Lecture 05
JDBC Database Access

JDBC- Java Database Connectivity

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.
- Test connection in Netbeans.

Contents

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)

3 languages:

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

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.

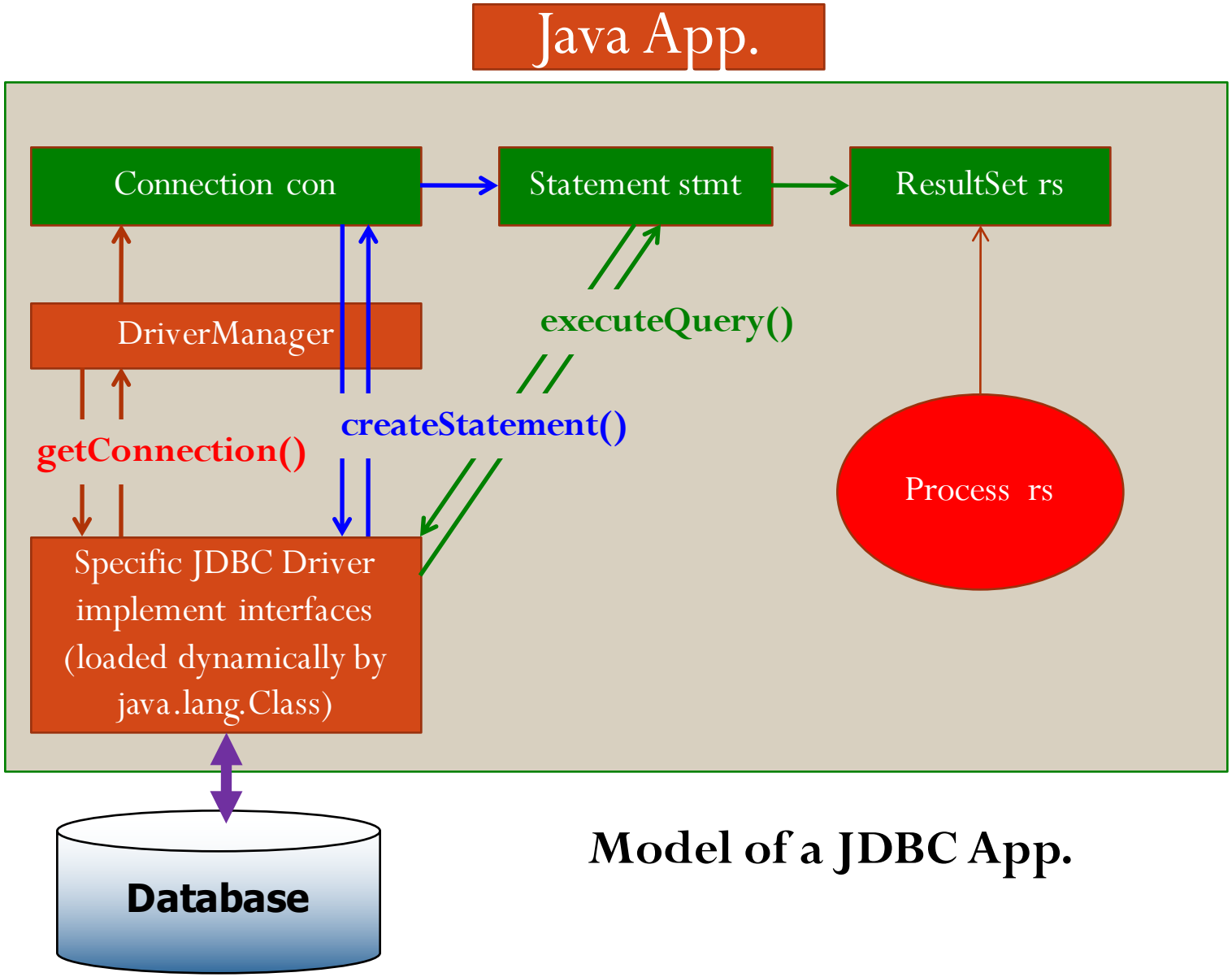
JDBC and JDBC Driver…

- 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

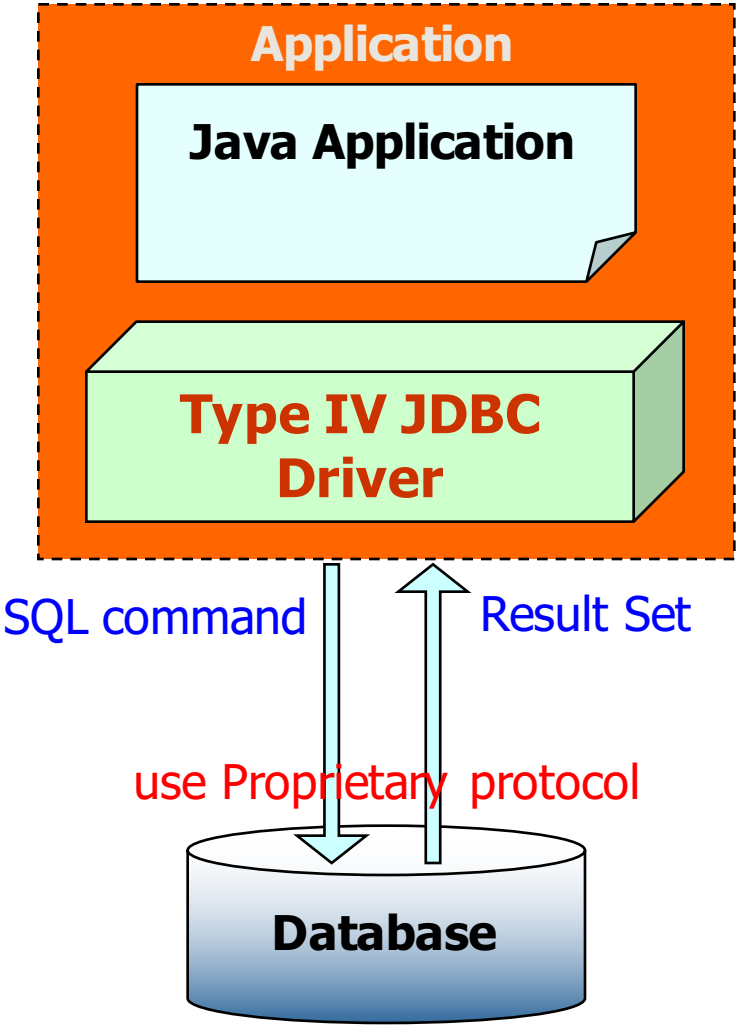
JDBC and JDBC Driver…



Model of a JDBC App.

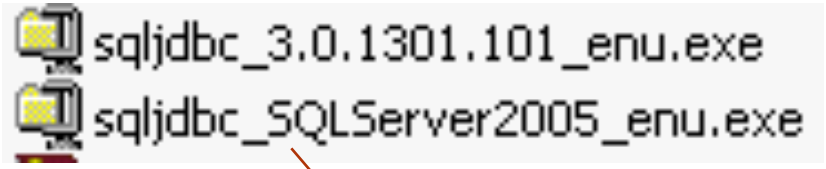
Type 4-Driver: Native Protocol

- 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



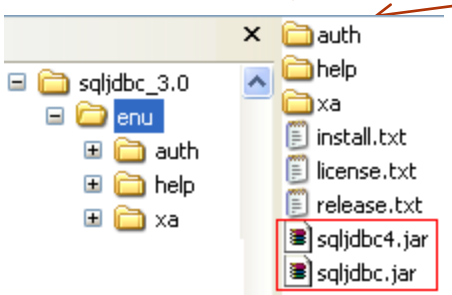
Download Type 4 SQL Server JDBC

Google : Microsoft SQL Server JDBC Driver



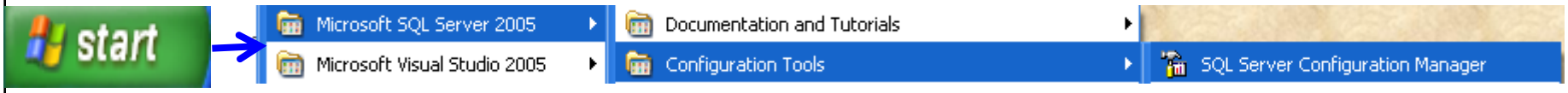
MS SQL Server 2008
MS SQL Server 2005

Setup

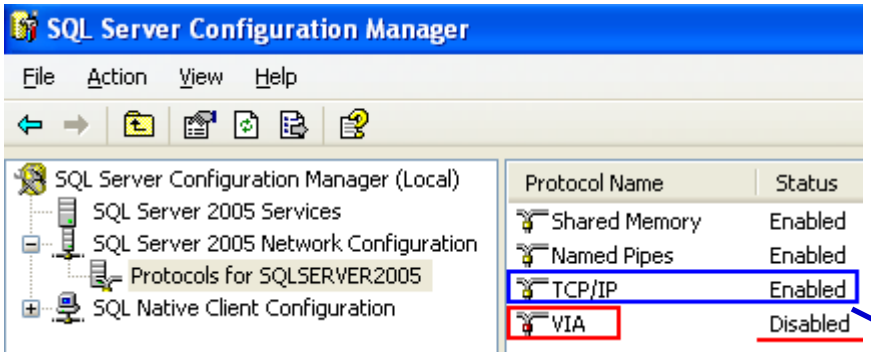


Latest Driver Release:	7.08
Last Update:	Oct 15, 2010
Java Version:	1.4 or higher for JDBC 3.0 1.6 or higher for JDBC 4.0
JDBC API Level:	3.0 / 4.0
Driver Type:	4
Supported DBMS:	MS SQL Server 6.5 - 2008 with all Service Packs (32 bit / 64 bit)
Download Size:	472 KB
Driver Size:	230 KB
Sun Certificate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

Configure Ports, Protocols for SQL Server

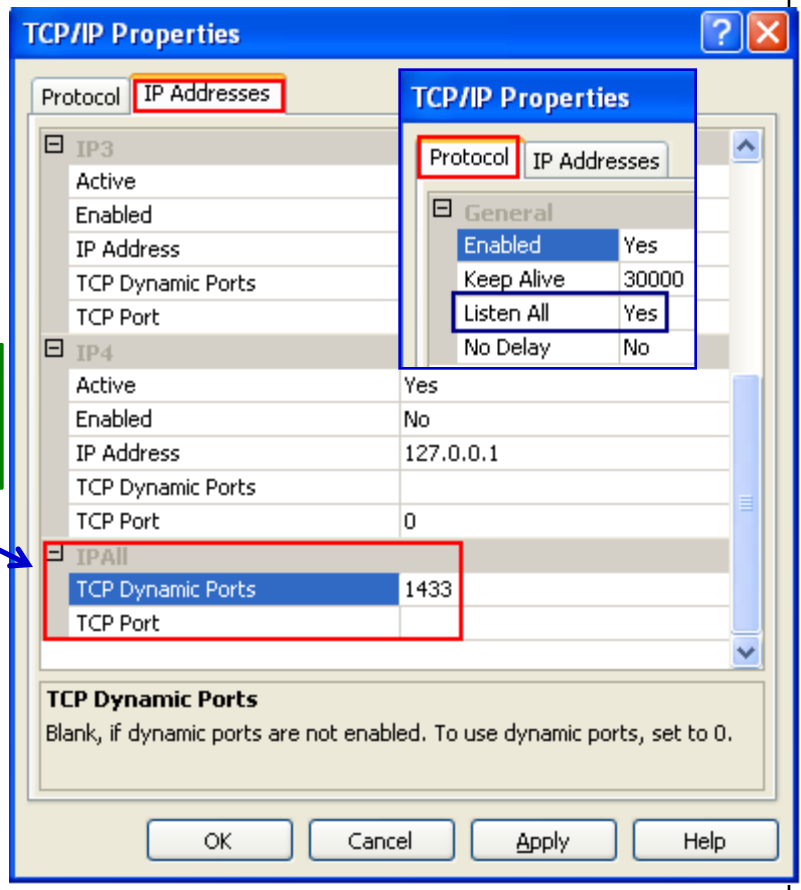


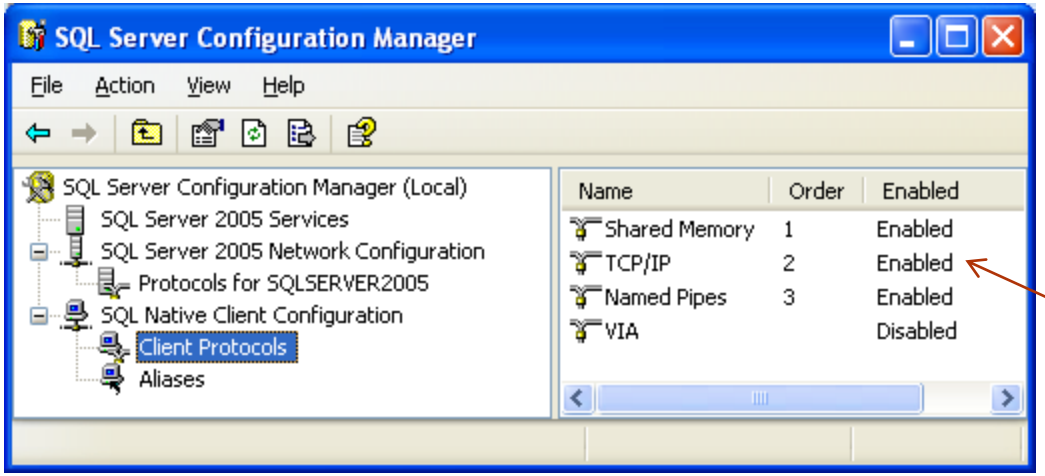
Enable Server protocols and port



Attention: Disable VIA

Right click

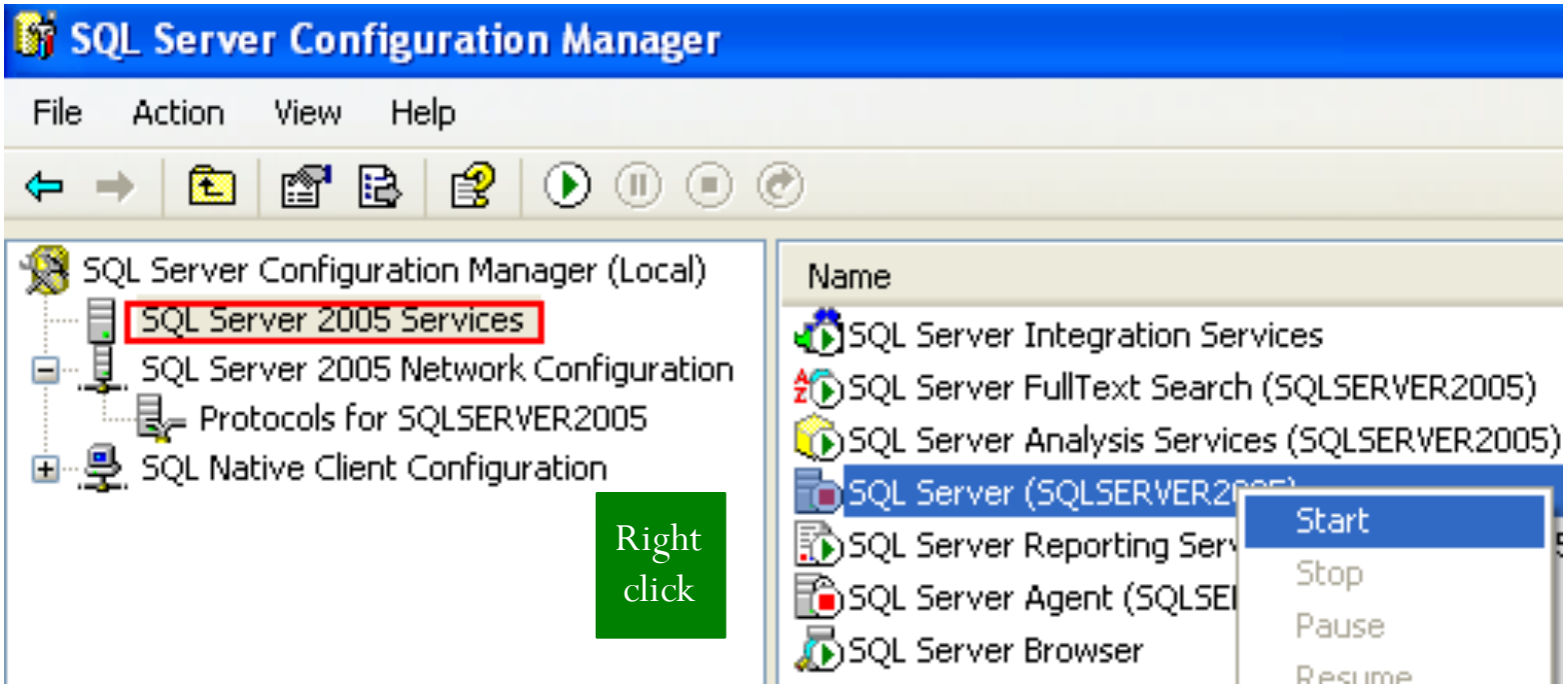




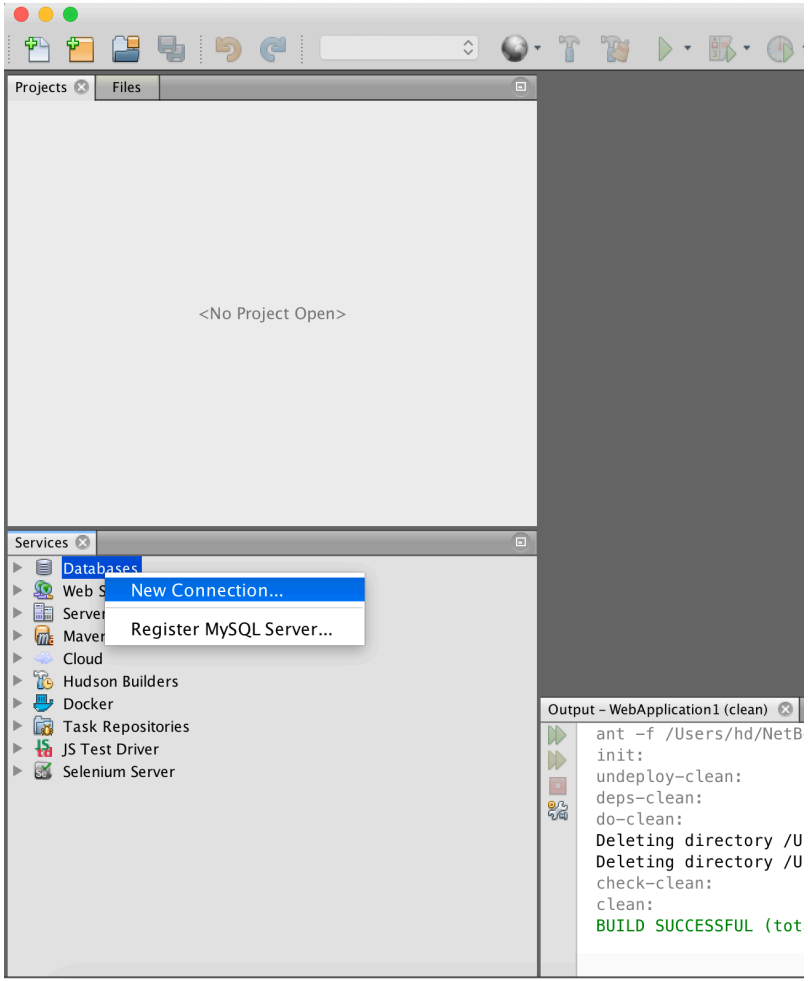
Enable client protocols and port

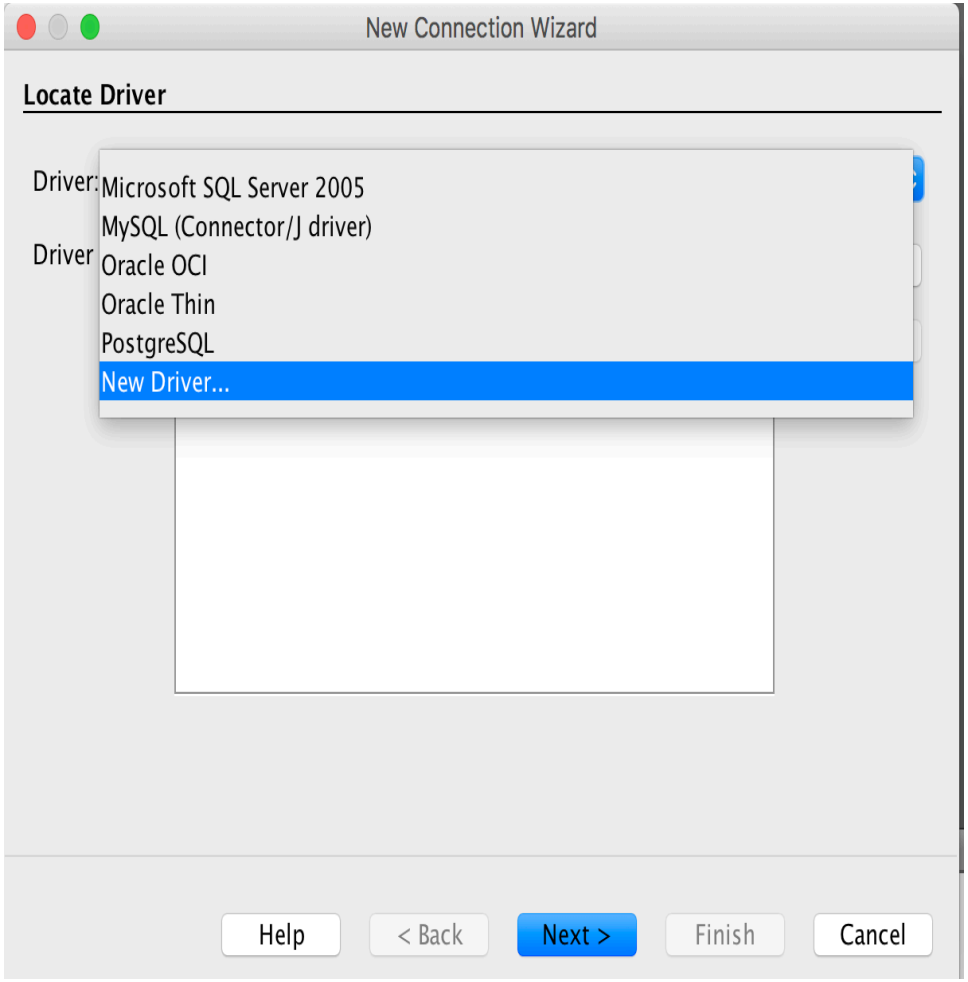
Configure Ports, Protocols for SQL Server…

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

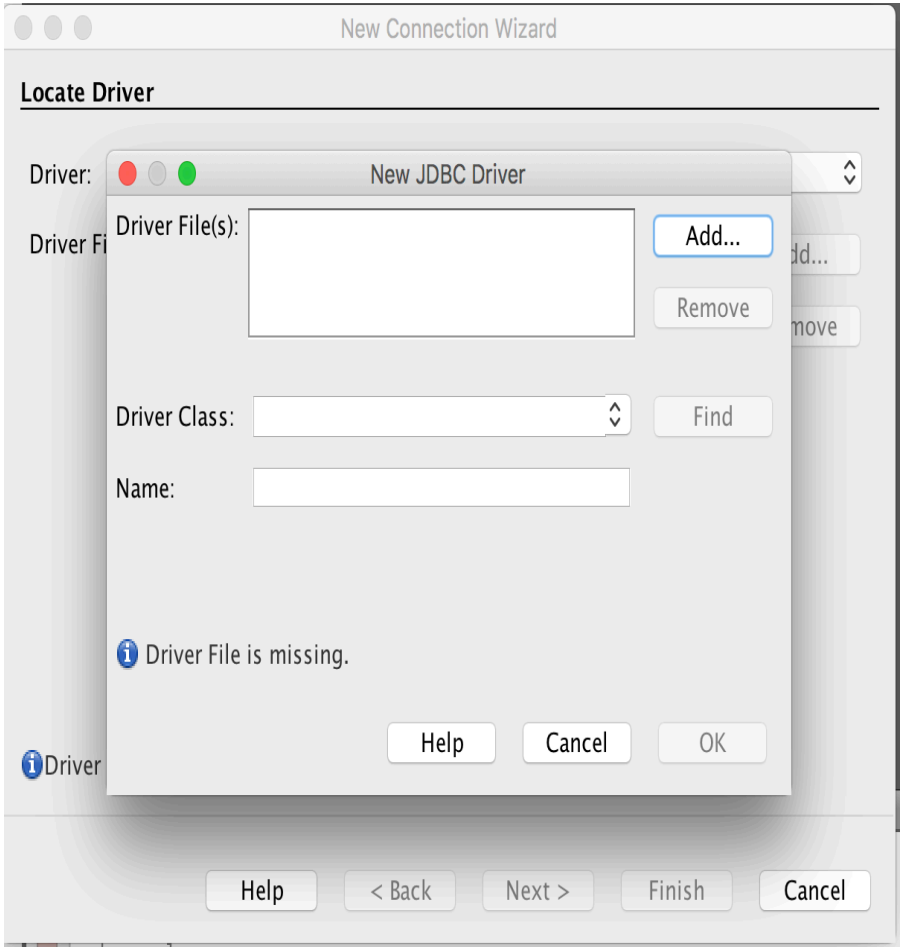


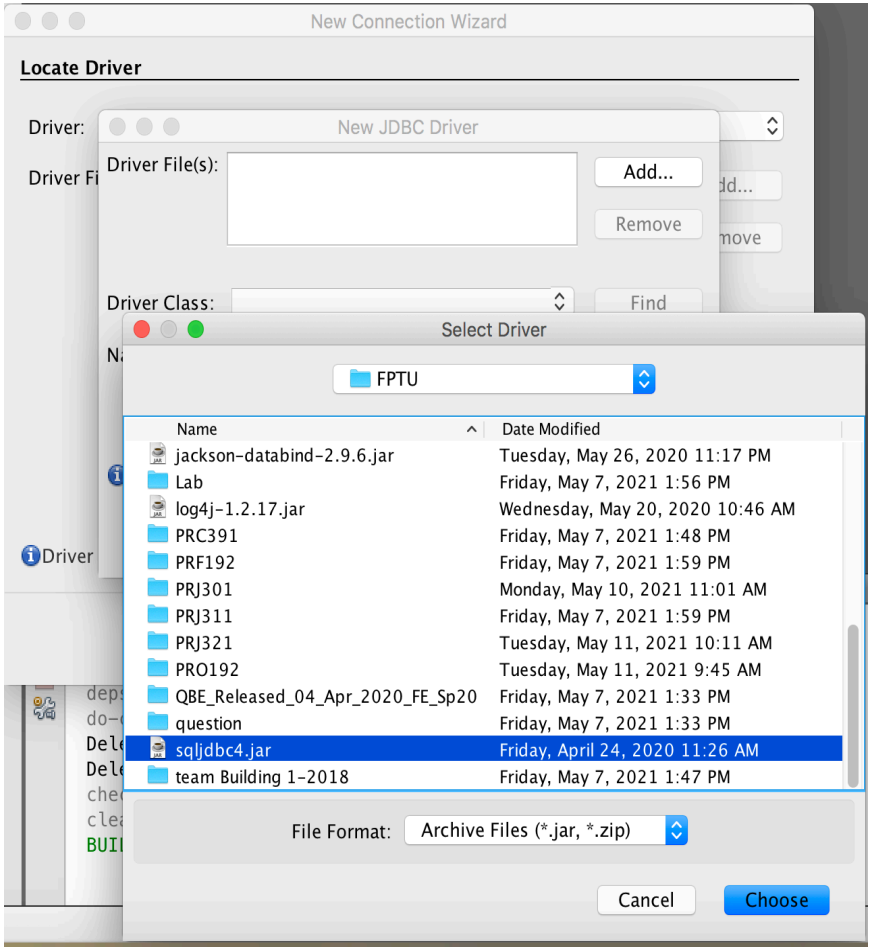
Test connection in Netbeans



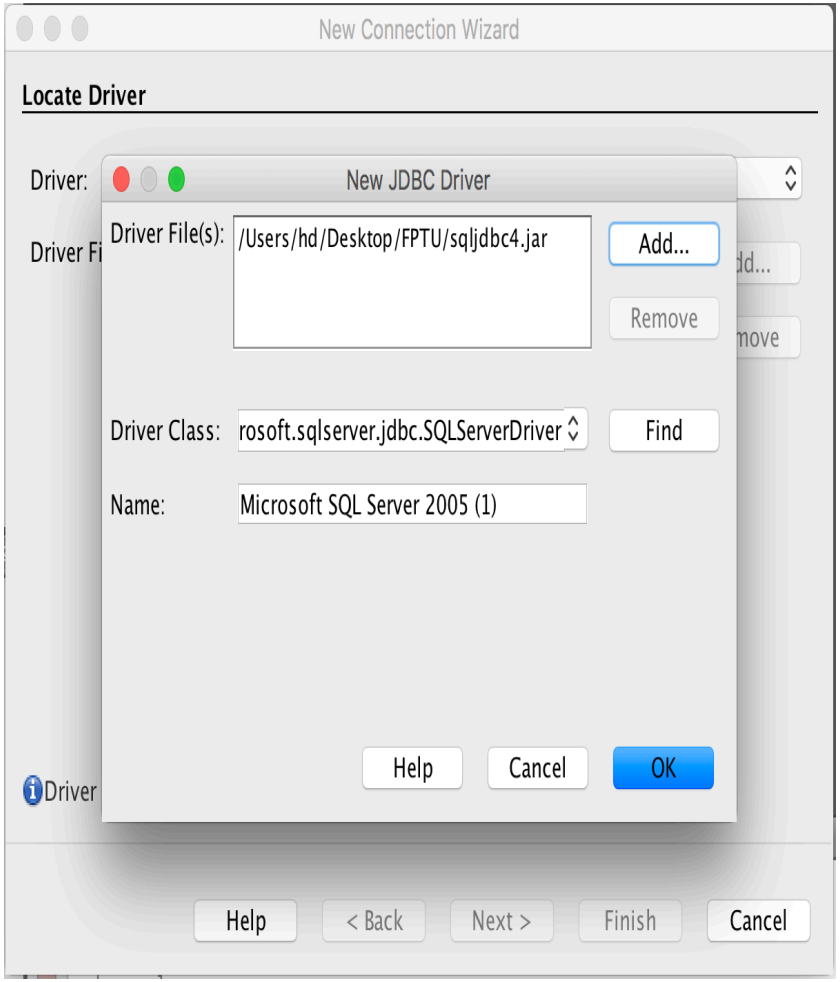


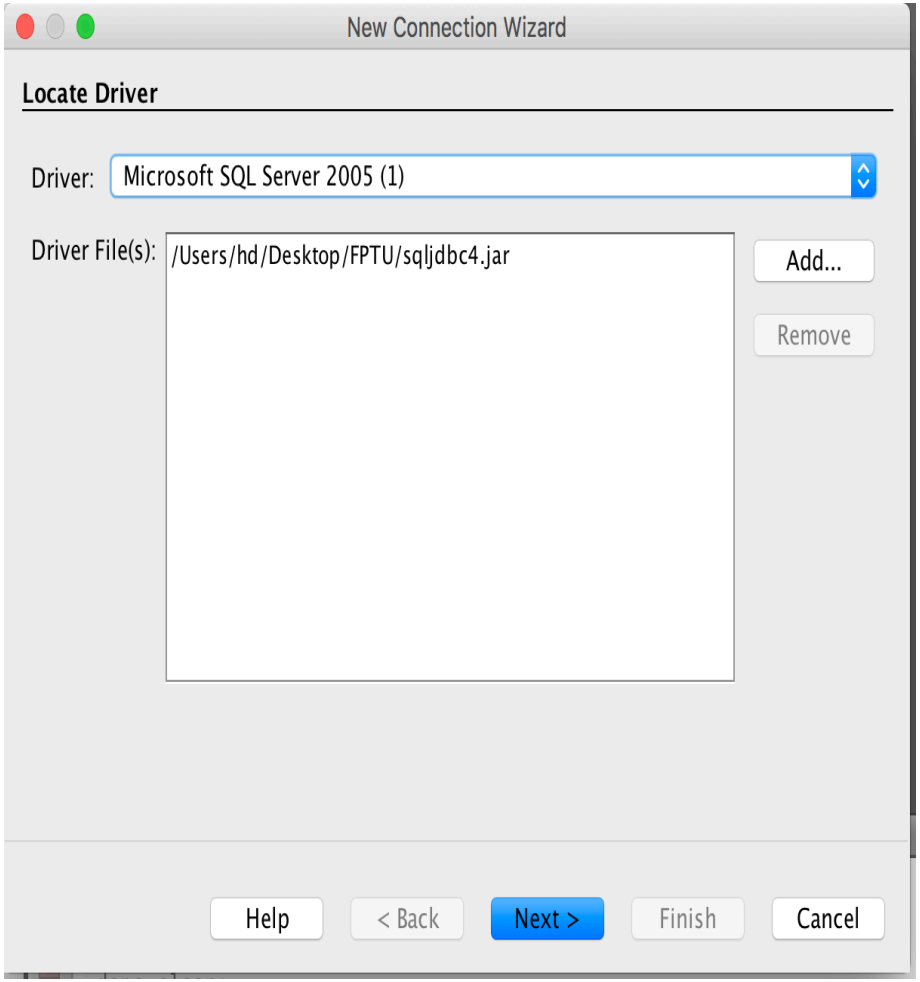
Test connection in Netbeans





Test connection in Netbeans





Test connection in Netbeans

New Connection Wizard

Customize Connection

Driver Name: Microsoft SQL Server 2005 on Microsoft SQL Server 2005 (1)

Host: localhost Port: 1433

Database: UserManagement

Instance Name:

User Name: sa

Password: ••••••••••••••••••

Remember password

Connection Properties | Test Connection

JDBC URL: jdbc:sqlserver://localhost:1433;databaseName=UserManagement

Connection Succeeded.

Help | < Back | Next > | Finish | Cancel

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 1,2 : Make connection

Utils.java

Source History

```java
/**
 *
 * @author HoaDoan
 */
package pe;

import java.io.Serializable;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Utils implements Serializable{
    public static Connection createConnection() throws ClassNotFoundException, SQLException{
        Connection con=null;
        Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
        String url ="jdbc:sqlserver://localhost:1433;databaseName=FU_DB";
        con = DriverManager.getConnection(url,"sa","12345678");
        return con;
    }
}
```

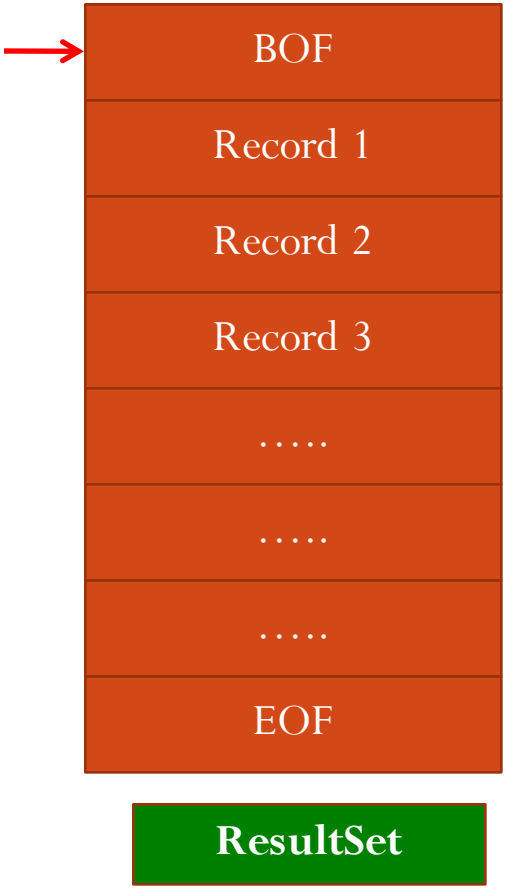
Step 3: Create &Execute a SQL statement


```
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.

Thank You