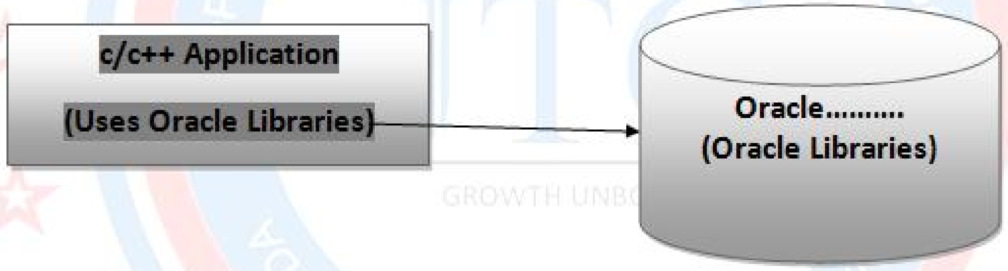
Introduction

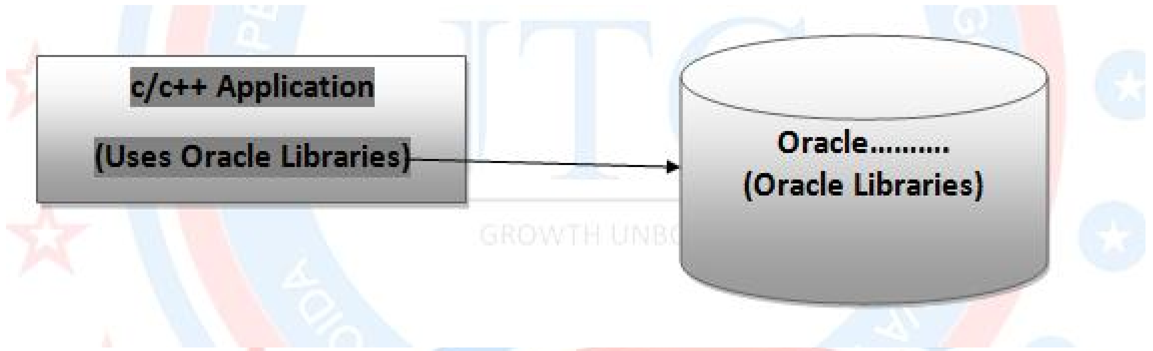
- JDBC is a technology which is used to intract with the databse from java Application.
- JDBC Technology is a part of java Standard Edition.
- JDBC is a Specification provided by java vendor and implemented by java Vendor or DB Vendor.

JDBC Versions

- JDBC 3.0 is released under J2SE 1.4.2.
- No updation under J2SE 5.0.
- JDBC 4.0 is released under Java SE 6.
- JDBC 4.1 is released under Java SE 7.
- JDBC 4.2 is released under Java SE 8.
- If you want to intract with database using c or c++, you need to use database specific libraries in your application directly.
- In the below diagram, c or c++ application wants to intract with Oracle database. So it is using Oracle libraries directly

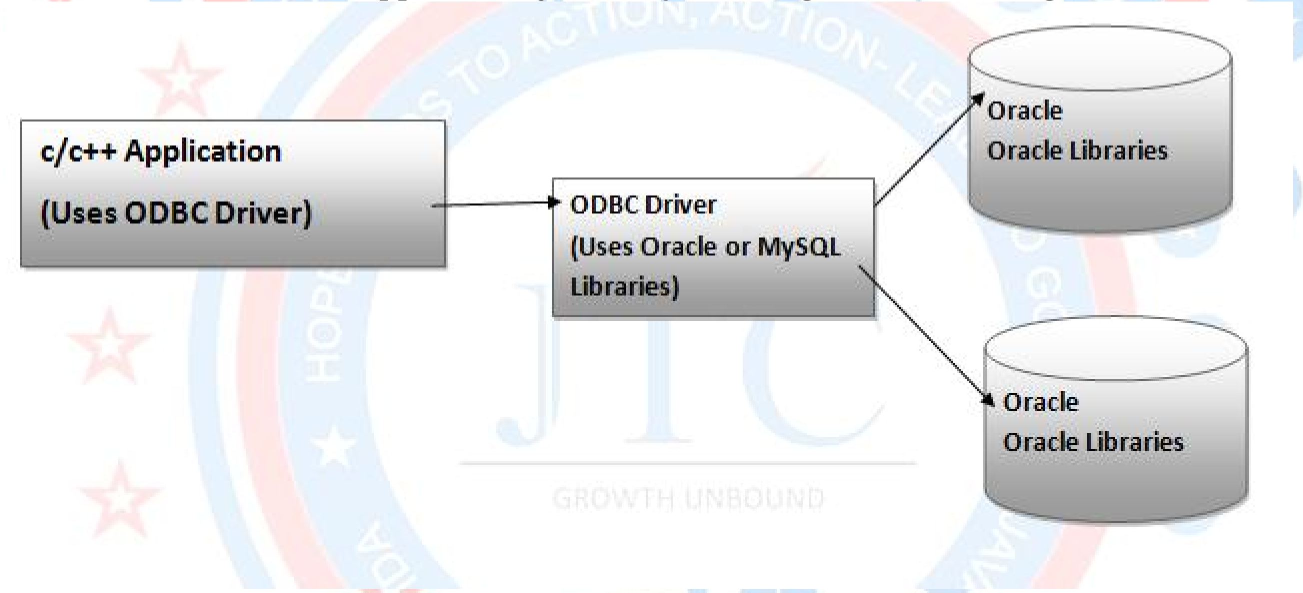


- Later when you migrate the database to another database then you then to rewrite the entire application using new database specific libraries.
- In the below diagram, your c or c++ application wants to intract with MySQL database. So you have to rewrite entire application using MySQL libraries.
- This increases the maintance of the application.



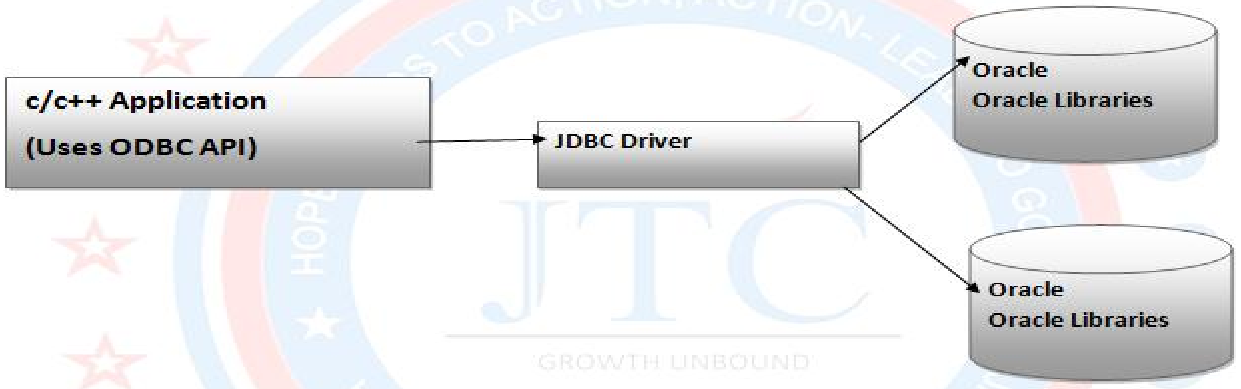
- To avoid the above said maintains problem, Microsoft has introduced ODBC Driver.
- ODBC stands for Open DataBase Connectivity.
- With ODBC driver, you no need to use database specific libraries directly in your application.
- Your application now intract with ODBC driver instead of using database specific librearies directly and ODBC Driver intracts with database specific libraries.

- Now when you migrate the database to another database then you no need to rewrite the entire application.you can just change ODBC Configuration.



- ODBC Driver setup is available only on windows operating system and also ODBC is not good in terms of performance.
- To avoid these limitations, SUN has provided JDBC API and JDBC Drivers.

- JDBC API and JDBC Drivers are Java Based Programs which runs on any Operating System.



- Java Program which is using JDBC API is called as JDBC Program.
- Two packages provided under JDBC API called:
    - Java.sql
    - Javax.sql
- Various classes and interfaces are provided under above two packages.

Java.sql package

DriverManager	Driver	Connection	Statement
PreparedStatement	CallableStatement	ResultSet	DatabaseMetadata
ResultSetMetadata	Types		

Javax.sql package

RowSet	JdbcRowSet	CachedRowSet	DataSource

Steps to Write JDBC Program
- Step 1: Load the Driver class.
- Step 2:Establish the Connection between JDBC Program and Database.
- Step 3:Prepare the SQL Statement.
- Step 4:Create the JDBC Statement.
- Step 5:Submit the SQL Statement to Database using JDBC Statement.
- Step 6:Process the result.
- Step 7:Close all the resources.

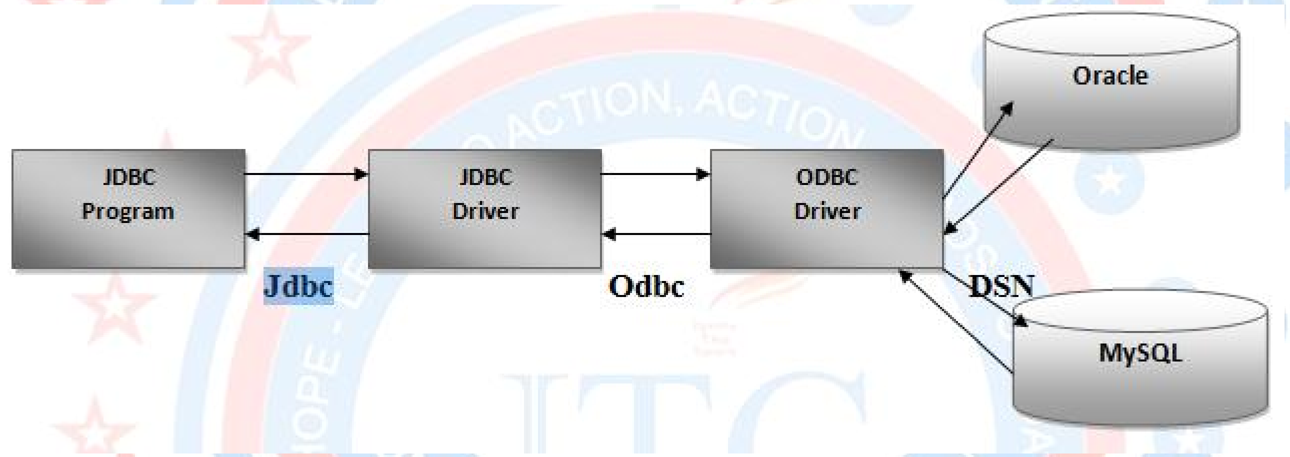
Types Of JDBC Drivers
- There are 4 types of JDBC Drivers.
    - Type 1 Driver   Jdbc ODBC Bridge Driver
    - Type 2 Driver   Partial Java and Partial Native Driver

- Type 3 Driver Net Protocol Driver
- Type 4 Driver Pure Java Driver

Type 1 Driver

Name	JDBC ODBC Bridge Driver
Vendor	Java Vendor
Driver Class	Sun.jdbc.odbc.JdbcOdbcDriver
URL	Jdbc:odbc:<Data Source Name>
Username	<Database Username>
Password	<Database Password>
Softeware Required	DB,Java,ODBC Drivers

Architecture:



Create the following table in Oracle and MySQL:
Create table jtcstudents(sid int primary key,sname varchar(10),email carchar(15),phone long);

Steps to Configure ODBC Data Source Name for Oracle

- Open controlpanel
- Open Administrative Tools
- Open Data Sources(ODBC)
- Click on add button under User DSN tab
- Select the Oracle in XE from the list

- Click on Finish
- Provide the following information
  - Data Source Name JTCORADSN
  - TNS Service Name XE
  - User Id system(Oracle username)
- To test the connection click on Test Connection
  - Provide the Oracle Password
  - Click o Ok
- Click on OK button on the Configration Window.
- Click on OK Button of ODBC Administrator Window

Steps to Configure ODBC Data Source Name for MySQL

- Install ODBC Driver usinng mysql-connector-odbc-5.2.5- win-32.msi from Student DVD
- Open Control Panel
- Open Administrative Tools
- Open Data Sources (ODBC)
- Click on Add button under User DSN tab
- Select the MySQL ODBC 5.2 ANSI Driver from the list
- Click on Finish
- Provide the following information
  - Data Source Name JTCMYDSN
  - TCP/IP Server localhost
  - User root (MYSQl username)
  - Password JTCindia (MYSQL password)
- Select the database from the list jtcdb
- To test the connection click on Test
  - Click on OK button on the Configration Window
- Click on OK Button of ODBC Administration Window

JTC 1.java

```
package com.jtcindia.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
/*
```

```
 * @Author:somprakash rai
 *@company: java training Center
 *@see:www.jtcindia.org
*/
public class Jtc1 {
        static{

                Connection con=null;
                Statement st=null;
                try{
                        //step 1:Load the drive class.
                        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                        //step2. Establish the Connection.
                        con=DriverManager.getConnection("jdbc:odbc:JTCORADSN",
"root","root");
                        // step 3: Prepare the SQL Statement.
                        String sql="insert into jtcstudents
values(99,'som','som@jtc.com','123345567'";
                        // Step 4: Create the JDBC Statement.
                        st=con.createStatement();
                        //Step 5: Submit the SQL Statement to Database using JDBC
Statement.
                        int x=st.executeUpdate(sql);
                        // Step 6:Process the result.
                        if(x==1){
                                System.out.println("Record inserted");
                        }else{
                                System.out.println("Record Not Inserted");
                        }
                }catch(Exception e){
                        e.printStackTrace();
                }finally{
                        // Step 7:Close all the resources.
                        try{
                                if(st!=null) st.close();
                                if(con!=null) con.close();
                        }catch(Exception e){
                                e.printStackTrace();
```

```
                    }
                }
            }

    }
```

JTC 2.java

```
package com.jtcindia.jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
/*
 * @Author:somprakash rai
 *@company: java training Center
 *@see:www.jtcindia.org
*/
public class Jtc2 {
        static{

                Connection con=null;
                Statement st=null;
                try{
//step 1:Load the drive class.
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
//step2. Establish the Connection.
        con=DriverManager.getConnection("jdbc:odbc:JTCORADSN", "root","root");
// step 3: Prepare the SQL Statement.
        String sql="insert into jtcstudents values(88,'som','som@jtc.com','123345567'";
// Step 4: Create the JDBC Statement.
        st=con.createStatement();
//Step 5: Submit the SQL Statement to Database using JDBC Statement.
                int x=st.executeUpdate(sql);
// Step 6:Process the result.
        if(x==1){
        System.out.println("Record inserted");
                        }else{
```

```
                System.out.println("Record Not Inserted");
                    }
                }catch(Exception e){
                    e.printStackTrace();
                }finally{
                    // Step 7:Close all the resources.
                    try{
                        if(st!=null) st.close();
                        if(con!=null) con.close();
                    }catch(Exception e){
                        e.printStackTrace();
                    }
                }
            }

}
```

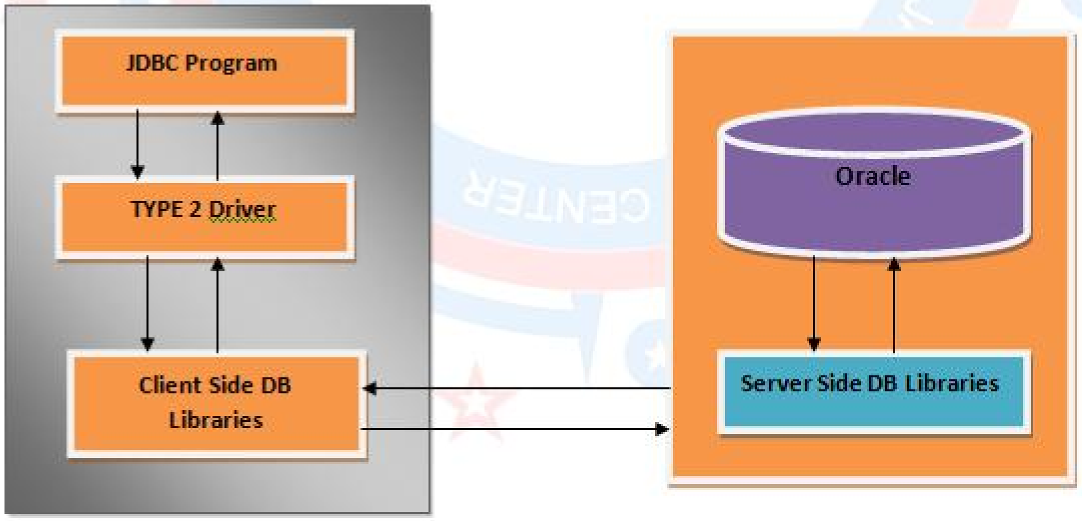
JTC3.java

```
package com.jtcindia.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
/*
 * @Author:somprakash rai
 *@company: java training Center
 *@see:www.jtcindia.org
*/
public class Jtc3 {
        static{

                Connection con=null;
                Statement st=null;
                try{

        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con=DriverManager.getConnection("jdbc:odbc:JTCORADSN", "root","root");
                        String sql="select * from jtcstudents";
                        st=con.createStatement();
                        ResultSet rs=st.executeQuery(sql);
                        while(rs.next()){
                                int sid=rs.getInt(1);
                                String sn=rs.getString(2);
                                String em=rs.getString(3);
                                String ph=rs.getString(4);
                                System.out.println(sid+"\t"+sn+"\t"+em+"\t"+ph);
                        }
                }catch(Exception e){
                        e.printStackTrace();
                }finally{
                        try{
                                if(st!=null) st.close();
                                if(con!=null) con.close();
                        }catch(Exception e){
                                e.printStackTrace();
                        }
                }
        }
}
```

Type 2 Driver:

Name	**Partial native partial java Driver**
Vendor	DB Vendor
Driver Class	**Oracle.jdbc.driver.OracleDriver**
URL	Jdbc:oracle:oci8:@hostname:port:serviceName Ex: jdbc:oracle:oci8:@localhost:1521:XE
Username	<Database Username>
Password	<Database Password>
Software Required	Database client Server Edition,Java

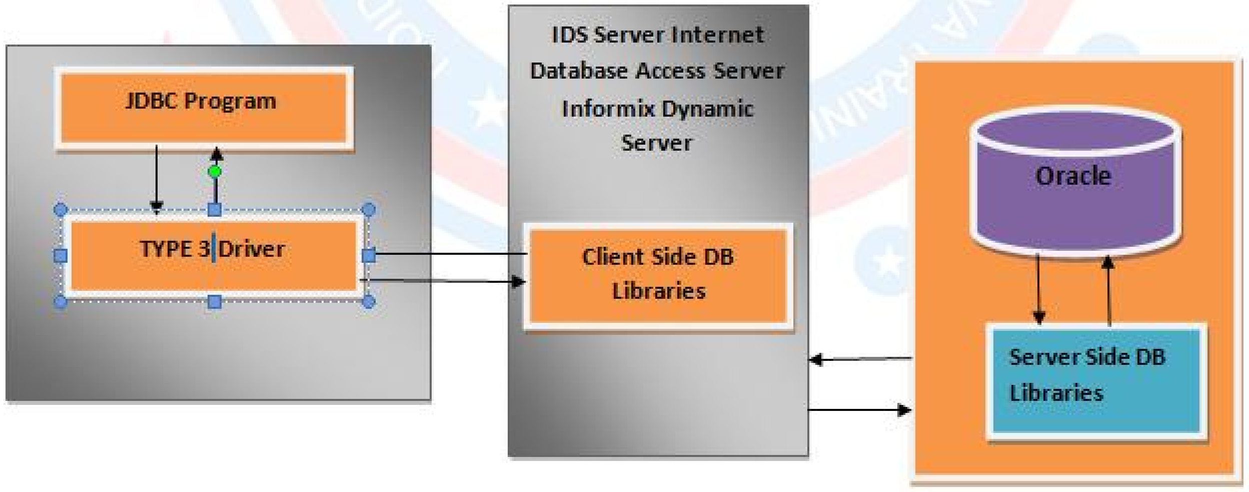
Architecture:



Type 3 Driver

Name	Net Protocol
Vendor	Java Vendor
Driver Class	Com.ids.Driver
URL	Jdbc:ids://hostname…
Username	<Database Username>
Password	<Database Password>
Softeware Required	IDS Server, Database Client Server Edition,java

Architecture:



Type 4 Driver

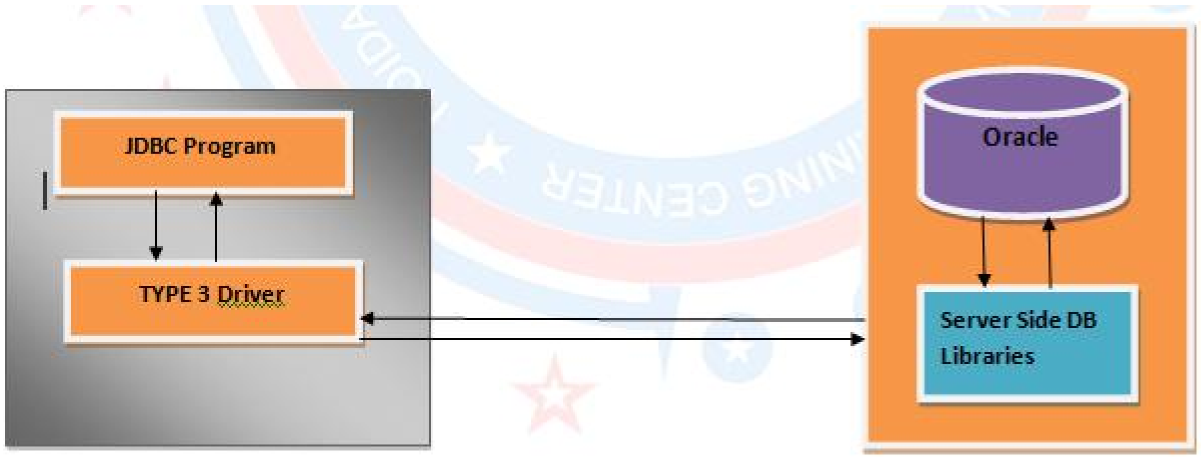
Name	Pure Java Driver
Vendor	Java Vendor
Username	<Database Username>
Password	<Database Password>
Softeware Required	Database,Java

For Oracle Driver

Driver class	**Oracle.jdbc.driver.OracleDriver**
Url	**Jdbc:oracle:thin:@<host>:<port>:<serviceName>** **Ex: jdbc:oracle:thin:@localhost:1521:XE**
Class Path	**Ojdbc14.jar** **Ojdbc6.jar**

For MySQL Driver

Driver class	**Com.mysql.jdbc.Driver**
Url	**Jdbc:mysql://<host>:<port>:/<dbName>** **Ex: jdbc:mysql://localhost:3306/jdbc**
Class Path	**Mysql.jar**



JDBCUtil.java

```
package jdbcType4Driver;

import java.sql.Connection;

import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
/**@Author:somprakash rai
*@company:java Training Center
*@see:www.jtcindia.org*/

public class JdbcUtil {
	public static Connection getOraConnection() throws SQLException,
ClassNotFoundException{
		Class.forName("oracle.jdbc.driver.OracleDriver");
		String url="jdbc:oracle:thin:@localhost:1521:XE";
		Connection con=DriverManager.getConnection(url,"system","jtcsom");
		return con;

	}

	public static Connection getMySQLConnection() throws ClassNotFoundException,
SQLException{
		Class.forName("com.mysql.jdbc.Driver");
		String url="jdbc:mysql://localhost:3306/jdbc";
		Connection con=DriverManager.getConnection(url,"root","root");
		return con;

	}
	public static void cleanup(Statement st,Connection con){
		try{
			if(st!=null) st.close();
```

```
                    if(con!=null) con.close();
            }catch(Exception e){
                    e.printStackTrace();
            }
    }


    public static void cleanup(ResultSet rs,Statement st,Connection con){
            try{
                    if(rs!=null) rs.close();
                    if(st!=null) st.close();
                    if(con!=null) con.close();
            }catch(Exception e){
                    System.out.println(e);
            }
    }

}
```

Jtc4.java	Jtc5.java

package jdbcType4Driver;
 	
import java.sql.Connection;
import java.sql.Statement;
 	
public class Jtc4 {
public static void main(String ar[]){
Connection con=null;
Statement st=null;
 	
try{
//con=JdbcUtil.getMySQLConnection();
con=JdbcUtil.getOraConnection();
String qry="insert into students values(77,'Som','som@jtc.com','9990399111')";
st=con.createStatement();
int x=st.executeUpdate(qry);
if(x==1){
System.out.println("Record inserted");
}else{
System.out.println("Record Not Inserted");
}
}catch(Exception e){
e.printStackTrace();
}finally{
JdbcUtil.cleanup(st, con);
}
	package jdbcType4Driver;
	
	import java.sql.Connection;
	import java.sql.ResultSet;
	import java.sql.Statement;
	/**@Author:somprakash rai
	*@company:java Training Center
	@see:www.jtcindia.org/
	public class Jtc5 {
	public static void main(String ar[]){
	Connection con=null;
	Statement st=null;
	ResultSet rs=null;
	
	try{
	//con=JdbcUtil.getMySQLConnection();
	con=JdbcUtil.getOraConnection();
	String qry="select * from students";
	st=con.createStatement();
	rs=st.executeQuery(qry);
	if(rs.next()){do{
	int id=rs.getInt(1);
	String name=rs.getString(2);
	String email=rs.getString(3);
	String phone=rs.getString(4);
	System.out.println(id+"\t"+name+"\t"+email+"\t"+phone);

}
}
	} while(rs.next());
	}else
	System.out.println("Record Not Inserted");
	
	}catch(Exception e){
	e.printStackTrace();
	}finally{
	JdbcUtil.cleanup(st, con);
	}
	}
	}

Pros and Cons of Types Of Drivers

	Advantages	Disadvantages
Type 1 Driver	• Type 1 is very easy to use and maintain. • Type 1 is suitable for migrating application to java without changing existing ODBC setup. • No Extra software is required for the Type 1 implementation. • Performance of the Type 1 is acceptable.	• Type 1 driver implementation is possible in window OS only becouse ODBC driver avaialable only with windows. • Performance of this driver is not excellent but acceptable.
Type 2 Driver	• Type 2 is faster than all other drivers.	• In type 2 both client and server machine will be have the database library. • When database is migrated then you will be grt much maintance because you need to re-install client side libraries in all the client machine.
Type 3 Driver	• In type 3,client side DB libraries are moved to middleware server called IDS server. • Because of this,client side maintance is reduce.	• You need to purches extra software called IDS server. • Because of having middlware server between your program and database server,performance will be reduced.
Type 4 Driver	• This driver is best among all the drivers and highly recommended to use	• Negglible.

JDBC Statement

There are 3 Type of JDBC Statement:

- **Statement**
- **PreparedStatement**
- **CallableStatement**

Statement :-

- **Statement is an interface available in java.sql package.**
- **Subclass of statement interface is provide by Drive vendor.**
- **You can create the Statement using the following methods of Connection interface.**
  - **public Statement createStatement()**
  - **public Statement createStatement(int , int)**
  - **public Statement createStatement(int ,int ,int)**

- **After creating the Statement object , you can call one of the following methods to submit the SQL Statement to Database.**
  - **public int executeUpdate(String sql)**
  - **public boolean execute(String sql)**
  - **public ResultSet executeQuery(String sql)**
- **Public int executeUpdate(String sql)When you want to submit insert or update or delete SQL Statements then use executeUpdate() method which returns the number of records inserted or updated or deleted.**
- **Public ResultSet executeQuery(String sql) When you want to submit Select SQL Statements then use executeQuery() method which returns the number of records fetched by select statement interms of ResultSet object.**

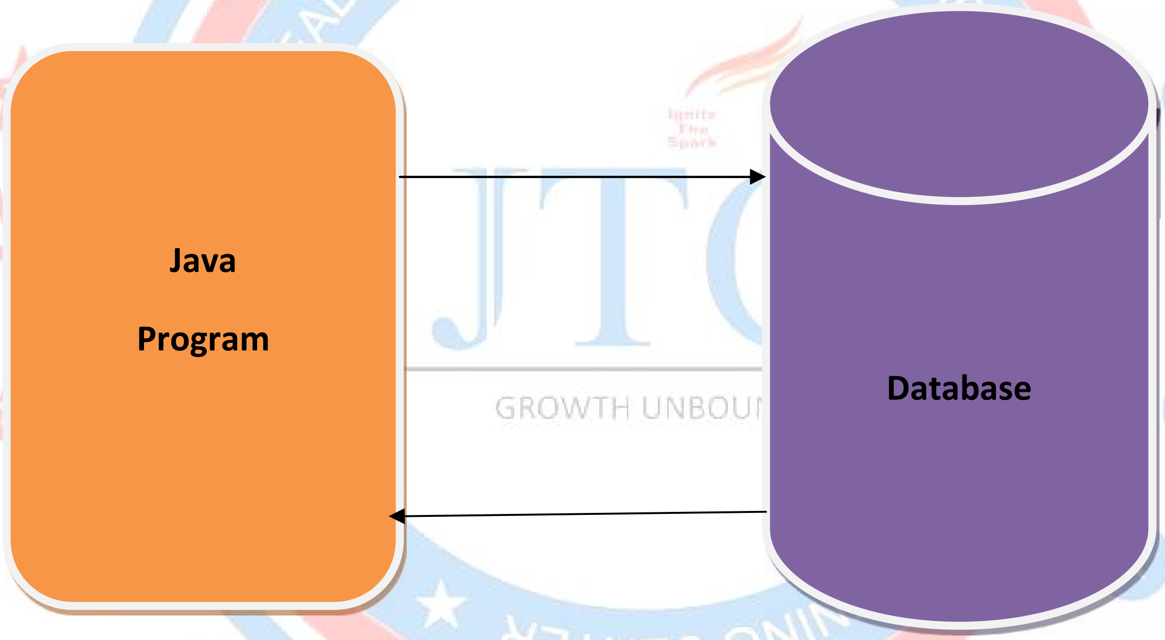
- **Public boolean execute(String sql)When you want to submit insert ,update,delete or select SQL Statements then use execute() method which returns the boolean value saying whether the ResultSet object is created or not(The SQL Statement is SELECT or not).**

  - if return value is true which means that SELECT SQL statement is submitted and ResultSet object is created.
    - Public ResultSet getResultSet()
  - If return value false which means that INSERT,UPDATE,or DELETE SQL statement is submitted and integer number is available which represent number of records inserted update or deleted.
    - Public int getUpdateCount()

- **Using the Single Statement Object , you can submit any type of SQL statement and any number of SQL statements.**
    - **ex:**

```
Statement st=con.createStatement();
 String sql1="insert ....";
 String sql2="update ....";
 String sql3="delete ....";
String sql4="select ....";
booleab b1=st.execute(sql1);
int x=st.executeUpdate(sql2);
int y=st.executeUpdate(sql3);
ResultSet rs=st.executeQuery(sql4);
```

- **When you submit the SQL Statement using Statement object then SQL Statement will be compiled and executed every time.**



- **Total time = req.time + compile time + exec time + res.time**
  = 5 ms+5 ms+5 ms+5 ms = 20 ms.
  1 SQL Stmt = 20 ms.
  100 times = 2000ms.
- **If you are providing dynamic values for the query then you need to use concatination operator, Formattor or StringBuffer etc to format the query.**
- **If you are providing the value that format is database dependent (May be Date) then you need to provide depending on Database.**

Jtc6.java

```
package jdbcType4Driver;

import java.sql.Connection;
import java.sql.Statement;
import java.util.Scanner;

public class Jtc6 {
public static void main(String arg[]){
Connection con=null;
Statement st=null;
try{
//con=JdbcUtil.getMySQLConnection();
con=JdbcUtil.getOraConnection();
Scanner sc=new Scanner(System.in);
System.out.println("Enter Id");
            int id=sc.nextInt();
            sc.nextLine();
System.out.println("Enter Name");
String name=sc.nextLine();
System.out.println("Enter Email:");
String email=sc.nextLine();
System.out.println("Enter Phone:");
String phone=sc.nextLine();
String qry=String.format("insert into students
values(%d,'%s','%s','%s')",id,name,email,phone)
;
System.out.println(qry);
st=con.createStatement();
int x=st.executeUpdate(qry);
if(x==1){
System.out.println("Record inserted
succesfully");
}else{
System.out.println("not inserted");

                                                }
                        }catch(Exception e){
                        e.printStackTrace();
            }finally{
            JdbcUtil.cleanup(st, con);                  }
            }

}
```

Jtc7.java

```
package jdbcType4Driver;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Scanner;

public class Jtc7 {
public static void main(String arg[]){
            Connection con=null;
            Statement st=null;
try{
//          con=JdbcUtil.getMySQLConnection();
            con=JdbcUtil.getOracleConnection();
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter Id");
            int id=sc.nextInt();
            sc.nextLine();
System.out.println("Enter Name");
String name=sc.nextLine();
System.out.println("Enter Email:");
String email=sc.nextLine();
System.out.println("Enter Phone:");
String phone=sc.nextLine();
String qry=String.format("select * from
students",id,name,email,phone);
System.out.println(qry);
ResultSet rs=st.executeQuery(qry);
            if(rs.next()){
            int id1=rs.getInt(1);
String name1=rs.getString(2);
String email1=rs.getString(3);
String phone1=rs.getString(4);
System.out.println(id1+"\t"+name1+"\t"+email1+"\t
"+phone1);

            }else{System.out.println("sorry,Student not
found");
                                    }catch(Exception e){
            e.printStackTrace();
                        }finally{JdbcUtil.cleanup(st, con);
                        }
            }
}
```

Jtc8.java	int id=rs.getInt(1);

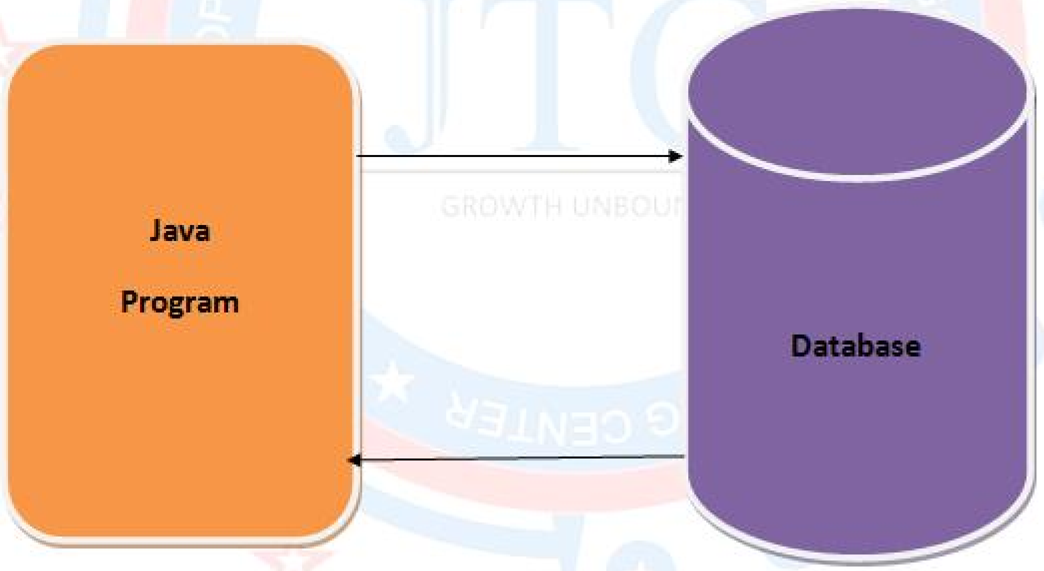
```
package jdbcType4Driver;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Scanner;

public class Jtc8 {
	public static void main(String ar[]){
		Connection con=null;
		Statement st=null;
		ResultSet rs=null;
		try{

	con=JdbcUtil.getOraConnection();
	Scanner sc=new Scanner(System.in);
System.out.println("Enter Query:");
	String qry=sc.nextLine();
	st=con.createStatement();
	boolean b1=st.execute(qry);
	if(b1){
	rs=st.getResultSet();
		if(rs.next()){
			do{
		String name=rs.getString(2);
		String email=rs.getString(3);
		String phone=rs.getString(4);
System.out.println(id+"\t"+name+"\t"+email+"\t"+p
hone);

		}while(rs.next());
					}
				}else{
		int x=st.getUpdateCount();
		System.out.println("Result:"+x);
					}

		}catch(Exception e){
				e.printStackTrace();
			}finally{
				JdbcUtil.cleanup(st, con);
			}

		}
}
```

===

2. PreparedStatement :

- PreparedStatement is an interface available in java.sql package and extends Statement interface.
- You can create the PreparedStatement using the following methods of Connection interface.
  - public PreparedStatement prepareStatement(sql)
  - public PreparedStatement prepareStatement(sql,int , int)
  - public PreparedStatement prepareStatement(sql,int ,int ,int)
- After creating the PreparedStatement object , you can call one of the following methods to submit the SQL Statement to Database.
  - public int executeUpdate()
  - public boolean execute()
  - public ResultSet executeQuery()
- Using the Single PreparedStatement Object , you can submit only one SQL statement.

ex:
```
String sql="insert ....";
PreparedStatement ps=con.prepareStatement(sql)
int x=ps.executeUpdate();
```



- When you submit the SQL Statement using PreparedStatement object then SQL Statement will be compiled only once first time and pre-compile SQL Statement will be executed every time.
- Total time = req.time + compile time + exec time + res.time
  = 5 ms+5 ms+5 ms+5 ms = 20 ms.
  First time -> 1 SQL Stmt = 20 ms.
  next onwards -> 5ms + 0 ms+ 5 ms +5 ms= 15 ms.
  101 times = 20 ms + 1500ms.=> 1520.
- PreparedStatement gives you the place holder mechanism for providing the data dynamic to the query.You need to use ? symbol for placeholder.
- To provide the value of place holder you need to invoke the following method depending the type of the value for place holder

```
public void setX(int paramIndex, X val)
        X can be Int,String,Long,Float,Date etc
```

- If you want to specify the date type value then create the object of java.sql.Date type and invoke the following method

```
public void setDate(int paramINdex,Date dt)

String sql="insert into jtcstudents values(?,?,?,?,?,?)";
ps=con.prepareStatement(sql);
ps.setInt(1, id);
ps.setString(2,nm);
ps.setString(3,eml);
ps.setLong(4,phn);
ps.setString(5,fee);
ps.setDate(6,dt);
```

Jtc9.java

```
package jdbcType4Driver;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Scanner;

public class Jtc9 {
        public static void main(String arg[]){
                Connection con=null;
                Statement st=null;
                PreparedStatement ps=null;
                try{
                //
        con=JdbcUtil.getMySQLConnection();

        con=JdbcUtil.getOraConnection();
                        Scanner sc=new
Scanner(System.in);
                        System.out.println("Enter
Id");
                        int id=sc.nextInt();
                        sc.nextLine();
                        System.out.println("Enter
Name");
                        String
name=sc.nextLine();
                        System.out.println("Enter
Email:");
                        String
email=sc.nextLine();
                        System.out.println("Enter
Phone:");
                        String
phone=sc.nextLine();
                        String
qry1=String.format("insert into students
values(?,?,?,?)");
System.out.println(qry1);
ps=con.prepareStatement(qry1);
                ps.setInt(1,id);
                ps.setString(2,name);
                ps.setString(3,email);
                ps.setString(4,phone);
        int x=ps.executeUpdate();
```

Jtc10.java

```
package jdbcType4Driver;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.Statement;
import java.util.Scanner;

public class Jtc10 {
        public static void main(String arg[]){
                Connection con=null;
                Statement st=null;
                CallableStatement cs=null;
                try{
                //
        con=JdbcUtil.getMySQLConnection();

        con=JdbcUtil.getOraConnection();
                        Scanner sc=new
Scanner(System.in);
                        System.out.println("Enter
Id");
                        int id=sc.nextInt();
                        sc.nextLine();
                        System.out.println("Enter
Name");
                        String name=sc.nextLine();
                        System.out.println("Enter
Email:");
                        String email=sc.nextLine();
                        System.out.println("Enter
Phone:");
                        String phone=sc.nextLine();

                                String
qry2=String.format("select * from students");

        ps=con.prepareStatement(qry2);
                        System.out.println(qry2);
                        ResultSet
rs=ps.executeQuery();
                        if(rs.next()){
                                int id1=rs.getInt(1);
                                String
name1=rs.getString(2);
                                String
email1=rs.getString(3);
```

```
            if(x==1){
System.out.println("Record inserted
succesfully");
                              }else{

            System.out.println("not inserted");

                                        }


                              }

                              String qry3=sc.nextLine();

            ps=con.prepareStatement(qry3);
                              boolean b1=ps.execute();
                              if(b1){

            rs=ps.getResultSet();
                                        if(rs.next()){
                                                  do{
                                                            int
id1=rs.getInt(1);

            String name1=rs.getString(2);

            String email1=rs.getString(3);

            String phone1=rs.getString(4);

            System.out.println(id1+"\t"+name1+"\t"
+email1+"\t"+phone1);

            }while(rs.next());
                                        }
                                        }else{
                                                  int
x1=ps.getUpdateCount();

            System.out.println("Result:"+x1);
                              }


                    }catch(Exception e){
                              e.printStackTrace();
                    }finally{
                              JdbcUtil.cleanup(st, con);
                    }
```

```
                                        String
phone1=rs.getString(4);

          System.out.println(id1+"\t"+name1+"\t"+ema
il1+"\t"+phone1);

                              }else{
System.out.println("sorry,Student not found");


                    }catch(Exception e){
                              e.printStackTrace();
                    }finally{
                              JdbcUtil.cleanup(st, con);
                    }


          }

}
```

```
        }

}
```

Jtc11.java

```
package jdbcType4Driver;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Scanner;

public class Jtc9 {
        public static void main(String arg[]){
                Connection con=null;
                Statement st=null;
                PreparedStatement ps=null;
                try{
                //
        con=JdbcUtil.getMySQLConnection();

        con=JdbcUtil.getOraConnection();
                        Scanner sc=new
Scanner(System.in);
                        System.out.println("Enter
Id");
                        int id=sc.nextInt();
                        sc.nextLine();
                        System.out.println("Enter
Name");
                        String
name=sc.nextLine();
                        System.out.println("Enter
Email:");
                        String
email=sc.nextLine();
                        System.out.println("Enter
Phone:");
                        String
phone=sc.nextLine();
ps=con.prepareStatement(qry3);
                        boolean
```

```
                        if(b1){

        rs=ps.getResultSet();
                                if(rs.next()){
                                        do{
                                                int
id1=rs.getInt(1);
                                                String
name1=rs.getString(2);
                                                String
email1=rs.getString(3);
                                                String
phone1=rs.getString(4);

        System.out.println(id1+"\t"+name1+"\t"+
email1+"\t"+phone1);

        }while(rs.next());
                                }
                                }else{
                                        int
x1=ps.getUpdateCount();

        System.out.println("Result:"+x1);
                        }
```

`b1=ps.execute();`	

CallableStatement :

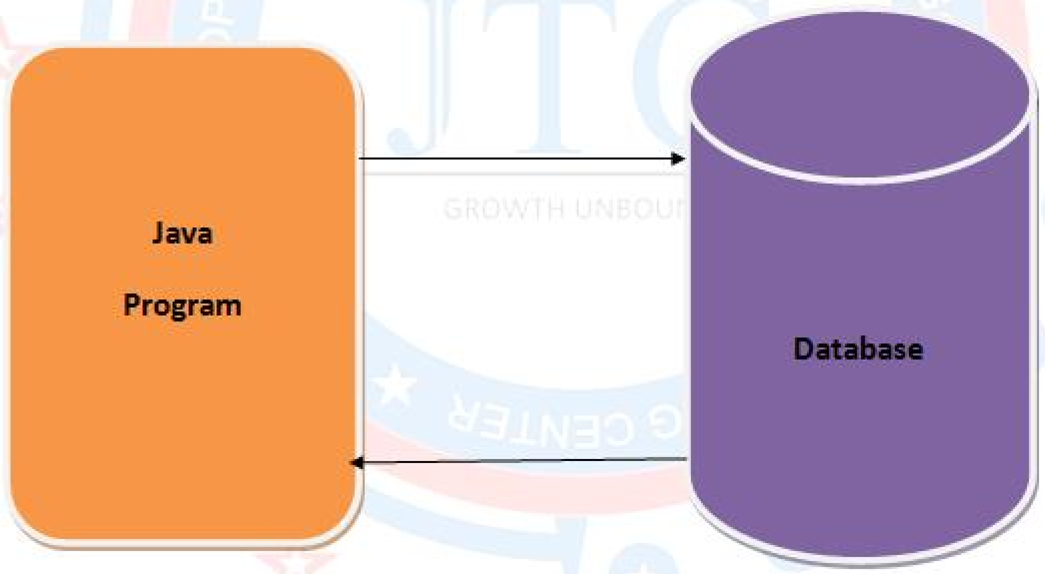
- **CallableStatement is an interface available in java.sql package and extends PreparedStatement interface.**
- **You can create the CallableStatement using the following methods of Connection interface.**
  - **CallableStatement prepareCall(String)**
  - **CallableStatement prepareCall(String,int , int)**

  - CallableStatement prepareCall(String,int ,int ,int)

- After creating the CallableStatement object , you can call one of the following methods to submit the SQL Statement to Database.
  - int executeUpdate()
  - boolean execute()
  - ResultSet executeQuery()
- CallableStatement is designed mainly to invoke the stored procedures running in the database.
- Stored procedure is pre-compiled procedure .i.e When you create the procedure then that procedure will be compiled and stored in database memory. When you make call to the procedure then that pre-compiled procedure will be executed directly.
- Using the Single CallableStatement Object , you can make a call to only one stored procedure.
  ex:

```
String sql="call p1(?,?)";
CallableStatement cs=con.prepareCall(sql)
cs.setInt(1,10);
cs.setInt(2,20);
int x=cs.executeUpdate();
```

- Use Stored Procedures when you want to run some logic in database.



with SP using CS.

- Total time = req.time + compile time + exec time + res.time
  = 5 ms+0 ms+20 ms+5 ms = 30 ms.101 times -> 3030ms.

with SQL using PS.

- Total time = req.time + compile time + exec time + res.time
  = 5 ms+0 ms+20 ms+5 ms = 30 ms.

One JDBC Program with 4 SQL.
4 SQL's -> 4 * 20 ms =80 ms. (With Statement)
-> 4 * 15 ms= 60 ms
101 times = 80 ms + 6000ms.=> 6080 ms.

- CallableStatement gives you the place holder mechanism.

A)jtcstudents table.
B)insertStudentInfo() Procedure

For MySQL:

```
delimiter $
create procedure insertStudentInfo(id int,nm varchar(20),eml varchar(50),phn long,fee float,dob date)
begin
insert into jtcstudents values(id,nm,eml,phn,fee,dob);
end;
$
delimiter ;
```

For Oracle:

```
create or replace procedure insertStudentInfo(id in int,nm in varchar,eml varchar,phn long,fee float,dob
date)
as
begin
insert into jtcstudents values(id,nm,eml,phn,fee,dob);
end;
/
```

Jtc12.java	Jtc13.java

package jdbcType4Driver;
 	
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.Statement;
import java.util.Scanner;
	package jdbcType4Driver;
	
	import java.sql.CallableStatement;
	import java.sql.Connection;
	import java.sql.Statement;
	import java.sql.Types;
	import java.util.Scanner;

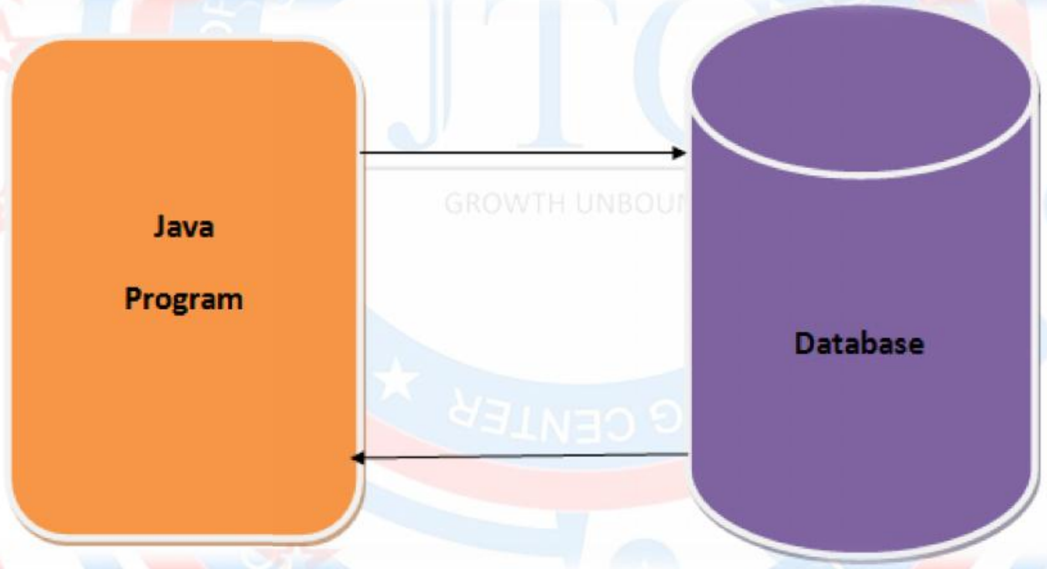
```
public class Jtc10 {
        public static void main(String arg[]){
                Connection con=null;
                Statement st=null;
                CallableStatement cs=null;
                try{
                //
        con=JdbcUtil.getMySQLConnection();

        con=JdbcUtil.getOraConnection();
                        Scanner sc=new
Scanner(System.in);

        System.out.println("Enter Id");
                        int id=sc.nextInt();
                        sc.nextLine();

        System.out.println("Enter Name");
                        String
name=sc.nextLine();

        System.out.println("Enter Email:");
                        String
email=sc.nextLine();

        System.out.println("Enter Phone:");
                        String
phone=sc.nextLine();


        cs=con.prepareCall("insert
students(?,?,?,?");
                        cs.setInt(1,id);
                        cs.setString(2,name);
                        cs.setString(3,email);
                        cs.setString(4,phone);
cs.execute();



        System.out.println("Record inserted
succesfully");

                        //
        System.out.println("not inserted");
```

```
public class CopyOfJtc101 {
        public static void main(String arg[]){
                Connection con=null;
                Statement st=null;
                CallableStatement cs=null;
                try{
                //
        con=JdbcUtil.getMySQLConnection();

        con=JdbcUtil.getOraConnection();
                        Scanner sc=new
Scanner(System.in);
                        System.out.println("Enter Id");
                        int id=sc.nextInt();
                        sc.nextLine();
                        System.out.println("Enter
Name");
                        String name=sc.nextLine();
                        System.out.println("Enter
Email:");
                        String email=sc.nextLine();
                        System.out.println("Enter
Phone:");
                        String phone=sc.nextLine();

                        cs=con.prepareCall("call
updateinfo(?,?,?");
                        cs.setInt(1,id);
                        cs.setString(3,email);

        cs.registerOutParameter(2,Types.VARCHAR);

        cs.registerOutParameter(3,Types.VARCHAR);
cs.execute();
String nm=cs.getString(name+"\t"+phone);
System.out.println("called Successfully");


        System.out.println("Record inserted
succesfully");

                        //        System.out.println("not
inserted");



                }catch(Exception e){
```

```
                }catch(Exception e){
                        e.printStackTrace();
                }finally{
                        JdbcUtil.cleanup(st, con);
                }

        }

}
```
	```
	                        e.printStackTrace();
	                }finally{
	                        JdbcUtil.cleanup(st, con);
	                }

	        }

	}
	```

Batch Updates :

- When you want to submit multiple types of SQL statements and No . of SQL statements to the Database at a time then use Batch Updates.



- Using Statement
  - For 1 SQL Statement=5ms+5ms+5ms+5ms=20ms
    For 100 SQL Statement=100*20=2000ms
- Using PreparedStatement
  - For 1SQL Statemen =5ms+0ms+5ms+5ms=15ms
    For 100 SQL Statements=100*15=1500ms
- In the above two cases, ou are trying to submit 100 SQL Statement. For submitting 100 SQL Statement, you need to Communication with the database 100 times.this increases number of round trips between your application and database which damages the application performance.
- Batch updates allows you to submit multiple SQL Statement to the databse at the time

Using Batch Update:

For 100 SQL Statement=5ms+100*5ms+100*5ms+5ms=1010ms.

- **Using Batch Updates with Statement:**
  - **You can submit multiple types of SQL Statements.**
  - **You can submit multiple SQL Statements.**
  - **You can reduce number of round trips between your application and database.**
  - **Which improve the application performance.**
  - **You can use insert, update and delete statement only.**
  - **You can not use SELECT statement.**
  - **Use the following methods of statement interface to implement Batch Updates.**
    - **Void addBatch(String)**
    - **Int[] executeBatch()**
    - **Void clearBatch()**
- **Using Batch Updates with Prepared Statement:**
  - **You can submit only siggle type of SQL Statement.**
  - **You can submit multiple SQL Statements.**
  - **You can reduce number of round trips between your application and database which improve the application performance.**
  - **You can use insert, update and delete statements only .**
  - **You can not use SELECT statement.**
  - **Use the following methods of preparedStaetement interface to implement Batch Updates.**
    - **Void addBatch()**
    - **Int[] executeBatch();**
    - **Void clearParameters()**

Program……………………………code…………..

With Statement

To add the Query in Batch

```
  public void addBatch(String sql)
```

To Clear the batches

```
  public void clearBatch()
```

To submit the Queries as batch

```
 public int[] executeBatch()

String sql1="insert into customers..";
String sql2="update customers ..";
String sql3="delete from customers ..";

st=con.createStatement();
st.addBatch(sql1);
st.addBatch(sql2);
st.addBatch(sql3);
```

```
int x[]=st.executeBatch();
```

With PreparedStatement

Jtc15.java

To add the Query in Batch

```
  public void addBatch()
```

To submit the Queries as batch

```
 public int[] executeBatch()
package com.jtc.b13.jdbc;
import java.sql.*;
import com.jtc.b13.jdbc.util.JDBCUtil;
public class Jtc10a {
public static void main(String[] args) {
Connection con = null;
PreparedStatement ps = null;
try {
con = JDBCUtil.getOracleConnection();
ps = con.prepareStatement("insert into jtcstudents (id,name,email) values(?,?,?)");
for (int i = 1; i <= 5; i++) {
int id = 500 + i;
String name = "Siva " + i;
String email = "siva" + i + "@jtc.org";
ps.setInt(1, id);
ps.setString(2, name);
ps.setString(3, email);
ps.addBatch();
}
int res[] = ps.executeBatch();
for (int i = 0; i < res.length; i++) {
System.out.println("Res :" + res[i]);
}
} catch (SQLException e) {
e.printStackTrace();
} finally {
JDBCUtil.cleanup(ps, con);
}}}
```

Can we submit select statement using batch update?
NO, because if second result set will be available then first result set object will not be used.

Can we use batch update with CallableStatement ?
NO

how to get the result from the CallableStatement if you invke any function?

call p1(?,?);
? = f1(?,?)

For MySQL you can specify the URL

jdbc:mysql://localhost:3333/acb13db

jdbc:mysql:///acb13db
default port 3333 & host localhost
jdbc:mysql://localhost/acb13db
default port 3333
jdbc:mysql://3333/acb13db
default host localhost
jdbc:mysql:///?
default port 3333 & host localhost
NO DATABASE SELECTED

ResultSet

- ResultSet is an interface available in java.sql package.
- Subclass of ResultSet interface is provided by Driver vendor.
- ResultSet object contains the records returned by select statements.
- resultSet object can be created by using the following method.
  - ResultSet rs=st.executeQuery("select ….");//Statement interface
  - ResultSet rs=st.executeQuery();//PreparedStatement.
- Assume that there is table called jtcstudents with 4 column –sid,sname,email,phone.
- Case1:
  - Strig sql="select * from Jtcstudents";
    Rs=st.executeQuery(sql);

Sid	Sname	Emai	Phone
001	Som	som@jtc,com	4321
002	Jtcsom	Jtcsom@gmail.com	87665432
003	Prakash	prakas@jtc.ocm	8765432

- Case2:
  String sql="select sid,phone from jtcstudents";
  Rs=st.executeQuery(sql);

Sid	Phone
001	00998765433
002	987654356789

003	09876543567987
004	98765434567
005	98765434567

- When ResultSet object is created then initially ResultSet cursor points to before to the first record.
- You can use the next() method to move the ResulgtSet pointer in the forward direction

```
Public boolean next(){
Check whether next record is available or not.
If Next record is avaialble then Moves the pointer to next record
return tru;
If Next record is not available then
        Moves the pointer to next position
    Return false;
}
```

- You can use the previous() method to move the ResultSet pointer in the reverse direction.

```
Public boolean previous(){
Check whether next record is available or not.
If Next record is avaialble then Moves the pointer to next record
return tru;
If Next record is not available then
        Moves the pointer to next position
    Return false;
}
```

- When ResultSet pointer is pointing one record then you can access the data of various column using getXXX() methods.

```
Public int getInt(int columnindex)
Public int getInt(String columnindex)
Public String getString(int columnindex)
Public String getString(String columnName) etc
```

Type Of ResulteSets:

- Depending on the ResultSet cursor movent, you devide the ResultSet into 2 Types.
  - Forward-Only ResultSets
  - Scrollable ResultSets

Forward-Only ResultSets:

- When Result is forward-only then:
  - Pointer can be moved in the forward direction only.
  - Pointer can be moved only once
  - Pointer can be moved in sequential order only.
- By default, ResultSets are forward only.

`St=con.createStatement();`
`Rs=st.executeQuery(“select…”);`
	Ps=con.preparedStatement(“select…..”);
	Rs=ps.executeQuery();

- You can explicity specify the ResultSets as forward only as follows:

```
st=con.createStatement(ResultSet.TYPE_FORWARD_ONLY, 0);
rs=st.executeQuery("select.....");

ps=con.prepareStatement("select...",ResultSet.TYPE_FORWARD_ONLY,0);
rs=ps.executeQuery();
```

- **You can use the following methods on forward only ResultSet.**
  Public boolean next()
  Public void close()
  Public boolean isBeforeFirst()
  Public boolean is AfterLast()
  Public boolean is First()
  Public boolean isLast()
  Public int getRow()
  Public xxx getXXX(int)
  Public XXX getXXX(String)
  Etc

Jtc17.java

```
package jdbcType4Driver;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;

public class Jtc13 {
        public static void main(String arg[]) {
                Connection con = null;
                Statement st = null;
                ResultSet rs = null;
                try {
                        con =
JdbcUtil.getOraConnection();
                        st =
con.createStatement(ResultSet.TYPE_SCROLL
_SENSITIVE,

        ResultSet.CONCUR_UPDATABLE);
                        rs =
st.executeQuery("select * from Students");

        System.out.println("**FORWORD
DIRECTION**");
                        while (rs.next()) {

        System.out.println(rs.getInt(1) + "\t" +
rs.getString(2) + "\t"
System.out.println("1st record**");
                        rs.first();

        System.out.println(rs.getInt(1) + "\t" +
rs.getString(2) + "\t"
                                +
rs.getString(3) + "\t" + rs.getString(4));
                        }
                        System.out.println("**4th
Record**");
                        rs.absolute(4);

        System.out.println(rs.getInt(1) + "\t" +
rs.getString(2) + "\t"
                                +
rs.getString(3) + "\t" + rs.getString(4));
                        System.out.println("**From
4th next 2ndRecord**");
                        rs.relative(2);

        System.out.println(rs.getInt(1) + "\t" +
rs.getString(2) + "\t"
                                +
rs.getString(3) + "\t" + rs.getString(4));

                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
```

+ rs.getString(3) + "\t" + rs.getString(4));
 	
 System.out.println("**REVERSE DIRECTION**");
 while (rs.previous()) {
 System.out.println(rs.getInt(1) + "\t" + rs.getString(2)+ "\t" + rs.getString(3) + "\t" + rs.getString(4));
	}

Scrollable ResultSets

- **When ResultSet is scrollable then:**
  - **Pointer can be moved in both forward and reverse direction.**
  - **Pointer cn be moved multiple times.**
  - **Pointer can be moved in random order.**
  - **Pointer can be moved in random order.**
- **By Default, ResultSets are not scrollable.**
- **You can explicty specify the ResultSets as scrollable ad follows:**

```
st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, 0);
rs=st.executeQuery("select.....");

ps=con.prepareStatement("select...",ResultSet.TYPE_SCROLL_SENSITIVE,0);
        rs=ps.executeQuery();
```

- You can use the following methods on forward only ResultSet.

public boolean next()
public void close()
public boolean isBeforeFirst()
public boolean is AfterLast()
public boolean is First()
public boolean isLast()
public int getRow()
public xxx getXXX(int)
public XXX getXXX(String)
public boolean beeforeFirst()
public boolean afterLast()
public boolean First()
public boolean Last()
public boolean First()
public boolean Last()
public boolean abslute()
public boolean relative()
public boolean previous()

Types of ResultSets:

- **Depending on the ResultSet Updation, you can divide the ResultSets into 2 types.**

- o **Read-Only ResultSets.**
- o **Updatable ResultSets or Dynamic ResultSets.**

Read-Only ResultSets:

- **When ResultSet is read-only then you can just access the data from ResultSet object by calling getter methods and you can not do any updations on the ResultSet object.**
- **Read-Only ResultSet is also called as static ResultSet.**
- **By Default, Resultset is are read-only.**

`st=con.createStatement();`
`rs=st.executeQuery("select...");`
	`ps=con.prepareStatement("select...");`
	`rs=ps.executeQuery();`

- **You can explicty specify the ResultSet as read-only as follows:**

```
st=con.createStatement(ResultSet.TYPE_FORWARD_ONLY,ResultSet.CONCUR_READ_ONLY);
            rs=st.executeQuery("select.....");

      ps=con.prepareStatement("select...",ResultSet.TYPE_FORWARD_ONLY,ResultSet.CONCUR_REA
D_ONLY);
            rs=ps.executeQuery();
```

- **You can use the following methods on Read-only ResultSet.**
  Public boolean next()
  Public void close()
  Public boolean isBeforeFirst()
  Public boolean is AfterLast()
  Public boolean is First()
  Public boolean isLast()
  Public int getRow()
  Public xxx getXXX(int)
  Public XXX getXXX(String)
  Etc

1)Forward-Only ResultSets :

When ResultSet is forward-only then you can move pointer only on the forward direction and only once.

Updatable ResultSets:

- **When ResultSet is Updatable then you can do the following operations on ResultSet object**
  - o **get the data from ResultSet**
  - o **insert records into ResultSet**
  - o **update the records of ResultSet**
  - o **delete the records from ResultSet.**
- **When ResultSet is Updatable then it must be scrollable.**

- **By deafult ResultSets are not Updatable.**
- **You can explicty specify the ResultSet as Updatable as Follows:**
  st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCURR_UPDATABLE);
  rs=st.executeQuery(sql)
  ps=con.prepareStatement(sql,ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCURR_UPDATABLE);
  rs=ps.executeQuery()

  In this code ,ResultSet created is the Scrollable and Updatable resultset.

- You can use the folowing methods onread-only ResultSet.
- ResultSet will not become updatable even when you use ResultSet .CONCUR_UPDATEABLE. IN the following.
  - When SELECT statement uses Aggregate functions.
  - When SELECT statement uses Aggregate function.
  - When SELECT statement use* inserted of colunm names.(in Oracle Only)

Jtc18.java

```
package jdbcType4Driver;


import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;

public class Jtc13 {
	public static void main(String arg[]) {
		Connection con = null;
		Statement st = null;
		ResultSet rs = null;
		PreparedStatement ps=null;
		try {
			con = JdbcUtil.getOraConnection();
			st = con.prepareStatement( null,ResultSet.TYPE_SCROLL_SENSITIVE,
					ResultSet.CONCUR_UPDATABLE);
			rs = st.executeQuery("select * from Students");
			System.out.println("**FORWORD DIRECTION**");
			while (rs.next()) {
				System.out.println(rs.getInt(1) + "\t" + rs.getString(2) + "\t"
						+ rs.getString(3) + "\t" + rs.getString(4));
				System.out.println("**REVERSE DIRECTION**");
				while (rs.previous()) {
					System.out.println(rs.getInt(1) + "\t" + rs.getString(2)
							+ "\t" + rs.getString(3) + "\t" +
rs.getString(4));
				}
```

```
                    System.out.println("1st record**");
                    rs.first();
                    System.out.println(rs.getInt(1) + "\t" + rs.getString(2) + "\t"
                              + rs.getString(3) + "\t" + rs.getString(4));
               }
               System.out.println("**4th Record**");
               rs.absolute(4);
               System.out.println(rs.getInt(1) + "\t" + rs.getString(2) + "\t"
                         + rs.getString(3) + "\t" + rs.getString(4));
               System.out.println("**From 4th next 2ndRecord**");
               rs.relative(2);
               System.out.println(rs.getInt(1) + "\t" + rs.getString(2) + "\t"
                         + rs.getString(3) + "\t" + rs.getString(4));

          } catch (Exception e) {
               e.printStacTrace();
          }
     }

}
```

CONSTANTS Defined in ResultSet interface

ResultSet type	**ResultSet Cuncurrency**	**ResultSet Holdability**
TYPE_FORWARD_ONLY 1003 TYPE_SCROLL_INSENSITIVE 1004 TYPE_SCROLL_SENSITIVE 1005	CONCURR_READ_ONLY 1007 CONCURR_UPDATABLE 1008	HOLD_CRSORS_OVER_COMMA CLOSE_CURSORS_AT_COMMIT

You can following mwthod of statement interface to find the ResultSet Type
    Public int getResultSetType()
You can use the following method pf statement interface to find the ResultSet concurrency
    Public int getResultSet Concurrency()
You can use the following method pf statement interface to find the ResultSet Holdablity.
    Public int getResultSetHoldability()
You can use the following method pf statement interface to create the statement object
    Public statement createStatement()
    Public statement createStatement(int rsType,int rsConcurrency)
    Public statement createStatement(int rstype,int rsConcurrency, int rsHoldability)

You can use the following method pf statement interface to create the PreparedStatement.

    Public statement createPreparedStatement()
    Public statement createPreparedStatement(int rsType,int rsConcurrency)

Public statement createPreparedStatement(int rstype,int rsConcurrency, int rsHoldability)

Q) in JDBC API or in java.sql package, sun has provided more interface like connection,statement,PreparedStatement,CallableStatement,ResultSet etc, How instance will be created?
ANS: For these interfaces, various Driver vendors has implemented various sub classess vendor specific sub classes will be instanciated depending on the JDBC driver.

MYSQL vendor has provided following are subclasses in com.mysql.jdbc.package.
JDBC4Connection
StatementImpl
JDBC4PreparedStatement
JDBC4CallableStatement
Oracle vendor has provided following are subclasses in oracle.jdbc.driver package:
T4connection
T4Cstatement
T4CpreparedStatement
T4TcallableStatement

DatabaseMetaData
- **DatabaseMetaData is an an interface available in java.sql package.**
- **Subclasses of databaseMetaData interface is provided by Driver vendor.**
- **DatabaseMetadat is used to get the information about your dtatabase i.e you can find whether database is supporting the required features or not.**
- **You can use the following method of connection interface to get the databaseMetaData object.**
  - **Public dtatabaseMetaData getMetaData()**

ResultSetMetadata
- ResultSetMetaData is an interface available in java.sql package.
- Subclasses of ResultSetMetadata interface is provided by Driver vendor.
- ResultSetMetatData is used to get the information about your ResultSet object.
- You can use the following of ResultSet interface to get the ResultSetMetaData. Object.
  - Public ResultSetMetaData getMetaData()

CURD Operation

Book.java package com.jtcJdbc;	BookService.java package com.jtcJdbc;

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class Book {
           String bid;
           String bname;
           String Author;
 String publication;
          double cost;
          int edition;

          public Book(){
          }


          public Book(String bname, String
Author, String publication, double cost,
                    int edition) {
                    super();
                    this.bname = bname;
                    this.Author = Author;
                    this.publication =
publication;
                    this.cost = cost;
                    this.edition = edition;
          }



          public String getBid() {
                    return bid;
          }


          public void setBid(String bid) {
                    this.bid = bid;
          }
```

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class BookService {

          public boolean verifyUser(String
un,String pw){
                    boolean valid=false;
                    Connection con=null;
                    ResultSet rs=null;
                    PreparedStatement ps=null;
                    try{

          con=JdbcUtil.getMySQLConnection(
);

          //con=JdbcUtil.getOraConnection();

          ps=con.prepareStatement("select *
from user_table_1 where username=? and
password=?");
                              ps.setString(1,un);
                              ps.setString(2,pw);
                              rs=ps.executeQuery();
                              if(rs.next()){
                                        valid=true;
                              }
                    }catch(Exception e){
                              e.printStackTrace();
                    }
                    return valid;

          }
          public int deleteBook(String bid){
```

```
	public String getBname() {
		return bname;
	}


	public void setBname(String
bname) {
		this.bname = bname;
	}


	public String getAuthor() {
		return Author;
	}


	public void setAuthor(String
Author) {
		this.Author = Author;
	}


	public String getPublication() {
		return publication;
	}


	public void setPublication(String
publication) {
		this.publication =
publication;
	}


	public double getCost() {
		return cost;
```

```
		Connection con=null;
		ResultSet rs=null;
		PreparedStatement ps=null;
		int x=0;
		try{

	con=JdbcUtil.getMySQLConnection(
);

	ps=con.prepareStatement("delete
from jtcbooks where bid=?");
		ps.setString(1,bid);
		x=ps.executeUpdate();
		}catch(Exception e){
			e.printStackTrace();
		}
		return x;

	}
	public  boolean addBook(Book bo){
		boolean added=false;
		Connection con=null;
		ResultSet rs=null;
		PreparedStatement ps=null;
		try{

	con=JdbcUtil.getMySQLConnection(
);

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

	ps.setString(1,bo.getNewBookId());

	ps.setString(2,bo.getBname());

	ps.setString(3,bo.getAuthor());
	}


	public void setCost(double cost) {
		this.cost = cost;
	}


	public int getEdition() {
		return edition;
	}


	public void setEdition(int edition) {
		this.edition = edition;
	}


	public String toString(){
		return
""+bid+"\t"+bname+"\t"+Author+"\t"+pu
blication+"\t"+cost+"\t"+edition;
	}


	public String getNewBookId() {
		Connection con=null;
		PreparedStatement ps=null;
		ResultSet rs=null;
		String bid="B-01";
		try{

	con=JdbcUtil.getMySQLConnectio
n();

	ps=con.prepareStatement("select
max( ");
			rs=ps.executeQuery();
```

```
	ps.setString(4,bo.getPublication());

	ps.setDouble(5,bo.getCost());

	ps.setInt(6,bo.getEdition());
			ps.executeUpdate();
			added=true;
		}catch(Exception e){
			e.printStackTrace();
		}
		return added;

	}
	public int updateBook(Book bo){
		int x=0;
		Connection con=null;
		ResultSet rs=null;
		PreparedStatement ps=null;
		try{

	con=JdbcUtil.getMySQLConnection(
);

	ps=con.prepareStatement("update
jtcbooks set
bname=?,Author=?,publication=?,cost=?,edi
tion=? where bid=?");

	ps.setString(1,bo.getBname());

	ps.setString(2,bo.getAuthor());

	ps.setString(3,bo.getPublication());

	ps.setInt(5,bo.getEdition());

	ps.setString(6,bo.getBid());
			x=ps.executeUpdate();
```

```
                        if(rs.next()){

        bid=rs.getString(1);
                                if(bid!=null){
                                        String
id=bid.substring(2);
                                        int
x=Integer.parseInt(id);
                                        x++;
                                        if(x<10)

        bid="B-0"+x;

        else

        bid="B-"+x;
                                }else{
                                        bid="B-
01";
                                        }
                                }

                }catch(Exception e){
                        e.printStackTrace();
                }
                return bid;
        }
}
```

Jtc21.java

```
package com.jtcJdbc;

import java.util.List;

public class CurdBook {
        //01202544333
        public static void main(String
ar[]){
                BookService bs=new
BookService();
                //Verify User
```

```
                }catch(Exception e){
                        e.printStackTrace();
                }
                return x;

        }
        public Book getBookByBid(String
bid){
                Book bo=null;
                Connection con=null;
                ResultSet rs=null;
                PreparedStatement ps=null;
                try{

        con=JdbcUtil.getMySQLConnection(
);

        ps=con.prepareStatement("select *
from jtcbooks where bid=?");
                        ps.setString(1,bid);
                        rs=ps.executeQuery();
                        if(rs.next()){

bo=getBookFromResultSet(rs);

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

                return bo;

        }
        public List<Book>
getBooksByBname(String bname){
                List<Book> al=new
```

```
		boolean valid=bs.verifyUser("som","jtcindia");
		if(valid){
	System.out.println("Login Success ! Redirecting to home page");
		}else{
		System.out.println("Login Success ! try Again");
			}
	//Adding the book
		Book b=new Book("Java","som","JTC",250,4);
		boolean res=bs.addBook(b);
		if(res){
	System.out.println("Book added successfully");
		}else{
	System.out.println("Error while adding book info");
		}
		//update Book
		Book bk=new Book("Jdbc","Sompraksh","Sp",250,5);
		bk.setBid("B-02");
		int a=bs.updateBook(bk);
		System.out.println("Book update:"+a);

	//Delete book
	int c=bs.deleteBook("B-01");
	System.out.println("Book Deleted:"+c);
	//Accessing Book By Bid
	System.out.println("**Book By Bid");
	Book bo=bs.getBookByBid("B-01");
	System.out.println(bo);
	//Accessing Book By Bname
	System.out.println("**Book By Bname");
	List<Book> list=bs.getBooksByBname("java");
	for(Book b1:list){
		System.out.println(b1);
	}
	//Accessing Book by Author
	System.out.println("**Book By Author");
	List<Book> list1=bs.getBooksByAuthor("som");
```

```
ArrayList<Book>();
		Connection con=null;
		ResultSet rs=null;
		PreparedStatement ps=null;
		try{

	con=JdbcUtil.getMySQLConnection();

	ps=con.prepareStatement("select * from jtcbooks where bname=?");
			ps.setString(1,bname);
			rs=ps.executeQuery();
			while(rs.next()){
				Book bo=getBookFromResultSet(rs);
				al.add(bo);
			}
		}catch(Exception e){
			e.printStackTrace();
		}
		return al;

	}
	public List<Book> getBooksByAuthor(String Author){
		List<Book> al=new ArrayList<Book>();
		Connection con=null;
		ResultSet rs=null;
		PreparedStatement ps=null;
		try{

	con=JdbcUtil.getMySQLConnection();

	ps=con.prepareStatement("select * from jtcbooks where Author=?");
```

```
        for(Book b1:list1){
                System.out.println(b1);
        }
        //Accessing Book By cost
        System.out.println("Book By Cost");
        List<Book>
list2=bs.getBooksByCost(250);
        for(Book b1:list2){
                System.out.println(b1);
        }
        //Accessing All Books
        System.out.println("All Books");
        List<Book> list3=bs.getAllBooks();
        for(Book b1:list3){
                System.out.println(b1);
        }

}
}
```

```
                        ps.setString(1,Author);
                        rs=ps.executeQuery();
                        while(rs.next()){
                                Book
bo=getBookFromResultSet(rs);
                                al.add(bo);
                        }
                }catch(Exception e){
                        e.printStackTrace();
                }
                return al;
        }
        public List<Book>
getBooksByCost(double cost){
                List<Book> al=new
ArrayList<Book>();
                Connection con=null;
                ResultSet rs=null;
                PreparedStatement ps=null;
                try{

        con=JdbcUtil.getMySQLConnection(
);

        ps=con.prepareStatement("select *
from jtcbooks where cost=?");
                        ps.setDouble(1,cost);
                        rs=ps.executeQuery();
                        while(rs.next()){
                                Book
bo=getBookFromResultSet(rs);
                                al.add(bo);
                        }
                }catch(Exception e){
                        e.printStackTrace();
                }
                return al;
        }
```

```
	public List<Book> getAllBooks(){
		List<Book> al=new
ArrayList<Book>();
		Connection con=null;
		ResultSet rs=null;
		PreparedStatement ps=null;
		try{

	con=JdbcUtil.getMySQLConnection(
);

	ps=con.prepareStatement("select *
from jtcbooks");
			rs=ps.executeQuery();
			while(rs.next()){
				Book
bo=getBookFromResultSet(rs);
				al.add(bo);
			}
		}catch(Exception e){
			e.printStackTrace();
		}
		return al;

	}
	private Book
getBookFromResultSet(ResultSet rs) throws
SQLException {
		Book bo=new Book();
		bo.setBid(rs.getString(1));
		bo.setBname(rs.getString(2));
		bo.setAuthor(rs.getString(3));

	bo.setPublication(rs.getString(4));
		bo.setCost(rs.getDouble(5));
		bo.setEdition(rs.getInt(6));
		return bo;
	}
```

	}

RowSets

- Rowset is an interface available in javax.sql package.
- RowSets interface is extending ResultSet interface.
- RowSets interface implemetion classes are provided by java vendor.
- RowSets functionality is similar to RowSet.

ResultSet	**RowSet**

ResultSet object can be created as follows:
Con=D.M.getConnection(url,un,pw);
St=con.createStatement();
Rs=st.exceuteQuer(Sql);
	RowSet object can be created as follows:
	RowSet jrs=new JdbcRowSetIml();
	Jrs.setUrl(url);
	Jrs.setUseranem(un); jrs.seyPassword(pw);
	Jrs.setCommand(sql);
	Jrs.exceute();

By default, ResultSets are forward-only and read only.	By Default, RowSets are scrollable and updatable.
ResultSet objects are connection oriented i.e you can access the ResultSet data as long as connection is available. Once Connection is closed, ResulteSet also will be closed automatically.	RowSets are connection less object i.e you can access the RowSet data without Connection.
ResultSet objects are not realizable for serialization.	RowSet objects are elizible for Serialization.

- Following are subtypes of RowSet interface:
  - JdbcRowSet interface.
  - CachedRowSet interface.
  - WebRowSet interface.
  - JoinRowSet interface.
- <u>**Types of RowSets:**</u>
  - Connected Rowsets

- Disconnected RowSets

Connected RowSets:

- Connected RowSets are like ResultSets i.e Connected RowSets need the Connection as long as you are accessing the RowSet data.
- You cannot seriate Connected RowSets.
- JdbcRowSet is connected RowSet.

Disconnected RowSets:

- Disconnected RowSets are not like ResultSets i.e Disconnected RowSets do not need the connection while you access the RowSet data.
- You can serialize disconnected RowSets.
- JdbcRowSet is disconnected RowSet.

```
Jtc22.java
package com.jtcindia.jdbc;

import java.io.FileOutputStream;

import java.io.ObjectOutputStream;
import java.sql.Connection;

import javax.sql.RowSet;

public class Jtc22 {
	public static void main(String arg[]){
		try{

	Class.forName("com.mysql.jdbc.Driver");
			RowSet rs=new
JdbcRowSetImpl();

	rs.setUrl("jdbc:mysql://localhost:3306/my
sql");
			rs.setUsername("root");
			rs.setPassword("root");
//			Connection
con=JdbcUtil.getMySQLConnection();
//			RowSet rs=new
JdbcRowSetImpl();
			rs.setCommand("select
sid,sname,email,phone from jtcstudents");
			rs.execute();
```

```
Jtc23.java
package com.jtcindia.jdbc;

import java.io.FileOutputStream;

import java.io.ObjectOutputStream;
import java.sql.Connection;

import javax.sql.RowSet;

public class Jtc23 {
	public static void main(String arg[]){
		try {

	Class.forName("oracle.jdbc.driver.OracleD
river");
			RowSet rs = new
CachedRowSetImpl();

	rs.setUrl("jdbc:oracle:thin:@localhost:152
1:XE");
			rs.setUsername("system");

	rs.setPassword("somsree");
			rs.setCommand("select
id,name,email,phone from jtcstudents");
			rs.execute();
			while (rs.next()) {
```

while(rs.next()){
System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+"\t"+rs.getString(4));
}
rs.absolute(2);
rs.updateString(2,"Sompraksh");
rs.updateString(3,"Sompraksh@jtc.com");
rs.updateRow();
System.out.println("Updated");
rs.beforeFirst();
System.out.println("Serializing JdbcRowSet");
FileOutputStream fos=new FileOutputStream("D:\\rowset.txt");
ObjectOutputStream oos=new ObjectOutputStream(fos);
oos.writeObject(rs);
System.out.println("Serialized");
}catch(Exception e){
e.printStackTrace();
}
}
}
	System.out.println(rs.getInt(1) + "\t" + rs.getString(2) + "\t"
	+ rs.getString(3) + "\t" + rs.getLong(4));
	}
	System.out.println("\n-- Reverse Order --");
	while (rs.previous()) {
	System.out.println(rs.getInt(1) + "\t" + rs.getString(2) + "\t"
	+ rs.getString(3) + "\t" + rs.getLong(4));
	}
	System.out.println("--absolute(3)--");
	rs.absolute(3);
	System.out.println(rs.getInt(1) + "\t" + rs.getString(2) + "\t"
	+ rs.getString(3) + "\t" + rs.getLong(4));
	rs.updateString(2, "Som");
	rs.updateString(3, "som@jtc.org");
	rs.updateRow();
	System.out.println("--Updated --");
	System.out.println("Serializing the CachedRowSetImpl --");
	FileOutputStream fos = new FileOutputStream("rs.txt");
	ObjectOutputStream oos = new ObjectOutputStream(fos);
	oos.writeObject(rs);
	System.out.println("Object Serialized");
	} catch (Exception e) {
	e.printStackTrace();
	}}}

```
Jtc24.java
package com.jtcindia.jdbc;
```

```
import java.io.FileOutputStream;

import java.io.ObjectOutputStream;
import java.sql.Connection;

import javax.sql.RowSet;

public class Jtc24 {
	public static void main(String arg[]){
		try {
			System.out.println("Deserializing the CachedRowSetImpl --");
			FileInputStream fis = new FileInputStream("rs.txt");
			ObjectInputStream ois = new ObjectInputStream(fis);
			Object obj = ois.readObject();
			System.out.println(obj);
			RowSet rs=(RowSet)obj;
			while(rs.next()){

	System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+"\t"+rs.getLong(4));
			}
			System.out.println("Object Serialized");
			} catch (Exception e) {
			e.printStackTrace();
			}}}
```

How the object of CachedRowSetImpl is being serialized if connection is not Serializable?

After accessing the data, connection will be closed.

What are the new feature in JDBC 4.0 ?
Will be discussed
How to read the data from Excel file using JDBC?
Steps Using Type I Driver

While Configuring the DSN in ODBC
- Select Driver do Microsoft Excel
- Click on finish
- Provide the DSN :STUDXLSPS
- Select the Version
- Click on Select Workbook
  - Select the Drive
  - Select the Directory
  - Select the Excel File
  - Uncheck the Read Only Check Box

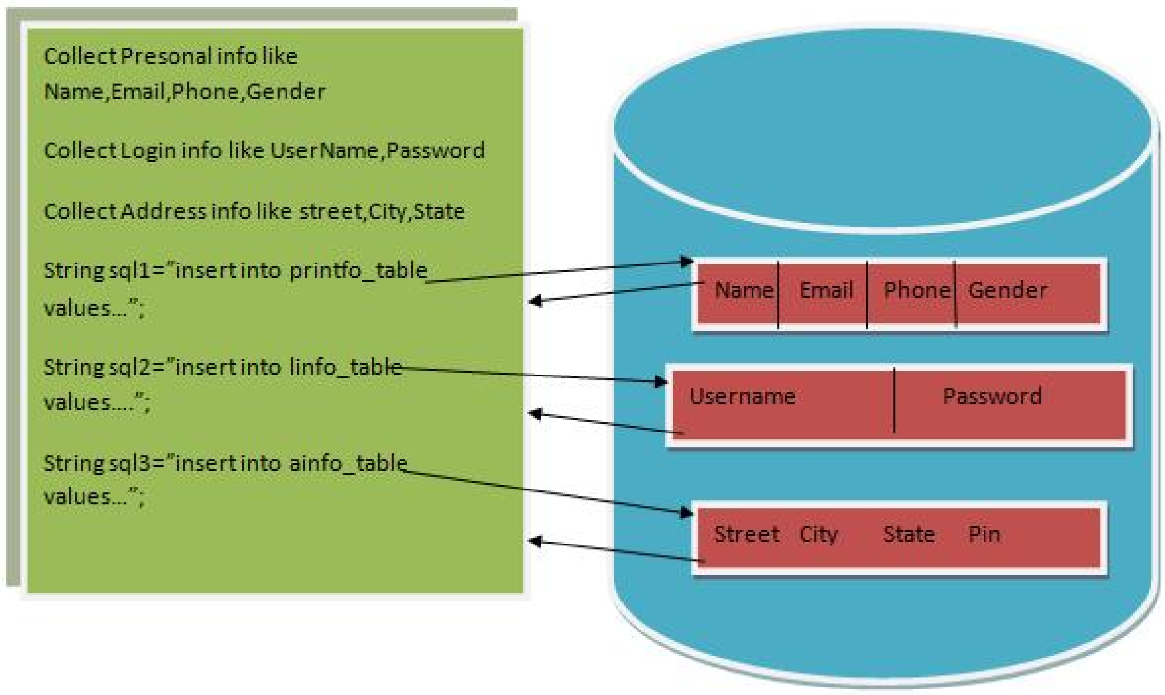
Click on OK Button
Click on OK Button
Click on OK Button

```
import java.sql.*;
public class ExcelReadTest {
public static void main(String[] args) {
Connection con = null;
Statement st = null;
ResultSet rs = null;
try {
con = DriverManager.getConnection("jdbc:odbc:STUDXLSPS");
st = con.createStatement();
rs = st.executeQuery("select * from [Sheet1$]");
while (rs.next()) {
System.out.println(rs.getInt(1) + "\t" + rs.getString(2) + "\t"
+ rs.getLong(4));
}
int x = st.executeUpdate("update [Sheet1$] set sid=101 where sid=1");
if (x == 1) {
System.out.println("Updated");
}else{
System.out.println("NOT Updated");
}
} catch (Exception e) {
e.printStackTrace();
} finally {
// CLOSE THE Resources
}}}
```

Transaction Management

- Transaction is the process of performing multiple database operations as one Atomic unit with All-Nothing Criteria.
- When All the database operations in the unit are successful then Transaction is successful and should be committed.

- When any one database operation in the unit is failed then Transaction is failed and should be rolled black.


Collect Presonal info like Name,Email,Phone,Gender
Collect Login info like UserName,Password
Collect Address info like street,City,State
String sql1="insert into printfo_table values...";
String sql2="insert into linfo_table values....";
String sql3="insert into ainfo_table values...";
Name
Email
Phone
Gender
Username
Password
Street
City
State
Pin

- When you implement Transactions properly in your application, it gaurantees ACID Properties.
  - A-Atomacity
  - C-Consistency
  - I-Isolation
  - D-Durability

Type Of Transactions

- Local Transactions
- Distributed Transactions

Local Transactions

- When a single Database is participating in the Txal Operations then it is called as Local Transactions.

Ex:

- Transfer the funds from one account to another account where two accounts are in same bank or same database.

Distributed Transactions

- When a two or more Database are participating in the Txal operations then it is called as Distributed Transactions.
  Ex:
  - Transfer the funds from one account to another where two account are in different banks or different database.

Note: Jdbc Supports only Local Transactions and doesn't support Distributed Transactions.

JDBC Tx Managament:

- Specifying the transactional Boundries,

```
Connection con=null;
Try{
Con.setAutoCommit(false);        //Transaction Begin
Op1;
Op2;
Op3;
Op4;
Con.commit();              //Transaction End
}catch(Exception e){
If(con!=null){
Con.rollback();            //Transaction End
}
}
```

- When multiple transaction are running concurrently then you may get some transactional concurrency problems
  - Dirty Read problem
  - Repeatable read problem
  - Phantom Read Problem
- You need to specify the transactional Isolation levels to solve these Transactional concurrency problems.
- There are 4 transactional Isolation levels which are defined as Constant in connection interface as follows:

- **TRANSACTION_READ_UNCOMMITED 1**
- **TRANSACTION_READ_COMMITED 2**
- **TRANSACTION_REPEATABLE_READ 4**
- **TRANSACTION_SERIALIZABLE 8**

- Use the following method to specify the required Transactional Isolation Level
  - Con.setTransactionIsolation(2);
  - Con.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITED);

Jtc25.java

```
package com.jtcindia.jdbc;
class InsufficientFundsException extends
Exception{
public InsufficientFundsException() {}
public String toString(){
return "Sufficient Funds are not
Available";
       }

}
class InvalidAccountNumberException
extends Exception{
       int accno;
public InvalidAccountNumberException() {}
InvalidAccountNumberException(int accno){
this.accno=accno;
       }
public String toString(){
return "Accno:"+accno+"is Not Found";
       }
}
public class Jtc25 {
public static void main(String arg[]){
              Account acc=new Account();
              acc.transfer(11,22,33);
       }

}
```

```
package com.jtcindia.jdbc;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import
javax.naming.InsufficientResourcesException;

import org.omg.CORBA.ExceptionList;

public class Account {
       int bal;
       int xbal,ybal,zbal;
public void transfer(int x,int y,int z){
              Connection con=null;
              PreparedStatement ps1=null;
              PreparedStatement ps2=null;
              try{
con=JdbcUtil.getMySQLConnection();
                     //START TRANSACTION
       con.setAutoCommit(false);
                     //OPERATION 1 check
destination account
ps1=con.prepareStatement("select bal from
account where accno=?");
ResultSet rs1=ps1.executeQuery();
if(rs1.next()){
xbal=rs1.getInt(1);
                     }else{
throw new InvalidAccountNumberException(x);
                     }
                     x=y+z;
                     //OPERATION 2 UPDATE
Destination Account
ps2=con.prepareStatement("update account set
bal=? where accno=?");
                     ps2.setInt(1, x);
                     ps2.setInt(2,y);
                     ps2.executeUpdate();
System.out.println("***"+x+"update");
```

	```
	                    //OPERATION 3 check
	source account
	                    ps1.setInt(1,y);
	                    rs1=ps1.executeQuery();
	                    if(rs1.next()){
	                          y=rs1.getInt(1);
	                    }else{
	                          throw new
	InvalidAccountNumberException(x);
	                    }
	                    if(ybal>=z){
	                          ybal=ybal=z;

	                    }else{
	                          throw new
	InsufficientFundsException();
	                    }
	                    //OPERATION 4 UPDATE
	Source Account
	                    ps2.setInt(1,ybal);
	                    ps2.setInt(2,x);
	                    ps2.executeUpdate();
	                    con.commit();
	System.out.println("**"+x+"update");
	System.out.println("**Fund Transfered");
	              }catch(Exception e){
	                    try{
	                          con.rollback();
	                    }catch(Exception e1){
	System.out.println(e);
	                    }
	              }
	        }
	}
	```

Connection Pooling

- JDBC supports two to manage the Connections.
  - DriverManager Connections.
  - DataSource Connections.

DriverManager Connections

- If you want to get the DriverManager Connections then you need to write the following code.
  - Code…..

- When you use DriverMAnager Connections, you will get to problems.
  - With DriverManager connections, you need to Hardcode the driver class,url,username and password in every program, when you want to change the database then you need to change all the programs which gives the maintainance problem.(Note: we have already solved this using JDNBCUtil).
  - When you call setConnection(url,un,pw) method on the DriverManager then new Connection will be created and returned. When you call close() method on the Connection which is taken from DriverManager then that Connections every time for every user is expensive and may damage your application performance.
- You can solve these problems with Datasource Connections.

Datasource Connection

- DataSource Connections are based on Connection pooling technique.
- Connection pool is special area which is containing set of reusable database connections i.e multiple database connections will be created and be placed in pool.
- Whenever you want to use the connection
  - You can just pick a connection from the pool
  - Use the connection for database operation
  - Return the connection to pool.
- If you want to use DataSource Connections, then you have to use some Web/Application Server.

Working with dates

Table Required:

Create table dataTest(id int, dop date);

Jtc26.java

```
package com.jtcindia.jdbc;

import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.Date;
import java.sql.Statement;
import java.text.SimpleDateFormat;
import java.util.Scanner;

public class Jtc26 {
	public static void main(String
arg[]){
	Scanner sc=new Scanner(System.in);
	System.out.println("Enter ID");
	int id=sc.nextInt();
```

Jtc27.java

```
package com.jtcindia.jdbc;

import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.text.SimpleDateFormat;
import java.util.Scanner;

public class Jtc27 {
        System.out.println("Enter Date");
        int d=sc.nextInt();
        System.out.println("Enter Month");
        int m=sc.nextInt();
        System.out.println("Enter Year");
        int y=sc.nextInt();
        Date dt=new Date(y-1900,m-1,d) ;
        Connection con=null;
        Statement st=null;
        try{

        con=JdbcUtil.getMySQLConnection();
                DatabaseMetaData
md=con.getMetaData();
                String
db=md.getDatabaseProductName();
                String dob="";
                System.out.println(db);
                if(db.equals("Oracle")){
                        SimpleDateFormat
f=new SimpleDateFormat("yyyy-MM-dd");
                        dob=f.format(dt);
                }
                st=con.createStatement();
                String
sql=String.format("insert into datetest
values(%d,'%d')",d,dob);
                st.executeUpdate(sql);

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

        }

        }

}
```

```
        public static void main(String arg[]){
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter ID");
        int id=sc.nextInt();
        System.out.println("Enter Date");
        int d=sc.nextInt();
        System.out.println("Enter Month");
        int m=sc.nextInt();
        System.out.println("Enter Year");
        int y=sc.nextInt();
        Date dt=new Date(y-1900,m-1,d) ;
        Connection con=null;
        Statement st=null;
        PreparedStatement ps=null;
        ResultSet rs=null;
        try{
        con=JdbcUtil.getMySQLConnection();
                DatabaseMetaData
md=con.getMetaData();
ps=con.prepareStatement("insert into datetest
values(?,?)");
                ps.setInt(1,id);
                ps.setInt(2, d);
                ps.executeUpdate();
                System.out.println("inserted");
                JdbcUtil.cleanup(ps, null);

ps=con.prepareStatement("select *from
datetest");
rs=ps.executeQuery();
        while(rs.next()){
        Date dob=rs.getDate(2);
        SimpleDateFormat f=new
SimpleDateFormat("dd-mm-yyyy");
                System.out.println(id+"\t"+d);
                }
        }catch(Exception e){
                e.printStackTrace();
        }
        }
}
```

Working with files

- When you want to store the files in database then do the following:
  - Define the column data type as BLOB/LONGBLOB(MySql)
  - Create the fileInputStream by representing the file and its path.
  - Invoke the following method with PreparedStatement.
    - Ps.setBinaryStream(index,fils);
- When you want to read the files from database then do the following:
  - Invoke the following method with ResultSet.
    - InputStream is=rs.getBinaryStream(1);
  - Create the FileOutputStream by representing the file the its path.
  - Read the data from InputStream and write to the FileOutputStream.

Note: it is not the good practice to store large files and Images in the databse. It may damage the performance because of reading and writing the Streams every time.

- **Best Paractice:**
  - Store the file or Image in the hard disk.
  - Store the filename with the path in the database.

<u>**Table Required:**</u>

Create table datatable(id int,name varchar(100),data blob); //Oracle

Create table datatable(id int,name varchar(100),data blob);

Jtc28.java
```java
package com.jtcindia.jdbc;

import java.io.File;
import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Scanner;
import java.util.StringTokenizer;

public class Jtc28 {
public static void main(String arg[]){
Scanner sc=new Scanner(System.in);
System.out.println("Enter file name with
path");
String filename=sc.nextLine();
      String abcfilename=filename;
```

Jtc29.java
```java
package com.jtcindia.jdbc;

import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Scanner;

public class Jtc29 {
public static void main(String arg[]){
Scanner sc=new Scanner(System.in);
System.out.println("Enter file Name");
String filename=sc.nextLine();
Connection con=null;
PreparedStatement ps=null;
```

```StringTokenizer tok=new StringTokenizer(filename,"\\"); while(tok.hasMoreTokens()) filename=tok.nextToken(); Connection con=null; PreparedStatement ps=null; ResultSet rs=null; FileInputStream fis=null; try{ con=JdbcUtil.getMySQLConnection(); String sql="insert into datatable(name,data) values(?,?)"; ps=con.prepareStatement(sql); ps.setString(1,filename); File image=new File(abcfilename); fis=new FileInputStream(image); ps.setBinaryStream(2,fis,(int)image.length()); ps.execute(); System.out.println("inserted"); }catch(Exception e){ e.printStackTrace(); } } }```	```FileOutputStream fos=null; try{ con=JdbcUtil.getMySQLConnection(); String sql="select name,data from database where name=?"; ps=con.prepareStatement(sql); ps.setString(1,filename); ResultSet rs=ps.executeQuery(); while(rs.next()){ File image=new File("D:\\"+filename); fos=new FileOutputStream(image); byte[] buffer=new byte[1]; InputStream is=rs.getBinaryStream(2); while(is.read(buffer)>0){ fos.write(buffer); } } System.out.println("file Accessed in D:\\"+filename); }catch(Exception e){ e.printStackTrace(); } } }```