JDBC

Introduction

- Provides java API that allows Java programs to access database management systems
- API consists of a set of interfaces and classes which enables java programs to execute SQL statements

JDBC Components

- **JDBC Drivers**: It is a collection of classes which implements interfaces defined in the JDBC API for opening database connections, interacting with database and closing database connections.
- **Connections**: Used to open a database connection. Most common method is getConnection() method of DriverManager class.
- **Statements**: The JDBC statements are used to execute the SQL or PL/SQL queries against the database. JDBC API defines the Statement, CallableStatement, and PreparedStatement types of statements.
- **ResultSets**: A query returns the data in the form of ResultSet. To read the query result date ResultSet provides a cursor that points to the current row in the result set.

Driver Types

- **JDBC-ODBC bridge drive**r: It converts JDBC method calls into ODBC function calls. It is also known as Type 1 driver.
- **Native-API driver**: It uses the client-side libraries of the database. It converts JDBC method calls into native calls of the database API. It is partially written in java. It is also known as Type 2 driver.
- **Network-Protocol driver**: It is a pure java driver which uses a middle-tier to converts JDBC calls directly or indirectly into database specific calls. Multiple types of databases can be accessed at the same time. It is a platform independent driver. It is also known as Type 3 or MiddleWare driver.
- **Thin driver**: Thin driver is a pure java driver which converts JDBC calls directly into the database specific calls. It is a platform independent driver. It is also known as Type 4 or Database-Protocol driver.

Steps to connect database in java using JDBC

- Load the JDBC driver
  - Class.forName("driverClassName");
  - For mysql - Class.forName("com.mysql.cj.jdbc.Driver");
- Create connection
  - Connection connection = DriverManager.getConnection(url, user, password)
- Create statement
  - Statement stmt=conn.createStatement();
- Execute statement
  - ResultSet resultSet = stmt.executeQuery(selectQuery);

Info for Connection with MySql DB

1. Driver class: com.mysql.jdbc.Driver.

2. Connection URL: "jdbc:mysql://hostname:port/dbname","username", "password

3. Username: Username of MySql database, default is root.

4.Password: Password of MySql database.

Example : Connection

```
import java.sql.*;


public class JDBCMysqlTest {
	//JDBC and database properties.
	private static final String DB_DRIVER = "com.mysql.jdbc.Driver";
	private static final String DB_URL = "jdbc:mysql://localhost:3306/dbname";
	private static final String DB_USERNAME = "root";
	private static final String DB_PASSWORD = "root";

	public static void main(String args[]){
		Connection conn = null;
		try{
			Class.forName(DB_DRIVER);
			 conn = DriverManager.getConnection(DB_URL, DB_USERNAME, DB_PASSWORD);
			 if(conn != null){
			     System.out.println("Successfully connected.");
			 }else{
			     System.out.println("Failed to connect.");
			 }
		}catch(Exception e){
			e.printStackTrace();
		}
	}
}
```

Example

```
import java.sql.*; public class SqlQueryDemo{
	public static void main(String arg[]){
		Connection connection = null;
		try {
			Class.forName("com.mysql.cj.jdbc.Driver");
			connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb", "mydbuser",
"mydbuser");
			Statement statement;
			statement = connection.createStatement();
			ResultSet resultSet;
			String myquery="select * from designation";
			resultSet = statement.executeQuery(myquery);
			int code;
			String title;
			while (resultSet.next()) {
				code = resultSet.getInt("code");
				title = resultSet.getString("title").trim();
				System.out.println("Code : " + code  + " Title : " + title);
			}
			resultSet.close();
			statement.close();
			connection.close();
		}catch (Exception exception){
			System.out.println(exception);
		}
	}
}
```

JDBC CallableStatement

- It is used to execute the store procedure and functions.
- CallableStatement interface provides the methods to execute the store procedure and functions.
- We can get a statement object by invoking the prepareCall() method of Connection interface.

JDBC CallableStatement Stored procedure IN parameter example.

```
CREATE OR REPLACE PROCEDURE insertEMPLOYEE(
        e_id IN EMPLOYEE.EMPLOYEE_ID%TYPE,
        e_name IN EMPLOYEE.NAME%TYPE,
        e_salary IN EMPLOYEE.SALARY%TYPE)
IS
BEGIN

  INSERT INTO EMPLOYEE ("EMPLOYEE_ID", "NAME", "SALARY")
  VALUES (e_id, e_name, e_salary);

  COMMIT;

END;
```

```
import java.sql.CallableStatement;
import java.sql.Connection;
import com.w3spoint.util.JDBCUtil;

public class JDBCTest {
	public static void main(String args[]){
		Connection conn = null;
		CallableStatement callableStatement = null;
		String proc = "{call insertEMPLOYEE(?,?,?)}";
		try{
			conn = JDBCUtil.getConnection();

			callableStatement = conn.prepareCall(proc);
			callableStatement.setInt(1, 5);
			callableStatement.setString(2, "Shveta");
			callableStatement.setInt(3, 100000);

			callableStatement.executeUpdate();

			callableStatement.close();
			conn.close();

		    System.out.println("Record inserted successfully.");
		}catch(Exception e){
			e.printStackTrace();
		}
	}
}
```

```
import java.sql.Connection;
import java.sql.DriverManager;

public class JDBCUtil {
	private static final String DB_DRIVER = "oracle.jdbc.driver.OracleDriver";
	private static final String DB_URL = "jdbc:oracle:thin:@localhost:1521:XE";
	private static final String DB_USERNAME = "system";
	private static final String DB_PASSWORD = "oracle";

	public static Connection getConnection(){
		Connection conn = null;
		try{

			Class.forName(DB_DRIVER);

			conn = DriverManager.getConnection(DB_URL, DB_USERNAME,
DB_PASSWORD);

			if(conn != null){
			   System.out.println("Successfully connected.");
			}else{
			   System.out.println("Failed to connect.");
			}
		}catch(Exception e){
			e.printStackTrace();
		}
		return conn;
	}
}
```

JDBC CallableStatement Stored procedure IN parameter example.

```
package com.mkyong.jdbc.callablestatement;
import java.math.BigDecimal;
import java.sql.*;
public class StoreProcedureOutParameter {
    public static void main(String[] args) {
        String createSP = "CREATE OR REPLACE PROCEDURE get_employee_by_id( "
                + " p_id IN EMPLOYEE.ID%TYPE, "
                + " o_name OUT EMPLOYEE.NAME%TYPE, "
                + " o_salary OUT EMPLOYEE.SALARY%TYPE, "
                + " o_date OUT EMPLOYEE.CREATED_DATE%TYPE) "
                + " AS "
                + " BEGIN "
                + "     SELECT NAME, SALARY, CREATED_DATE INTO o_name, o_salary, o_date from
EMPLOYEE WHERE ID = p_id; "
                + " END;";
        String runSP = "{ call get_employee_by_id(?,?,?,?) }";
        try (Connection conn = DriverManager.getConnection(
                "jdbc:oracle:thin:@localhost:1521:orcl", "system", "Password123");
            Statement statement = conn.createStatement();
            CallableStatement callableStatement = conn.prepareCall(runSP)) {
           statement.execute(createSP);
```

Contd..

Contd..

```
callableStatement.setInt(1, 3);
            callableStatement.registerOutParameter(2, java.sql.Types.VARCHAR);
            callableStatement.registerOutParameter(3, Types.DECIMAL);
            callableStatement.registerOutParameter(4, java.sql.Types.DATE);

            callableStatement.executeUpdate();

            String name = callableStatement.getString(2);
            BigDecimal salary = callableStatement.getBigDecimal(3);
            Timestamp createdDate = callableStatement.getTimestamp(4);
            System.out.println("name: " + name);
            System.out.println("salary: " + salary);
            System.out.println("createdDate: " + createdDate);
        } catch (SQLException e) {
            System.err.format("SQL State: %s\n%s", e.getSQLState(), e.getMessage());
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

JDBC CallableStatement Stored procedure batch update

```
CREATE OR REPLACE PROCEDURE insertEMPLOYEE(
        e_id IN EMPLOYEE.EMPLOYEE_ID%TYPE,
        e_name IN EMPLOYEE.NAME%TYPE,
        e_salary IN EMPLOYEE.SALARY%TYPE)
IS
BEGIN

  INSERT INTO EMPLOYEE ("EMPLOYEE_ID", "NAME", "SALARY")
  VALUES (e_id, e_name, e_salary);

  COMMIT;

END;
```

```
import java.sql.CallableStatement;
import java.sql.Connection;
import com.w3spoint.util.JDBCUtil;

public class JDBCTest {
	public static void main(String args[]){
		Connection conn = null;
		CallableStatement callableStatement = null;
		String proc = "{call insertEMPLOYEE(?,?,?)}";
		try{
			conn = JDBCUtil.getConnection();

			callableStatement = conn.prepareCall(proc);

			callableStatement.setInt(1, 7);
			callableStatement.setString(2, "Harish Yadav");
			callableStatement.setInt(3, 50000);
			callableStatement.addBatch();

			callableStatement.setInt(1, 8);
			callableStatement.setString(2, "Abhishek Rathor");
			callableStatement.setInt(3, 50000);
			callableStatement.addBatch();

			callableStatement.executeBatch();

			callableStatement.close();
			conn.close();
		      System.out.println("Records inserted successfully.");
		}catch(Exception e){
			e.printStackTrace();
		}
	}
}
```