JDBC

Study Guide

Author
Srinivas Dande

1.1 Introduction

- JDBC is a technology which is used to intract with the database 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 as well as various database Vendor.

1.1.1 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.

1.1.2 JDBC API

- The Java Database Connectivity (JDBC) API provides universal data access from the Java programs.
- Java Program which uses 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

1.1.3 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.

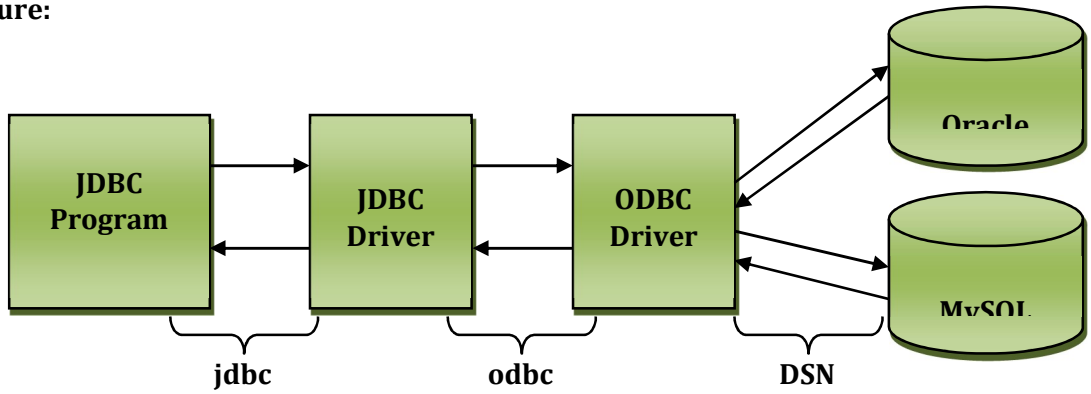
1.2 Types of JDBC Drivers

- There are four types of JDBC Drivers.
  - Type I Driver — JDBC ODBC Bridge Driver
  - Type II Driver — Partial Java and Partial Native Driver
  - Type III Driver — Net Protocol Driver
  - Type IV Driver — Pure Java Driver

1.2.1 Type I 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>
Software Required	DB, Java, ODBC Drivers

Architecture:

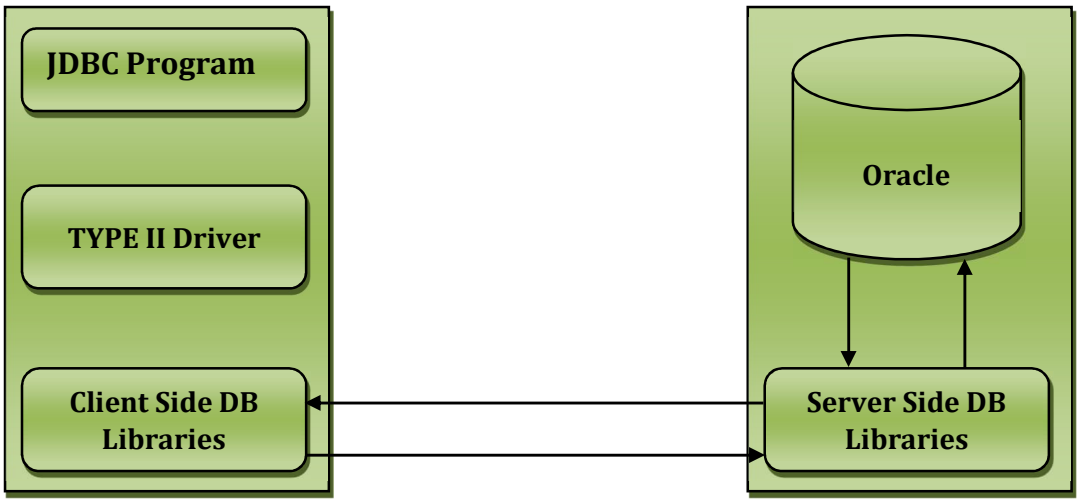


Note: Type I driver support is removed from Java 8.

1.2.2 Type II Driver

Name	Partial Java and Partial Native Driver
Vendor	Database 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

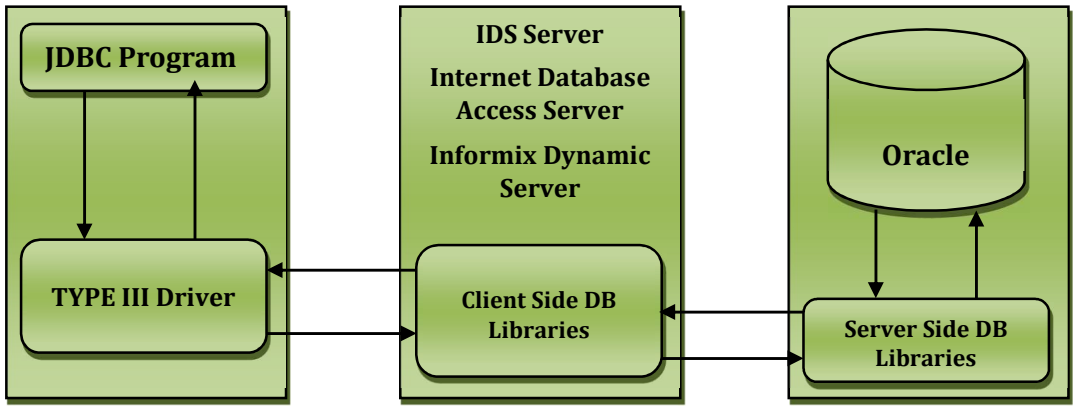
Architecture:



1.2.3 Type III Driver

Name	Net Protocol Driver
Vendor	IDS Software Vendor
Driver Class	com.ids.Driver
URL	jdbc:ids://hostname…
Username	<Database Username>
Password	<Database Password>
Software Required	IDS Server, Database Client Server Edition, Java

Architecture:



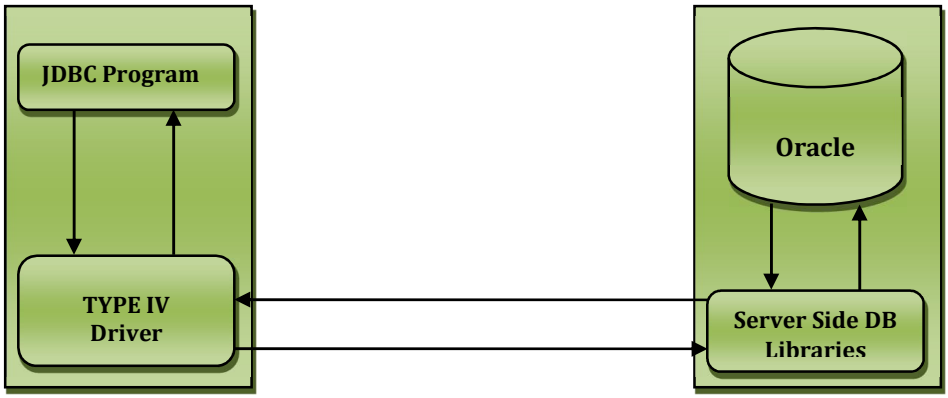
1.2.4 Type IV Driver

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

For Oracle	
Driver Class	oracle.jdbc.driver.OracleDriver
URL	jdbc:oracle:thin:@<host>:<port>:<serviceName> ex: jdbc:oracle:thin:@localhost:1521:XE
Classpath	ojdbc14.jar or ojdbc6.jar

For MySQL	
Driver Class	com.mysql.jdbc.Driver
URL	jdbc:mysql://<host>:<port>/<dbName> ex: jdbc:mysql://localhost:3306/jlcdb
Classpath	mysql.jar

Architecture:



1.3 Comparision between JDBC Drivers

Pros and Cons of Types of Drivers		
	Advantages	**Disadvantages**
Type I Driver	◆Type I is very easy to use and maintain. ◆Type I is suitable for migrating application to Java without changing existing ODBC setup. ◆No extra software is required for the Type I implementation. ◆Performance of the Type I is acceptable.	◆Type I driver implementation is possible in window OS only because ODBC drivers available only with windows. ◆Performance of this driver is not excellent but acceptable.
Type II Driver	◆Type II is faster than all other drivers.	◆In Type II both client and server machine will have the database library. ◆ When database is migrated then you will get much maintenance because you need to re-install client side libraries in all the client machines.
Type III Driver	◆In Type III, client side DB libraries are moved to middleware server called IDS server. ◆Because of this, client side maintenance is reduced.	◆You need to purchase extra software called IDS server. ◆Because of having middleware server between your program and database server, performance will be reduced.
Type IV Driver	◆This driver is best among all the drivers and highly recommendable to use.	◆Negligible.

Note: Type I driver support is removed from Java 8.

Writing First JDBC Program:

A) Make Sure that MySQL5/8 is installed .

```
create database myjdbcdb;

create table mycustomers(
cid int primary key,
cname char(15),
email char(15),
phone int,
city char(15)
);
```

B) Make sure that JDK8 is installed.

C) Make sure that Eclipse is installed.

D) Make sure that You have Mysql jar file

mysql-connector-java-8.0.13.jar (MySQL 8)
mysql-connector-java-5.1.45.jar (MySQL 5)

E) Start Coding

1) Open the Eclipse with workspace - MyLabs
2) Create the Java Project
3) Add the JDK1.8 which is installed by you.
4) Add the MySQL Jar to Project ClassPath.
5) Create the package called **com.coursecube.jdbc**
6) Create the Java Class - **Lab1 with main() method**
7) Write the Code with the following Steps

Step 1: Load the Driver Class
Step 2: Open the Connection
Step 3: Prepare SQL Statement
Step 4: Create the JDBC Statement
Step 5: Submit SQL to DB
Step 6: Process Results
Step 7: Close the Resources.

8) Run the Program

Lab1.java

```
package com.jlcindia.jdbc;

import java.sql.*;

/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab1 {
public static void main(String[] args) {

Connection con=null;
Statement st=null;

try {

//Step 1: Load the Driver Class
Class.forName("com.mysql.jdbc.Driver");

//Step 2: Open the Connection
con = DriverManager.getConnection("jdbc:mysql://localhost:3306/myjdbcdb", "root",
"srinivas");

//Step 3: Prepare SQL Statement
String SQL="insert into mycustomers values(106,'ds','ds@jlc',12345,'Blore')";

//Step 4: Create the JDBC Statement
st=con.createStatement();

//Step 5: Submit SQL to DB
int x=st.executeUpdate(SQL);

//Step 6: Process Results
if(x==1) {
System.out.println("Customer Record is Inserted Succesfully");
}else {
System.out.println("Sorry, Customer Record is Not Inserted");
}
```

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

finally {
//Step 7: Close the Resources.
try {
if(st!=null)
st.close();
if(con!=null)
con.close();
}catch(Exception e) {
e.printStackTrace();
}
}

}
}
```

Lab2.java

```
package com.jlcindia.jdbc;

import java.sql.*;
/*
* @Author     : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */

public class Lab2 {
public static void main(String[] args) {

Connection con=null;
Statement st=null;
ResultSet rs=null;

try {
```

```
//Step 1: Load the Driver Class
Class.forName("com.mysql.jdbc.Driver");

//Step 2: Open the Connection
con = DriverManager.getConnection("jdbc:mysql://localhost:3306/myjdbcdb", "root",
"srinivas");

//Step 3: Prepare SQL Statement
String SQL="select * from mycustomers ";


//Step 4: Create the JDBC Statement
st=con.createStatement();

//Step 5: Submit SQL to DB
rs=st.executeQuery(SQL);

//Step 6: Process Results

while(rs.next()) {
int cid= rs.getInt(1);
String cname=rs.getString(2);
String email=rs.getString(3);
int phone=rs.getInt(4);
String city=rs.getString(5);

System.out.println(cid+"\t"+cname+"\t"+email+"\t"+phone+"\t"+city);
}

System.out.println("----Done-----");

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

```
finally {
//Step 7: Close the Resources.
try {
if(rs!=null)
rs.close();
if(st!=null)
st.close();
if(con!=null)
con.close();
}catch(Exception e) {
e.printStackTrace();
}
}

}
}
```

1.4 JDBCUtil class

JDBCUtil.java

```
package com.jlcindia.jdbc.util;

import java.sql.*;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class JDBCUtil {
public static Connection getConnection() {

Connection con = null;
try {
Class.forName("com.mysql.jdbc.Driver");
con = DriverManager.getConnection("jdbc:mysql://localhost:3306/myjdbcdb", "root",
"srinivas");
} catch (Exception ex) {
ex.printStackTrace();
```

```
}

return con;
}

public static void cleaup(Statement st, Connection con) {

try {
if (st != null)
st.close();

if (con != null)
con.close();

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

}

public static void cleaup(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) {
e.printStackTrace();
}

}
}
```

Lab3.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab3 {
public static void main(String[] args) {
Connection con=null;
Statement st=null;

try {
con=JDBCUtil.getConnection();

String SQL="insert into mycustomers values(107,'hello','hello@jlc',12345,'Blore')";
st=con.createStatement();
int x=st.executeUpdate(SQL);

if(x==1) {
System.out.println("Customer Record is Inserted Succesfully");
}else {
System.out.println("Sorry, Customer Record is Not Inserted");
}
}catch(Exception ex) {
ex.printStackTrace();
}finally {
JDBCUtil.cleaup( st, con);
}
}
}
```

Lab4.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab4 {
public static void main(String[] args) {
Connection con=null;
Statement st=null;
ResultSet rs=null;
try {
con=JDBCUtil.getConnection();
String SQL="select * from mycustomers ";
st=con.createStatement();
rs=st.executeQuery(SQL);

while(rs.next()) {
int cid= rs.getInt(1);
String cname=rs.getString(2);
String email=rs.getString(3);
int phone=rs.getInt(4);
String city=rs.getString(5);
System.out.println(cid+"\t"+cname+"\t"+email+"\t"+phone+"\t"+city);
}
System.out.println("----Done-----");
}catch(Exception ex) {
ex.printStackTrace();
}finally {
JDBCUtil.cleaup(rs, st, con);
}
}  }
```

Introduction to JDBC - Questions

Q1) What is JDBC? What is latest version of JDBC available?

Q2) What is JDBC API?

Q3) What are the classes and interfaes available in JDBC API?

Q4) Who has provided JDBC API?

Q5) Who has provided implementation of JDBC API?

Q6) What are the steps to write JDBC Program?

Q7) What is JDBC Driver?

Q8) How many types of JDBC Drivers are available? Which is best?

Q9) Which version of Java has excluded TYPE I Driver?

Q10) What will happen when driver class is loaded?

Q11) I have loaded both Oracle and mysql drivers.Which database connection will be established when I call getConnection (...) method?

Q12) I have loaded Oracle driver and trying to get the connection with MySQL URL . What will happen?

Code:

```
Class.forName ("oracle.jdbc.driver.OracleDriver");
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/jlcdb","root","srinivas")
```

Q13) Can I register the Driver Explicitly?

Q14) Can I unregister the Driver?

Q15) How can I find list of drivers registered?

Q16) Can I establish two database connections at a time?

Q17) What are the difference among the following 3 getConnection() methods ?

```
public static Connection getConnection(String url)
public static Connection getConnection(String url, String uname, String pword)
public static Connection getConnection(String url, Properties info)
```

Q18) What the DriverManager.getConnection() method is doing?

Q19) In JDBC API or in java.sql package, SUN has given more interfaces like Connection, Statement, ResulSet etc. How Instances will be created for those interfaces?

1.5 JDBC Statements

- There are 3 types of JDBC Statements:
  1) Statement
  2) PreparedStatement
  3) CallableStatement

1.5.1 Statement

- Statement is an interface available in java.sql package.
- Subclass of Statement interface is provided by Driver vendor.
- You can create the Statement object using the following methods of Connection interface:
  - public Statement createStatement()
  - public Statement createStatement(int , int)
  - public Statement createStatement(int ,int ,int)

- You can call one of the following methods on Statement object to submit the SQL Statement to Database:
  - public int executeUpdate(String sql)
  - public ResultSet executeQuery(String sql)
  - public boolean execute(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 ResultSet object contains the records returned by SELECT statement.

- **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.
  - **If returned value is true which means that SELECT SQL statement is submitted and ResultSet object is created.**
    - Use the following method of Statement to get the ResultSet object
      - **public ResultSet getResultSet()**

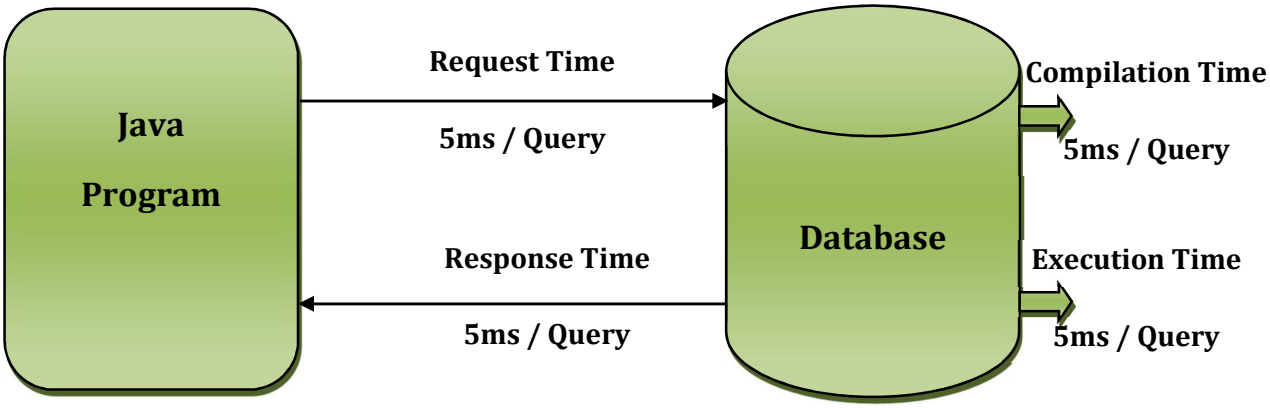
- **If returned value is false which means that INSERT, UPDATE or DELETE SQL statement is submitted and integer number is available which represents number of records inserted,updated or deleted.**
  - Use the following method of Statement to get the integer number available
    - **public int getUpdateCount()**

- Using the Single Statement Object, you can submit any type of SQL statements 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 SQL Stmts = 2000ms.

- When you provide dynamic values for the SQL Statement then you need to use concatenation operator, Formatter or format() of String class etc to format the query.

```
int sid = 99;
String nm = "Sri";
String em = "sri@jlc.com";
long phn = 8105815599L;
```

- String sql="insert into jlcstudents values ("+sid+",'"+nm+"','"+em+"',"+phn+")";
- String sql = String.format("insert into jlcstudents values (%d,'%s','%s',%d)", sid,nm, em, phn);

Lab5.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab5 {
public static void main(String[] args) {
Connection con=null;
Statement st=null;

int mycid=109;
String mycname="bbb";
String myemail="bbb@jlc";
int myphone=222;
String mycity="Blore";

try {
con=JDBCUtil.getConnection();

String SQL=String.format("insert into mycustomers
values(%d,'%s','%s',%d,'%s')", mycid,mycname,myemail,myphone,mycity);
System.out.println(SQL);
```

```
st=con.createStatement();
int x=st.executeUpdate(SQL);

if(x==1) {
System.out.println("Customer Record is Inserted Succesfully");
}else {
System.out.println("Sorry, Customer Record is Not Inserted");
}

}catch(Exception ex) {
ex.printStackTrace();
}finally {
JDBCUtil.cleaup( st, con);
}
}
}
```

Lab6.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab6 {
public static void main(String[] args) {

Connection con=null;
Statement st=null;
ResultSet rs=null;

String mycity="Blore";

try {
con=JDBCUtil.getConnection();
```

```
String SQL=String.format("select * from mycustomers where city='%s'",mycity);
System.out.println(SQL);

st=con.createStatement();

rs=st.executeQuery(SQL);


while(rs.next()) {
int cid= rs.getInt(1);
String cn=rs.getString(2);
String em=rs.getString(3);
int ph=rs.getInt(4);
String ci=rs.getString(5);

System.out.println(cid+"\t"+cn+"\t"+em+"\t"+ph+"\t"+ci);
}

System.out.println("----Done-----");

}catch(Exception ex) {
ex.printStackTrace();
}finally {
JDBCUtil.cleaup(rs, st, con);
}
}
}
```

Statement Questions

Q20) What are the types of JDBC Statements?

Q21) What is Statement?

Q22) Is there any super type defined for Statement?

Q23) Who is responsible to define subclass for Statement?

Q24) How to get / create the obejct of Statement type?

Q25) Do we need to provide any SQL Statement while creating the Statement?

Q26) What methods can be used from Statement to submit the SQL Query to database?

Q27) What is the difference among executeUpdate (), executeQuery () and execute ()?

Q28) How many Queries can be submited using one Statement object?

Q29) How many types of queries I can submit using one Statement object?

Q30) When excatly the SQL Statemet will be submitted to the database?

Q31) How many times SQL Statenment will be compiled/ verified with Statement Object?

Q32) How to use dynamic value to the SQL Statement in the case of Statement object?

1.5.2 PreparedStatement

- PreparedStatement is an interface available in java.sql package.
- PreparedStatement is extending Statement interface.
- Subclass of PreparedStatement interface is provided by Driver vendor.
- You can create the PreparedStatement object using the following methods of Connection interface:
  - public PreparedStatement prepareStatement(sql)
  - public PreparedStatement prepareStatement(sql,int , int)
  - public PreparedStatement prepareStatement(sql,int ,int ,int)

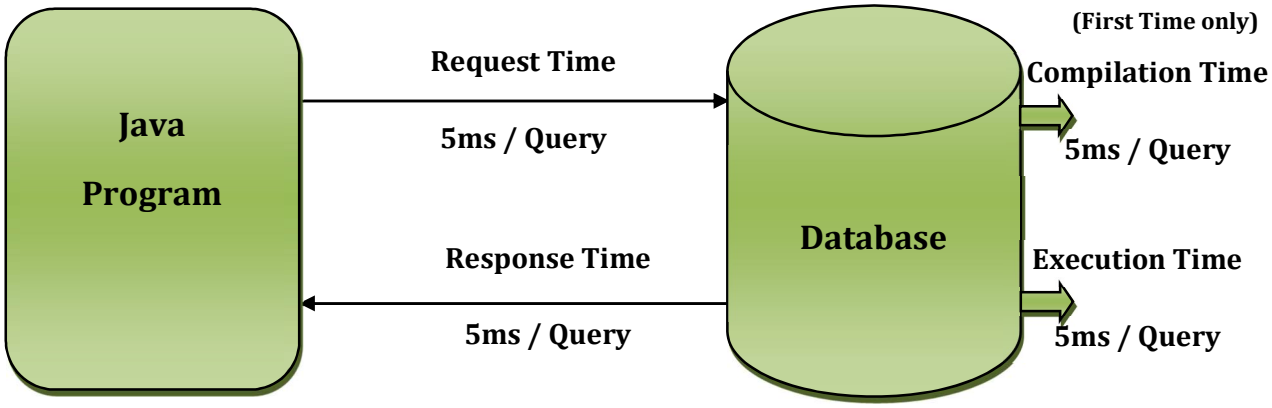
- You can call one of the following methods on PreparedStatement object to submit the SQL Statement to Database:
  - public int executeUpdate()
  - public ResultSet executeQuery()
  - public boolean execute()

- 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-compiled 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 = 20 ms.

2nd onwards = 5ms + 0 ms+ 5 ms +5 ms
= 15 ms.

100 SQL Statements = 20 ms + 99 * 15 ms.
= 20 ms + 1485 ms.
= 1505ms.

- PreparedStatement gives you the place holder mechanism for providing the data dynamically to the SQL Statement.You need to use ? symbol for **placeholder**.
- These Placeholders also called as Parameters.
- JDBC Supprots only **Positional Parameters**.
- JDBC Does not Supprot **Named Parameters**.

- To provide the value for placeholder, you need to invoke the setter methods depending on the placeholder data type.

    - public void setInt(int paramIndex, int val)
    - public void setString(int paramIndex, String val)
    - public void setLong(int paramIndex, long val)
    - public void setDouble(int paramIndex, double val)
      etc.

Lab7.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;

/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */

public class Lab7 {
public static void main(String[] args) {

Connection con=null;
PreparedStatement ps=null;
```

```
int mycid=109;
String mycname="bbb";
String myemail="bbb@jlc";
int myphone=222;
String mycity="Blore";

try {

con=JDBCUtil.getConnection();
String SQL="insert into mycustomers values(?,?,?,?,?)";

ps=con.prepareStatement(SQL);

ps.setInt(1, mycid);
ps.setString(2,mycname);
ps.setString(3,myemail);
ps.setInt(4,myphone);
ps.setString(5,mycity);

int x=ps.executeUpdate();

if(x==1) {
System.out.println("Customer Record is Inserted Succesfully");
}else {
System.out.println("Sorry, Customer Record is Not Inserted");
}


}catch(Exception ex) {
ex.printStackTrace();
}finally {
JDBCUtil.cleaup( ps, con);
}
}
}
```

Lab8.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */

public class Lab8 {
public static void main(String[] args) {

Connection con=null;
PreparedStatement ps=null;
ResultSet rs=null;

String mycity="Blore";

try {
con=JDBCUtil.getConnection();

String SQL="select * from mycustomers where city=?";
ps=con.prepareStatement(SQL);
ps.setString(1, mycity);

rs=ps.executeQuery();

while(rs.next()) {
int cid= rs.getInt(1);
String cn=rs.getString(2);
String em=rs.getString(3);
int ph=rs.getInt(4);
String ci=rs.getString(5);
System.out.println(cid+"\t"+cn+"\t"+em+"\t"+ph+"\t"+ci);
}

System.out.println("----Done-----");
```

```
}catch(Exception ex) {
ex.printStackTrace();
}finally {
JDBCUtil.cleaup(rs, ps, con);
}
}
}
```

Lab9.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab9 {
public static void main(String[] args) {

Connection con=null;
Statement st=null;
ResultSet rs=null;

String SQL1="insert into mycustomers values(201,'aa','aa@jlc',111,'Blore') ";
String SQL2="update mycustomers set phone=5599 where phone=12345";
String SQL3="select * from mycustomers";

try {
con=JDBCUtil.getConnection();
st=con.createStatement();

//boolean b=st.execute(SQL1);
//boolean b=st.execute(SQL2);
boolean b=st.execute(SQL3);
```

```
if(b==false) {
System.out.println("Submited - Updatable Op");
int x=st.getUpdateCount();
System.out.println("Records Updated : " + x);
}else {
System.out.println("Submited - Select Op");
rs=st.getResultSet();

while(rs.next()) {
int cid= rs.getInt(1);
String cn=rs.getString(2);
String em=rs.getString(3);
int ph=rs.getInt(4);
String ci=rs.getString(5);
System.out.println(cid+"\t"+cn+"\t"+em+"\t"+ph+"\t"+ci);
}
}
}catch(Exception ex) {
ex.printStackTrace();
}finally {
JDBCUtil.cleaup(rs, st, con);
}
}
}
```

Lab10.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab10 {
public static void main(String[] args) {
```

```
Connection con=null;
PreparedStatement ps=null;
ResultSet rs=null;

String SQL1="insert into mycustomers values(201,'aa','aa@jlc',111,'Blore') ";
String SQL2="update mycustomers set phone=12345 where phone=111";
String SQL3="select * from mycustomers where city='Blore'";

try {
con=JDBCUtil.getConnection();
//ps=con.prepareStatement(SQL1);
//ps=con.prepareStatement(SQL2);
ps=con.prepareStatement(SQL3);

boolean b=ps.execute();

if(b==false) {
System.out.println("Submited - Updatable Op");
int x=ps.getUpdateCount();
System.out.println("Records Updated : " + x);
}else {
System.out.println("Submited - Select Op");
rs=ps.getResultSet();

while(rs.next()) {
int cid= rs.getInt(1);
String cn=rs.getString(2);
String em=rs.getString(3);
int ph=rs.getInt(4);
String ci=rs.getString(5);
System.out.println(cid+"\t"+cn+"\t"+em+"\t"+ph+"\t"+ci);
}
}
}catch(Exception ex) {
ex.printStackTrace();
}finally {
JDBCUtil.cleaup(rs, ps, con);
}
}
}
```

PreparedStatement Questions

Q33) What is PreparedStatement?

Q34) In which package the PreparedStatement is defined?

Q35) Is there any super type defined for PreparedStatement?

Q36) Who is responsible to define subclass for PreparedStatement?

Q37) How to get / create the obejct of PreparedStatement type?

Q38) While creating the PreparedStatement do I need to provide any SQL Statement?

Q39) Which methods of PreparedStatement can be used to submit the SQL Query to database?

Q40) How many Query I can submit using one PreparedStatement object?

Q41) How many types of queries I can submit using one PreparedStatement object?

Q42) When you submit the SQL Statenment to database using PreparedStatement then how many times the SQL Statement will be compiled/ verified?

Q43) How to use dynamic value to the SQL Statement in the case of PreparedStatement object?

Q44) What is the difference between Statement and PreparedStatement?

Q45) What is the benifit of PreparedStatement over Statement?

1.5.3 CallableStatement

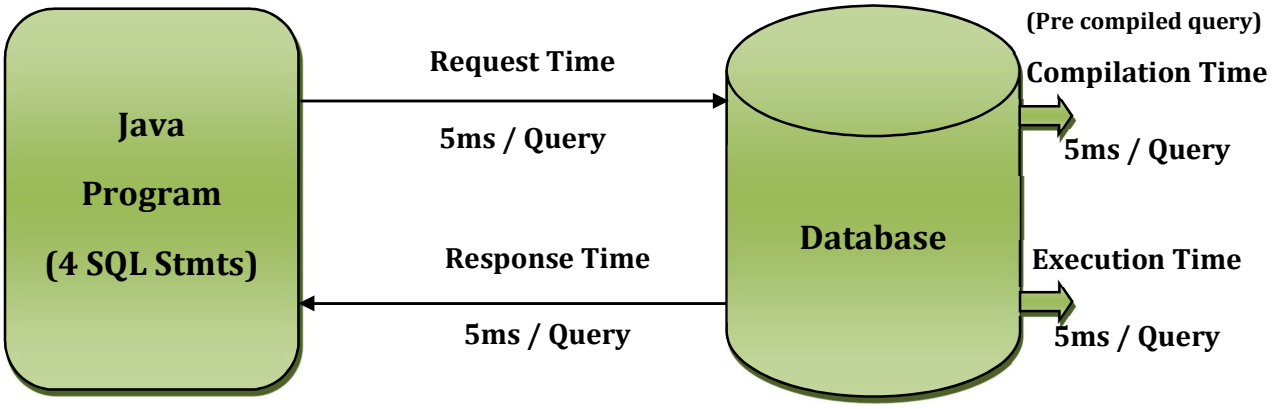
- CallableStatement is an interface available in java.sql package.
- CallableStatement is extending PreparedStatement interface.
- Subclass of CallableStatement interface is provided by Driver vendor.
- You can create the CallableStatement object using the following methods of Connection interface:
    - public CallableStatement prepareCall(sql)
    - public CallableStatement prepareCall(sql,int , int)
    - public CallableStatement prepareCall(sql,int ,int ,int)

- You can call the following method on CallableStatement object to submit the SQL Statement to Database:
    - public boolean execute()

- Using the Single CallableStatement Object, you can submit call to only one procedure.

```
ex:
String sql="call p1(?)";
CallableStatement cs=con.prepareCall(sql);
cs.setInt(1,10);
cs.execute();
```

- When you submit the call to stored procedure using CallableStatement object then pre-compiled stored procedure will be executed directly.



- Total time = req.time + compile time + exec. time + res.time
  = 5 ms+0 ms+ 4* 5 ms+5 ms
  = 5 ms+0 ms+20 ms+5 ms = 30 ms.

- 4 SQL Statements * 25 times
  = 30ms * 25 = 750ms.

- CallableStatement gives you the place holder mechanism for providing the data dynamically to the procedure parameters. You need to use ? symbol for placeholder.

- To provide the value for placeholder, you need to invoke the setter methods depending on the placeholder data type.
  - public void setInt(int paramIndex, int val)
  - public void setString(int paramIndex, String val)
  - public void setLong(int paramIndex, long val)
  - public void setDouble(int paramIndex, double val)
    etc

- You need to use following method to specify the OUT parameter
  - public void registerOutParameter(int parameterIndex, int sqlType)
    - sqlType is a constant from java.sql.Types class
- You need to use the following method to access the result of OUT Parameter
  - public int getInt(int paramIndex)
  - public String getString(int paramIndex)
  - public long getLong(int paramIndex)
  - public double getDouble(int paramIndex)

Procedure required for Lab11:

```
delimiter ##
create procedure p1(IN a int,IN b int,OUT c int,OUT d int)
begin
set c=a+b;
set d=a*b;
end;
##
delimiter  ;
```

Lab11.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author     : Srinivas Dande
* @Company : Java Learning Center
* @See           : myjavalearningcenter.com
* */
public class Lab11 {
public static void main(String[] args) {

Connection con=null;
CallableStatement cs=null;
try {
con=JDBCUtil.getConnection();
String SQL="{call p1(?,?,?,?)}";

cs=con.prepareCall(SQL);
cs.setInt(1, 10);
cs.setInt(2,20);
cs.registerOutParameter(3, Types.INTEGER);
cs.registerOutParameter(4, Types.INTEGER);

cs.execute();

int sum=cs.getInt(3);
int mul=cs.getInt(4);
System.out.println("Sum : "+sum);
System.out.println("Mul : "+mul);
}catch(Exception ex) {
ex.printStackTrace();
}finally {
JDBCUtil.cleaup(cs, con);
}
}
}
```

Procedure required for Lab12, Lab13, Lab14:

1) Create the following table

```
create table jlcstudents(
sid int primary key,
sname char(20) NOT NULL,
email char(20) NOT NULL,
totalfee double DEFAULT 25000,
feepaid double  NOT NULL,
m1 int,
m2 int,
m3 int,
total int,
average double,
status char(10),
grade char(5)
);
```

2) Insert some Sample Records

```
insert into jlcstudents(sid,sname,email,feepaid,m1,m2,m3)
values(101,'Sri','Sri@jlc',5000,60,40,90);

insert into jlcstudents(sid,sname,email,feepaid,m1,m2,m3)
values(102,'sd','sd@jlc',15000,80,80,80);

insert into jlcstudents(sid,sname,email,feepaid,m1,m2,m3)
values(103,'ds','ds@jlc',10000,20,30,40);

insert into jlcstudents(sid,sname,email,feepaid,m1,m2,m3)
values(104,'vas','vas@jlc',2000,95,96,97);

insert into jlcstudents(sid,sname,email,feepaid,m1,m2,m3)
values(105,hello,'hello@jlc',12000,99,99,99);
```

3) Write the Stored Procedure for finding Balance

```
delimiter ##
create procedure findBalance(IN mysid int,OUT mybal double)
begin
declare tfee double;
declare fpaid double;

select totalfee,feepaid into tfee,fpaid from jlcstudents where sid=mysid;
set mybal=tfee-fpaid;

end;
##
delimiter  ;
```

4) Write the Stored Procedure for finding Grade and Update the Table without OUT parameters

```
delimiter ##
create procedure findGrade(IN mysid int)
begin
declare mm1 int;
declare mm2 int;
declare mm3 int;
declare mytotal int;
declare myaverage double;
declare mystatus char;
declare mygrade char;

select m1,m2,m3 into mm1,mm2,mm3 from jlcstudents where sid=mysid;
set mytotal = mm1+mm2+mm3;
set myaverage = mytotal/3;
```

```
if myaverage>=70 then
set mystatus ='P';
else
set mystatus ='F';
end if;

if myaverage >= 90 then
set mygrade= 'A';
elseif myaverage >= 80 then
set mygrade= 'B';
else
set mygrade= 'C';
end if;
set mygrade =  trim(mygrade);
set mystatus = trim(mystatus);

update jlcstudents set total=mytotal, average=myaverage, status = mystatus ,
grade=mygrade where sid=mysid;

end;
##
delimiter  ;
```

5) Write the Stored Procedure for finding Grade and Update the Table with OUT parameters

```
delimiter ##
create  procedure findMyGrade(IN mysid int,OUT  mytotal int, OUT myaverage
double,OUT mystatus char, OUT mygrade char)
begin
declare mm1 int;
declare mm2 int;
declare mm3 int;

select m1,m2,m3 into mm1,mm2,mm3 from jlcstudents where sid=mysid;
```

```
set mytotal = mm1+mm2+mm3;
set myaverage = mytotal/3;

if myaverage>=70 then
set mystatus ='P';
else
set mystatus ='F';
end if;

if myaverage >= 90 then
set mygrade= 'A';
elseif myaverage >= 80 then
set mygrade= 'B';
else
set mygrade= 'C';
end if;

set mygrade =  trim(mygrade);
set mystatus = trim(mystatus);

update jlcstudents set total=mytotal, average=myaverage, status = mystatus ,
grade=mygrade where sid=mysid;

end;
##
delimiter  ;
```

Lab12.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab12 {
public static void main(String[] args) {

Connection con=null;
CallableStatement cs=null;

int mysid= 102;
try {

con=JDBCUtil.getConnection();
String SQL="{call findBalance(?,?)}";

cs=con.prepareCall(SQL);
cs.setInt(1, mysid);
cs.registerOutParameter(2, Types.DOUBLE);

cs.execute();

double bal=cs.getDouble(2);
System.out.println("Balance : "+bal);
}catch(Exception ex) {
ex.printStackTrace();
}finally {
JDBCUtil.cleaup(cs, con);
}
}
}
```

Lab13.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab13 {
public static void main(String[] args) {

Connection con=null;
CallableStatement cs=null;

int mysid= 102;

try {
con=JDBCUtil.getConnection();
String SQL="{call findGrade(?)}";

cs=con.prepareCall(SQL);
cs.setInt(1, mysid);

cs.execute();
System.out.println("Done- Call Completed : ");
}catch(Exception ex) {
ex.printStackTrace();
}finally {
JDBCUtil.cleaup(cs, con);
}
}
}
```

Lab14.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab14 {
public static void main(String[] args) {

Connection con=null;
CallableStatement cs=null;

int mysid= 104;

try {
con=JDBCUtil.getConnection();
String SQL="{call findMyGrade(?,?,?,?,?)}";

cs=con.prepareCall(SQL);
cs.setInt(1, mysid);
cs.registerOutParameter(2, Types.INTEGER);
cs.registerOutParameter(3, Types.DOUBLE);
cs.registerOutParameter(4, Types.CHAR);
cs.registerOutParameter(5, Types.CHAR);

cs.execute();

int total= cs.getInt(2);
double avg = cs.getDouble(3);
String status= cs.getString(4);
String grade= cs.getString(5);

System.out.println("Total : \t"+total);
System.out.println("Avg : \t"+avg);
System.out.println("Status : \t"+status);
System.out.println("Grade : \t"+grade);
```

```
}catch(Exception ex) {
ex.printStackTrace();
}finally {
JDBCUtil.cleaup(cs, con);
}
}
}
```

CallableStatement Questions

Q46) What is CallableStatement?

Q47) In which package the CallableStatement is defined?

Q48) Is there any super type defined for CallableStatement?

Q49) Who is responsible to define subclass for CallableStatement?

Q50) How to get / create the obejct of CallableStatement type?

Q51) While creating the CallableStatement do I need to provide any SQL Statement?

Q52) What methods of CallableStatement can be used to call the procedure from database?

Q53) When I call the procedure from database using CallableStatement then how many times the SQL Statement will be compiled/ verified?

Q54) How to use dynamic value to the procedure in the case of CallableStatement object?

Q55) How can you call the procedure from Java application using input parameter?

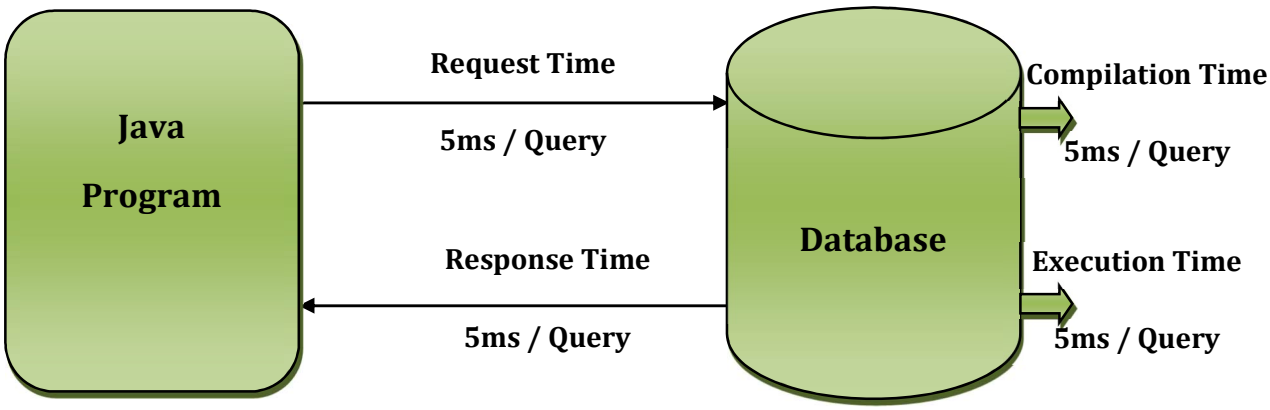
Q56) How can you call the procedure from Java application using output parameter of procedure?

Q57) How to get the value of output parameter of the procedure?

1.6 Batch Updates

- When you submit multiple queries to database one by one then lot of time will be wasted for request and response.

Ex:



Using Statement

For 1 SQL Statement = 5ms +5ms + 5ms + 5ms =20 ms
For 100 SQL Statements = 100 * 20= 2000ms

Using PreparedStatement

For 1 SQL Statement = 5ms +0ms + 5ms + 5ms =15 ms
For 100 SQL Statements = 100 * 15= 1500ms

- In the above two cases, you are trying to submit 100 SQL Statements. For submitting 100 SQL Statements, you need to communicate with the database 100 times. This increseases number of round trips between your application and database which damages the application performance.
- **Batch updates allows you to submit multiple SQL statements as One Batch to the Database at a time.**

Using Batch Updates

For 100 SQL Statements = 5ms +100 * 5ms + 100 * 5ms + 5ms
= 5ms +500ms + 500ms + 5ms
=1010 ms

- **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 improves the application performance.
  - You can use insert, update and delete statements 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 PreparedStatement:**
  - You can submit only single type of SQL Statement.
  - You can submit multiple SQL Statements.
  - You can reduce number of round trips between your application and database which improves the application performance.
  - You can use insert, update and delete statements only.
  - You can not use SELECT statement.
  - Use the following methods of PreparedStatement interface to implement Batch Updates.
    - void addBatch()
    - int [] executeBatch()
    - void clearParameters()

Using Statement	**Using PreparedStatement**

sql1="insert into jlcstudents..";
sql2="update jlcstudents ..";
sql3="delete from jlcstudents ..";
 	
st=con.createStatement();
st.addBatch(sql1);
st.addBatch(sql2);
st.addBatch(sql3);
 	
int x[]=st.executeBatch();
	sql = "insert into jlcstudents values (?,?)";
	ps=con.prepareStatement(sql);
	ps.setInt(1,99);
	ps.setString(2,"Sri");
	ps.addBatch();
	
	ps.setInt(1,88);
	ps.setString(2,"Vas");
	ps.addBatch();
	
	int x[]=ps.executeBatch();

Lab15.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab15 {
public static void main(String[] args) {

Connection con=null;
Statement st=null;
try {
con=JDBCUtil.getConnection();

String SQL1="insert into mycustomers values(501,'aa','aa@jlc',111,'Blore')";
String SQL2="insert into mycustomers values(502,'bb','bb@jlc',222,'Blore')";
String SQL3="insert into mycustomers values(503,'cc','cc@jlc',222,'Blore')";
String SQL4="update mycustomers set phone=55555 where cid=101";
String SQL5="update mycustomers set phone=99999 where city='Hyd'";
String SQL6="delete from mycustomers  where cid=201";

st=con.createStatement();

st.addBatch(SQL1);
st.addBatch(SQL2);
st.addBatch(SQL3);
st.addBatch(SQL4);
st.addBatch(SQL5);
st.addBatch(SQL6);

int results[]= st.executeBatch();

for(int x : results) {
System.out.println(x);
}
```

```
System.out.println("Done !!! ");

}catch(Exception ex) {
ex.printStackTrace();
}finally {
JDBCUtil.cleaup(st, con);
}
}
}
```

Lab16.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author     : Srinivas Dande
* @Company : Java Learning Center
* @See           : myjavalearningcenter.com
* */
public class Lab16 {
public static void main(String[] args) {

Connection con=null;
PreparedStatement ps=null;
Object [][] mycustomers= {
{504,"dd","dd@jlc",444,"Hyd"},
{505,"ee","ee@jlc",555,"Hyd"},
{506,"ff","ff@jlc",666,"Blore"},
};

try {
con=JDBCUtil.getConnection();

String SQL="insert into mycustomers values(?,?,?,?,?)";

ps=con.prepareStatement(SQL);
```

```
for( Object [] mycust : mycustomers) {
ps.setInt(1, (int)mycust[0]);
ps.setString(2, (String) mycust[1]);
ps.setString(3,  (String) mycust[2]);
ps.setInt(4, (int) mycust[3]);
ps.setString(5,  (String) mycust[4]);

ps.addBatch();
}

int results[]= ps.executeBatch();

for(int x : results) {
System.out.println(x);
}

System.out.println("Done !!! ");

}catch(Exception ex) {
ex.printStackTrace();
}finally {
JDBCUtil.cleaup(ps, con);
}
}
}
```

Lab17.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab17 {
public static void main(String[] args) {
```

```
Connection con=null;
PreparedStatement ps=null;

Object [][] mycustomers= {
{507,"dd","dd@jlc",444,"Hyd"},
{508,"ee","ee@jlc",555,"Hyd"},
{509,"ff","ff@jlc",666,"Blore"},
{510,"ff","ff@jlc",666,"Blore"},
};

try {
con=JDBCUtil.getConnection();

String SQL="insert into mycustomers values(?,?,?,?,?)";
ps=con.prepareStatement(SQL);

for( Object [] mycust : mycustomers) {
int paramNum=1;
for(Object obj:mycust) {
ps.setObject(paramNum++, obj);
}

ps.addBatch();
}

int results[]= ps.executeBatch();

for(int x : results) {
System.out.println(x);
}
System.out.println("Done !!! ");
}catch(Exception ex) {
ex.printStackTrace();
}finally {
JDBCUtil.cleaup(ps, con);
}
}
}
```

Batch Updates Questions

Q58) What are Batch Updates? Or what are the advantages of Batch Update?

Q59) How to use batch update with Statement?

Q60) How to use batch update with PreparedStatement?

Q61) Can I submit insert statement using Batch Update?

Q62) Can I submit update statement using Batch Update?

Q63) Can I submit delete statement using Batch Update?

Q64) Can I submit select statement using Batch Update?

Q65) Can I submit different type of SQL Statement with Batch Update using Statement?

Q66) Can I submit different type of SQL Statement with Batch Update using PreparedStatement?

1.7 ResultSets

- 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 methods:
  - ResultSet rs=st.executeQuery("select ... "); // Statement interface
  - ResultSet rs=ps.executeQuery(); // PreparedStatement interface

- Assume that there is a table called jlcstudents with 4 columns - sid,sname,email,phone.
- Case 1:

```
String sql = "select * from jlcstudents";
rs = st.executeQuery(sql);
```

- Case 2:

```
String sql = "select sid,phone from jlcstudents";
rs = st.executeQuery(sql);
```

- 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 ResultSet pointer in the forward direction.

```
public boolean next() {
        Checks whether next record is available or not.
        If Next record is available  then
            moves the pointer to next Record
                return true;
        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() {
        Checks whether previous record is available or not.
        If Previous record is available  then
            moves the pointer to previous Record
                return true;
        If Previous record is not available  then
            moves the pointer to previous position
                return false;
}
```

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

```
public int getInt(int columnIndex)
public int getInt(String columnName)
public String getString(int columnIndex)
public String getString(String columnName)
etc
```

Types of ResultSets:

- Depending on the ResultSet cursor movements, you can divide the ResultSet into 2 types.
  - Forward-Only ResultSets
  - Scrollable ResultSets

1.7.1 Forward-Only ResultSets

- When ResultSet 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.prepareStatement("select .... "); rs=ps.executeQuery()

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

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

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

- You can use the following methods on Forward-only ResultSet:

```
public  boolean next()                  public  void close()
public  boolean isBeforeFirst()         public  boolean isAfterLast()
public  boolean isFirst()               public  boolean isLast()
public  int getRow()                    public  XXX getXXX(int)
public  XXX getXXX(String)              etc
```

1.7.2 Scrollable ResultSets

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

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

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

- You can use the following methods on Scrollable ResultSet.

```
public  boolean next()
public  void close()
public  boolean isBeforeFirst()
public  boolean isAfterLast()
public  boolean isFirst()
public  boolean isLast()
public  int getRow()
public  void beforeFirst()
public  void afterLast()
public  boolean first()
public  boolean last()
public  boolean absolute(int)
public  boolean relative(int)
public  boolean previous()
public  XXX getXXX(int)
public  XXX getXXX(String)
etc
```

Lab18.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab18 {
public static void displayRow(ResultSet rs) throws SQLException{
int cid= rs.getInt(1);
String cn=rs.getString(2);
String em=rs.getString(3);
int ph=rs.getInt(4);
String ci=rs.getString(5);

System.out.println(cid+"\t"+cn+"\t"+em+"\t"+ph+"\t"+ci);
}
public static void main(String[] args) {

Connection con=null;
Statement st=null;
ResultSet rs=null;
try {
con=JDBCUtil.getConnection();
String SQL="select * from mycustomers";
st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_
ONLY);
rs=st.executeQuery(SQL);

System.out.println("RS Type : "+st.getResultSetType());
System.out.println("RS Updatability : "+st.getResultSetConcurrency());
System.out.println("RS Holdability : "+st.getResultSetHoldability());

System.out.println("---------Forward Order-------");
while(rs.next()) {
displayRow(rs);
}
```

```
System.out.println("---------Reverse Order-------");
while(rs.previous()) {
displayRow(rs);
}

System.out.println(rs.isBeforeFirst());
System.out.println(rs.isFirst());
rs.first();
System.out.println(rs.isBeforeFirst());
System.out.println(rs.isFirst());

System.out.println("1st Record");
displayRow(rs);

System.out.println(rs.isAfterLast());
System.out.println(rs.isLast());
rs.last();
System.out.println(rs.isAfterLast());
System.out.println(rs.isLast());

System.out.println("Last Record");
displayRow(rs);

rs.absolute(5);

System.out.println("5th Record");
displayRow(rs);

rs.relative(2);
System.out.println("7th Record");
displayRow(rs);

rs.relative(-3);
System.out.println("4th Record");
displayRow(rs);

System.out.println("----Done-----");
```

```
}catch(Exception ex) {
ex.printStackTrace();
}finally {
JDBCUtil.cleaup(rs, st, con);
}
}
}
```

Types of ResultSets:

- Depending on the ResultSet updations, you can divide the ResultSets into 2 types.
  - Read-only ResultSets.
  - Updatable ResultSets or Dynamic ResultSets.

1.7.3 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, ResultSets are read-only.

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

- You can explicitly specify the ResultSets as read-only as follows:

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

   ps=con.prepareStatement(sql,
ResultSet.TYPE_FORWARD_ONLY,ResultSet.CONCUR_READ_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 isAfterLast()
public  boolean isFirst()          public  boolean isLast()
public  int getRow()               public  XXX getXXX(int)
public  XXX getXXX(String)
etc
```

1.7.4 Updatable ResultSets

- When ResultSet is Updatable then you can do the following operations on ResultSet object
  - Access the data from ResultSet
  - Insert records into ResultSet
  - Update the records of ResultSet
  - Delete the records from ResultSet.

- When ResultSet is Updatable then it must be scrollable.
- By default, ResultSets are not updatable.
- You can explicitly specify the ResultSets as updatable as follows:

```
st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
        rs=st.executeQuery(sql)

ps=con.prepareStatement(sql,ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE
);
        rs=ps.executeQuery()
```

- You can use the following methods on Updatable ResultSet.

```
public  boolean next()
public  void close()
public  boolean isBeforeFirst()
public  boolean isAfterLast()
public  boolean isFirst()
public  boolean isLast()
public  int getRow()
public  void beforeFirst()
public  void afterLast()
public  boolean first()
public  boolean last()
public  boolean absolute(int)
public  boolean relative(int)
public  boolean previous()
public  XXX getXXX(int)
public  XXX getXXX(String)
public void updateXXX(int index,XXX val)
public void moveToInsertRow()
public viod insertRow();
public void updateRow();
public void deleteRow();
etc
```

- ResultSet can not be updatable even if you use ResultSet.CONCUR_UPDATABLE in the following cases:
  - When SELECT statement uses JOINS.
  - When SELECT statement uses Aggregate functions.
  - When SELECT statement uses * instead of column names. (In Oracle only)

Lab19.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab19 {
public static void main(String[] args) {

Connection con=null;
Statement st=null;
ResultSet rs=null;

try {
con=JDBCUtil.getConnection();
String SQL="select * from mycustomers";
st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDAT
ABLE);

rs=st.executeQuery(SQL);

System.out.println("RS Type : "+st.getResultSetType());
System.out.println("RS Updatability : "+st.getResultSetConcurrency());
System.out.println("RS Holdability : "+st.getResultSetHoldability());

//Insert Record into RS
rs.moveToInsertRow();
rs.updateInt(1, 502);
rs.updateString(2, "hai");
rs.updateString(3, "hai@jlc");
rs.updateInt(4, 555);
rs.updateString(5, "Blore");

rs.insertRow(); //Insert the Row in DB
```

```
//Update the Row 6
rs.absolute(6);
rs.updateString(2, "test");
rs.updateString(3, "test@jlc");
rs.updateInt(4, 666);

rs.updateRow(); //update the row with DB

//Delete the Row 8
rs.absolute(8);
rs.deleteRow();

//Delete the Row 9
rs.absolute(9);
rs.deleteRow();

rs.beforeFirst();
System.out.println("---------Forward Order-------");
while(rs.next()) {
int cid= rs.getInt(1);
String cn=rs.getString(2);
String em=rs.getString(3);
int ph=rs.getInt(4);
String ci=rs.getString(5);
System.out.println(cid+"\t"+cn+"\t"+em+"\t"+ph+"\t"+ci);
}
System.out.println("----Done-----");
}catch(Exception ex) {
ex.printStackTrace();
}finally {
JDBCUtil.cleaup(rs, st, con);
}
}
}
```

CONSTANTS Defined in ResultSet interface		
ResultSet Type	ResultSet Concurrency	ResultSet Holdability
TYPE_FORWARD_ONLY 1003 TYPE_SCROLL_INSENSITIVE 1004 TYPE_SCROLL_SENSITIVE 1005	CONCURR_READ_ONLY 1007 CONCURR_UPDATABLE 1008	HOLD_CURSORS_OVER_COMMIT 1 CLOSE_CURSORS_AT_COMMIT 2

- You can use the following method of Statement interface to find the ResultSet Type:
  public int getResultSetType()

- You can use the following method of Statement interface to find the ResultSet Concurrency:
  public int getResultSetConcurrency()

- You can use the following method of Statement interface to find the ResultSet Holdability:
  public int getResultSetHoldability()

- You can use the following methods of Connection interface to create the Statement:
  Statement createSatement()
  Statement createSatement(int rsType, int rsConcurrency)
  Statement createSatement(int rsType, int rsConcurrency, int rsHoldability)

- You can use the following methods of Connection interface to create the PreparedStatement:

PreparedStatement prepareSatement(String sql)
PreparedStatement prepareSatement(String sql, int rsType, int rsConcurrency)
PreparedStatement prepareSatement(String sq, int rsType, int rsConcurrency, int rsHoldability)

ResultSets Questions

Q67) What is ResultSet?

Q68) In which package, ResultSet is available?

Q69) Who has defined the subclass of ResultSet?

Q70) How can you get the Object of ResultSet type?

Q71) What does the ResultSet represent?

Q72) What are the types of ResultSet available as per Cursor Movement?

Q73) What is Forward Only ResultSet?

Q74) How can you get the Forward Only ResultSet?

Q75) Can I call the following method with Forward Only ResultSet?

a. previous()
b. first()
c. last()
d. absolute()
e. relative()

Q76) What is Scrollable ResultSet?

Q77) How can I get the Scrollable ResultSet?

Q78) Can I call the following method with Scrollable ResultSet?

a. previous()
b. first()
c. last()
d. absolute()
e. relative()

Q79) What are the types of ResultSet available as per Updatable Operation?

Q80) What is Read Only ResultSet?

Q81) How can you get the Read Only ResultSet?

Q82) Can I call the following method with Read Only ResultSet?

a. moveToInsertRow()
b. updateRow()
c. deleteRow()

d. insertRwo()
e. updateX(int colIndex,X value)

Q83) What is Updatable ResultSet?

Q84) How can you get the Updatable ResultSet?

Q85) Can I call the following method with Updatable ResultSet?

a. moveToInsertRow()
b. updateRow()
c. deleteRow()
d. insertRwo()
e. updateX(int colIndex,X value)

Q86) What is the default type of ResultSet?

Q87) What are the constants defined to specify the ResultSet type?

Q88) What is the default concurrency of ResultSet?

Q89) What are the constants defined to specify the ResultSet concurrency?

Q90) What is ResultSet holdability?

Q91) What are the constants defined to specify the ResultSet holdability?

1.8 ResultSetMetaData

- ResultSetMetaData is an interface available in java.sql package.
- Subclass of ResultSetMetaData interface is provided by Driver vendor.
- ResultSetMetaData is used to get the information about your ResultSet object.
- You can use the following method of ResultSet to get the ResultSetMetaData object.
    - **public ResultSetMetaData getMetaData()**

Lab20.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab20 {
public static void main(String[] args) {

Connection con = null;
Statement st = null;
ResultSet rs = null;

try {
con = JDBCUtil.getConnection();
String SQL = "select * from mycustomers";
st = con.createStatement();
rs = st.executeQuery(SQL);

ResultSetMetaData rsmd = rs.getMetaData();

int cc= rsmd.getColumnCount();
System.out.println("Col Count : "+cc);

rs.first();
```

```
for(int i=1;i<=cc;i++) {
		String colName= rsmd.getColumnName(i);
		System.out.println(colName);
		String colLabel = rsmd.getColumnLabel(i);
		System.out.println(colLabel);
		String colType = rsmd.getColumnTypeName(i);
		System.out.println(colType);
		String colClsName = rsmd.getColumnClassName(i);
		System.out.println(colClsName);
		String tabName = rsmd.getTableName(i);
		System.out.println(tabName);
		String catName = rsmd.getCatalogName(i);
		System.out.println(catName);
		System.out.println("------------");
}

System.out.println("----Done-----");

} catch (Exception ex) {
ex.printStackTrace();
} finally {
JDBCUtil.cleaup(rs, st, con);
}
}
}
```

1.9 DatabaseMetaData

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

Lab21.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab21 {
public static void main(String[] args) {

Connection con = null;

try {
con = JDBCUtil.getConnection();
DatabaseMetaData dbmd= con.getMetaData();

System.out.println(dbmd.getDatabaseProductName());
System.out.println(dbmd.getDatabaseProductVersion());
System.out.println(dbmd.getDatabaseMajorVersion());
System.out.println(dbmd.getDatabaseMinorVersion());
System.out.println(dbmd.getDefaultTransactionIsolation());//2
System.out.println(dbmd.getDriverName());
System.out.println(dbmd.getJDBCMajorVersion());
System.out.println(dbmd.getJDBCMinorVersion());

System.out.println(dbmd.supportsBatchUpdates());
System.out.println(dbmd.supportsFullOuterJoins());
System.out.println(dbmd.supportsTransactions());
```

```
System.out.println(dbmd.supportsGroupBy());
System.out.println(dbmd.supportsMultipleTransactions());
System.out.println(dbmd.supportsMultipleTransactions());
System.out.println(dbmd.supportsMultipleResultSets());

System.out.println("----Done-----");
} catch (Exception ex) {
ex.printStackTrace();
} finally {
JDBCUtil.cleaup(null, con);
}
}
}
```

MetaData Questions

Q92) What is DatabaseMetaData?

Q93) In which package the DatabaseMetaData is available?

Q94) Who has defined the subclass for DatabaseMetaData?

Q95) How can you get the object of DatabaseMetaData type?

Q96) What is the use of DatabaseMetadata?

Q97) How can I access the Database Product Name?

Q98) How can I access the Database Product Version?

Q99) How can I access the Driver Name?

Q100) How can I access the Driver Version?

Q101) How can I check whether Database supports batch update or not?

Q102) How can I check whether Database supports Full Outer Join or not?

Q103) How can I check whether Database supports Transactions or not?

Q104) How can I check whether Database supports supportsGroupBy or not?

Q105) What is the DefaultTransactionIsolation of MYSQL?

Q106) What is ResultSetMetaData?

Q107) In which package the ResultSetMetaData is available?

Q108) Who has defined the subclass for ResultSetMetaData?

Q109) How can I get the object of ResultSetMetaData type?

Q110) What is the use of ResultSetMetaData?

Q111) How can I get the number of columns available in ResultSet?

Q112) How can I access the Name of the columns available in ResultSet?

Q113) How can I access the type of the columns available in ResultSet?

Q114) How can I access the Table of the columns available in ResultSet?

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

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

Types of Transactions

- Local Transactions
- Distributed Transactions

Local Transactions

- When a Single Database is participating in the Transactional Operations then it is called as Local Trnsactions.
  Ex:
    - Transfer the funds from one account to another account where two accounts are in same bank or same database.

Distributed Transactions

- When two or more Databases are participating in the Transactional Operations then it is called as Distributed Transactions.
  Ex:
    - Transfer the funds from one account to another account where two accounts are in different banks or different databases.

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

JDBC Transaction Management:

- **Specifying the Transactional Boundaries.**

```
Connection con=null;
try{
con=...;
con.setAutoCommit(false);          // Transaction Begin
  op1;
  op2;
  op3;
con.commit();                      // Transaction End
}catch(Exception e){
if(con!=null){
 con.rollback();                   // Transaction End
}
}
```

- When multiple transactions 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 Constants in Connection interface as follows.

  - **TRANSACTION_READ_UNCOMMITTED 1**
  - **TRANSACTION_READ_COMMITTED 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_COMMITTED);**

DB Work for Lab22:

6) Create the following table

```
create table myaccounts(
accno int primary key,
atype char(2),
bal double
);
```

7) Insert some Sample Records

```
insert into myaccounts values(101,'SA',25000);
insert into myaccounts values(102,'SA',5000);
```

Lab22.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
class InvalidAccountNumberException extends RuntimeException {
int accno;

InvalidAccountNumberException() { }

InvalidAccountNumberException(int accno) {
this.accno = accno;
}

public String toString() {
return "Accno : " + accno + "  is Not Found";
}
}
```

```
class InsufficientFundsException extends RuntimeException {
InsufficientFundsException() {
}

public String toString() {
return "Sufficient Funds are Not Available";
}
}

class AccountService {

void fundsTransfer(int saccno, int daccno, double amt) {

Connection con = null;
PreparedStatement ps1 = null;
PreparedStatement ps2 = null;
ResultSet rs = null;

try {
con = JDBCUtil.getConnection();

con.setAutoCommit(false); // Tx Begin

String SQL1 = "select bal from myaccounts where accno=?";
String SQL2 = "update myaccounts set bal=? where accno=?";
ps1 = con.prepareStatement(SQL1);
ps2 = con.prepareStatement(SQL2);

// Deposit
ps1.setInt(1, daccno); // 102
rs = ps1.executeQuery(); //OP1-Select
if (rs.next()) {
double bal = rs.getInt(1);
bal = bal + amt;

ps2.setDouble(1, bal);
ps2.setInt(2, daccno);
ps2.executeUpdate(); // OP2-Update
```

```
} else {
throw new InvalidAccountNumberException();
}


// Withdraw
ps1.setInt(1, saccno); // 101
rs = ps1.executeQuery(); //OP3-Select
if (rs.next()) {
double bal = rs.getInt(1);
if (bal >= amt) {
bal = bal - amt;

ps2.setDouble(1, bal);
ps2.setInt(2, saccno);
ps2.executeUpdate(); // OP4-Update
} else {
throw new InsufficientFundsException();
}
} else {
throw new InvalidAccountNumberException();
}
con.commit(); // End of Tx
System.out.println("----Done-----");

} catch (Exception ex) {
ex.printStackTrace();

try {
con.rollback(); // End of Tx
} catch (SQLException e) {
e.printStackTrace();
}

} finally {
JDBCUtil.cleaup(rs,ps1, con);
}
}
}
```

```
public class Lab22 {
public static void main(String[] args) {

AccountService as = new AccountService();
as.fundsTransfer(101, 102, 5000);
}
}
```

JDBC Transaction Management Questions

Q115) What is Transaction?

Q116) What Transaction Guarantees?

Q117) What are the transactional concurrency problems?

Q118) What are the Transactional Isolation Levels?

Q119) What is the Default auto commit status in JDBC?

Q120) What is the Default auto commit status in MySQL?

Q121) What is the Default auto commit status in Oracle?

Q122) What will happen when auto commit is true?

Q123) How to manage the Transaction in JDBC?

Q124) How to begin the Transaction in JDBC?

Q125) How to end the Transaction in JDBC?

Q126) What will hapen when i am not specifying the Isolation Level with JDBC?

Q127) How can I get Database Vendor Specific Default Transactional Isolation Level?

Q128) What is the Default Transactional Isolation Level of MySQL?

Q129) What is the Default Transactional Isolation Level of Oracle?

11. RowSets

- RowSet is an interface available in javax.sql package.
- RowSet interface is extending ResultSet interface.
- RowSet interface implementation classes are provided by Java vendor.
- RowSet funcationality is similar to ResultSet.

ResultSet	RowSet
ResultSet object is used to store records returned by SELECT SQL Statement.	RowSet object is also used to store records returned by SELECT SQL Statement.

ResultSet object can be created as follows:
 	
con=D.M.getConnection(url,un,pw);
st=con.createStatement();
rs=st.executeQuery(sql);
	RowSet object can be created as follows:
	
	RowSet jrs = new JdbcRowSetImpl();
	jrs.setUrl(url);
	jrs.setUsername(un);
	jrs.setPassword(pw);
	jrs.setCommand(sql);
	jrs.execute();

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 , ResultSet also will be closed automatically.	RowSets are connection less objects i.e you can access the RowSet data without Connection.
ResultSet objects are not eligible for Serialization.	RowSet objects are eligible for Serialization.

- **Following are subtypes of RowSet interface:**
  - JdbcRowSet interface.
  - CachedRowSet interface.
  - WebRowSet interface.
  - JoinRowSet interface.

- **Types of Rowsets:**
  - Connected RowSets
  - Disconnected RowSets

Connected RowSets:

- These rowsets establish a connection with the database and retain it.
- Connected RowSets are like ResultSets i.e Connected RowSets needs the connection as long as you are accessing the RowSet data.
- You can not serialize connected rowsets.
- JdbcRowSet is connected rowSet.

Disconnected RowSets:

- These rowsets establishes a connection, executes a query and closes the connection.
- Disconnected RowSets are not like ResultSets i.e Disconnected RowSets do not need the connection always while you access the RowSet data.
- You can serialize disconnected rowsets.
- CachedRowSet is disconnected rowset.

Lab23.java

```
package com.jlcindia.jdbc;

import java.sql.SQLException;
import javax.sql.RowSet;
import com.sun.rowset.CachedRowSetImpl;
import com.sun.rowset.JdbcRowSetImpl;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab23 {
public static void main(String[] args) {

RowSet myrowset = null;
try {

String SQL = "select * from mycustomers where city='Blore'";
```

```
//myrowset = new JdbcRowSetImpl();
myrowset = new CachedRowSetImpl();
myrowset.setUrl("jdbc:mysql://localhost:3306/myjdbcdb");
myrowset.setUsername("root");
myrowset.setPassword("srinivas");
myrowset.setCommand(SQL);

myrowset.execute(); // ***

while (myrowset.next()) {
int cid = myrowset.getInt(1);
String cn = myrowset.getString(2);
String em = myrowset.getString(3);
int ph = myrowset.getInt(4);
String ci = myrowset.getString(5);

System.out.println(cid + "\t" + cn + "\t" + em + "\t" + ph + "\t" + ci);
}
System.out.println("----Done-----");
} catch (Exception ex) {
ex.printStackTrace();
} finally {
try {
myrowset.close();
} catch (SQLException e) {
e.printStackTrace();
}
}
}
}
```

RowSets – Questions

Q131) What is RowSet?

Q132) What is the super type for RowSet?

Q133) How to get the object of RowSet?

Q134) How many sub types of RowSet interface available?

Q135) What is the default type of RowSet?

Q136) What is the default concurrency of RowSet?

Q137) Can I serialize the JdbcRowSet?

Q138) Can I serialize the CachedRowSet?

Q139) What is the difference between ResultSet and RowSet?

12. Connection Pooling

- JDBC supports two ways 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.

  Class.forName("oracle.jdbc.driver.OracleDriver");
  con=DriverManager.getConnection("jdbc:mysql://localhost:3306/myjdbcdb","root","srinivas");

- When you use DriverManager Connections, You will get 2 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 problems. (Note : We have alreday solved this using JDBCUtil).

  - When you call getConnection(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 Connection will be closed permenently. Creating and destroying connections every time for every user is expensive and may damage your application performance.

- You can solve these problems with DataSource Connections.

DataSource Connections

- DataSource Connections are Pooled Connections.
- Connection pool is special area which is containing set of reusable database conections i.e multiple database connections will be created and will 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.
- Following are Third Party DataSources / Connection Pooling Algorithms
  - Hikari DataSource
  - Tomcat DataSource
  - DBCP2
  - DBCP
  - C3P0

Lab24.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import org.apache.commons.dbcp.BasicDataSource;
import com.jlcindia.jdbc.util.JDBCUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab24 {
public static void main(String[] args) {

Connection con = null;
PreparedStatement ps=null;
ResultSet rs=null;
try {
BasicDataSource myds = new BasicDataSource();

myds.setDriverClassName("com.mysql.jdbc.Driver");
myds.setUrl("jdbc:mysql://localhost:3306/myjdbcdb");
myds.setUsername("root");
myds.setPassword("srinivas");
myds.setInitialSize(10);
myds.setMaxActive(100);
con=myds.getConnection();

String SQL = "select * from mycustomers";
ps=con.prepareStatement(SQL);
rs=ps.executeQuery();

while (rs.next()) {
int cid = rs.getInt(1);
String cn = rs.getString(2);
String em = rs.getString(3);
int ph = rs.getInt(4);
String ci = rs.getString(5);
System.out.println(cid + "\t" + cn + "\t" + em + "\t" + ph + "\t" + ci);
}
```

```
System.out.println("----Done-----");
} catch (Exception ex) {
ex.printStackTrace();
} finally {
JDBCUtil.cleanup(rs ,ps, con);
}

}
}
```

Connection Pooling - Questions

Q140) What are the ways available to manage the Connections in JDBC?

Q141) What are the advantages of DataSource Connections over DriverManager Connections?

Q142) What are the Third Party Connection Pooling Algorithms available?

Q143) How to Use DBCP Connection Pooling?

DataSourceUtil.java

```
package com.jlcindia.jdbc.util;

import java.sql.*;
import org.apache.commons.dbcp.BasicDataSource;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class DataSourceUtil {

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) {
        e.printStackTrace();
        }

}

public static Connection getConnection() {

        Connection con = null;
        try {
        BasicDataSource myds = new BasicDataSource();
        //myds.setDriverClassName("com.mysql.jdbc.Driver");
        myds.setUrl("jdbc:mysql://localhost:3306/myjdbcdb");
        myds.setUsername("root");
        myds.setPassword("srinivas");
        myds.setInitialSize(10);
        myds.setMaxActive(100);

        con = myds.getConnection();
        } catch (Exception ex) {
        ex.printStackTrace();
        }

        return con;

}

}
```

13. Working with dates

- Database tables you are interacting with can have the columns of Date type.
- When you are **inserting values into Date type column** then you have to use the following
  - ps.setDate(3,dob);
- When you are **selecting values into Date type column** then you have to use the following
  - Date dob= rs.getDate(3);

DB Work for Lab25:

1) Create the following table

```
create table mystudents(
sid int primary key,
sname char(10),
dob date);
```

Lab25.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.DataSourceUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab25 {
public static void main(String[] args) {
Connection con = null;
PreparedStatement ps=null;

int dd = 16;
int mm = 6;
int yy = 1979;
Date dob = new Date(yy - 1900, mm - 1, dd); //1
```

```
try {

con=DataSourceUtil.getConnection();

String SQL = "insert into mystudents values(?,?,?)";
ps=con.prepareStatement(SQL);
ps.setInt(1, 103);
ps.setString(2, "Srinivas");
ps.setDate(3,dob); //2

int x= ps.executeUpdate();

System.out.println(x);

System.out.println("----Done-----");
} catch (Exception ex) {
ex.printStackTrace();
} finally {
DataSourceUtil.cleanUp(ps, con);
}

}
}
```

Lab26.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import java.text.SimpleDateFormat;
import com.jlcindia.jdbc.util.DataSourceUtil;
/*
* @Author     : Srinivas Dande
* @Company : Java Learning Center
* @See           : myjavalearningcenter.com
* */

public class Lab26 {
public static void main(String[] args) {
```

```
Connection con = null;
PreparedStatement ps=null;
ResultSet rs=null;

try {
con=DataSourceUtil.getConnection();

String SQL = "select * from mystudents";
ps=con.prepareStatement(SQL);

rs=ps.executeQuery();

while(rs.next()) {
int sid=rs.getInt(1);
String sname=rs.getString(2);
Date dob= rs.getDate(3); //1
SimpleDateFormat sdf = new SimpleDateFormat("dd-MMM-yyyy");
String mydob = sdf.format(dob); //2

System.out.println(sid+"\t"+sname+"\t"+mydob);
}
System.out.println("----Done-----");

} catch (Exception ex) {
ex.printStackTrace();
} finally {
DataSourceUtil.cleanUp(ps, con);
}

}
}
```

14. JDBC New Features

New Features added in JDBC4.0

1) Auto-loading of JDBC driver class
2) SQL exception handling enhancements
3) Connection management enhancements

New Features added in JDBC4.1

1) try with resource/ Automatic Resource Management.
2) RowSetFactory Interface and RowSetProvider Class

New Features added in JDBC4.2

1) Addition of REF_CURSOR support
2) Addition of the java.sql.JDBCType Enum
3) Add support for large update counts

14.1. Auto-loading of JDBC driver class

- Before to JDBC4.0 , you need to get the Connection object as follows:
  - **Class.forName("com.mysql.jdbc.Driver");**
  - **con=DriverManager.getConnection(url,un,pw);**

- **DriverManager.getConnection()** method is doing the following tasks:
  - Takes the URL.
  - Identifies the Database related to the URL given
  - Checks whether the Driver class related to given URL is loaded or not.
  - If loaded then creates the Connection interface sub class object provided by that database vendor.
  - If not loaded then gives the SQLException with message - No suitable Driver found.

- From JDBC4.0 onwards, you need to get the Connection object as follows:
  - **con=DriverManager.getConnection(url,un,pw);**

- **DriverManager.getConnection()** method is doing the following tasks:
  - Takes the URL.
  - Identifies the Database related to the URL given.
  - Loads the required driver class automatically when driver class is not loaded (if the driver class is found in application classpath).
  - Creates the Connection interface sub class object provided by that database vendor.

14.2. SQL exception handling enhancements

- From JDBC 4.0 onwards, SQLException object is an Iterator which can hold multiple Exception objects raised in JDBC code.

Before JDBC4.0	From JDBC4.0

try{
...
}catch(SQLException ex){
ex.printStackTrace();
}
}
	try{
	...
	}catch(SQLException ex){
	for(Throwable e:ex){
	System.out.println(e);
	e.printStackTrace();
	}
	}

14.3. Connection management enhancements

- You have to use Database vendor provided DataSource class to get the connection object.
- For Oracle Database :

```
OracleDataSource ods = new OracleDataSource();
ods.setURL("jdbc:oracle:thin:@localhost:1521:XE");
ods.setUser("system");
ods.setPassword("srinivas");
con = ods.getConnection();
```

14.4. try with Resource

- **Upto Java 6**, when you are initializing resources to establish the connection between Java program and IO Device or DB Engine etc then you must have to close the resources explicitly using finally block.
- **From Java 7**, new feature added called **try-with resources** which helps to close or clean the resources automatically.

Syntax:

```
try(<Resources Declaration and Initialization>) {
…
}catch(<throwableType> refVar){
…
}
```

Before Java 7	**From Java 7**

Connection con = null;
Statement st = null;
try{
con = DM.getConnection(…);
st = con.createStatement();
…
}catch(SQLException e){
…
}finally{
if(st!=null)
st.close();
if(con!=null)
con.close();
}
	try(
	Connection con =DM.getConnection("");
	Statement st = con.createStatement();
	)
	{
	…
	}catch(SQLException e){
	…
	}

- A resource is as an object that must be closed after the program execution is completed.
- The try-with-resources statement ensures that each resource is closed at the end of the statement. Any object that implements java.lang.AutoCloseable can be used as a resource.
- When a class is implementing java.lang.AutoCloseable interface then only those class objects can be used with try-with resources for auto cleanup.
- With try-with resources try block can be used without catch and finally blocks also.

Lab27.java

```
package com.jlcindia.jdbc;

import java.sql.*;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class Lab27 {
public static void main(String[] args) {

String URL="jdbc:mysql://localhost:3306/myjdbcdb";
String UN="root";
String PW="srinivas";
String SQL="select * from mycustomers ";

try(Connection con =DriverManager.getConnection(URL,UN,PW);
Statement st=con.createStatement();
ResultSet rs=st.executeQuery(SQL)  ) {

while(rs.next()) {
int cid= rs.getInt(1);
String cn=rs.getString(2);
String em=rs.getString(3);
int ph=rs.getInt(4);
String ci=rs.getString(5);

System.out.println(cid+"\t"+cn+"\t"+em+"\t"+ph+"\t"+ci);
}
System.out.println("----Done-----");
}catch(Exception ex) {
ex.printStackTrace();
}
}
}
```

14.5 Addition of REF_CURSOR support

- The REF_CURSOR data type is supported by several databases. To return a REF_CURSOR from a stored procedure you can use this type.
- To check whether JDBC Driver supports REF_CURSOR you can invoke the supportsRefCursors () method with DatabaseMetaData.

1.14.6 Addition of the java.sql.JDBCType Enum

- This enum identifies generic SQL Types, called JDBC Types. You can use JDBCType in place of the constants defined in Types.

1.14.7 Add support for large update counts

- JDBC methods returned only int value for the update count.
- Since datasets continue to grow, this can cause problems.
- So following methods are added to the Statement that return a long value for the update count.
    - public long getLargeUpdateCount()
    - public long getLargeMaxRows()
    - public long[] executeLargeBatch()
    - public long executeLargeUpdate(java.lang.String)

Lab28.java

```
package com.jlcindia.jdbc;

import java.sql.*;
import com.jlcindia.jdbc.util.DataSourceUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */

public class Lab28 {
public static void main(String[] args) {
```

```
Connection con = null;
PreparedStatement ps=null;

try {

con=DataSourceUtil.getConnection();

String SQL = "update mycustomers set city='Bangalore'";
ps=con.prepareStatement(SQL);

long x= ps.executeLargeUpdate();

System.out.println(x);

System.out.println("----Done-----");
} catch (Exception ex) {
ex.printStackTrace();
} finally {
DataSourceUtil.cleanUp(ps, con);
}
}
}
```

New Features - Questions

Q144) What are the new features of JDBC 4.0?

Q145) What are the new features of JDBC 4.1?

Q146) What are the new features of JDBC 4.2?

Q147) How can we restrict the user to access only specific data from table?

15. DAO and JDBC Example

Lab29: Files required

1. Lab29.java	2. DAOFactory.java
3. Customer.java	4. DataSourceUtil.java
5. CustomerDAO.java	6. CustomerDAOImpl.java

Lab29.java

```
package com.jlcindia.client;

import java.util.List;
import com.jlcindia.dao.CustomerDAO;
import com.jlcindia.dao.DAOFactory;
import com.jlcindia.dto.Customer;
/*
* @Author     : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */

public class Lab29 {
public static void main(String[] args) {

CustomerDAO custDAO= DAOFactory.getCustomerDAO();

//1.Add Customer
Customer mycust1=new Customer(110,"cc","cc@jlc",33333,"Delhi");

int x= custDAO.addCustomer(mycust1);
System.out.println(x);

//2.Update Customer
Customer mycust2=new Customer(101,"Srinivas","sri@jlc",12345,"Blore");
int y= custDAO.updateCustomer(mycust2);
System.out.println(y);
```

```
//3.Delete Customer
int z= custDAO.deleteCustomer(502);
System.out.println(z);

//4.Get Customer By Cid
Customer cust1 = custDAO.getCustomerByCid(101);
System.out.println(cust1);

Customer cust2 = custDAO.getCustomerByCid(201);
System.out.println(cust2);

//5.Get Customer By Email
Customer cust3 = custDAO.getCustomerByEmail("test@jlc");
System.out.println(cust3);

Customer cust4 = custDAO.getCustomerByEmail("test@cc");
System.out.println(cust4);

//6.Get All Customers
List<Customer> mylist1= custDAO.getAllCustomers();
for(Customer mycust : mylist1) {
System.out.println(mycust);
}

//7.Get Customer By City
List<Customer> mylist2= custDAO.getCustomersByCity("Bangalore");
for(Customer mycust : mylist2) {
System.out.println(mycust);
}
}
}
```

2) DAOFactory.java

```
package com.jlcindia.dao;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class DAOFactory {

        static CustomerDAO customerDAO;
        static {
                customerDAO = new CustomerDAOImpl();
        }
        public static CustomerDAO getCustomerDAO() {
                return customerDAO;
        }
}
```

3) Customer.java

```
package com.jlcindia.dto;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
//DTO or TO
public class Customer {

private int cid;
private String cname;
private String email;
private int phone;
private String city;

public Customer() {}
public Customer(int cid, String cname, String email, int phone, String city) {
        super();
        this.cid = cid;
        this.cname = cname;
        this.email = email;
```

```
		this.phone = phone;
		this.city = city;
}
public int getCid() {
		return cid;
}
public void setCid(int cid) {
		this.cid = cid;
}
public String getCname() {
		return cname;
}
public void setCname(String cname) {
		this.cname = cname;
}
public String getEmail() {
		return email;
}
public void setEmail(String email) {
		this.email = email;
}
public int getPhone() {
		return phone;
}
public void setPhone(int phone) {
		this.phone = phone;
}
public String getCity() {
		return city;
}
public void setCity(String city) {
		this.city = city;
}
@Override
public String toString() {
		return cid + ", " + cname + ", " + email + ", " + phone + ", " + city;
}
}
```

4) DataSourceUtil.java

```
package com.jlcindia.util;

import java.sql.*;
import org.apache.commons.dbcp.BasicDataSource;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class DataSourceUtil {

public static Connection getConnection() {
        Connection con = null;
        try {
        BasicDataSource myds = new BasicDataSource();
        //myds.setDriverClassName("com.mysql.jdbc.Driver");
        myds.setUrl("jdbc:mysql://localhost:3306/myjdbcdb");
        myds.setUsername("root");
        myds.setPassword("srinivas");
        myds.setInitialSize(10);
        myds.setMaxActive(100);

        con = myds.getConnection();
        } catch (Exception ex) {
        ex.printStackTrace();
        }
        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) {
        e.printStackTrace();
        }
}
}
```

5) CustomerDAO.java

```
package com.jlcindia.dao;

import java.util.List;
import com.jlcindia.dto.Customer;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public interface CustomerDAO {
        public int addCustomer(Customer cust);
        public int updateCustomer(Customer cust);
        public int deleteCustomer(int cid);

        public Customer getCustomerByCid(int cid);
        public Customer getCustomerByEmail(String email);
        public Customer getCustomersByPhone(int phone);

        public List<Customer> getAllCustomers();
        public List<Customer> getCustomersByCity(String city);

         public Customer getCustomersByPhone(int phone) ;
         public List<Customer> getCustomersByCname(String cname) ;
}
```

6) CustomerDAOImpl.java

```
package com.jlcindia.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;
import com.jlcindia.dto.Customer;
import com.jlcindia.util.DataSourceUtil;
/*
* @Author     : Srinivas Dande
* @Company : Java Learning Center
* @See           : myjavalearningcenter.com
* */
public class CustomerDAOImpl implements CustomerDAO {

@Override
public int addCustomer(Customer cust) {
System.out.println("--addCustomer--");
Connection con = null;
PreparedStatement ps = null;
int x = 0;

try {
con = DataSourceUtil.getConnection();
String SQL = "insert into mycustomers values(?,?,?,?,?)";
ps = con.prepareStatement(SQL);

ps.setInt(1, cust.getCid());
ps.setString(2, cust.getCname());
ps.setString(3, cust.getEmail());
ps.setInt(4, cust.getPhone());
ps.setString(5, cust.getCity());

x = ps.executeUpdate();

} catch (Exception ex) {
ex.printStackTrace();
} finally {
```

```
DataSourceUtil.cleanUp(ps, con);
}

return x;
}

@Override
public int updateCustomer(Customer cust) {
System.out.println("--updateCustomer--");
Connection con = null;
PreparedStatement ps = null;
int x = 0;

try {
con = DataSourceUtil.getConnection();
String SQL = "update mycustomers set cname=? , email=? , phone=?, city=? where cid=?";
ps = con.prepareStatement(SQL);

ps.setString(1, cust.getCname());
ps.setString(2, cust.getEmail());
ps.setInt(3, cust.getPhone());
ps.setString(4, cust.getCity());
ps.setInt(5, cust.getCid());

x = ps.executeUpdate();

} catch (Exception ex) {
ex.printStackTrace();
} finally {
DataSourceUtil.cleanUp(ps, con);
}

return x;
}
```

```
@Override
public int deleteCustomer(int cid) {
System.out.println("--deleteCustomer--");
Connection con = null;
PreparedStatement ps = null;
int x = 0;

try {
con = DataSourceUtil.getConnection();
String SQL = "delete from mycustomers where cid=?";
ps = con.prepareStatement(SQL);

ps.setInt(1, cid);

x = ps.executeUpdate();

} catch (Exception ex) {
ex.printStackTrace();
} finally {
DataSourceUtil.cleanUp(ps, con);
}

return x;
}

@Override
public Customer getCustomerByCid(int cid) {
System.out.println("--getCustomerByCid--");

Connection con = null;
PreparedStatement ps = null;
ResultSet rs = null;

Customer cust = null;
try {
con = DataSourceUtil.getConnection();
String SQL = "select * from mycustomers where cid=?";
ps = con.prepareStatement(SQL);
ps.setInt(1, cid);
```

```
rs = ps.executeQuery();

if (rs.next()) {
cust = new Customer();
cust.setCid(rs.getInt(1));
cust.setCname(rs.getString(2));
cust.setEmail(rs.getString(3));
cust.setPhone(rs.getInt(4));
cust.setCity(rs.getString(5));
}

} catch (Exception ex) {
ex.printStackTrace();
} finally {
DataSourceUtil.cleanUp(rs, ps, con);
}

return cust;
}

@Override
public Customer getCustomerByEmail(String email) {
System.out.println("--getCustomerByEmail--");

Connection con = null;
PreparedStatement ps = null;
ResultSet rs = null;

Customer cust = null;
try {
con = DataSourceUtil.getConnection();
String SQL = "select * from mycustomers where email=?";
ps = con.prepareStatement(SQL);
ps.setString(1, email);

rs = ps.executeQuery();

if (rs.next()) {
cust = new Customer();
```

```
cust.setCid(rs.getInt(1));
cust.setCname(rs.getString(2));
cust.setEmail(rs.getString(3));
cust.setPhone(rs.getInt(4));
cust.setCity(rs.getString(5));
}

} catch (Exception ex) {
ex.printStackTrace();
} finally {
DataSourceUtil.cleanUp(rs, ps, con);
}

return cust;
}

@Override
public List<Customer> getAllCustomers() {
System.out.println("--getAllCustomers--");

Connection con = null;
PreparedStatement ps = null;
ResultSet rs = null;

List<Customer> mylist = new ArrayList<>(); //Empty
try {
con = DataSourceUtil.getConnection();
String SQL = "select * from mycustomers";
ps = con.prepareStatement(SQL);

rs = ps.executeQuery();


while (rs.next()) {
Customer  cust = new Customer();

cust.setCid(rs.getInt(1));
cust.setCname(rs.getString(2));
cust.setEmail(rs.getString(3));
cust.setPhone(rs.getInt(4));
```

```
cust.setCity(rs.getString(5));

mylist.add(cust);
}

} catch (Exception ex) {
ex.printStackTrace();
} finally {
DataSourceUtil.cleanUp(rs, ps, con);
}

return mylist;
}

@Override
public List<Customer> getCustomersByCity(String city) {
System.out.println("--getCustomersByCity--");

Connection con = null;
PreparedStatement ps = null;
ResultSet rs = null;

List<Customer> mylist = new ArrayList<>(); //Empty
try {
con = DataSourceUtil.getConnection();
String SQL = "select * from mycustomers where city=?";
ps = con.prepareStatement(SQL);
ps.setString(1, city);
rs = ps.executeQuery();

while (rs.next()) {
Customer  cust = new Customer();

cust.setCid(rs.getInt(1));
cust.setCname(rs.getString(2));
cust.setEmail(rs.getString(3));
cust.setPhone(rs.getInt(4));
cust.setCity(rs.getString(5));

mylist.add(cust);
```

```
}

} catch (Exception ex) {
ex.printStackTrace();
} finally {
DataSourceUtil.cleanUp(rs, ps, con);
}

return mylist;
}

@Override
public Customer getCustomersByPhone(int phone) {
  //You need to implement this
return null;
}

@Override
public List<Customer> getCustomersByCname(String cname) {
  //You need to implement this
return null;
}

}
```

DAO ,Template and RowMapper Example

Lab30: Files required

1. Lab30.java	Same as Lab29
2. DAOFactory.java	Same as Lab29
3. Customer.java	Same as Lab29
4. DataSourceUtil.java	Same as Lab29
5. RowMapper.java	**New in Lab30**
6. CustomerRowMapper.java	**New in Lab30**
7. JdbcTemplate.java	**New in Lab30**
8. CustomerDAO.java	Same as Lab29
9. CustomerDAOImpl.java	**Updated in Lab30**

5) RowMapper.java

```
package com.jlcindia.rowmappers;

import java.sql.ResultSet;
import java.sql.SQLException;

/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */

public interface RowMapper {
public Object mapRow(ResultSet rs) throws SQLException;
}
```

6) CustomerRowMapper.java

```
package com.jlcindia.rowmappers;

import java.sql.ResultSet;
import java.sql.SQLException;
import com.jlcindia.dto.Customer;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class CustomerRowMapper implements RowMapper {
@Override
public Object mapRow(ResultSet rs)  throws SQLException {
Customer       cust = new Customer();
cust.setCid(rs.getInt(1));
cust.setCname(rs.getString(2));
cust.setEmail(rs.getString(3));
cust.setPhone(rs.getInt(4));
cust.setCity(rs.getString(5));
return cust;
}
}
```

7) JdbcTemplate.java

```
package com.jlcindia.template;

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

import com.jlcindia.rowmappers.RowMapper;
import com.jlcindia.util.DataSourceUtil;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
```

```
public class JdbcTemplate {

//Update() method has to used for doing
//Updatable Operations - Insert,Update,Delete  with any Table
public static int update(String SQL,Object ...params) {
System.out.println("JdbcTemplate - update()");
Connection con = null;
PreparedStatement ps = null;

int x = 0;

try {
con = DataSourceUtil.getConnection();
ps = con.prepareStatement(SQL); //5

for(int i=0;i<params.length;i++) {
ps.setObject(i+1, params[i]);
}

x = ps.executeUpdate();

} catch (Exception ex) {
ex.printStackTrace();
} finally {
DataSourceUtil.cleanUp(ps, con);
}

return x;
}
```

```
//Any Select Operation with Any table which return 0 or 1 record
public static Object queryForObject(String SQL,RowMapper rowMapper, Object
...params) {
System.out.println("JdbcTemplate - queryForObject()");

Connection con = null;
PreparedStatement ps = null;
ResultSet rs = null;

Object obj = null;
try {
con = DataSourceUtil.getConnection();
ps = con.prepareStatement(SQL);

for(int i=0;i<params.length;i++) {
ps.setObject(i+1, params[i]);
}

rs = ps.executeQuery();

if (rs.next()) {
obj=rowMapper.mapRow(rs);
}

} catch (Exception ex) {
ex.printStackTrace();
} finally {
DataSourceUtil.cleanUp(rs, ps, con);
}

return obj;
}
```

```
//Any Select Operation with Any table which return 0 or many records
public static List queryForList(String SQL,RowMapper rowMapper, Object
...params) {
System.out.println("JdbcTemplate - queryForList()");

Connection con = null;
PreparedStatement ps = null;
ResultSet rs = null;

List mylist = new ArrayList();
try {
con = DataSourceUtil.getConnection();
ps = con.prepareStatement(SQL);

for(int i=0;i<params.length;i++) {
ps.setObject(i+1, params[i]);
}

rs = ps.executeQuery();

while (rs.next()) {
Object obj= rowMapper.mapRow(rs);
mylist.add(obj);
}

} catch (Exception ex) {
ex.printStackTrace();
} finally {
DataSourceUtil.cleanUp(rs, ps, con);
}

return mylist;
}
}
```

9) CustomerDAOImpl.java

```
package com.jlcindia.dao;

import java.util.List;
import com.jlcindia.dto.Customer;
import com.jlcindia.rowmappers.CustomerRowMapper;
import com.jlcindia.rowmappers.RowMapper;
import com.jlcindia.template.JdbcTemplate;
/*
* @Author    : Srinivas Dande
* @Company : Java Learning Center
* @See          : myjavalearningcenter.com
* */
public class CustomerDAOImpl implements CustomerDAO {

@Override
public int addCustomer(Customer cust) {
String SQL = "insert into mycustomers values(?,?,?,?,?)";
int x = JdbcTemplate.update(SQL, cust.getCid(), cust.getCname(), cust.getEmail(),
cust.getPhone(),
cust.getCity());
return x;
}

@Override
public int updateCustomer(Customer cust) {
String SQL = "update mycustomers set cname=? , email=? , phone=?, city=? where
cid=?";
int x = JdbcTemplate.update(SQL, cust.getCname(), cust.getEmail(), cust.getPhone(),
cust.getCity(),
cust.getCid());
return x;
}

@Override
public int deleteCustomer(int cid) {
String SQL = "delete from mycustomers where cid=?";
int x = JdbcTemplate.update(SQL, cid);
return x;
}
```

```
@Override
public Customer getCustomerByCid(int cid) {
String SQL = "select * from mycustomers where cid=?";
RowMapper rowMapper = new CustomerRowMapper();
Customer cust = (Customer) JdbcTemplate.queryForObject(SQL, rowMapper, cid);
return cust;
}

@Override
public Customer getCustomerByEmail(String email) {
String SQL = "select * from mycustomers where email=?";
RowMapper rowMapper = new CustomerRowMapper();
Customer cust = (Customer) JdbcTemplate.queryForObject(SQL, rowMapper, email);
return cust;
}

@Override
public Customer getCustomersByPhone(int phone) {
String SQL = "select * from mycustomers where phone=?";
RowMapper rowMapper = new CustomerRowMapper();
Customer cust = (Customer) JdbcTemplate.queryForObject(SQL, rowMapper, phone);
return cust;
}

@Override
public List<Customer> getAllCustomers() {
String SQL = "select * from mycustomers";
RowMapper rowMapper = new CustomerRowMapper();
List mylist = JdbcTemplate.queryForList(SQL, rowMapper);
return mylist;
}

@Override
public List<Customer> getCustomersByCity(String city) {
String SQL = "select * from mycustomers where city=?";
RowMapper rowMapper = new CustomerRowMapper();
List mylist = JdbcTemplate.queryForList(SQL, rowMapper,city);
return mylist;
}
```

```
@Override
public List<Customer> getCustomersByCname(String cname) {
String SQL = "select * from mycustomers where cname=?";
RowMapper rowMapper = new CustomerRowMapper();
List mylist = JdbcTemplate.queryForList(SQL, rowMapper,cname);
return mylist;
}

}
```