CMR INSTITUTE OF TECHNOLOGY	USN									
Internal Assesment Test – II										
Sub:	Advanced Java Programming							Code:	20MCA33	
Date:	7/2/23	Duration:	90 mins	Max Marks:	50	Sem:	III	Branch:	MCA	
Answer Any One FULL Question from each part.										

		Marks	OBE CO	OBE RBT
	Part – I			
1	Explain the different type of JDBC drivers	10	CO4	L2
2	Write a java JSP program to accept customer information through a HTML. Also create Java bean class, populate the bean and display the same information through JSP.	10	CO3	L4
	Part – II			
3	With an example program describe the various steps involved in connecting a java application with database.	10	CO5	L3
4.a.	How is JAR files created and used? Explain with different switches how to work with JAR files?	10	CO3	L2
4.b.	Explain Scrollable Result Set with a code snippet	5	CO4	L2
5	Develop a program to insert following data into customer table in database. Using prepared Statement object. Table consists of cust_id , int(5), cust_name varchar(20), city varchar(20)	10	CO4	L4
6	Explain the following page directive attributes along with example program. a)import b)error page and isErrorPage c)contentType d)buffer and autoFlush	10	CO3	L2
	Part – IV			
7	Write a JSP Program which uses jsp:include and jsp:forward action to display a Webpage.	10	CO6	L4

8	List and explain the Built-in annotations of JAVA	10	CO5	L2
	Part – V			
9	Write the short note about the following Prepared statement Batch Update	10	CO4	L2
10	Discuss the advanced JDBC data types	10	CO4	L2

1. **Explain the different type of JDBC drivers.**

Type 1: JDBC-to-ODBC Driver

 Microsoft created ODBC (Open Database Connection), which is the basis from which Sun created JDBC. Both have similar driver specifications and an API.

 The JDBC-to-ODBC driver, also called the JDBC/ODBC Bridge, is used to translate DBMS calls between the JDBC specification and the ODBC specification.

 MS Access and SQL Server contains ODBC driver written in C language using pointers, but java does not support the mechanism to handle pointers.

 So JDBC-ODBC Driver is created as a bridge between the two so that JDBC-ODBC bridge driver translates the JDBC API to the ODBC API.

Drawbacks of Type-I Driver:

o ODBC binary code must be loaded on each client.

o Transaction overhead between JDBC and ODBC.

o It doesn‟t support all features of Java.

o It works only under Microsoft, SUN operating systems.

Type 2: Java/Native Code Driver or Native-API Partly Java Driver

 It converts JDBC calls into calls on client API for DBMS.

 The driver directly communicates with database servers and therefore some database client software must be loaded on each client machine and limiting its usefulness for internet

 The Java/Native Code driver uses Java classes to generate platform- specific code that is code only understood by a specific DBMS.

Ex: Driver for DB2, Informix, Intersoly, Oracle Driver, WebLogic drivers

Drawbacks of Type-I Driver:

o Some database client software must be loaded on each client machine

o Loss of some portability of code.

o Limited functionality

o The API classes for the Java/Native Code driver probably won‟t work with another manufacturer‟s DBMS.

Type 3: Net-Protocol All-Java Driver

 It is completely implemented in java, hence it is called pure java driver. It translates the JDBC calls into vendor‟s specific protocol which is translated into DBMS protocol by a middleware server

 Also referred to as the Java Protocol, most commonly used JDBC driver.

 The Type 3 JDBC driver converts SQL queries into JDBC- formatted statements, in-turn they are translated into the format required by the DBMS.

Ex: Symantec DB

Drawbacks:

 It does not support all network protocols.

 Every time the net driver is based on other network protocols.

Type 4: Native-Protocol All-Java Driver or Pure Java Driver

 Type 4 JDBC driver is also known as the Type 4 database protocol.

 The driver is similar to Type 3 JDBC driver except SQL queries are translated into the format required by the DBMS.

 SQL queries do not need to be converted to JDBC-formatted systems.

 This is the fastest way to communicated SQL queries to the DBMS.

 Here the driver uses network protocol this protocol is already built-into the database engine; here the driver talks directly to the database using java sockets. This driver is better than all other drivers, because this driver supports all network protocols.

 Use Java networking libraries to talk directly to database engines

Ex: Oracle, MYSQL

Only disadvantage: need to download a new driver for each database engine

2. **Write a java JSP program to accept customer information through a HTML. Also create Jav bean class, populate the bean and display the same information through JSP.**

student.java

```
package program8;
public class Cust
{
        public String cname;
        public String cid;

        public void setcname(String name)
        {
                cname=name;
```

```
        }

        public String getcname()
        {
                return cname;
        }
        public void setcid(String no)
        {
                cid=no;
        }
        public String getcid()
        {
                return cid;
        }
}
```

display.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<!-- Using the studb bean -->
<jsp:useBean id ="studb" scope = "request" class =
"program8.Cust"></jsp:useBean> Student Name : <jsp:getProperty
name="studb" property="cname"/><br/>
Roll No. : <jsp:getProperty name="studb" property="cid"/><br/>
</body>
</html>
```

first.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

```
<title>Insert title here</title>
</head>
<body>
<!-- Create the bean studb and set the property -->
<jsp:useBean id="studb" scope="request" class="program8.Cust"></jsp:useBean>
<jsp:setProperty name="studb" property='*'/>
<jsp:forward page="display.jsp"></jsp:forward>
</body>
</html>
```

index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<!-- send the form data to first.jsp -->
<form action="first.jsp">
Customer Name : <input type="text" name =
"cname"> Customer ID : <input type="text"
name = "cid">
<input type = "submit" value="Submit"/>
</form>
</body>
</html>
```

3. With an example program describe the various steps involved in connecting a java application with database.

The following 5 steps are the basic steps involve in connecting a Java application with

Database using JDBC.

Register the Driver

Create a Connection

Create SQL Statement

Execute SQL Statement

Closing the connection

Register the Driver

Class.forName() is used to load the driver class explicitly.

Example to register with JDBC-ODBC Driver

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Create a Connection

getConnection() method of DriverManager class is used to create a connection.

Syntax

getConnection(String url)

getConnection(String url, String username, String password)

getConnection(String url, Properties info)

Example establish connection with Oracle Driver

```
Connection con = DriverManager.getConnection
("jdbc:oracle:thin:@localhost:1521:XE","username","password");
```

Create SQL Statement

createStatement() method is invoked on current Connection object to create a SQL Statement.

Syntax

public Statement createStatement() throws SQLException

Example to create a SQL statement

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

Execute SQL Statement

executeQuery() method of Statement interface is used to execute SQL statements.

Syntax

public ResultSet executeQuery(String query) throws SQLException

Example to execute a SQL statement

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

Closing the connection

After executing SQL statement you need to close the connection and release the session.

The close() method of Connection interface is used to close the connection.

Syntax

public void close() throws SQLException

Example of closing a connection

```
con.close();
import java.sql.*;
class OracleCon{
public static void main(String args[]){
try{
//step1 load the driver class
```

```
Class.forName("oracle.jdbc.driver.OracleDriver");
//step2 create the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
//step3 create the statement object
Statement stmt=con.createStatement();
//step4 execute query
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
//step5 close the connection object
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

4.a. How is JAR files created and used? Explain the different switches how to work with JAR files?

JAR files are packaged with the ZIP file format, so you can use them for tasks such as lossless data compression, archiving, decompression, and archive unpacking. These tasks are among the most common uses of JAR files, and you can realize many JAR file benefits using only these basic features.

Creating a JAR File

The basic format of the command for creating a JAR file is:

jar cf *jar-file input-file(s)*

The options and arguments used in this command are:

The c option indicates that you want to *create* a JAR file.

The f option indicates that you want the output to go to a *file* rather than to stdout. jar-file is the name that you want the resulting JAR file to have. You can use any filename for a JAR file. By convention, JAR filenames are given a .jar extension, though this is not required. The input-file(s) argument is a space-separated list of one or more files that you want to include in your JAR file. The input-file(s)argument can contain the wildcard * symbol. If any of the "input-files" are directories, the contents of those directories are added to the JAR archive recursively.

The c and f options can appear in either order, but there must not be any space between them. This command will generate a compressed JAR file and place it in the current directory. c - creates a new or empty archive pm the Std output

t - lists the table of contents from std output

X file – it extracts all files or just the named files

f - The argument following this option specifies a JAR file to work v

- It generates verbose output on stderr

m - It includes manifest information from a specified manifest file

0 - it indicates ‘store only’ without using ZIP compression

M - it specifies that a manifest file should not be created for the entries

u - It updates an exisiting JAR file by adding files or changing the

Manifest
To create a JAR file jar cf *jar-file input-file(s)*
To view the contents of a JAR file jar tf *jar-file*
To extract the contents of a JAR file jar xf *jar-file*
To extract specific files from a JAR file jar xf *jar-file archived-file(s)*
To run an application packaged as a JAR file
(requires the Main-class manifest header) java -jar *app.jar*

Benefits of JAR

• Security: You can digitally sign the contents of a JAR file.
• Decreased download time: for Applets and Java Web Start
• Compression: efficient storage
• Packaging for extensions: extend JVM (example Java3D)
• Package Sealing: enforce version consistency
o all classes defined in a package must be found in the same JAR file • Package Versioning: hold data like like vendor and version information • Portability: the mechanism for handling JAR files is a standard part of the Java platform's core API

4.b. Explain scrollable resultset with a code snippet.

In JDBC 2.1 API the virtual cursor can be moved backwards or positioned at a specific row.

 Six methods are there for Resultset object.
 They are first(), last(), previous(), absolute(), relative() and getrow().
 first() Moves the virtual cursor to the first row in the Resultset.
 last() Positions the virtual cursor at the last row in the Resultset
 previous() Moves the virtual cursor to the previous row.
 absolute() Positions the virtual cursor to a specified row by the an integer value passed to the method.
 relative() Moves the virtual cursor the specified number of rows contained in the parameter. The parameter can be positive or negative integer.
 getRow() Returns an integer that represents the number of the current row in the Resultset.
 To handle the scrollable ResultSet , a constant value is passed to the Statement object that is created using the createStatement(). Three constants.

TYPE_FORWARD_ONLY restricts the virtual cursor to downward movement
TYPE_SCROLL_INSENSITIVE and TYPE_SCROLL_SENSITIVE (Permits the virtual cursor to Move in any direction)

```
try {
String query = "SELECT FirstName,LastName FROM Customers";
Statement stmt;
ResultSet rs;
stmt = con.createStatement();
rs = stmt.executeQuery (query);
while(rs.next()){
        rs.first();
        rs.previous();
        rs.absolute(10);
        rs.relative(-2);
        rs.relative(2);
System.out.println(rs.getString(1) + rs. getString (2));
}
stmt.close();}catch ( Exception e ){}
```

5. Develop a program to insert following data into music table in database. Using prepared Statement object. Table consists of music_id , int(5), music_name varchar(20), music_author varchar(20)

```
package j2ee.p9;
import java.sql.*;
import java.io.*;

public class Studentdata {

        public static void main(String[] args) {
                Connection con;
                PreparedStatement pstmt;
                Statement stmt;
                ResultSet rs;
 String music_name,music_author;
                Integer music_id,
 try
                {
              Class.forName("com.mysql.jdbc.Driver"); // type1 driver

                        try{

        con=DriverManager.getConnection("jdbc:mysql://127.0.0.1/mca","root","s
ystem"); //  type1 access connection
                        BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
                         do
                         {
                        System.out.println("\n1. Insert.\n2. Select.5. Exit.\nEnter your
```

```
choice:");
                    int choice=Integer.parseInt(br.readLine());
                        switch(choice)
                        {
                    case 1: System.out.print("Enter music id :");
                            music_id =Integer.parseInt(br.readLine());
                            System.out.print("Enter music name :");
                            music_name=br.readLine();
System.out.print("Enter music author :"); music_author=br.readLine();
pstmt=con.prepareStatement("insert into music values(?,?,?)");
pstmt.setInt(1,music_id);
                            pstmt.setString(2,music_name);
                            pstmt.setString(3,music_author);
                            pstmt.execute();
                            System.out.println("\nRecord Inserted
successfully.");
                              break;
                              case 2:
                            stmt=con.createStatement();
                            rs=stmt.executeQuery("select *from music ");
                            if(rs.next())
                                    {
                            System.out.println("Music ID \t Music Name \t
Music author\n--------------------------------");
                                      do
                                      {
                                  music_id=rs.getInt(1);
                                  music_name=rs.getString(2);
                                  music_author=rs.getString(3);

            System.out.println(music_id+"\t"+music_name+"\t"+music_author);
                            }while(rs.next());
                                      }
                                      else
                                  System.out.println("Record(s) are not
available in database.");
                                      break;

                            case 3: con.close(); System.exit(0);
                            default: System.out.println("Invalid choice, Try
again.");
                              }//close of switch
                              }while(true);
                              }//close of nested try
                    catch(SQLException e2)
                              {
                            System.out.println(e2);
                              }
                              catch(IOException e3)
                              {
```

```
                                        System.out.println(e3);
                                        }
                        }//close of outer try
                        catch(ClassNotFoundException e1)
                        {
                                System.out.println(e1);
                        }
                }
    }
```

6. Explain the following page directive attributes along with example program.
a)import
b)error page and isErrorPage
c)contentType
d)buffer and autoFlush

1)import

The import attribute is used to import class,interface or all the members of a package.It is similar to import keyword in java class or interface.

2. Example of import attribute

```
1. <html>
2. <body>
3.
4. <%@ page import="java.util.Date" %>
5. Today is: <%= new Date() %>
6.
7. </body>
8. </html>
```

2. errorPage

The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.

2. Example of errorPage attribute

```
1. //index.jsp
2. <html>
3. <body>
4.
5. <%@ page errorPage="myerrorpage.jsp" %>
6.
7.  <%= 100/0 %>
8.
9. </body>
10. </html>
```

3.isErrorPage

The isErrorPage attribute is used to declare that the current page is the error page.

1. *Note: The exception object can only be used in the error page.*

4. Example of isErrorPage attribute

```
1. //myerrorpage.jsp
2. <html>
3. <body>
4.
5. <%@ page isErrorPage="true" %>
6.
7.  Sorry an exception occured!<br/>
8. The exception is: <%= exception %>
9.
10. </body>
11. </html>
```

4.contentType

The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.The default value is "text/html;charset=ISO-8859-1".

2. Example of contentType attribute

```
1. <html>
2. <body>
3.
4. <%@ page contentType=application/msword %>
5. Today is: <%= new java.util.Date() %>
6.
7. </body>
8. </html>
```

5. buffer

The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page.The default size of the buffer is 8Kb.

2. Example of buffer attribute

```
1. <html>
2. <body>
3.
4. <%@ page buffer="16kb" %>
5. Today is: <%= new java.util.Date() %>
6.
7. </body>
8. </html>
```

6.Autoflush

The autoFlush attribute controls whether the output buffer should be automatically flushed when it is full (the default) or whether an exception should be raised when the buffer overflows (autoFlush="false"). Use of this attribute takes one of the following two forms.

<%@ page autoflush=“false” %>

7. Write a JSP Program which uses jsp:include and jsp:forward action to display a Webpage.

ndex.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<!-- send the form data to login.jsp and the get method is used -->
<form method="get" action="login.jsp">
UserName : <input type="text"
name ="name"><br> Password
: <input type="password" name
="pass"><br>
<input type="Submit" value ="Submit"/><br>
</form>
</body>
</html>
```

login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
//Getting the input name from the html form
and storing in String ‘uname’ String uname =
request.getParameter("name");
//Getting the input pass from the html form
and storing in String ‘upass’ String upass =
request.getParameter("pass");
if(uname.equals("admin") && upass.equals("admin"))
{
```

```
 %>
<jsp:forward page="main.jsp"></jsp:forward>


<%
}
else
{
out.println("Wrong Credentials Username and
Password"+"<br>"); out.println("Enter Corrects Username
and Password.. Try again" +"<br><br>");%>

   <jsp:include page="index.jsp"></jsp:include>
<%
}%>
</body>
</html>
```

main.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
// Getting the input name from the html form
and storing in String 'un'--> String
un=request.getParameter("name");
// Getting the input pass from the html form
and storing in String 'pw'--> String
pw=request.getParameter("pass");
%>
<h1>welcome:<%=un%></h1>
<h1>your user name is:<%=un%></h1>
<h1>your password is:<%=pw%></h1>
</body>
</html>
```

8. List and explain the Built-in annotations of JAVA

These four are the annotations imported from **java.lang.annotation**: **@Retention**, **@Documented**, **@Target**,and **@Inherited**.

· **@Override**, **@Deprecated**, and **@SuppressWarnings** are included in **java.lang**.

1. @Retention

@Retention is designed to be used only as an annotation to another annotation. It specifies the retention policy.

· A retention policy determines at what point annotation should be discarded.
· Java defined 3 types of retention policies through java.lang.annotation.RetentionPolicy enumeration. It has SOURCE, CLASS and RUNTIME.
· Annotation with retention policy SOURCE will be retained only with source code, and discarded during compile time.
· Annotation with retention policy CLASS will be retained till compiling the code, and discarded during runtime.
· Annotation with retention policy RUNTIME will be available to the JVM through runtime.
· The retention policy will be specified by using java built-in annotation @Retention, and we have to pass the retention policy type.

The default retention policy type is CLASS.

2. @Documented

The **@Documented** annotation is a marker interface that tells a tool that an annotation is to be documented. It is designed to be used only as an annotation to an annotation declaration. By default, annotation are not included in javadoc(is a documentation generator). But if @document is used, it then will be processed by javadoc like toolas and the annotation type information will also be included in generated document .

3. @Target

The **@Target** annotation specifies the types of declarations to which an annotation can be applied. It is designed to be used only as an annotation to another annotation. **@Target** takes one argument, which must be a constant from the **ElementType** enumeration. This argument specifies the types of declarations to which the annotation can be applied. The constants are shown here along with the type of declaration to which they correspond.

Target Constant Annotation Can Be Applied To
ANNOTATION_TYPE Another annotation
CONSTRUCTOR Constructor
FIELD Field
LOCAL_VARIABLE Local variable
METHOD Method
PACKAGE Package
PARAMETER Parameter
TYPE Class, interface, or enumeration

we can specify one or more of these values in a **@Target** annotation. To specify multiple values, we must specify them within a braces-delimited list. For example, to specify that an annotation applies only to fields and local variables, we can use this **@Target** annotation: @Target({ ElementType.FIELD, ElementType.LOCAL_VARIABLE }) **4. @Inherited**

@Inherited is a marker annotation that can be used only on another annotation declaration. it affects only annotations that will be used on class declarations. **@Inherited** causes the annotation for a superclass to be inherited by a subclass.

Therefore, when a request for a specific annotation is made to the subclass, if that annotation is not present in the subclass,then its superclass is checked. If that annotation is present in the superclass, and if it is annotated with **@Inherited**, then that annotation will be returned.

```
java.lang.annotation.Inherited

@Inherited
public @interface MyAnnotation {

}
@MyAnnotation
public class MySuperClass { ... }
public class MySubClass extends MySuperClass { ... }
```

In this example the class MySubClass inherits the annotation @MyAnnotation because MySubClassinherits from MySuperClass, and MySuperClass has a @MyAnnotation annotation.

5. @Override

@Override is a marker annotation that can be used only on methods. A method annotated with **@Override** must override a method from a superclass. If it doesn't, a compile-time error will result. It is used to ensure that a superclass method is actually overridden, and not simply overloaded.

6. @Deprecated

@Deprecated is a marker annotation. It indicates that a declaration is obsolete and has been replaced by a newer form. This annotation is used to mark a class, method or field as deprecated, meaning it should on longer be used If your code uses deprecated classes, methods or fields the compiler will give you a warning.

```
@Deprecated
Public class MyComponent
{
}
```

The use of the @Deprecated annotation above the class declaration marks the class as deprecated.

The use of the @Deprecated annotation above the fieldclass declaration marks the field as deprecated.

7. **@SuppressWarnings** specifies that one or more warnings that might be issued by the compiler are to be suppressed. The warnings to suppress are specified by name, in string form. This annotation can be applied to any type of declaration.

@SuppressWarnings

– Makes the compiler suppress warnings for a given methods

– If a method class a deprecated method, or makes an insecure type

case, the compiler may generate a warning.

– You can suppress these warnings by annotating the method containing the code with the @SuppressWarnings annotation

```
@ SuppressWarnings
public void methodWithWarning()
{
    }
```

9. Write the short note about the following

i.Prepared statement

ii.Batch Update

The preparedStatement object allows you to execute parameterized queries. A SQL query can be precompiled and executed by using the PreparedStatement object. · Ex: Select * from publishers where pub_id=? Here a query is created as usual, but a question mark is used as a placeholder for a value· that is inserted into the query after the query is compiled. The preparedStatement() method of Connection object is called to return the· PreparedStatement object.

Ex: PreparedStatement stat; stat= con.prepareStatement("select * from publisher where pub_id=?")

```
import java.sql.*;

public class JdbcDemo {
   public static void main(String args[]){
     try{
       Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
       Connection con=DriverManager.getConnection("jdbc:odbc:MyDataSource","khutub","");
       PreparedStatement pstmt;
       pstmt= con.prepareStatement("select * from employee whereUserName=?");
       pstmt.setString(1,"khutub");
       ResultSet rs1=pstmt.executeQuery();
       while(rs1.next()){
          System.out.println(rs1.getString(2));
       }
     } // end of try
     catch(Exception e){System.out.println("exception"); }
   } //end of main
   } // end of class
```

Batch Updates

A batch update is a batch of updates grouped together, and sent to the database in one "batch", rather than sending the updates one by one.

Sending a batch of updates to the database in one go, is faster than sending them one by one, waiting for each one to finish. There is less network traffic involved in sending one batch of updates (only 1 round trip), and the database might be able to execute some of the updates in parallel. The speed up compared to executing the updates one by one, can be quite big.

You can batch both SQL inserts, updates and deletes. It does not make sense to batch select statements.

There are two ways to execute batch updates:

1. Using a Statement
2. Using a PreparedStatement

i) Add Batch
ii) Clear Batch
iii) Execute Batch

`Statement` object is used to execute batch updates. You do so using the `addBatch()` and `executeBatch()` methods. Here is an example:

```
Statement statement = null;

try{
 statement = connection.createStatement();

 statement.addBatch("update people set firstname='John' where id=123");
statement.addBatch("update people set firstname='Eric' where id=456");
statement.addBatch("update people set firstname='May' where id=789");

 int[] recordsAffected = statement.executeBatch();
} finally {
 if(statement != null) statement.close();
}
```

First you add the SQL statements to be executed in the batch, using the `addBatch()` method.

Then you execute the SQL statements using the `executeBatch()`. The `int[]` array returned by the `executeBatch()` method is an array of `int` telling how many records were affected by each executed SQL statement in the batch

10 Discuss the advanced JDBC data types

1. BLOB

· The JDBC type `BLOB` represents an SQL3 `BLOB` (Binary Large Object).
· A JDBC `BLOB` value is mapped to an instance of the `Blob` interface in the Java programming language.
· A `Blob` object logically points to the BLOB value on the server rather than containing its binary data, greatly improving efficiency.
· The `Blob` interface provides methods for materializing the `BLOB` data on the client when that is desired.

2. CLOB

· The JDBC type `CLOB` represents the SQL3 type `CLOB` (Character Large Object). · A JDBC `CLOB` value is mapped to an instance of the `Clob` interface in the Java programming language.

· A `Clob` object logically points to the `CLOB` value on the server rather than containing its character data, greatly improving efficiency.
· Two of the methods on the `Clob` interface materialize the data of a `CLOB` object on

the client. **3. ARRAY**

· The JDBC type `ARRAY` represents the SQL3 type `ARRAY`.
· An `ARRAY` value is mapped to an instance of the `Array` interface in the Java programming language.
· An `Array` object logically points to an `ARRAY` value on the server rather than containing the elements of the `ARRAY` object, which can greatly increase efficiency.
· The `Array` interface contains methods for materializing the elements of the `ARRAY` object on the client in the form of either an array or a `ResultSet` object.

Example : ResultSet rs = stmt.executeQuery("SELECT NAMES FROM STUDENT"); rs.next();
Array stud_name=rs.getArray("NAMES");

4. DISTINCT

· The JDBC type DISTINCT represents the SQL3 type `DISTINCT`.
· For example, a `DISTINCT` type based on a `CHAR` would be mapped to a `String` object, and a `DISTINCT` type based on an SQL `INTEGER` would be mapped to an `int`.
· The `DISTINCT` type may optionally have a custom mapping to a class in the Java programming language.
· A custom mapping consists of a class that implements the interface `SQLData` and an entry in a `java.util.Map` object.

5. STRUCT

· The JDBC type `STRUCT` represents the SQL3 structured type.
· An SQL structured type, which is defined by a user with a `CREATE TYPE` statement, consists of one or more attributes. These attributes may be any SQL data type, built-in or user defined.
· A `Struct` object contains a value for each attribute of the`STRUCT` value it represents.
· A custom mapping consists of a class that implements the interface `SQLData` and an entry in a `java.util.Map` object.

6. REF

· The JDBC type `REF` represents an SQL3 type `REF<structured type>`. · An SQL `REF` references (logically points to) an instance of an SQL structured type, which the `REF` persistently and uniquely identifies.
· In the Java programming language, the interface `Ref` represents an

SQL `REF`.

7. JAVA_OBJECT

· The JDBC type `JAVA_OBJECT`, makes it easier to use objects in the Java programming language as values in a database.
· `JAVA_OBJECT` is simply a type code for an instance of a class defined in the Java programming language that is stored as a database object.
· The `JAVA_OBJECT` value may be stored as a serialized Java object, or it may be stored in some vendor-specific format.
· The type `JAVA_OBJECT` is one of the possible values for the column `DATA_TYPE` in the `ResultSet` objects returned by various `DatabaseMetaData` methods, including `getTypeInfo`, `getColumns`, and `getUDTs`.
· Values of type `JAVA_OBJECT` are stored in a database table using the method `PreparedStatement.setObject`.
· They are retrieved with They are retrived with the methods `ResultSet.getObject` or `CallableStatement.getObject` and updated with
the `ResultSet.updateObject` method.

For example, assuming that instances of the class `Engineer` are stored in the column `ENGINEERS` in the table `PERSONNEL`, the following code fragment, in which stmt is a `Statement` object, prints out the names of all of the engineers.