UNIT-VIII :DATABASE ACCESS

8.1 Database programming using JDBC

1. JDBC API provides java objects for working with database drivers,connections,sql statements.
2. Write java code for sql databases.
3. JDBC includes statement classes for supporting different execution models of queries.
4. JDBC modeled after ODBC specifications for database access API
5. Supports SQL functionality
6. Enable access to wide range of relational DBMS products.

JDBC API consists of two main interfaces

(i) API for application writers

(ii) Lower level driver API for driver written JDBC driver classes implement these lower level APIC interfaces for a particular database engine.

7. Driver implementations comes in four flavors:

- Type1(JDBC-ODBC bridge+ODBC driver)
- Type2(partial JDBC driver)
- Type3(pure Java JDBC driver for database middle ware)
- Type4(Pure database driver with direct database connection.

8.1.1 Database programming using JDBC involves following steps

1. Load JDBC driver /get datasource object using JNDI
2. Create database connection
3. Create statement on connection object and execute statement using executeQuery()
4. Process result using ResultSet object

Example program 8.1

Step1:Creating a database. You can either create the database outside of Java, via tools supplied by the database vendor, or via SQL statements fed to the database from a Java program.

Step1: Create Database

Lets connect to the 'localhost' under the user 'root'.

Run the following script, which creates a table called 'students' under the 'test' DB.

```
/* Table structure for students */
CREATE TABLE students
(name varchar(32) default NULL,
class varchar(32) default NULL,
grade char(2) default NULL);
```

…Continued

Run the following script, which inserts the data into the ‘students’ table.

```
/* Table data for test.students*/
INSERT INTO students VALUES
('John','Math','A'),
('Mary','Math','A'),
('Tom','Math','C'),
('Tom','English','C+'),
('John','English','A'),
('Mary','English','B'),
('Tom','Science','D-'),
('John','Science','A'),
('Mary','Science','B+');
```

Step2:java source code:-

```
import java.sql.*;
public class MySqlJdbc {
public static Connection dbconn;
public static Statement stmt;
public static ResultSet rset;
public static void main(String[] args) {
try         {
Class.forName ("com.mysql.jdbc.Driver").newInstance();

dbconn = DriverManager.getConnection ("jdbc:mysql://localhost:3306/test","root","");
```

```
stmt = dbconn.createStatement();
String qry = "Select * from students";


rset = stmt.executeQuery(qry);


while (rset.next())       {
System.out.println ("Name: " + rset.getString(1) + "\t Class: " + rset.getString(2) + "\t Grade: "
+ rset.getString(3));
}
}
catch (SQLException sqle) {
System.out.println(sqle);
System.out.println("Sql exception..");
}
catch(ClassNotFoundException cnfe) {
System.out.println(cnfe);
System.out.println("Class not found exception..");
}
catch (Exception ex) {
System.out.println (ex.getMessage());
System.out.println(" try exception..");
}
finally
{
try {
rset.close();
stmt.close();
dbconn.close();
}
catch (SQLException sqle) {
System.out.println ("Exception from finally block.");
}
}
```

} // end of main() method

} // end of class definition

Step3: Compiling and running the program.

Set the class path to include mysql.jar file using the following command at the prompt.

C:\JDBC>set classpath=%classpath%;c:\JDBC\mysql.jar;

Compile the program using the javac tool.

C:\JDBC>javac MySqlJdbc.java

…Continued

Run the program using the java tool.

C:\JDBC>java MySqlJdbc

Name: John Class: Math Grade: A

Name: Mary Class: Math Grade: A

Name: Tom Class: Math Grade: C

Name: Tom Class: English Grade: C+

Name: John Class: English Grade: A

Name: Mary Class: English Grade: B

Name: Tom Class: Science Grade: D-

Name: John Class: Science Grade: A

Name: Mary Class: Science Grade: B+

8. 2 Studying Javax.sql.* package:-

It includes classes and interfaces that allow:
The creation of DataSource object

- Row sets
- Connection pooling
- Distributed transaction.

8.2.1 Data Source :-

- Javax.sql.DataSource allows database to be named and registered with in JNDI server.
- It allows calling application to access database instead of connection URL.
- DataSource is created by application server so that client can access it using JNDI.

8.2.2 Row Set :

- Javax.sql.RowSet object set properties for event notification.
- RowSet object holds set of rows contained either in ResultSet object or in the some sort of tabular data.
- RowSet interface implemented by component wants to be notified about events that occur on particular RowSet object.
- RowSetMetaData derived form ResultSetMetaData interface, provides information about columns in RowSet object.

8.2.3 Connection pooling:

The Connection pool is a cache of database connections maintained in database's memory, so that connection can be reused when database receives future requests for data.

The classes and interfaces for connection pooling are:

- ConnnectionPoolDataSource(interface)
- PooledConnection(interface)
- ConnectionEvent(class)
- ConnectionEventListener(interfaces)

8.2.4 Distributed Transaction

It is a system, which typically relies on external transaction manager such as software components that implement standard Java Transaction API functionality to coordinate individual transaction. Resource manager manges data or resource in distributed transaction.

8.3 Accessing a database from a JSP page

A JSTL includes number of actions for database access to make it easy and to assist the developer to develop simple database-driven JSP applications.

Action elements provide following features:

1. *Using a connection pool for better performance and scalability.*
2. *Supporting queries,updates and inserts*
3. *Handling most common data type conversions*
4. *Supporting combination of database operations in one transaction*

8.3.1 specifying a data source for JSTL Actions:

JSTL database actions require data source object, so as to establish connection with database. Data source can be specified in four ways:-

1. <context-param> element of deployment descriptor .
2. <sql:setDataSource> JSTL action

3. application scope variable
4. JNDI service.

8.3.2 Database Access tags :

JSTL tags define action element

<sql:query>	It specify query statement ,datasource ,scope
<sql:param>	It specify parameter values
<sql:update>	It execute SQL insert,update ,delete statements
<sql:transaction>	It establishes transaction context for <sql:query>,<sql:param>

Example Program 8.3.2

Example program on JSTL tags of database

Step 1:set datasource in web.xml

```
<sql:setDataSource var=”example” scope=”application”
driver=”sun.jdbc.odbc.jdbcOdbcDriver”
url=”jdbc:odbc:example”
user=”scott”
password=”tiger” />
```

stpe2:Database access page:

```
<%@ taglib predfix=”c” uri=”http://java.sun.com/jsp/jstl/core” %>
<%@ taglib predfix=”c” uri=”http://java.sun.com/jsp/jstl/sql” %>
<%@ taglib predfix=”c” uri=”http://java.sun.com/jsp/jstl/fmt” %>
<sql:query var=”empdbInfo”>
SELECT * FROM Employee
WHERE UserName=?
<sql:param value=”${param.userName}”/>
</sql:query>
<fmt:ParseDate value=”${param.empDate}” var=”parsedEmpDate” pattern=”YYYY-MM-dd”/>
<jsp:useBean id=”now” class=”java.util.Date”/>
<c:choose >
<C:when test=”${empDbInfo.rowCount==0}”>
```

```
<sql:update>
INSERT INTO Emplyee
(UserName,Password,FirstName,LastName,Dept,EmpDate,EmailAddr,ModDate”) VLAUES
(?,?,?,?,?,?,?,?)
<sql:param value=”${param.UserName}”/>
<sql:param value=”${param.password}”/>
<sql:param value=”${param.firstName}”/>
<sql:param value=”${param.lastName}”/>
<sql:param value=”${param.dept}”/>
<sql:dateParam value=”${parsedEmpDate}” type=”date”/>
<sql:param value=”${param.emailAddr}”/>
<sql:dateParam value=”${now}”/>
</sql:update>
</c:when>
<c:otherwise.
<sql:update>


UPDATE Employee
SET Password=?,FirstName=?,LastName=?,Dept=?,EmpDate=?,EmilAddr=?,ModDate=?
WHERE UserName =?
<sql:param value=”${param.password}”/>
<sql:param value=”${param.firstName}”/>
<sql:param value=”${param.lastName}”/>
<sql:param value=”${param.dept}”/>
<sql:dateParam value=”${parsedEmpDate}” type=”date”/>
<sql:param value=”${param.emailAddr}”/>
<sql:dateParam value=”${now}”/>
<sql:param value=”${param.UserName}”/>
</sql:update>
</c:otherwise></c:chhose>
<sql:query var=”newEmpDbInfo” scope=”session”>
SELECt * FROM Employee
WHERE UserName= ?
<sql:param value=”${param.userName}”/>
</sql:query>
<c:redirect url=”confirmation.jsp”/>
```

8.4 Application-specifc Database Actions:

- Complex web application development requires application specific custom actions.
- It separates application presentation logic from business logic.

- It migrates application to an enterprise JavaBeans architecture to interact with other type of clients than web browsers.

8.5 Introduction Struts Framework :-

Framework provides libraries for database access, template's, validation, internationalization for promoting code reuse.

Struts Framework follows Model2MVC architecture

1. Controller servlet receives all requests.
2. Searches properties file for appropriate request handler ,depending upon request URL.
3. Invokes request handler returned by step2 and pass control to it.
4. Request handler handles request, including business logic methods invocations,form validation.
5. Model component interacts with database to retrieve/save data.
6. Controller sets result in appropriate scope.
7. Controller dispatches request to appropriate view depending on results.
8. View is sent to user response.

Example program 8.5

Example program on Callable statement

CallSP.java

```
import java.sql.*;
import java.io.*;
public class CallSP
{
public static Connection conn;
public static Statement stmt;
public static ResultSet rset;
public static void main(String args[])
{
try
{
Class.forName("com.mysql.jdbc.Driver");
System.out.println("Mysql Driver is loaded...");
conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/test", "root", "root");
CallableStatement cstmt = conn.prepareCall("call student(?)");
```

```
cstmt.setInt(1, 3);
ResultSet rset = cstmt.executeQuery();
rset.next();
System.out.println(rset.getString(2));
}
catch (SQLException sqle)
{
System.out.println(sqle.getMessage());
}
catch (ClassNotFoundException cnfe)
{
System.out.println("Class not found...");
}
}
}
```

Misc. Programs

Example program for inserting record :-

```
import java.sql.*;
import java.io.*;

public class InsertRecord
{
public static Connection dbconn;
public static Statement stmt;

public static void main(String[] args)
{
try
{
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
```

```
String dataSourceName = "MyDSN";
String dbURL = "jdbc:odbc:" + dataSourceName;
dbconn = DriverManager.getConnection(dbURL, "","");

stmt = dbconn.createStatement();

int sno = 1;
while (sno <= 10)
{
DataInputStream in = new DataInputStream(System.in);
System.out.println("Enter Name: ");
String n = in.readLine();
String qry = "Insert into Candidates (S_No, Name) Values (" + sno + ", '" + n.trim() + "');";

int i = stmt.executeUpdate(qry);
System.out.println(i + " record is inserted into 'Candidates' successfully..");

sno++;
}

stmt.close();
dbconn.close();
}
catch (SQLException sqle)
{
System.out.println(sqle);
System.out.println("Sql exception..");
}
catch(ClassNotFoundException cnfe)
{
System.out.println(cnfe);
System.out.println("Class not found exception..");
```

```
}

catch (Exception ex)
{
System.out.println (ex.getMessage());
System.out.println(" try exception..");
}
}
}
```

Compile and run the program as follows:

```
C:\JDBC>javac InsertRecord.java
Note: InsertRecord.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\JDBC>java InsertRecord
Enter Name:
Raju
1 record is inserted into 'Candidates' successfully..
Enter Name:
Ramesh
1 record is inserted into 'Candidates' successfully..
Enter Name:
Raghu
1 record is inserted into 'Candidates' successfully..
……
……
```

Example program for Accessing a record

```
import java.sql.*;
public class AccessJdbc
{
```

```
public static Connection dbconn;
public static Statement stmt;
public static ResultSet rset;

public static void main(String[] args)
{
try
{
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();

String dataSourceName = "MyDSN";
String dbURL = "jdbc:odbc:" + dataSourceName;
dbconn = DriverManager.getConnection(dbURL, "","");


stmt = dbconn.createStatement();
String qry = "Select * from Candidates";
rset = stmt.executeQuery(qry);

while (rset.next())
{
System.out.println("S.No: " + rset.getString("S_No") + "\nName: " + rset.getString("Name"));
System.out.println();
}

rset.close();
stmt.close();
dbconn.close();
}
catch (SQLException sqle)
{
```

```
System.out.println(sqle);
System.out.println("Sql exception..");
}

catch(ClassNotFoundException cnfe)
{
System.out.println(cnfe);
System.out.println("Class not found exception..");
}
catch (Exception ex)
{
System.out.println (ex.getMessage());
System.out.println(" try exception..");
}
}
}
```

Example program on Prepared statement

PrepStmt.java

```
import java.sql.*;
import java.io.*;
public class PrepStmt
{
public static Connection dbconn;
public static PreparedStatement pstmt;
public static void main(String[] args)
{
try
{
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
String dataSourceName = "MyDSN";
String dbURL = "jdbc:odbc:" + dataSourceName;
```

```
dbconn = DriverManager.getConnection(dbURL, "","");

String qry = "Insert into Candidates (S_No, Name, SkillSet) Values (?, ?, ?)";
pstmt = dbconn.prepareStatement(qry);

int sno = 11;
while (sno <= 15)
{
DataInputStream in = new DataInputStream(System.in);
System.out.println("Enter Name:");
String n = in.readLine();
System.out.println("Enter Skill Set:");
String sk = in.readLine();
pstmt.setInt(1, sno);
pstmt.setString(2, n.trim());
pstmt.setString(3, sk.trim());
int i = pstmt.executeUpdate();
System.out.println(i + " record is inserted successfully..");
sno++;
}

pstmt.close();
dbconn.close();
}
catch (SQLException sqle)
{
System.out.println(sqle);
System.out.println("Sql exception..");
}
catch(ClassNotFoundException cnfe)
{
System.out.println(cnfe);
```

```
System.out.println("Class not found exception..");
}

catch (Exception ex)
{
System.out.println (ex.getMessage());
System.out.println("try exception..");
}
}
}
```

```
C:\JDBC_Slides>javac PrepStmt.java
Note: PrepStmt.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
C:\JDBC_Slides>java PrepStmt
Enter Name:
Nikhath
Enter Skill Set:
ABCD
1 record is inserted successfully..
Enter Name:
Anjum
Enter Skill Set:
Oracle
1 record is inserted successfully..
Enter Name:
……
……
```

Example program on ResultsetMeta data

```
import java.sql.*;
public class RSMD
```

```
{
public static Connection dbconn;
public static Statement stmt;
public static ResultSet rset;
public static ResultSetMetaData rsmd;

public static void main(String[] args)
{
try
{
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
String dataSourceName = "MyDSN";
String dbURL = "jdbc:odbc:" + dataSourceName;
dbconn = DriverManager.getConnection(dbURL, "","");
stmt = dbconn.createStatement();
String qry = "Select * from Items";
rset = stmt.executeQuery(qry);
rsmd = rset.getMetaData();
int i = 1;
while (i <= rsmd.getColumnCount())
{
String columnName = rsmd.getColumnName(i);
String columnType = rsmd.getColumnTypeName(i);

System.out.print("The name of the column " + i + " is: '");
System.out.print(columnName);
System.out.println("'");
System.out.print("The data type of the column " + i + " is: ");
System.out.println(columnType);
i++;
}
```

```
rset.close();
stmt.close();
dbconn.close();
}
catch (SQLException sqle)
{
System.out.println(sqle);
System.out.println("Sql exception..");
}
catch(ClassNotFoundException cnfe)
{
System.out.println(cnfe);
System.out.println("Class not found exception..");
}
catch (Exception ex)
{
System.out.println(ex.getMessage());
System.out.println(" try exception..");
}
}
}
```

```
C:\JDBC_Slides>javac RSMD.java

C:\JDBC_Slides>java RSMD
The name of the column 1 is: 'Item_No'
The data type of the column 1 is: INTEGER
The name of the column 2 is: 'Item_Name'
The data type of the column 2 is: VARCHAR
```