Database Design & Applications

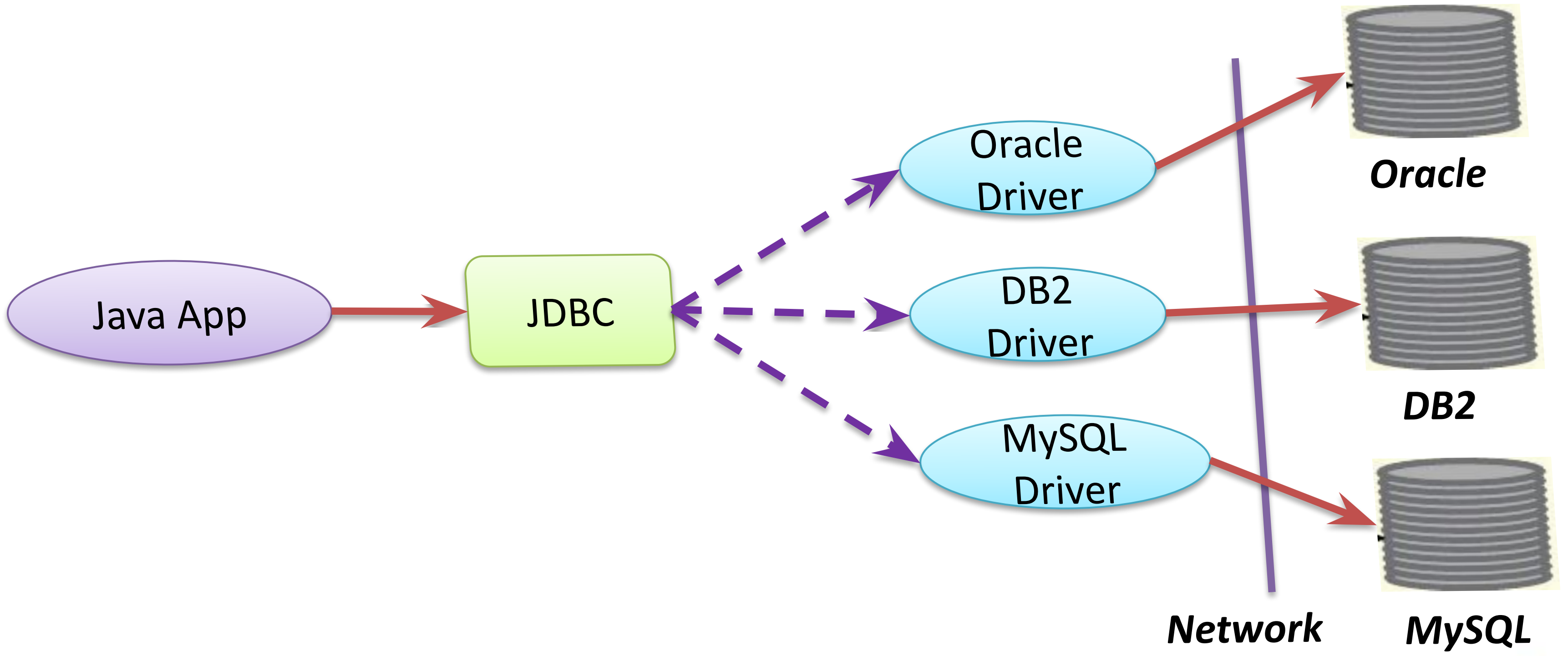
Java Database Connectivity

In this module You will learn

- Java Database Connectivity
- Database Connectivity Architecture
- JDBC APIs
- Database Access Steps
- Example

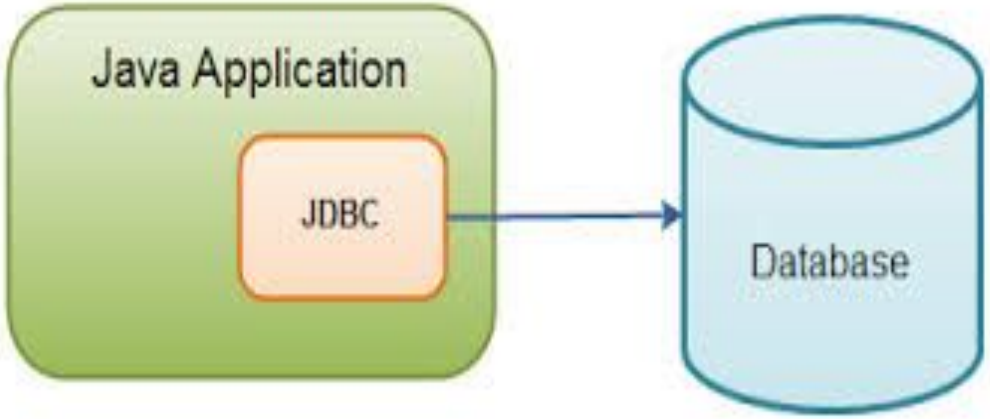
Java Database Connectivity

Using JDBC we can persist data in the database from Java application.



JDBC

- JDBC provides the facility of Connecting to any Database and to manipulate data in the database through Java Programs.
- The JDBC API consists of a set of interfaces and classes written in the Java programming language.
- Interfaces and classes are present in java.sql package
- Java code calls JDBC library which loads the JDBC Driver that talks to a specific database



JDBC API

DriverManager	• This interface manages a list of database drivers.
Driver	• This interface handles the communications with the database server.
Connection	• Interface with the methods for contacting a database.
Statement	• Used to submit the SQL statements to the database.
ResultSet	• These objects hold data retrieved from a database after you execute an SQL query using Statement objects
SQLException	• This class handles any errors that occur in a database application.

Database Access Steps

- Load the Driver
- Define the Connection URL
- Establish the Connection
- Get the statement Object
- Execute the Queries
- Process the Results
- Close the Connection

Download Microsoft JDBC driver for SQL server

- To enable a Java program connects to Microsoft SQL Server database, we need to have a suitable JDBC driver present in the classpath
- JDBC database URL for SQL Server: jdbc:sqlserver://[serverName[\instanceName][:portNumber]][;property=value[;property=value]]
- Where:
  - **serverName:** host name or IP address of the machine on which SQL server is running.
  - **instanceName:** name of the instance to connect to on serverName. The default instance is used if this parameter is not specified.
  - **portNumber:** port number of SQL server, default is 1433. If this parameter is missing, the default port is used.
  - **property=value:** specify one or more additional connection properties.

Download Microsoft JDBC driver for SQL server

- SQL Server has two authentication modes:
  - Windows authentication: using current Windows user account to log on SQL Server. This mode is for the case both the client and the SQL server are running on the same machine. We specify this mode by adding the property **integratedSecurity=true** to the URL.
  - SQL Server authentication: using a SQL Server account to authenticate. We have to specify username and password explicitly for this mode.
- Following are some examples:
  - Connect to default instance of SQL server running on the same machine as the JDBC client, using Windows authentication:
  - **jdbc:sqlserver://localhost;integratedSecurity=true;**
  - Connect to an instance named **sqlexpress** on the host **dbServer**, using SQL Server authentication:
    - **jdbc:sqlserver://dbHost\sqlexpress;user=sa;password=secret**
    - Connect to a named database **testdb** on **localhost** using Windows authentication:
    - **jdbc:sqlserver://localhost:1433;databaseName=testdb;integratedSecurity=true;**

Register JDBC driver for SQL Server and establish connection

The JDBC driver class of SQL Server is **com.microsoft.sqlserver.jdbc.SQLServerDriver**, so to register this driver, use the following statement:

```
DriverManager.registerDriver(new
com.microsoft.sqlserver.jdbc.SQLServerDriver());
```

OR:

```
 Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
```

However, that is not required since JDBC 4.0 (JDK 6.0) because the driver manager can detect and load the driver class automatically as long as a suitable JDBC driver present in the classpath.

Establish Connection

- To make a connection, call the method getConnection() of the DriverManager class.
- Example 1: Connects the user sa with password secret to the instance sqlexpress on localhost:

```
String dbURL = "jdbc:sqlserver://localhost\\sqlexpress;user=sa;password=secret";
Connection conn = DriverManager.getConnection(dbURL);
if (conn != null) {
   System.out.println("Connected");
}
```

Establish Connection

- Example 2: Pass username and password as arguments to the method getConnection():

```
String dbURL = "jdbc:sqlserver://localhost\\sqlexpress";
String user = "sa";
String pass = "secret";
conn = DriverManager.getConnection(dbURL, user, pass);
```

Establish Connection

- Example 3: use a java.util.Properties object to store connection properties:

```
String dbURL = "jdbc:sqlserver://localhost\\sqlexpress";
Properties properties = new Properties();
properties.put("user", "sa");
properties.put("password", "secret");
conn = DriverManager.getConnection(dbURL, properties);
```

Connection: Example

```
package net.codejava.jdbc;

import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.SQLException;

/**
 * This program demonstrates how to establish database
connection to Microsoft SQL Server.  */

public class JdbcSQLServerConnection {

  public static void main(String[] args) {

    Connection conn = null;

    try {

      String dbURL = "jdbc:sqlserver://localhost\\sqlexpress";
      String user = "sa";
      String pass = "secret";
      conn = DriverManager.getConnection(dbURL, user, pass);
```

```
 if (conn != null) {
        DatabaseMetaData dm = (DatabaseMetaData)
conn.getMetaData();
        System.out.println("Driver name: " + dm.getDriverName());
        System.out.println("Driver version: " + dm.getDriverVersion());
        System.out.println("Product name: " +
dm.getDatabaseProductName());
        System.out.println("Product version: " +
dm.getDatabaseProductVersion());
      }} catch (SQLException ex) {
      ex.printStackTrace();
    } finally {
      try {
        if (conn != null && !conn.isClosed()) {
          conn.close();
        }
      } catch (SQLException ex) {
        ex.printStackTrace();
      }
    }
  }
}
```

Creating Statement Object

- Used to create SQL statement
- Object of the statement can be acquired with the help of Connection Object
- Useful when static SQL statements are used at runtime.

```
Statement st=connection.createStatement();
```

Methods in Statement

- int executeUpdate(String SQL)
  - Returns the numbers of rows affected on executing the SQL statement.
  - Used for DML operation – insert, update, delete
- ResultSet executeQuery(String SQL)
  - Returns a ResultSet object.
  - Used for executing select statement
- boolean execute(String SQL)
  - Returns a boolean value of true if a ResultSet object can be retrieved; otherwise, it returns false.
  - Can be used for any DB operation

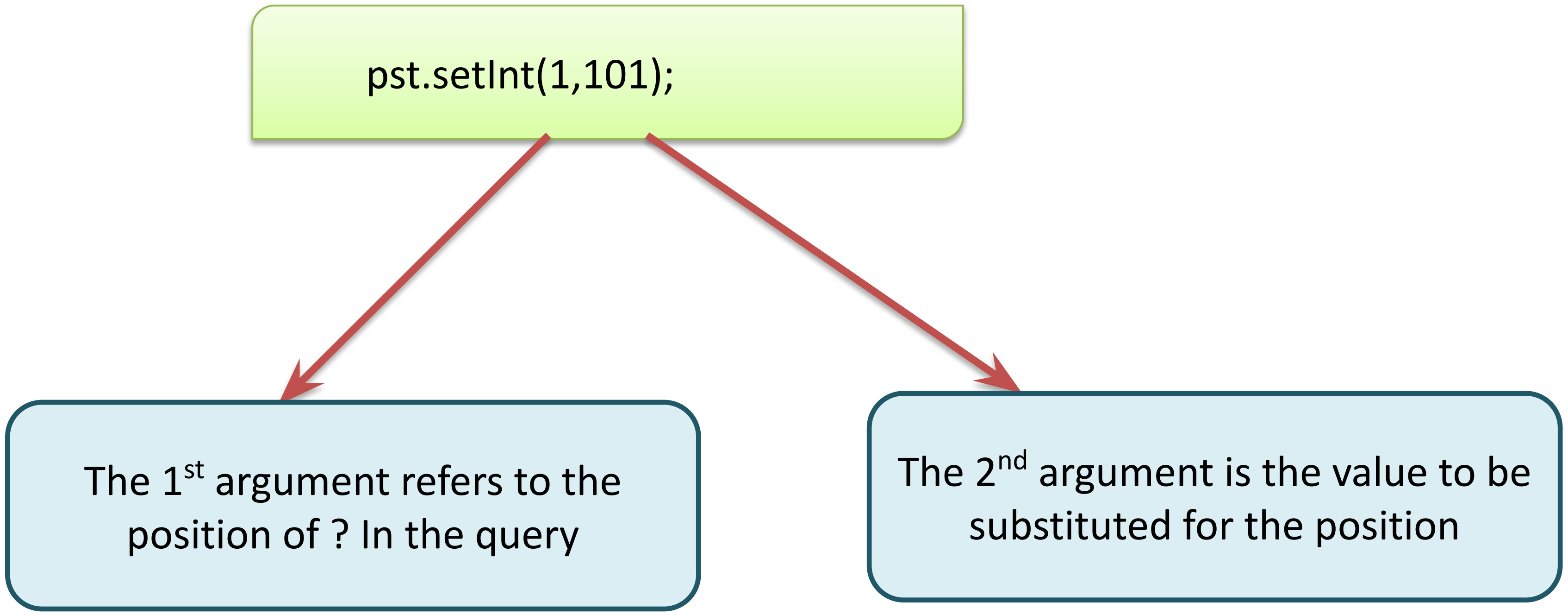
Prepared Statement

- The PreparedStatement interface extends the Statement interface which comes with added functionality.
- The PreparedStatement accepts parameters for which values can be assigned at runtime.
- All parameters in JDBC are represented by the “**?**” symbol, which is known as the parameter marker.

```
PreparedStatement pst = connection.prepareStatement(“select * from      employees where
Employee_id = ?”);
```

Prepared Statement

- The **setXXX()** method binds the user value to the parameters, where **XXX** represents the Java data type of the user value.



```
pst.setInt(1,101);
```

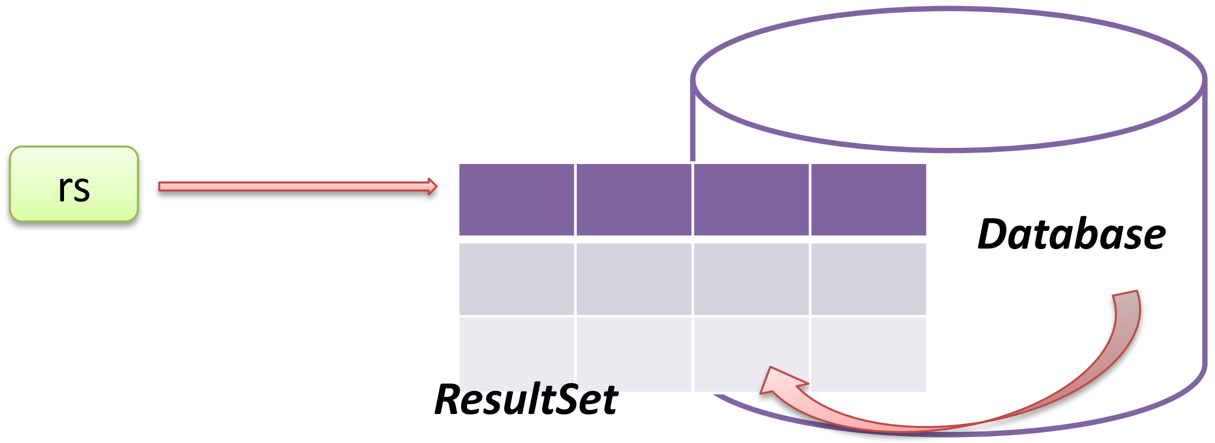
The 1st argument refers to the position of ? In the query

The 2nd argument is the value to be substituted for the position

Working with ResultSet

- Resultset interface represents the result set of a database query
- It is available in java.sql package
- A ResultSet object contains a cursor which points to the current row in the result set

```
Statement st=connection.createStatement();
ResultSet rs= st.executeQuery(“select * from employees”);
```



Methods in ResultSet

- public boolean next()
  - Moves the cursor to the next row.
  - This method returns false if there are no more rows in the result set

```
Statement st=connection.createStatement();
ResultSet rs= st.executeQuery(“select * from employee ”);
while(rs.next())
{
   …..
   …..
}
```

While loop will get executed until all the employee details have been read

Methods to Retrieve Result Data

- public XXX getXXX(String ColumnName)
  - Returns the value from the given column
- public XXX getXXX(int columnIndex)
  - Returns the value from the given column index
  - Column index is the order of columns in the select query
  - Column index starts from 1

Present for all Java data types except for char

Close the connection

- The close() method of Connection interface is used to close the connection.
- By closing connection object, statement and ResultSet will be closed automatically.

```
connection.close();
```

It is ***strongly recommended*** *that an application explicitly commits or rolls back an active transaction prior to calling the close method.*

ResultSet MetaData

- The meta information about ResultSet can be obtained using ResultSetMetaData
- ResultSetMetaData is used to get the column name, total number of columns, column type, table Name

```
Statement st=connection.createStatement();
ResultSet rs= st.executeQuery(“select * from employee”);
ResultSetMetatData  rsm =  rs.getMetaData();
```

ResultSet MetaData - Methods

Method	Description
public int getColumnCount()	it returns the total number of columns in the ResultSet object.
public String getColumnName(int index)	it returns the column name of the specified column index.
public String getColumnTypeName(int index)	it returns the column type name for the specified index.
public String getTableName(int index)	it returns the table name for the specified column index.

```
System.out.println("Total columns: "+rsm.getColumnCount());
System.out.println("Column Name of 1st column: "+rsm.getColumnName(1));
System.out.println("Column Type Name of 1st column: "+rsm.getColumnTypeName(1));
```

Complete Example

```
package com.amazon.atlas22.dao;
import java.sql.Connection;import java.sql.DriverManager;
import java.sql.Statement;
public class DB {
	Connection connection;
	public DB(){
	try {			Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
	System.out.println("1. Loaded the Driver...");
		}
		catch (Exception e)
	{
			System.err.println("Something Went Wrong: "+e);
	}
	}
```

Complete Example

```
public void createConnectionToDB()
{
String connectionURLString =
"jdbc:sqlserver://learnoa-atlas-server.database.windows.net:1433;database = db-atlas22;
user=atlas@learnoa-atlas-server; password=@tl@s123; encrypt=true; trustServerCertificate=false;
hostNameInCertificate=*.database.windows.net; loginTimeout=30;";
try   {
connection = DriverManager.getConnection(connectionURLString); System.out.println("2.
Connection Created with DataBase...");
        }
catch (Exception e)
         {
        System.err.println("Something Went Wrong: "+e);
        }
}
```

Complete Example

```
public void executeSQLQuery()
{
Try      {
          String sql = "INSERT INTO Customers VALUES(3, 'Leo', '+91 99999 14565', 'leo@example.com', '6655    Country Homes','')";
          Statement statement = connection.createStatement();
          statement.executeUpdate(sql);
          System.out.println("3. SQL Statement Executed...");
          }
catch (Exception e)
          {System.err.println("Something Went Wrong: "+e);
          }
}
public void closeConnectionToDB()
 {
try {
          connection.close();
          System.out.println("4. Connection Closed with DataBase...");
}
catch (Exception e)
 {        System.err.println("Something Went Wrong: "+e);// TODO: handle exception
          }         }         }
```

THANK YOU!