JDBC NOTES PART-1

ADVANCED JAVA

With Core Java knowledge we can develop Stand Alone Applications.

The Applications which are running on a Single Machine are called *Stand Alone Applications.*
Eg: Calculator, MS Word
Any Core Java Application
If we want to develop Web Applications then we should go for Advanced Java.

The Applications which are providing Services over the Web are called *Web Applications.*

Eg: gmail.com, facebook.com, jspiderbtm.com

In Java we can develop Web Applications by using the following Technologies...

- **JDBC**
- **Servlets**
- **JSP's**

Where ever Presentation Logic is required i.e. to display something to the End User then we should go for JSP i.e. JSP meant for View Component.

Eg: display login page
display inbox page
display signup page
etc..

Where ever some Processing Logic is required then we should go for Servlet i.e. Servlet meant for Processing Logic/ Business Logic. Servlet will always work internally.

Eg: Verify User
Communicate with Database
Process End User's Data
etc..

From Java Application (Normal Java Class OR Servlet) if we want to communicate with Database then we should go for JDBC.

Eg: To get Employee Information from Database
To get Information from Database

Current Versions Are:

JDBC 4.2 V
Servlets 3.1 V
JSP's 2.3 V

Steps for JDBC Application:

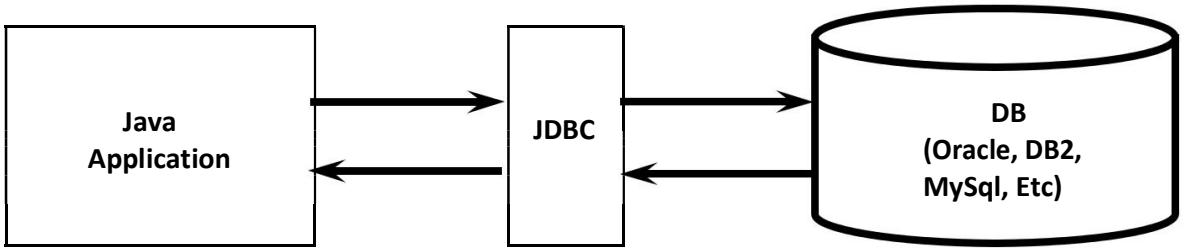
1. Load and Register Driver
2. Establish Connection between Java Application and Database
3. Create Statement Object
4. Send and Execute SQL Query
5. Process Results from ResultSet
6. Close Connection

Demo Program:

```
1)  import java.sql.*;
2)  public class JdbcDemo
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Class.forName("com.mysql.jdbc.Driver");
7)        Connection con =
    DriverManager.getConnection("jdbc:mysql://localhost:3306/adam","root","root");
8)        Statement st = con.createStatement();
9)        ResultSet rs= st.executeQuery("select * from employees");
10)       while(rs.next())
11)       {
12)          System.out.println(rs.getInt(1)+".."+rs.getString(2)+".."+rs.getDouble(3)+"..."+rs.get
    String(4));
13)       }
14)       con.close();
15)    }
16) }
```

JDBC

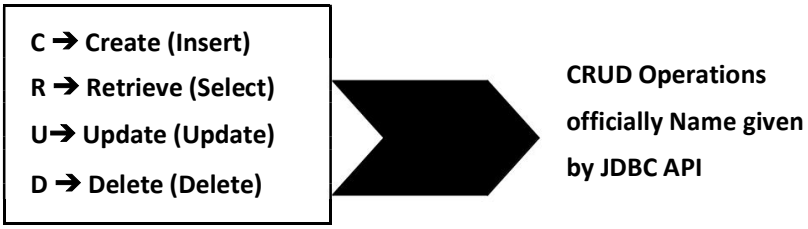
- **JDBC is a Technology, which can be used to communicate with Database from Java Application.**



- **JDBC is the Part of Java Standard Edition (J2SE|JSE)**
- **JDBC is a Specification defined by Java Vendor (Sun Micro Systems) and implemented by Database Vendors.**
- **Database Vendor provided Implementation is called "Driver Software".**

JDBC Features:

1) **JDBC API is Standard API. We can communicate with any Database without rewriting our Application i.e. it is Database Independent API.**
2) **JDBC Drivers are developed in Java and hence JDBC Concept is applicable for any Platform. i.e. JDBC Is Platform Independent Technology.**
3) **By using JDBC API, we can perform basic CRUD Operations very easily.**



We can also perform Complex Operations (like Inner Joins, Outer Joins, calling Stored Procedures etc) very easily by using JDBC API.

4) **JDBC API supported by Large Number of Vendors and they developed multiple Products based on JDBC API.**

List of supported Vendors we can check in the link
http://www.oracle.com/technetwork/java/index-136695.html

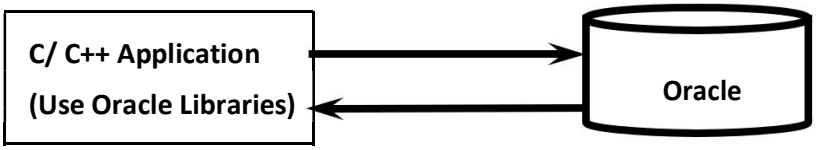
JDBC Versions:

- **JDBC 3.0 is Part J2SE 1.4**
- **No Update in Java SE 5.0**
- **JDBC 4.0 is Part Java SE 6.0**

- **JDBC 4.1 is Part Java SE 7.0**
- **JDBC 4.2 is Part Java SE 8.0**

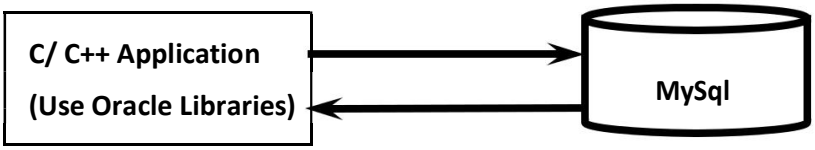
Evolution of JDBC:

- **If we want to communicate with Database by using C OR C++, compulsory we have to use database specific Libraries in our Application directly.**



- **In the above Diagram C OR C++ Application uses Oracle specific Libraries directly.**

- **The Problem in this Approach is, if we want to migrate Database to another Database then we have to rewrite Total Application once again by using new Database specific Libraries.**

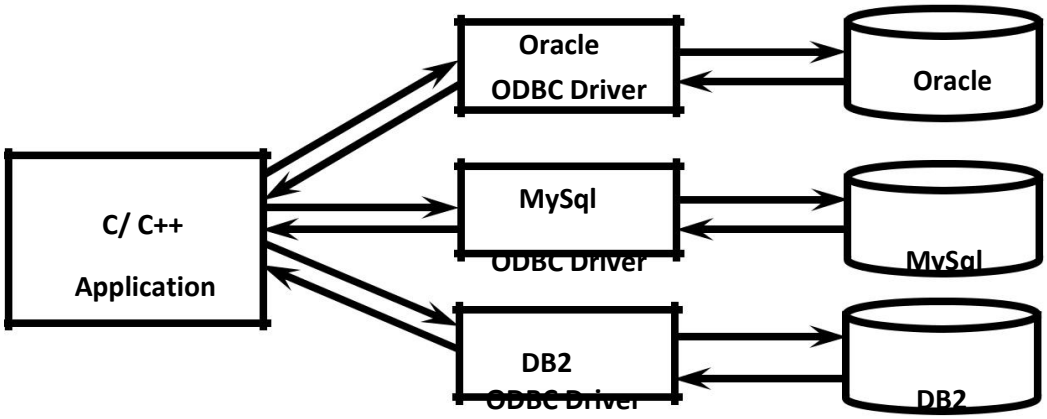


- **The Application will become Database Dependent and creates Maintenance Problems.**

- **To overcome this Problem, Microsoft People introduced "ODBC" Concept in 1992. It is Database Independent API.**

- **With ODBC API, Application can communicate with any Database just by selecting corresponding ODBC Driver.**

- **We are not required to use any Database specific Libraries in our Application. Hence our Application will become Database Independent.**



Limitations of ODBC:

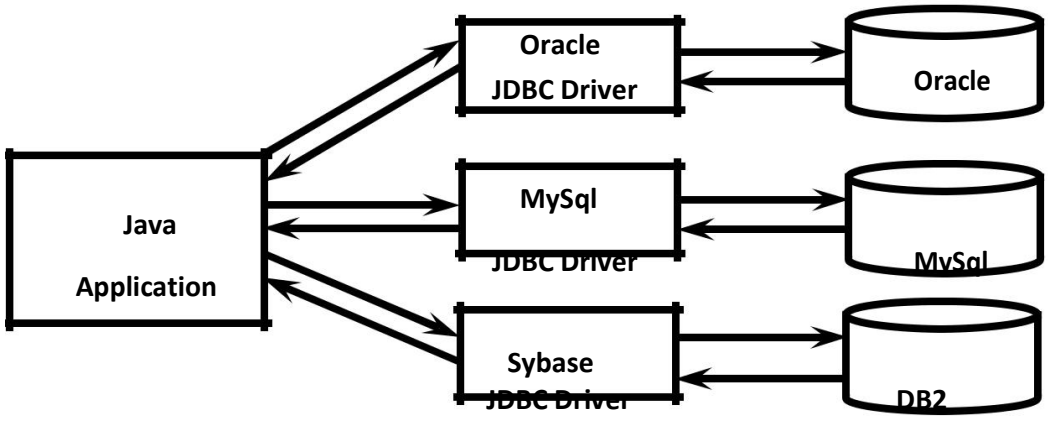
1) **ODBC Concept will work only for Windows Machines. It is Platform Dependent Technology.**
2) **ODBC Drivers are implemented in C Language. If we use ODBC for Java Applications, then Performance will be down because of internal conversions from Java to C and C to Java.**

- **Because of above Reasons, ODBC Concept is not suitable for Java Applications.**

- **For Java Applications, SUN People introduced JDBC Concept.**

- **JDBC Concept Applicable for any Platform. It is Platform Independent Technology.**

- **JDBC Drivers are implemented in Java. If we use JDBC for Java Applications, then internal Conversions are not required and hence there is no Effect on Performance.**

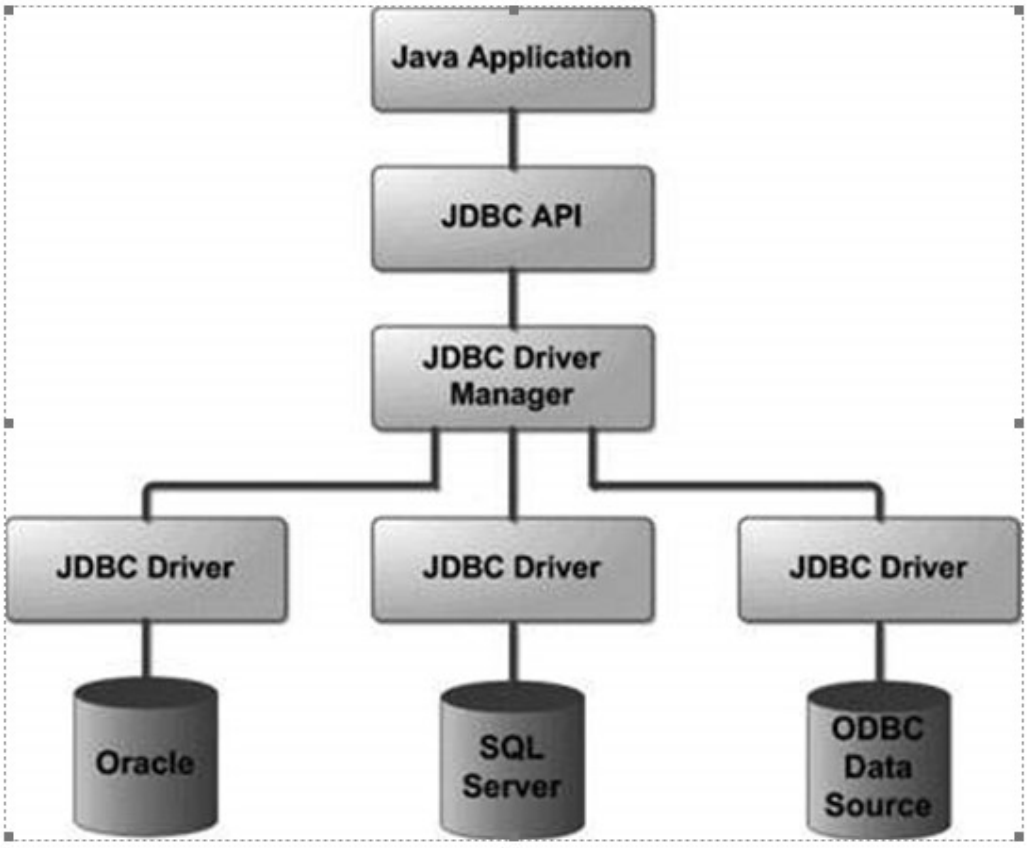


***<u>Note:</u>**

1) **ODBC Concept is applicable for any Database and for any Language, but only for Windows Platform.**
2) **JDBC Concept is Applicable for any Platform and for any Database, but only for Java Language.**

<u>Differences Between JDBC and ODBC</u>

ODBC	JDBC
1) ODBC Stands for Open Database Connectivity	1) JDBC Stands for Java Database Connectivity
2) Introduced by Microsoft.	2) Introduced by Sun Micro Systems.
3) We can Use ODBC for any Languages like C, C++, Java, Etc.	3) We can Use JDBC only for Java Language.
4) We can use ODBC only for Windows Platforms.	4) We can use JDBC for any Platform.
5) Mostly ODBC Drivers are developed in Native Languages like C OR C++.	5) Mostly JDBC Drivers are developed in Java.
6) For Java Applications, it is not recommended to use ODBC because Performance will be Down due to Internal Conversions and Application will become Platform Dependent.	6) For Java Applications, it is highly recommended to use JDBC because there is no Performance Problems and Platform Dependency Problems.



JDBC ARCHITECTURE

- **JDBC API provides DriverManager to our Java Application.**

- **Java Application can communicate with any Database with the help of DriverManager and Database Specific Driver.**

DriverManager:

- **It is the Key Component in JDBC Architecture.**
- **DriverManager is a Java Class present in *java.sql* Package.**
- **It is responsible to manage all Database Drivers available in our System.**

- **DriverManager is responsible to register and unregister Database Drivers. DriverManager.registerDriver(Driver);**
  DriverManager.unregisterDriver(Driver);

- **DriverManager is responsible to establish Connection to the Database with the help of Driver Software.**
  Connection con = DriverManager.getConnection (jdbcurl, username, pwd);

Database Driver:

- **It is the very Important Component of JDBC Architecture.**
- **Without Driver Software we cannot touch Database.**
- **It acts as Bridge between Java Application and Database.**
- **It is responsible to convert Java Calls into Database specific Calls and Database specific Calls into Java Calls.**

Note:

1) **Java Application is Database Independent but Driver Software is Database Dependent. Because of Driver Software only Java Application will become Database Independent.**
2) **Java Application is Platform Independent but JVM is Platform Dependent. Because of JVM only Java Application will become Platform Independent.**

JDBC API

- **JDBC API provides several Classes and Interfaces.**
- **Programmer can use these Classes and Interfaces to communicate with the Database.**
- **Driver Software Vendor can use JDBC API while developing Driver Software.**
- **JDBC API defines 2 Packages**

1) java.sql Package:

It contains basic Classes and Interfaces which can be used for Database Communication.

Interfaces	Classes
Driver	1) DriverManager
Connection	Date
Statement	Time
PreparedStatement	TimeStamp
CallableStatement	Types
ResultSet	
ResultSetMetaData	
DataBaseMetaData	

2) javax.sql Package:

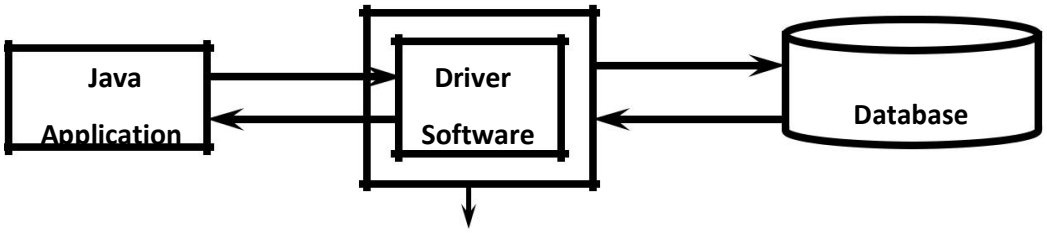
It defines more advanced Classes and Interfaces which can be used for Database Communication.

There are multiple Sub Packages are also available

- **javax.sql.rowset;**
- **javax.sql.rowset.serial;**
- **javax.sql.rowset.spi;**

Interfaces	Classes
1) DataSource	1) ConnectionEvent
2) RowSet	2) RowSetEvent
3) RowSetListener	3) StatementEvent
4) ConnectionEventListener	::::::::::::::::::
5) StatementEventListener	

- **Programmers are not responsible to provide Implementation for JDBC API Interfaces.**
- **Most of the times Database Vendor is responsible to provide Implementation as the Part of Driver Software.**
- **Every Driver Software is a Collection of Classes implementing various Interfaces of JDBC API, which can be used to communicate with a particular Database.**



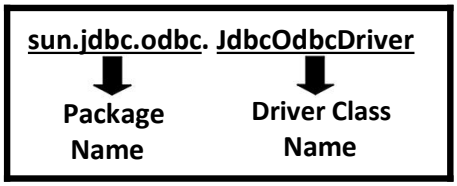
Implementation Classes Of JDBC API

- **For Example, Driver Software of Oracle means Collection of Implementation Classes of JDBC API, which can be used to communicate with Oracle Database.**

- **Every Driver Software is identified with some Special Class which is nothing but Driver Class. It is the Implementation Class of Driver Interface present in. *java.sql* Package.**

- **As the Part of JDK, SUN People provided one Built-In Driver Software which implements JDBC API, which is nothing but Type-1 Driver (JDBC-ODBC Bridge Driver).**

- **The corresponding Driver Class Name is:**



Difference between Driver Interface, Driver Class and Driver Software:

1) Driver Interface:

This Interface present in *java.sql* Package.
This Interface acts as Requirement Specification to implement Driver Class.

2) Driver Class:

It is the Implementation Class of Driver Interface
Eg: com.mysql.jdbc.Driver

3) Driver Software:

- **It is the Collection of Implementation Classes of various Interfaces present in JDBC API.**
- **It acts as Bridge between Java Application and Database.**
- **It is responsible to convert Java Calls into Database specific Calls and Database specific Calls into Java Calls.**

- **Usually Driver Softwares are available in the Form of jar File.**
  Eg:
  - **ojdbc14.jar**
  - **mysql-connector.jar**

- **Driver Softwares can be provided by the following Vendors**
  - **Java Vendor (Until 1.7 Version Only)**
  - **Database Vendor**
  - **Third Party Vendor**

Steps for developing JDBC Application

1. Load and register Driver Class
2. Establish Connection between Java Application and Database
3. Create Statement Object
4. Send and execute SQL Query
5. Process Result from ResultSet
6. Close Connection

Step 1: Load and Register Driver Class

JDBC API is a Set of Interfaces defined by Java Vendor.

Database Vendor is responsible to provide Implementation. This Group of Implementation Classes is nothing but "Driver Software".

We have to make this Driver Software available to our Java Program. For this we have to place corresponding Jar File in the Class Path.

Note:

Every Driver Software is identified by some special Class, which is nothing but Driver Class.

For Type-4 Driver, the corresponding mysql Driver Class Name is

```
com.mysql.jdbc.Driver
```

We can load any Java Class by using *Class.forName()* Method. Hence by using the same Method we can load Driver Class.

```
Class.forName("com.mysql.jdbc.Driver”);
```

Whenever we are loading Driver Class automatically Static Block present in that Driver Class will be executed.

```
1)  class Driver
2)  {
3)     static
4)     {
5)        com.mysql.jdbc.Driver driver= new com.mysql.jdbc.Driver ();
6)        DriverManager.registerDriver(driver);
7)     }
8)  }
```

Because of this Static Block, whenever we are loading automatically registering with *DriverManager* will be happened. Hence we are not required to perform this activity explicitly.

If we want to register explicitly without using *Class.forName()* then we can do as follows by using *registerDriver()* Method of *DriverManager* Class.

```
com.mysql.jdbc.Driver driver= new com.mysql.jdbc.Driver ();
DriverManager.registerDriver(driver);
```

Note: From JDBC 4.0 V (Java 1.6 V) onwards Driver Class will be loaded automatically from Class Path and we are not required to perform this step explicitly.

Step-2: Establish Connection between Java Application and Database

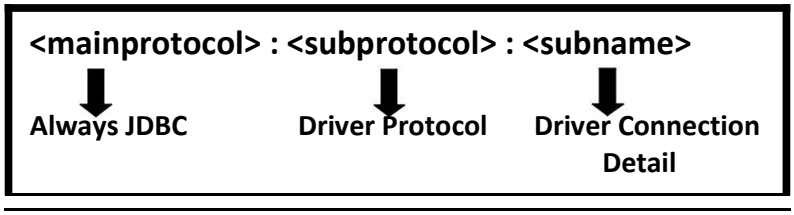
Once we loaded and registered Driver, by using that we can establish Connection to the Database. For this *DriverManager* Class contains *getConnection()* Method.

```
public static Connection getConnection(String jdbcurl, String username, String pwd) throws
SQLException
```

Eg: Connection con= DriverManager.getConnection(jdbcurl,username,pwd);

"Jdbcurl" represents URL of the Database.
***username* and *pwd* are Credentials to connect to the Database.**

JDBC URL Syntax:



For Type-4 Driver, JDBC URL is: **jdbc:mysql://localhost:3306**

Eg: **Connection con= DriverManager.getConnection("jdbc:mysql://localhost:3306","root","root");**

Note:

DriverManager will use Driver Class internally to connect with Database.
DriverManager Class *getConnection()* Method internally calls Driver Class *connect()* Method.

Write A Java Program To Establish Connection To The Oracle Database By Using Type-4 Driver?

```
1)  import java.sql.*;
2)  public class DbConnectDemo1
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Class.forName("com.mysql.jdbc.Driver");
    Connection con=
    DriverManager.getConnection("jdbc:mysql://localhost:3306","root","root");
7)        if(con != null)
8)        {
9)           System.out.println("Connection established Successfully");
10)       }
11)       else
12)       {
13)          System.out.println("Connection not established");
14)       }
15)    }
16) }
```

Note:

To Compile and Run above Program we are required to Place/Set any Jar File in the Class Path/Buildpath

***Q. Connection is an interface, then how we can get Connection Object?

We are not getting Connection Object and we are getting its Implementation Class Object.

This Implementation Class is available as the Part of Driver Software. Driver Software Vendor is responsible to provide Implementation Class.

We can print corresponding Class Name as follows SOP(con.getClass().getName());
o/p: driver Implementation Class Connection

Q.What Is The Advantage Of Using Interface Names In Our Application Instead Of Using Implementation Class Names?

Interface Reference can be used to hold implemented Class Object. This Property is called Polymorphism.

Connection ➔ sun.jdbc.odbc.JdbcOdbcConnection ➔ Type-1
Connection ➔ orcale.jdbc.OracleT4Connection ➔ Type-2

In JDBC Programs, Interface Names are fixed and these are provided by JDBC API. But Implementation Classes are provided by Driver Software Vendor and these Names are varied from Vendor to Vendor.

If we Hard Code Vendor provided Class Names in our Program then the Program will become Driver Software Dependent and won't work for other Drivers.

If we want to change Driver Software then Total Program has to rewrite once again, which is difficult. Hence it is always recommended to use JDBC API provided Interface Names in our Application.

Step-3: Creation of Statement Object

Once we established Connection between *Java Application* and *Database*, we have to prepare SQL Query and we have to send that Query to the Database. Database Engine will execute that Query and send Result to Java Application.

To send SQL Query to the Database and to bring Results from Database to Java Application some Vehicle must be required, which is nothing but Statement Object.

We can create Statement Object by using *createStatement()* Method of Connection Interface.

```
public Statement createStatement();
```

Eg: Statement st = con.createStatement();

Step-4: Prepare, Send and Execute SQL Query

According to Database Specification, all SQL Commands are divided into following Types...

1. **DDL (Data Definition Language) Commands:**
   Eg: Create Table, Alter Table, Drop Table Etc

2. **DML (Data Manipulation Language)**
   Commands: Eg: Insert, Delete, Update

3. **DQL (Data Query Language)**
   Commands: Eg: Select

According to Java Developer Point of View, all SQL Operations are divided into 2 Types...

1. **Select Operations (DQL)**
2. **Non-Select Operations (DML, DDL Etc)**

Select Operations and Non-Select Operations

Select Operations:

Whenever we are performing Select Operation then we will get a Group of Records as Result.

Eg: `select * from employee;`

Non-Select Operations:

Whenever we are performing Non-Select Operation then we will get Numeric Value that represents the Number of Rows affected.

Eg: `update employee set  name=adam  where no=1;`

Once we create Statement Object, we can call the following Methods on that Object to execute our Queries.

1. executeQuery()
2. executeUpdate()
3. execute()

1) executeQuery() Method:

We can use this Method for Select Operations.

Because of this Method Execution, we will get a Group of Records, which are represented by ResultSet Object.

Hence the Return Type of this Method is ResultSet.

```
public ResultSet executeQuery(String sqlQuery) throws SQLException
```

Eg: `ResultSet rs = st.executeQuery("select * from employee");`

2) executeUpdate() Method:

We can use this Method for Non-Select Operations (Insert|Delete|Update)

Because of this Method Execution, we won't get a Group of Records and we will get a Numeric Value represents the Number of Rows effected. Hence Return Type of this Method is int

public int executeUpdate(String sqlQuery)throws SQLException

Eg: int rowCount = st.executeUpdate("delete from employees where id=101");
SOP("The number of employees deleted:"+rowCount);

3) execute() method:

We can use this Method for both Select and Non-Select Operations.

If we don't know the Type of Query at the beginning and it is available dynamically at runtime then we should use this execute() Method.

public boolean execute(String sqlQuery)throws SQLException

Eg:

```
1)  boolean b = st.execute("dynamically provided query");
2)
3)  if(b==true)//select query
4)  {
5)    ResultSet rs=st.getResultSet();
6)    //use rs to get data
7)  }
8)  else// non-select query
9)  {
10)   int rowCount=st.getUpdateCount();
11)   SOP("The number of rows effected:"+rowCount);
12) }
```

executeQuery() Vs executeUpdate() Vs execute():

1. If we know the Type of Query at the beginning and it is always Select Query then we should use "*executeQuery()* Method".

2. If we know the Type of Query at the beginning and it is always Non-Select Query then we should use *executeUpdate()* Method.

3. If we don't know the Type of SQL Query at the beginning and it is available dynamically at Runtime (May be from *Properties File* OR From *Command Prompt* Etc) then we should go for execute() Method.

Note:

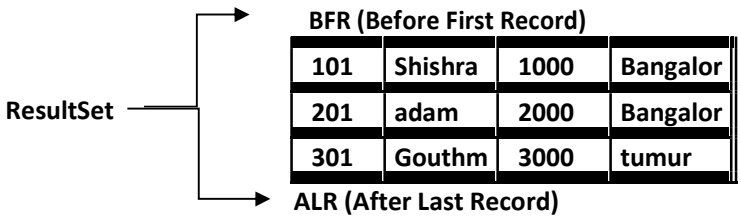
Based on our Requirement we have to use corresponding appropriate Method.

- st.executeQuery();
- st.executeUpdate();
  st.execute();
- st.getResultSet();
- st.getUpdateCount();

Step-5: Process Result from ResultSet

After executing Select Query, Database Engine will send Result back to Java Application. This Result is available in the form of ResultSet.

i.e. ResultSet holds Result of executeQuery() Method, which contains a Group of Records.
By using ResultSet we can get Results.



ResultSet is a Cursor always locating Before First Record (BFR).
To check whether the next Record is available OR not, we have to use *rs.next()* Method.

```
public boolean next()
```

This Method Returns True if the next Record is available, otherwise returns False.

```
1)  while(rs.next())
2)  {
3)     read data from that record
4)  }
```

If next Record is available then we can get Data from that Record by using the following Getter Methods.

1. getXxx(String columnName)
2. getXxx(int columnIndex)

Like getInt(), getDouble(), getString() etc..

Note:
In JDBC, Index is always one based but not Zero based i.e. Index of First Column is 1 but not 0.

```
1)  while(rs.next())
2)  {
3)    SOP(rs.getInt("ENO")+".."+rs.getString("ENAME")+".."+rs.getDouble("ESAL")+".."+rs.getS
    tring("EADDR"));
4)          OR
5)    SOP(rs.getInt(1)+".."+rs.getString(1)+".."+rs.getDouble(3)+".."+rs.getString(4));
6)   }
```

Note:

Readability wise it is recommended to use Column Names, but Performance wise it is recommended to use Column Index. (Because comparing Numbers is very easy than comparing String Values)

Hence if we are handling very large Number of Records then it is highly recommended to use Index.

If we know Column Name then we can find corresponding Index as follows...

```
int columnIndex=rs.findColumn(String columnName);
```

Conclusions:

1. ResultSet follows "Iterator" Design Pattern.
2. ResultSet Object is always associated with Statement Object.
3. Per Statement only one ResultSet is possible at a time. if we are trying to open another ResultSet then automatically first ResultSet will be closed.

Eg:

```
Statement st = con.createStatement();
RS rs1 = st.executeQuery("select * from Depart");
RS rs2 = st.executeQuery("select * from employees");
```

In the above Example Rs1 will be closed automatically whenever we are trying to open Rs2.

Step 6: Close the Connection

After completing Database Operations it is highly recommended to close the Resources whatever we opened in reverse order of opening.

1. rs.close();

It closes the ResultSet and won't allow further processing of ResultSet

2. st.close();

It closes the Statement and won't allow sending further Queries to the Database.

3. con.close();

It closes the Connection and won't allow for further Communication with the Database.

Conclusions:

- **Per Statement only one ResultSet is possible at a time.**
- **Per Connection multiple Statement Objects are possible.**
- **Whenever we are closing Statement Object then automatically the corresponding ResultSet will be closed.**
- **Similarly, whenever we are closing Connection Object automatically corresponding Statement Objects will be closed.**

- **Hence we required to use only *con.close();***

1.7 Version: try With Resources

Usually we will close the Resources inside finally Block.

```
1)  try
2)  {
3)     Open Database Connection
4)  }
5)  catch(X e)
6)  {
7)  }
8)  finally
9)  {
10)    Close That Database Connection
11) }
```

But in Java 1.7 Version try with Resources Concept introduced.

The Advantage of this Concept is, whatever Resources we opened as the Part of *try* Block, will be closed automatically once Control reaches End of *try* Block either Normally OR Abnormally. We are not required to close explicitly.

Hence until 1.6 Version *finally* Block is just like Hero but from 1.7 Version onwards *finally* Block became Zero.

```
try (Resource)
{
}
```

Eg:

```
try (Connection con = DM.getConnection(-,-,-))
{
        Use con based on our Requirement
        Once Control reaches End of try Block, automatically con will be
        closed, we are not required to close explicitly
}
```

Formatting SQL Queries With Dynamic Input

```
String sqlQuery="insert into employees values(100,adam,1000,Ban)"; If
```

Data is available in the following Variables eno, ename, esal, eaddr

```
String sqlQuery="insert into employees values("+eno+",'"+ename+"',"+esal+",'"+eaddr+"')";
```

It is highly recommended to use String Class *format()* Method while writing SQL Queries with Dynamic Input.

```
String sqlQuery = String.format("insert into employees values
(%d,'%s',%f,'%s')", eno,ename,esal,eaddr);
```

JDBC Information:

In general, we can use Type-4 Driver to communicates with MySQL Database which is provided by MySQL Vendor, and its Name is connector/J

Jar File: Driver Software is available in the following Jar File.

`mysql-connector-java-5.1.41-bin.jar`

We have to download separately from MySql Web Site.

jdbc url: `jdbc:mysql://localhost:3306/adamdb`
`jdbc:mysql:///adamdb`

If MySQL is available in Local System then we can specify JDBC URL as above.

Driver Class Name: `com.mysql.jdbc.Driver`
User Name: `root`
pwd: `root`

We required to Set Class Path of MySql Driver Jar File

Variable Name: `CLASSPATH`
Variable Value: `D:\mysql-connector-java-bin.jar;.;`

Program to Demonstrate JDBC with MySql Database

```
1)  import java.sql.*;
2)  public class JdbcMySQLDemo
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Class.forName("com.mysql.jdbc.Driver");
```

```
7)        Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/adam","root","root");
8)        Statement st = con.createStatement();
9)        ResultSet rs =st.executeQuery("select * from employees");
10)       while(rs.next())
11)       {
12)          System.out.println(rs.getInt(1)+".."+rs.getString(2)+".."+rs.getDouble(3)+".."+rs.get
String(4));
13)       }
14)       con.close();
15)    }
16) }
```