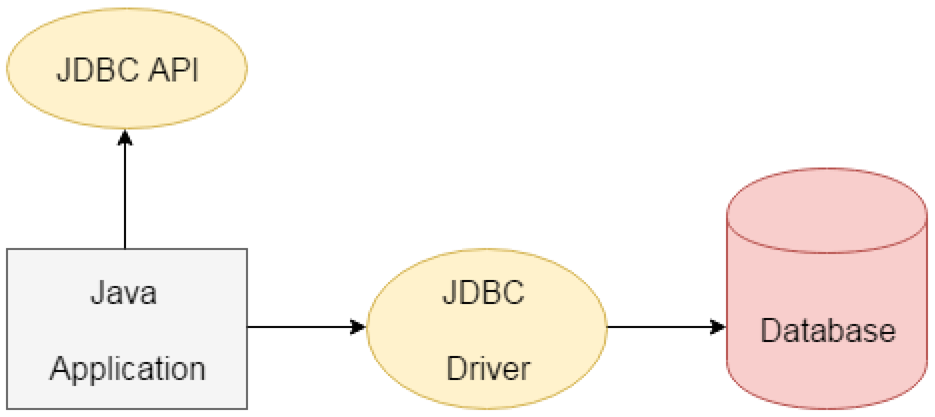
Java JDBC Tutorial

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, and
- Thin Driver

We have discussed the above four drivers in the next chapter.

We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft.



The current version of JDBC is 4.3. It is the stable release since 21st September, 2017. It is based on the X/Open SQL Call Level Interface. The **java.sql** package contains classes and interfaces for JDBC API. A list of popular *interfaces* of JDBC API are given below:

- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface
- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

A list of popular *classes* of JDBC API are given below:

- DriverManager class
- Blob class
- Clob class
- Types class

Why Should We Use JDBC

Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

We can use JDBC API to handle database using Java program and can perform the following activities:

1. Connect to the database
2. Execute queries and update statements to the database
3. Retrieve the result received from the database.

Java Database Connectivity with 5 Steps

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

- Register the Driver class
- Create connection object
- Create statement object
- Execute queries
- Close connection

1) Register the driver class

The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of forName() method

```
public static void forName(String className)throws ClassNotFoundException
```

Example to register the OracleDriver class

Here, Java program is loading oracle / MySql driver to esteblish database connection.

```
      Class.forName("oracle.jdbc.driver.OracleDriver"); //oracle

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

2) Create the connection object

The **getConnection()** method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

```
public static Connection getConnection(String url)throws SQLException

public static Connection getConnection(String url,String name,String p
assword) throws SQLException
```

Example to establish connection with the Oracle / MySql database

```
      Connection con=DriverManager.getConnection(  "jdbc:oracle:thin:@l
      ocalhost:1521:xe","system","password");  //oracle

Connection con=DriverManager.getConnection( "jdbc:mysql://localhost:3306/sono
o","root","root");  //MySql
//here sonoo is database name, root is username and password
```

3) Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

```
      public Statement createStatement()throws SQLException
```

Example to create the statement object

```
      Statement stmt=con.createStatement();
```

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method

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

Example to execute query

```
ResultSet rs=stmt.executeQuery("select * from emp");
 while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

5) Close the connection object

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

Syntax of close() method

```
public void close()throws SQLException
```

Example to close connection

```
con.close();
```

DriverManager class

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

Useful methods of DriverManager class

Method	Description
1) public static void registerDriver(Driver driver):	is used to register the given driver with DriverManager.
2) public static void deregisterDriver(Driver driver):	is used to deregister the given driver (drop the driver from the list) with DriverManager.
3) public static Connection getConnection(String url):	is used to establish the connection with the specified url.
4) public static Connection getConnection(String url,String userName,String password):	is used to establish the connection with the specified url, username and password.

Connection interface

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback() etc.

By default, connection commits the changes after executing queries.

Commonly used methods of Connection interface:

1) public Statement createStatement(): creates a statement object that can be used to execute SQL queries.

2) public Statement createStatement(int resultSetType,int resultSetConcurrency): Creates a Statement object that will generate ResultSet objects with the given type and concurrency.

3) public void setAutoCommit(boolean status): is used to set the commit status.By default it is true.

4) public void commit(): saves the changes made since the previous commit/rollback permanent.

5) public void rollback(): Drops all changes made since the previous commit/rollback.

6) public void close(): closes the connection and Releases a JDBC resources immediately.

Statement interface

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

Commonly used methods of Statement interface:

The important methods of Statement interface are as follows:

1) public ResultSet executeQuery(String sql): is used to execute SELECT query. It returns the object of ResultSet.

2) public int executeUpdate(String sql): is used to execute specified query, it may be create, drop, insert, update, delete etc.

3) public boolean execute(String sql): is used to execute queries that may return multiple results.

4) public int[] executeBatch(): is used to execute batch of commands.

ResultSet interface

The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

By default, ResultSet object can be moved forward only and it is not updatable.

But we can make this object to move forward and backward direction by passing either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in createStatement(int,int) method as well as we can make this object as updatable by:

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,   ResultSet.CONCUR_UPD
ATABLE);
```

Commonly used methods of ResultSet interface

1) public boolean next():	is used to move the cursor to the one row next from the current position.
2) public boolean previous():	is used to move the cursor to the one row previous from the current position.
3) public boolean first():	is used to move the cursor to the first row in result set object.
4) public boolean last():	is used to move the cursor to the last row in result set object.
5) public boolean absolute(int row):	is used to move the cursor to the specified row number in the ResultSet object.
6) public boolean relative(int row):	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
7) public int getInt(int columnIndex):	is used to return the data of specified column index of the current row as int.
8) public int getInt(String columnName):	is used to return the data of specified column name of the current row as int.

9) public String getString(int columnIndex):	is used to return the data of specified column index of the current row as String.
10) public String getString(String columnName):	is used to return the data of specified column name of the current row as String.

Example of Scrollable ResultSet

Let’s see the simple example of ResultSet interface to retrieve the data of 3rd row.

```
import java.sql.*;
class FetchRecord{
public static void main(String args[])throws Exception{

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system",
"oracle");
Statement stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_U
PDATABLE);
ResultSet rs=stmt.executeQuery("select * from emp765");

//getting the record of 3rd row
rs.absolute(3);
System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));

con.close();
}}
```

PreparedStatement interface

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

Let's see the example of parameterized query:

```
String sql="insert into emp values(?,?,?)";
```

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

Why use PreparedStatement?

Improves performance: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

How to get the instance of PreparedStatement?

The prepareStatement() method of Connection interface is used to return the object of PreparedStatement. Syntax:

```
public PreparedStatement prepareStatement(String query)throws SQLException{}
```

Methods of PreparedStatement interface

The important methods of PreparedStatement interface are given below:

Method	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

Example of PreparedStatement interface that inserts the record

First of all create table as given below:

1. create table emp(id number(10),name varchar2(50));

Now insert records in this table by the code given below:

```
import java.sql.*;
class InsertPrepared{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system",
"oracle");

PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");
stmt.setInt(1,101);//1 specifies the first parameter in the query
stmt.setString(2,"Ratan");

int i=stmt.executeUpdate();
System.out.println(i+" records inserted");

con.close();

}catch(Exception e){ System.out.println(e);}

}
}
```

Example of PreparedStatement interface that updates the record

```
PreparedStatement stmt=con.prepareStatement("update emp set name=? where id=?");
stmt.setString(1,"Sonoo");//1 specifies the first parameter in the query i.e. name
stmt.setInt(2,101);

int i=stmt.executeUpdate();
System.out.println(i+" records updated");
```

Example of PreparedStatement interface that deletes the record

```
PreparedStatement stmt=con.prepareStatement("delete from emp where id=?");
stmt.setInt(1,101);

int i=stmt.executeUpdate();
System.out.println(i+" records deleted");
```

Example of PreparedStatement interface that retrieve the records of a table

```java
PreparedStatement stmt=con.prepareStatement("select * from emp");
ResultSet rs=stmt.executeQuery();
while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

Example of PreparedStatement to insert records until user press n

```java
import java.sql.*;
import java.io.*;
class RS{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system",
"oracle");

PreparedStatement ps=con.prepareStatement("insert into emp130 values(?,?,?)");

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

do{
System.out.println("enter id:");
int id=Integer.parseInt(br.readLine());
System.out.println("enter name:");
String name=br.readLine();
System.out.println("enter salary:");
float salary=Float.parseFloat(br.readLine());

ps.setInt(1,id);
ps.setString(2,name);
ps.setFloat(3,salary);
int i=ps.executeUpdate();
System.out.println(i+" records affected");

System.out.println("Do you want to continue: y/n");
String s=br.readLine();
if(s.startsWith("n")){
break;
}
}while(true);

con.close();
}}
```

Java ResultSetMetaData Interface

The metadata means data about data i.e. we can get further information from the data.

If you have to get metadata of a table like total number of column, column name, column type etc. , ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.

Commonly used methods of ResultSetMetaData interface

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

How to get the object of ResultSetMetaData:

The getMetaData() method of ResultSet interface returns the object of ResultSetMetaData. Syntax:

```
public ResultSetMetaData getMetaData()throws SQLException
```

Example of ResultSetMetaData interface :

```
import java.sql.*;
class Rsmd{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

PreparedStatement ps=con.prepareStatement("select * from emp");
ResultSet rs=ps.executeQuery();
ResultSetMetaData rsmd=rs.getMetaData();

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

con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

```
Output:Total columns: 2
       Column Name of 1st column: ID
       Column Type Name of 1st column: NUMBER
```

JDBC RowSet

The instance of **RowSet** is the java bean component because it has properties and java bean notification mechanism. It is introduced since JDK 5.

It is the wrapper of ResultSet. It holds tabular data like ResultSet but it is easy and flexible to use.

The implementation classes of RowSet interface are as follows:

- JdbcRowSet
- CachedRowSet
- WebRowSet
- JoinRowSet
- FilteredRowSet

Let's see how to create and execute RowSet.

```
JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
rowSet.setUsername("system");
rowSet.setPassword("oracle");

rowSet.setCommand("select * from emp400");
rowSet.execute();
```

It is the new way to get the instance of JdbcRowSet since JDK 7.

Advantage of RowSet

The advantages of using RowSet are given below:

1. It is easy and flexible to use
2. It is Scrollable and Updatable bydefault

Simple example of JdbcRowSet

Let's see the simple example of JdbcRowSet without event handling code.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.sql.RowSetEvent;
import javax.sql.RowSetListener;
import javax.sql.rowset.JdbcRowSet;
import javax.sql.rowset.RowSetProvider;

public class RowSetExample {
        public static void main(String[] args) throws Exception {
                Class.forName("oracle.jdbc.driver.OracleDriver");

    //Creating and Executing RowSet
        JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
        rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
        rowSet.setUsername("system");
        rowSet.setPassword("oracle");

        rowSet.setCommand("select * from emp400");
        rowSet.execute();

    while (rowSet.next()) {
                        // Generating cursor Moved event
                        System.out.println("Id: " + rowSet.getString(1));
                        System.out.println("Name: " + rowSet.getString(2));
                        System.out.println("Salary: " + rowSet.getString(3));
                }

        }
}
```

Full example of Jdbc RowSet with event handling

To perform event handling with JdbcRowSet, you need to add the instance of **RowSetListener** in the addRowSetListener method of JdbcRowSet.

The RowSetListener interface provides 3 method that must be implemented. They are as follows:

```
1) public void cursorMoved(RowSetEvent event);
2) public void rowChanged(RowSetEvent event);
3) public void rowSetChanged(RowSetEvent event);
```

Let's write the code to retrieve the data and perform some additional tasks while cursor is moved, cursor is changed or rowset is changed. The event handling operation can't be performed using ResultSet so it is preferred now.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.sql.RowSetEvent;
import javax.sql.RowSetListener;
import javax.sql.rowset.JdbcRowSet;
import javax.sql.rowset.RowSetProvider;

public class RowSetExample {
        public static void main(String[] args) throws Exception {
                Class.forName("oracle.jdbc.driver.OracleDriver");

    //Creating and Executing RowSet
    JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
    rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
    rowSet.setUsername("system");
    rowSet.setPassword("oracle");

        rowSet.setCommand("select * from emp400");
        rowSet.execute();

    //Adding Listener and moving RowSet
    rowSet.addRowSetListener(new MyListener());

                while (rowSet.next()) {
                      // Generating cursor Moved event
                      System.out.println("Id: " + rowSet.getString(1));
                      System.out.println("Name: " + rowSet.getString(2));
                      System.out.println("Salary: " + rowSet.getString(3));
               }

        }
}

class MyListener implements RowSetListener {
     public void cursorMoved(RowSetEvent event) {
               System.out.println("Cursor Moved...");
     }
    public void rowChanged(RowSetEvent event) {
               System.out.println("Cursor Changed...");
    }
    public void rowSetChanged(RowSetEvent event) {
               System.out.println("RowSet changed...");
    }
}
```