JDBC

- Satya Kaveti

Small Codes

Programming Simplified

A SmlCodes.Com Small presentation

In Association with Idleposts.com

For more tutorials & Articles visit SmlCodes.com

So called TITLE

First published on Aug 2016, Published by **SmlCodes.com**

Author Credits

Name : **Satya Kaveti**

Email : satyakaveti@gmail.com

Website : smlcodes.com, satyajohnny.blogspot.com

Digital Partners

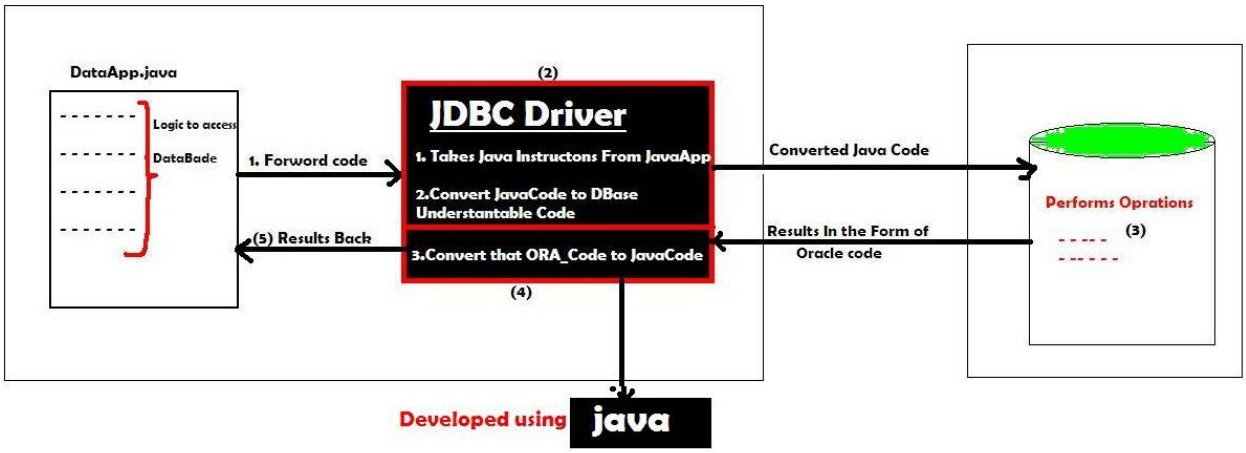
Table of Content

2. JDBC

2.1 Introduction

Before JDBC, ODBC API was the database API to connect and execute 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 (writtn in Java).

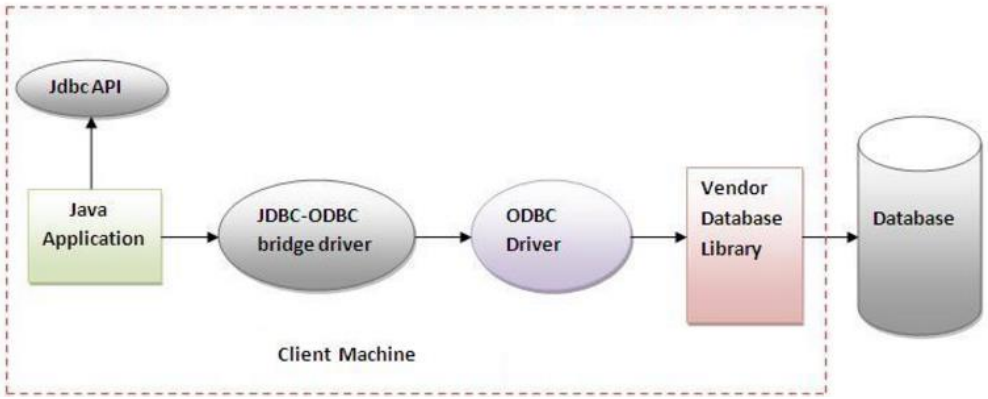


There are 4 types of JDBC drivers:

1. **JDBC-ODBC bridge driver**
2. **Native-API driver** **(partially java driver)**
3. **Network Protocol driver** **(fully java driver)**
4. **Thin driver** **(fully java driver)**

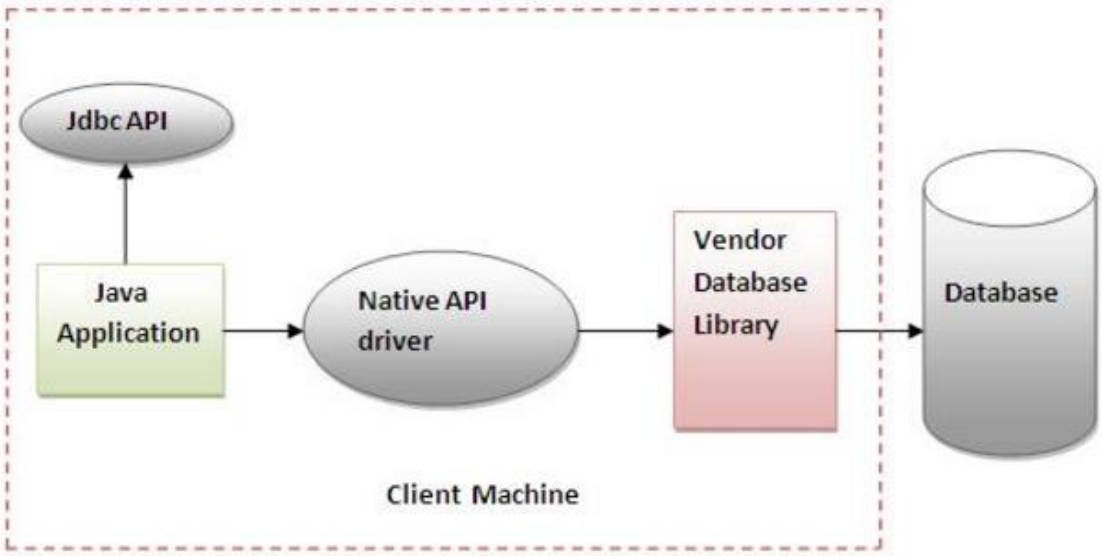
1. JDBC-ODBC bridge driver (Type-1)

- The JDBC-ODBC bridge driver uses **ODBC driver to connect to the database**.
- The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.
- Can be easily connected to **ANY database.**
- Performance degraded because **JDBC method call is converted into the ODBC calls**
- **ODBC driver needs** to be installed on the **client machine**
- Sun provided ODBC driver name : **sun.jdbc.odbc.JdbcOdbcDriver**



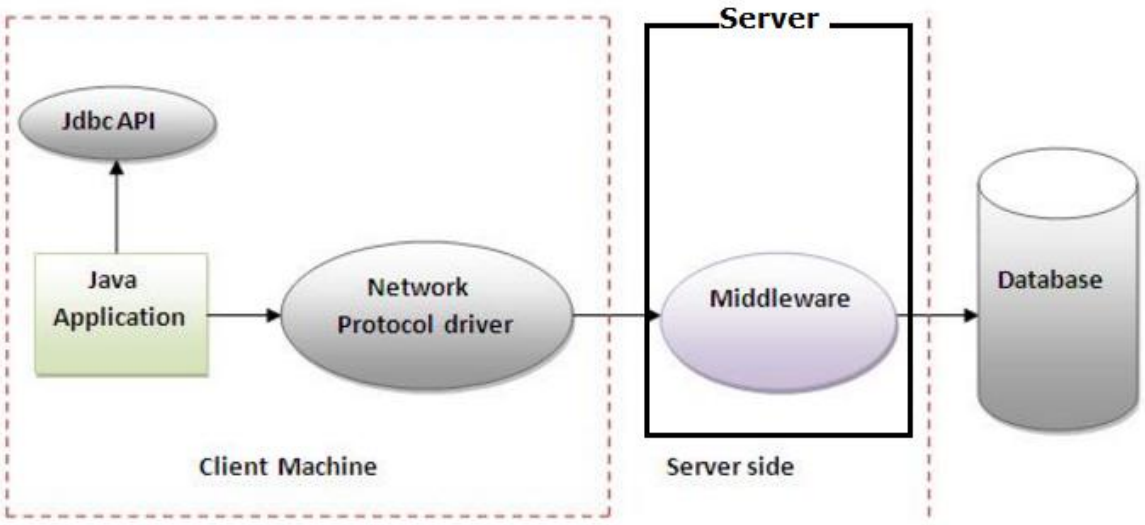
2. Native API driver (Type-1)

- The Native API driver uses the client-side libraries of the database.
- For **MySQL they have own Native API Driver, similarly for ORACLE, Postgres etc,**
- The driver converts JDBC (Java) method calls into native calls (MySQL, Oracle) of the database API.
- It is **NOT FULLY written entirely in java**
- The Native driver needs to be installed on the each client machine.



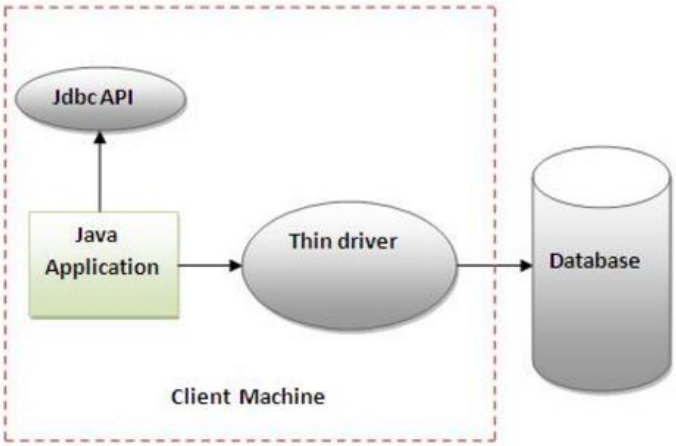
3. Network Protocol driver (Type-3)

- The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol.
- It is fully written in java.
- Used in **Connection Pooling**
- **Network support is required on client machine.**
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.



4. Thin driver (Type -4)

- The thin driver converts JDBC calls directly into the vendor-specific database protocol.
- It is **fully written in Java language**
- Better performance than all other drivers.
- **No software is required at client side or server side**.
- **com.mysql.jdbc.Driver** (MySQL), **oracle.jdbc.driver.OracleDriver**(ORACLE)



Steps to connect to the database

There are 5 steps to connect...

1. **Register the driver class**

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

2. **Creating connection**

   Connection con=DriverManager.getConnection ("url","system","password");

   Connection con=DriverManager.getConnection ("url "); //2nd Way

3. **Creating statement**

   Statement stmt=con.createStatement ();

4. **Executing queries**

   ResultSet rs=stmt.executeQuery("select * from emp");

5. **Closing connection**

   con.close();

cid	name	addr
101	Satya	HYD
102	Ravi	VIJ
103	RAKESH	CHENNEI
104	Surya	BANG

Table Data

```
101   Satya     HYD
102   Ravi      VIJ
103   RAKESH    CHENNEI
104   Surya     BANG
```

Output Data

```
public class JDBC {
     public static void main(String[] args) throws Exception {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con = DriverManager.getConnection
                    ("jdbc:mysql://localhost:3306/mydb", "root", "123456");

            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM customer");
            while (rs.next())
System.out.println(rs.getInt(1)+":"+rs.getString(2)+ "   " +
rs.getString(3));
            con.close();
     }
    }
```

In above Example we use getConnection("URL","UNAME","PWD") to connect with database.we can use directly without giving username/pwd also

There are two ways to connect java application with the access database.

- **Without DSN (Data Source Name) → above**
- **With DSN** → jdbc:odbc:mydsn (mydsn is DSN)

Creating DSN

Start > Administrative Tools > Data Sources (ODBC). →Add → (.mdb)

2.2 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().**

Methods

1) **public static void registerDriver(Driver driver):**
2) **public static void deregisterDriver(Driver driver):**
3) **public static Connection getConnection(String url):**
4) **public static Connection getConnection(String url,String uname,String pwd)**

2.3 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.

- **Statement createStatement()**
- **PreparedStatement prepareStatement(String sql)**
- **CallableStatement prepareCall(String sql)**
- **Blob createBlob()**
- **Clob createClob()**
- **String getSchema()**
- **DatabaseMetaData getMetaData()**

- **void close()**
- **void commit()**
- **void rollback()**
- **void setAutoCommit(boolean autoCommit)**
- **boolean getAutoCommit()**

All above statement related methods can have

- **ResultSet.TYPE_SCROLL_SENSITIVE**→ used to move ResultSet Both Directions
- **ResultSet.CONCUR_UPDATABLE**→ used to perform DML operations on ResultSet

2.4 Statement interface

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

- **public ResultSet executeQuery(String sql):**
  Is used to execute **SELECT query**. It returns the object of ResultSet.
- **public int executeUpdate(String sql):**
  Is used to execute DML, **CREATE, DROP, INSERT, UPDATE, DELETE** etc.
- **public boolean execute(String sql):**
  Is used to execute queries that may return multiple results. **>1** if success, **0** on fail
- **public int[] executeBatch():**
  Is used to execute batch of commands.

2.5 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**.
    - **public XXX getXXX(int rowIndex):**
    - **public XXX getXXX(String rowName):**
    - public boolean next()
    - public boolean previous()

- public boolean first()
- public boolean last()

executeQuery (String sql) example

We use this method to execute SELECT queries

```
public class JDBC {
	public static void main(String[] args) throws Exception {
		Class.forName("com.mysql.jdbc.Driver");
		Connection con = DriverManager.getConnection
("jdbc:mysql://localhost:3306/mydb", "root", "123456");

		Statement stmt = con.createStatement();
		ResultSet rs = stmt.executeQuery("SELECT * FROM customer");
		while (rs.next()) {
System.out.println(rs.getInt(1)+""+rs.getString(2)+" "+ rs.getString(3));
		}
		con.close();
	}
}
```

```
101  Satya  HYD
102  Ravi  VIJ
103  RAKESH  CHENNEI
104  Surya  BANG
```

executeUpdate (String sql) example

We use this method to **NON-SELECT** Queries like UPDATE, DELETE, etc

Returns 1 if → success

Returns 0 if → Failure

```
public class JDBC {
	public static void main(String[] args) throws Exception {
		String url = "jdbc:mysql://localhost:3306/mydb";
		Class.forName("com.mysql.jdbc.Driver");
	Connection con = DriverManager.getConnection(url, "root", "123456");
		Statement stmt = con.createStatement();

	String qry = "UPDATE `customer` SET `name`='Ram' WHERE  `cid`=102";
		int res = stmt.executeUpdate(qry);
		if (res > 0)
			System.out.println("Success is :" + res);
		else
			System.out.println("Failure is :" + res);
		con.close();
	}
}
```

```
Success is :1
Failure is :0
```

Boolean execute example

We can use **execute()** method in **both SELECT & NON-SELECT queries.**

1. SELECT

It returns TRUE on SELECT queres we can get ResultSet by calling below method

ResultSet rs = statement.getResultSet ()

2. NON-SELECT

It returns FALSE on NON-SELECT queres. we can get **Int** value by calling below method

int i = statement.getUpdateCount();

```
public class JDBC {
	public static void main(String[] args) throws Exception {
	String url = "jdbc:mysql://localhost:3306/mydb";
	Class.forName("com.mysql.jdbc.Driver");
	Connection con = DriverManager.getConnection(url, "root", "123456");
	Statement stmt = con.createStatement();

	// String qry = "SELECT * FROM customer";
	String qry = "UPDATE `customer` SET `name`='Ram' WHERE `cid`=102";

		boolean flag = stmt.execute(qry);
		if (flag == true) {
			System.out.println("SELECT QUERY\n --------");
			ResultSet rs = stmt.getResultSet();
			while (rs.next()) {
		System.out.println(rs.getString(1) + ":" + rs.getString(2));
			}
		}
		else {
			System.out.println("NON-SELECT QUERY\n --------");
			int i = stmt.getUpdateCount();
			System.out.println("Result is : " + i);
		}
	}
}
```

```
SELECT QUERY
 --------
101:Satya
102:Ram
103:RAKESH
104:Surya

NON-SELECT QUERY
 --------
Result is : 1
```

executeBatch(String sql) example

```
public class BatchDemo {
	public static void main(String[] args) throws Exception {

		String url = "jdbc:mysql://localhost:3306/mydb";
		Class.forName("com.mysql.jdbc.Driver");
	Connection con = DriverManager.getConnection(url, "root", "123456");

		Statement st = con.createStatement();
		st.addBatch("insert into  student   values(81, 'Syam', 'mtm')");
		st.addBatch("insert into  student   values(11, 'ram', 'mum')");
		st.addBatch("insert into  student   values(14, 'bam', 'kuk')");
		st.addBatch("insert into  student   values(44, 'pram', 'secu')");

		int rs[] = st.executeBatch();
		int sum = 0;
		for (int i = 0; i < rs.length; i++) {
			sum = sum + i;
		}
		System.out.println(sum + "Record are UPDATED using BATCH");
	}
}
```

Scrollabe Resultset(String sql) example

By Default ResultSet Object is not SCROLLABLE & NOT UPDATABLE.to make ResultSet Object to move both Directions we need to configure TYPE & MODE Values

Possible TYPE Values

- **ResultSet.TYPE_SCROLL_SENSITIVE** → **(Update Possible)**
- **ResultSet.TYPE_SCROLL_INSENSITIVE** → **(Default)**

Possible MODE Values

- **ResultSet.CONCUR_READ_ONLY** → **(Update Possible)**
- **ResultSet.CONCUR_UPDATABLE** → **(Default)**

Methods applicable on Scrolable ResultSet Object

- **int getRow()** → Returns ROW INDEX
- **boolean first()** → Keep CURSOR at 1st Record
- **boolean last()** → Keep CURSOR at LAST Record
- **boolean next()** → Moves Cursor to Forward
- **boolean previous()** → Moves Cursor to Backword
- **boolean absolute(int +/-)**→ Moves Cursor to given Index on ResultSet
- **boolean relative**(int +/-)→ Moves Cursor to given Index, based on current Row

```
public class JDBC {
	public static void main(String[] args) throws Exception {
	String url = "jdbc:mysql://localhost:3306/mydb";
	Class.forName("com.mysql.jdbc.Driver");
	Connection con = DriverManager.getConnection(url, "root", "123456");
	Statement st = con.createStatement
		(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);

	ResultSet rs = st.executeQuery("select * from customer");
			System.out.println("From Using Next\n------");

		while (rs.next()) {
			System.out.println(rs.getString(1)+":"+rs.getString(2));
		}

		System.out.println("\nFrom Using Previous ");
		while (rs.previous()) {
			System.out.println(rs.getString(1)+":"+rs.getString(2));
		}

		System.out.println("randomly.................. ");

rs.first();
System.out.println(rs.getRow()+"First:"+rs.getString(1)+":"+rs.getString(2));


rs.last();
System.out.println(rs.getRow()+"Last: "+rs.getString(1)+":"+rs.getString(2));

rs.absolute(4);//from starting point to 4 records
System.out.println(rs.getRow()+"Absolute:"+rs.getString(1)+":"+rs.getString(2
));

rs.relative(-2); //from here to 2 points back
System.out.println(rs.getRow()+"relative:"+rs.getString(1)+":"+rs.getString(2
));
	}
}
```

```
From Using Next
------
101:Satya
102:Ram
103:RAKESH
104:Surya

From Using Previous
104:Surya
103:RAKESH
102:Ram
101:Satya
randomly..................
1First: 101:Satya
4Last: 104:Surya
4Absolute Record : 104:Surya
2relative Record : 102:Ram
```

In above example we used on for SCROLLING resultset on both ditections using `ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY.`

If we want perform UPDATE operations & SCROLLING also, we have to use ResultSet.TYPE_`SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE`

Steps to Perform Insert/ UPDATE /Delete Operations on ResultSet

1. Select the Records

```
while (rs.next())
{
    System.out.println(rs.getRow()+""+rs.getString(1)+","+ rs.getString(2));
}
```

2. Perform INSERT Operation

```
System.out.println("1.INSERT OPERATION\n-----");
rs.moveToInsertRow(); // creates Empty Record
rs.updateInt(1, 200);
rs.updateString(2, "SACHIN");
rs.updateString(3, "MUMBAI");
rs.insertRow(); // Inserts Row
```

3. Perform UPDATE Operation

```
System.out.println("\n2.UPDATE OPERATION\n-----");
rs.absolute(2); // move to row to update
rs.updateString(3, "KOLKATA");
rs.updateRow();
```

4. Perform DELETE Operation

```
System.out.println("\n3.DELTE OPERATION\n-----");
rs.absolute(1); // move to row to DELETE
rs.deleteRow();
```

Example

```
public class JDBC {
      public static void main(String[] args) throws Exception {
            String url = "jdbc:mysql://localhost:3306/mydb";
            Class.forName("com.mysql.jdbc.Driver");
            Connection con = DriverManager.getConnection(url, "root",
"123456");
            Statement st = con.createStatement
            ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
            ResultSet rs = st.executeQuery("select * from customer");

            while (rs.next()) {
            System.out.println(rs.getRow() + "->" + rs.getString(1) + "," +
rs.getString(2));
            }

            System.out.println("1.INSERT OPERATION\n-----");
            rs.moveToInsertRow(); // creates Empty Record
            rs.updateInt(1, 200);
            rs.updateString(2, "SACHIN");
            rs.updateString(3, "MUMBAI");
            rs.insertRow(); // Inserts Row
```

```
            System.out.println("\n2.UPDATE OPERATION\n-----");
            rs.absolute(2); // move to row to update
            rs.updateString(3, "KOLKATA");
            rs.updateRow();

            System.out.println("\n3.DELTE OPERATION\n-----");
            rs.absolute(1); // move to row to DELETE
            rs.deleteRow();
        }
}
```

cid	name	addr
101	Satya	HYD
102	Ram	VIJ
103	RAKESH	CHENNEI
104	Surya	BANG

Before

cid	name	addr
102	Ram	KOLKATA
103	RAKESH	CHENNEI
104	Surya	BANG
200	SACHIN	MUMBAI

After

200 - Inserted

102 - Updated

101 - Deleted

2.6 PreparedStatement

The PreparedStatement interface is a sub interface of Statement. It is used to execute parameterized query.The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

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

(?) values will be set by calling the setter methods of PreparedStatement.

Method	Description
public void setInt(int index,int value)	sets integer value to the given parameter index.
public void setString(int index, String val)	sets String value to the given parameter index.
public void setFloat(int index, float value)	sets float value to the given parameter index.
public void setDouble(int index, double val)	sets double value to the given parameter index.
public int executeUpdate()	Uses for create, drop, insert, update, delete queries
public ResultSet executeQuery()	Executes the select query.
Boolean execute()	It returns TRUE for SELECT queries, FALSE for update, delete Queries
void addBatch()	
ParameterMetaData getParameterMetaData()	Retrieves the number, types and properties of this PreparedStatement object's parameters
ResultSetMetaDatagetMetaData()	contains information about the columns of the ResultSet object

void setBlob(int index, Blob x) **void setBlob(int index,InputStream is)** **void setBinaryStream(int i,InputStream is)** **getBinarayStream("column");**	Inserts Binary Large Object Videos,Audios
void setClob(int index, Clob x) **void setClob(int index, Reader reader)** **getCharacterStream("column");**	Inserts character Large Object Files

java.sql.Statement	**java.sql.PreparedStatement**
Statement is used for executing a static SQL statement in java JDBC.	PreparedStatement is used for executing a precompiled SQL statement in java JDBC.
java.sql.Statement cannot accept parameters at runtime in java JDBC.	java.sql.PreparedStatement can be executed repeatedly, it can accept different parameters at runtime in java JDBC.
java.sql.Statement is slower as compared to PreparedStatement in java JDBC.	java.sql.PreparedStatement is faster because it is used for executing precompiled SQL statement in java JDBC.

```
public class JDBC {
	public static void main(String[] args) throws Exception {
		String url = "jdbc:mysql://localhost:3306/mydb";
		String u = "root";
		String p = "123456";
		PreparedStatement ps=null;
		Class.forName("com.mysql.jdbc.Driver");
		Connection con = DriverManager.getConnection(url, u, p);

	  ps = con.prepareStatement("insert into customer values(?,?,?)");
		ps.setInt(1, 143);
		ps.setString(2, "sri");
		ps.setString(3, "mum");
		int res = ps.executeUpdate();
		System.out.println("Result    :     " + res);

	  ps =con.prepareStatement("SELECT * FROM  customer c WHERE c.cid<? ");
		ps.setInt(1, 200);
		ResultSet rs = ps.executeQuery();
		while (rs.next()) {
			System.out.println(rs.getString(2));
		}
	}
}
```

```
Result      :     1
Ram
RAKESH
Surya
sri
```

2.7 BLOB (BinaryLarge Obejects)

We use BLOB objects to store binary data like images, videos etc

We have following methods to Save & Retrive BLOB Objects

To Save

- **void setBinaryStream(int parameterIndex, InputStream x)**
- **void setBinaryStream(int parameterIndex, InputStream x, int length)**
- **void setBlob(int index, Blob x)**
- **void setBlob(int index,InputStream is)**

To Retrive

- **Blob getBlob(int columnIndex)**
- **Blob getBlob(String columnLabel)**
- **InputStream getBinaryStream(int columnIndex)**
- **InputStream getBinaryStream(String column)**

Steps:

1. **Read Image/ video data by using InputStream**
   FileInputStream fis=**new** FileInputStream("d:\\g.jpg");

2. **Create PreparedStatement Object to write insert image query**
   PreparedStatement ps=con.prepareStatement("insert into imgtable values(?,?)");

3. **Set parameter values**
   ps.setInt(1, 101);
   ps.setBinaryStream(2,fis);

4. **Execute Query**
   int i=ps.executeUpdate();

5. To get image from table➔ execute Select Quey , call on rs object
   FileInputSteam fs= rs.getBinaraysStream("column");

6. Choose Location to Store new Image

```
FileOutputStream fos = new FileOutputStream("res/newpict.gif");
```

sno	name	img
100	johny	0x4749463839617300480F53F00FF00CCFF0099FF006...

Example BlobInsert Operation

```
public class BlobInsert {
	public static void main(String[] args) throws Exception {
		String url = "jdbc:mysql://localhost:3306/mydb";
		String u = "root";
		String p = "123456";
		Class.forName("com.mysql.jdbc.Driver");
		Connection con = DriverManager.getConnection(url, u, p);
PreparedStatement ps = con.prepareStatement("insert into blobtest
values(?,?,?)");

		File f = new File("res/img.gif");
		FileInputStream fis = new FileInputStream(f);

		ps.setInt(1, 100);
		ps.setString(2, "johny");
		ps.setBinaryStream(3, fis, (int) f.length());
		ps.executeUpdate();
		System.out.println("Record is Inserted");
	}
}
```

Example BlobInsert Operation

```
public class BlobRetrive {
	public static void main(String[] args) throws Exception {
		String url = "jdbc:mysql://localhost:3306/mydb";
		String u = "root";
		String p = "123456";
		Class.forName("com.mysql.jdbc.Driver");
		Connection con = DriverManager.getConnection(url, u, p);
		Statement st = con.createStatement();
		ResultSet rs = st.executeQuery("select * from blobtest");

		if (rs.next()) {
			InputStream in = rs.getBinaryStream("img");
		FileOutputStream fos = new FileOutputStream("res/newpict.gif");

			int bytesRead = 0;
			byte[] buffer = new byte[4096];
			while ((bytesRead = in.read(buffer)) != -1) {
				fos.write(buffer, 0, bytesRead);
			}
			System.out.println("photo is stored in newpict.gif");
			fos.close();
			in.close();
			rs.close();
			st.close();
			con.close();
		} // if
	}// main
}// class
```

2.8 CLOB

We use CLOB objects to store character data like txt files,word files etc

We have following methods to Save & Retrive CLOB Objects

To Save

- **void setCharacterStream(int parameterIndex, InputStream x)**
- **void setCharacterStream(int parameterIndex, InputStream x, int length)**
- **void setClob(int index, Clob x)**
- **void setClob(int index,InputStream is)**

To Retrive

- **Blob getClob(int columnIndex)**
- **Blob getClob(String columnLabel)**
- **InputStream getCharacterStream (int columnIndex)**
- **InputStream getCharacterStream (String column)**

Steps:

1. **Read File data by using InputStream**
   FileInputStream fis=**new** FileInputStream("d:\\g.jpg");

2. **Create PreparedStatement Object to write insert image query**
   PreparedStatement ps=con.prepareStatement("insert into filetable values(?,?)");

3. **Set parameter values**
   ps.setInt(1, 101);
   ps.setCharacterStream (2,fis);

4. **Execute Query**
   int i=ps.executeUpdate();

5. To get image from table➔ execute Select Quey , call on rs object
   FileInputSteam fs= rs. getCharacterStream ("column");

2.9 CallableStatement

CallableStatement interface is used to call the **stored procedures and functions**.

We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.

Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

Stored Procedure	Function
is used to perform business logic.	is used to perform calculation.
must not have the return type.	must have the return type.
may return 0 or more values.	may return only one values.
We can call functions from the procedure.	Procedure cannot be called from function.
It supports input and output parameters.	Function supports only input parameter.
Exception handling using try/catch block can be used in stored procedures.	Exception handling using try/catch can't be used in user defined functions.

We use following method on Connection object to get CallableStatement Object

```
public CallableStatement prepareCall("{ call procedurename(?,?...?)}");

CallableStatement cs=con.prepareCall("{call myprocedure(?,?)}");
```

Example procedeure

```
create or replace function sum (n1 in number,n2 in number)
return number
is
temp number(8);
    begin
        temp :=n1+n2;
    return temp;
end;
/
```

In above example n1, n2 are Input Paramaters & temp is the Output parameter

To set Input Paramaters we use setXXX(int index, Value) methods

```
cs.setInt(1, 10);
cs.setInt(2, 20);
```

To set Output Paramaters we use registerOutParameter(int index, Type.TYPE) method

```
cs.registerOutParameter(1, Types.INTEGER);
```

int	Types.INTEGER
long	Types.LONG
String	Types.STRING
Date	Types.DATE
.....	Types.....

To excute CallableStatement we use **execute()** method

```
cs.execute();
```

To Get results we use **getXXX(int outputParamIndex)** method

```
cs.getInt(1);
```

```
public class FuncSum {
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");

CallableStatement stmt=con.prepareCall("{?= call sum4(?,?)}");
stmt.setInt(2,10);
stmt.setInt(3,43);
stmt.registerOutParameter(1,Types.INTEGER);
stmt.execute();

System.out.println(stmt.getInt(1));

}
}
```

```
Output: 53
```

2.10 Metadata

We have 3 types of metadata in jdbc

1) **DataBaseMetaData**
2) **ResultSetMetaData**
3) **ParameterMetaData**

1. DataBaseMetaData

We can get database meta like database details, driver name, name of total number of tables, no.of views etc. by using DataBaseMetaData class, we can get by using Connection Object

```
DatabaseMetaData getMetaData()
DatabaseMetaData dm = con.getMeteData()
```

- **String getDriverName()**
- **String getDriverVersion()**
- **String getURL()**
- **String getUserName()**
- **String getDatabaseProductName()**
- **String getDatabaseProductVersion()**
- **int getDatabaseMajorVersion()**
- **int getDatabaseMinorVersion()**

```
public class JDBC {
	public static void main(String[] args) throws Exception {
		String url = "jdbc:mysql://localhost:3306/mydb";
		String u = "root";
		String p = "123456";
		Class.forName("com.mysql.jdbc.Driver");

		Connection con = DriverManager.getConnection(url, u, p);
		DatabaseMetaData dm = con.getMetaData();
		System.out.println("Driver    :     "+dm.getDriverName());
		System.out.println("DriverVersion: "+dm.getDriverVersion());
		System.out.println("URL : "+dm.getURL());
		System.out.println("UserName : "+dm.getUserName());
		System.out.println("DatabseName : "+dm.getDatabaseProductName());
		System.out.println("DVersion : "+dm.getDatabaseProductVersion());
		System.out.println("Major : "+dm.getDatabaseMinorVersion());
		System.out.println("Minor : "+dm.getDatabaseMajorVersion());

	}
}
```

```
Driver  :       MySQL-AB JDBC Driver
DriverVersion: mysql-connector-java-5.1.18 ( Revision: tonci.grgin@oracle.com-20110930151701-
jfj14ddfq48ifkfq )
URL : jdbc:mysql://localhost:3306/mydb
UserName : root@localhost
DatabseName : MySQL
DVersion : 5.6.26
Major : 6
Minor : 5
```

2. ResultSetMetaData

- The metadata means data about data
- We can get ReselutSet meta information like no.of columns, column data,table name etc
    - **String getTableName(int column)**
    - **int getColumnCount()**
    - **String getColumnName(int column)**
    - **int getColumnType(int column)**

```
public class JDBC {
	public static void main(String[] args) throws Exception {
		String url = "jdbc:mysql://localhost:3306/mydb";
		String u = "root";
		String p = "123456";
		Class.forName("com.mysql.jdbc.Driver");
		Connection con = DriverManager.getConnection(url, u, p);
		Statement st = con.createStatement();
		ResultSet rs = st.executeQuery("select * from customer");
		ResultSetMetaData rm = rs.getMetaData();
		System.out.println(rm.getTableName(1)); // customer
		System.out.println(rm.getColumnCount());// 3
		System.out.println(rm.getColumnName(2));// name
		System.out.println(rm.getColumnType(2));// 12
	}
}
```

3. ParameterMetaData

Used to get information about the types and properties for each parameter marker in a **PreparedStatement** object

- **int getParameterCount()**
- **int getParameterType(int param)**
- **String getParameterTypeName(int param)**

2.11 Batch Processing

Instead of executing a single query, we can execute a group of queries.The java.sql.Statement and java.sql.PreparedStatement interfaces provide methods for batch processing

- **void addBatch(String query)** ➔ It adds query into batch.
- **int[] executeBatch()** ➔ It executes the batch of queries.

```
Statement stmt=con.createStatement();
stmt.addBatch("insert into user420 values(190,'abhi',40000)");
stmt.addBatch("insert into user420 values(191,'umesh',50000)");
stmt.executeBatch();//executing the batch
```

```
public class JDBC {
	public static void main(String[] args) throws Exception {
		String url = "jdbc:mysql://localhost:3306/mydb";
		String u = "root";
		String p = "123456";
		PreparedStatement ps = null;
		Class.forName("com.mysql.jdbc.Driver");
		Connection con = DriverManager.getConnection(url, u, p);
		Statement st = con.createStatement();
		st.addBatch("insert into  student    values(81, 'Syam', 'mtm')");
		st.addBatch("insert into  student    values(11, 'ram', 'mum')");
		st.addBatch("insert into  student    values(14, 'bam', 'kuk')");
		st.addBatch("insert into  student    values(44, 'pram', 'sec')");

		int rs[] = st.executeBatch();
		int sum = 0;
		for (int i = 0; i < rs.length; i++) {
			sum = sum + i;
		}
		System.out.println(sum + "Record are UPDATED using BATCH");
	}
}
```

2.12 Transcations

In JDBC, **Connection interface** provides methods to manage transaction.

Method	Description
Void setAutoCommit(boolean status)	If it is true each transaction is committed bydefault.
void commit()	Commits the transaction.
void rollback()	Cancels the transaction.

```
class FetchRecords{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","u","p");
    con.setAutoCommit(false);
            Statement stmt=con.createStatement();
            stmt.executeUpdate("insert into user420 values(190,'abhi',40000)");
            stmt.executeUpdate("insert into user420 values(191,'umesh',50000)");
            con.commit();
            con.close();  }
            }
```

2.13 RowSet Interfcae

ResultSet Disdavantages

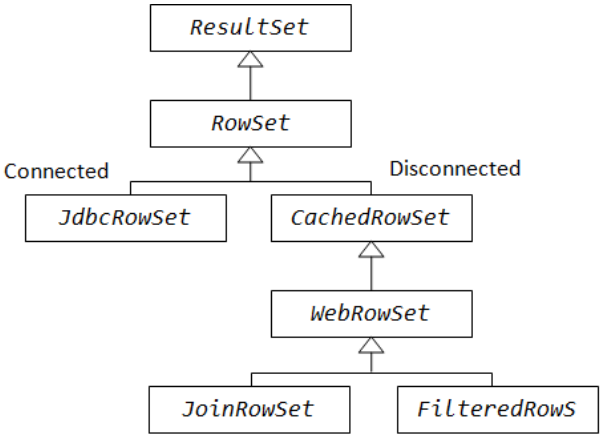
- ResultSet Object is not serilaizable , because it is always maintains a connection with DB
- We can't pass theResultset object from one class to other class across the network.

Rowset

- **RowSet extends the ResultSet interface** so it has all the methods of ResultSet
- RowSet **can be serialized** because it doesn't have a connection to any database
- RowSet **Object can be sent from one class to another across the network**.

We have following types of RowSets

1. **JdbcRowSet**
2. **CachedRowSet**
3. **WebRowSet**
4. **JoinRowSet**
5. **FilteredRowSet**



Features	JdbcRowSet	CacheRowSet	WebRowSet
Scrollable	✓	✓	✓
Updateable	✓	✓	✓
Connected	✓	✓	✓
Disconnected		✓	✓
Serializable		✓	✓
Generate XML			✓
Consume XML			✓

```
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();
```

2.14 Notes