-09-09	♂	student	edit delete
Elon	2019-09-09	♀	teacher	edit delete
Musk	2020-09-09	♂	worker	edit delete
Melon	2019-08-09	♂	student	edit delete
Kais	2020-07-08	♀	worker	edit delete

Insert

Demo 3…

- **Add MS SQL Server JDBC to the NetBeans:**



Demo 3...

DBContext.java

Source | History

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package dal;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;
import model.Human;
import model.HumanType;

public class DBContext {
    Connection connection;

    public DBContext()
    {
        try {
            String user = "sa";
            String pass = "sa";
            String url = "jdbc:sqlserver://localhost:1433;databaseName=Human";
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
            connection = DriverManager.getConnection(url, user, pass);
        } catch (ClassNotFoundException | SQLException ex) {
            Logger.getLogger(DBContext.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```

Demo 3...

Human.java

Source | History

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package model;
7
8  import java.sql.Date;
9
10 /**
11  *
12  * @author sonnt
13  */
14 public class Human {
15     private int ID;
16     private String Name;
17     private Date dob;
18     private boolean Gender;
19     private HumanType type;
20
21     public int getID() {...3 lines }
24
25     public void setID(int ID) {...3 lines }
28
29     public String getName() {...3 lines }
32
33     public void setName(String Name) {...3 lines }
36
37     public Date getDob() {...3 lines }
40
41     public void setDob(Date dob) {...3 lines }
44
45     public boolean isGender() {...3 lines }
48
49     public void setGender(boolean Gender) {...3 lines }
52
53     public HumanType getType() {...3 lines }
56
57     public void setType(HumanType type) {...3 lines }
60
61 }
```

Human

	Column Name	Data Type	Allow Nulls
🔑	humanid	int	☐
	humanname	nvarchar(100)	☑
	humandob	date	☑
	humangender	bit	☑
	typeid	int	☑
			☐

Name	DOB	Gender	Type
Steve	2020-09-09	♂	student
Elon	2019-09-09	♀	teacher
Musk	2020-09-09	♂	worker
Melon	2019-08-09	♂	student
Kais	2020-07-08	♀	worker

Type student

Demo 3...

ListController.java

Source | History

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package controller.human;
7
8  import dal.DBContext;
9  import java.io.IOException;
   import java.io.PrintWriter;
11 import java.util.ArrayList;
12 import javax.servlet.ServletException;
13 import javax.servlet.http.HttpServlet;
14 import javax.servlet.http.HttpServletRequest;
15 import javax.servlet.http.HttpServletResponse;
16 import model.Human;
17
18 /**...4 lines */
22 public class ListController extends HttpServlet {
23     /** Processes requests for both HTTP <code>GET</code> and <code>POST</code> ...9 lines */
32     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
33             throws ServletException, IOException {
34         DBContext db = new DBContext();
35         ArrayList<Human> humans = db.getHumans();
36         request.setAttribute("humans", humans);
37         request.getRequestDispatcher("../view/human/list.jsp").forward(request, response);
38     }
39
40     HttpServlet methods. Click on the + sign on the left to edit the code.
78
79 }
```

Demo 3...

DBContext.java

Source | History

```
35      public ArrayList<Human> getHumans()
36      {
37          ArrayList<Human> humans = new ArrayList<>();
38          try {
39              String sql = "SELECT h.humanid,h.humanname,h.humandob,h.humangender,ht.typeid,ht.typename "
40                      + " FROM Human h INNER JOIN HumanType ht ON h.typeid = ht.typeid";
41              PreparedStatement statement = connection.prepareStatement(sql);
42              ResultSet rs = statement.executeQuery();
43              while(rs.next())
44              {
45                  Human h = new Human();
46                  h.setID( rs.getInt("humanid")  );
47                  h.setName(rs.getString("humanname")  );
48                  h.setDob(rs.getDate("humandob")  );
49                  h.setGender(rs.getBoolean("humangender")  );
50
51                  HumanType ht = new HumanType();
52                  ht.setTypeiD(rs.getInt("typeid")  );
53                  ht.setName(rs.getString("typename")  );
54                  h.setType(ht);
55                  humans.add(h);
56              }
57          } catch (SQLException ex) {
58              Logger.getLogger(DBContext.class.getName()).log(Level.SEVERE, null, ex);
59          }
60
61          return humans;
62      }
```

Demo 3…

HumanType.java

Source | History

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package model;
7
8  public class HumanType {
9      private int typeiD;
10     private String Name;
11
12     public int getTypeiD() {...3 lines }
15
16     public void setTypeiD(int typeiD) {...3 lines }
19
20     public String getName() {...3 lines }
23
24     public void setName(String Name) {...3 lines }
27
28 }
```

HumanType

	Column Name	Data Type	Allow Nulls
🔑	typeid	int	☐
	typename	nvarchar(100)	☑
			☐

Demo 3...

InsertController.java

Source | History

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package controller.human;
7
8  import dal.DBContext;
9  import java.io.IOException;
10 import java.io.PrintWriter;
11 import java.sql.Date;
12 import java.util.ArrayList;
13 import javax.servlet.ServletException;
14 import javax.servlet.http.HttpServlet;
15 import javax.servlet.http.HttpServletRequest;
16 import javax.servlet.http.HttpServletResponse;
17 import model.Human;
18 import model.HumanType;
19
20 /**...4 lines */
24 public class InsertController extends HttpServlet {
25
26     // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the left to edit the code.">
27     /** Handles the HTTP <code>GET</code> method ...8 lines */
35     @Override
36     protected void doGet(HttpServletRequest request, HttpServletResponse response)
37             throws ServletException, IOException {
38         DBContext db = new DBContext();
39         ArrayList<HumanType> types = db.getTypes();
40         request.setAttribute("types", types);
41         request.getRequestDispatcher("../view/human/insert.jsp").forward(request, response);
42     }
43
44     /** Handles the HTTP <code>POST</code> method ...8 lines */
52     @Override
53     protected void doPost(HttpServletRequest request, HttpServletResponse response)
54             throws ServletException, IOException {...22 lines }
76
77     /**
78      * Returns a short description of the servlet.
79      *
80      * @return a String containing servlet description
81      */
82     @Override
83     public String getServletInfo() {...3 lines }// </editor-fold>
86
87 }
```

Demo 3…

DBContext.java

Source | History

```
64
65      public ArrayList<HumanType> getTypes()
66      {
67          ArrayList<HumanType> types = new ArrayList<>();
68          try {
69              String sql = "SELECT [typeid]\n" +
70                           "      ,[typename]\n" +
71                           "  FROM [HumanType] ";
72              PreparedStatement statement = connection.prepareStatement(sql);
73              ResultSet rs = statement.executeQuery();
74              while(rs.next())
75              {
76                  HumanType ht = new HumanType();
77                  ht.setTypeiD(rs.getInt("typeid")  );
78                  ht.setName(rs.getString("typename")  );
79                  types.add(ht);
80              }
81          } catch (SQLException ex) {
82              Logger.getLogger(DBContext.class.getName()).log(Level.SEVERE, null, ex);
83          }
84
85          return types;
86      }
```

Summary

- Introduction to databases
- Relational Database Overview
- JDBC and JDBC Drivers
- Steps to develop a JDBC application.
- Demonstrations

Thank You