BACHOLER OF COMPUTER APPLICATIONS

SEMESTER 4

DCA2202

JAVA PROGRAMMING

Unit 13

Java Data Base Connectivity (JDBC)

Table of Contents

1. INTRODUCTION

In this unit, we shall discuss the database handling mechanisms in Java using Java Data Base Connectivity Application Program Interface (JDBC API).

1.1 Objectives

After studying this unit, you should be able to:

- *Describe JDBC concepts*
- *Manage databases in Java*
- *Explain the mechanism for connecting to a back end database*
- *Load the ODBC driver*

2. DATABASE MANAGEMENT

The **JDBC API (Java Data Base Connectivity Application Program Interface**) can access any kind of tabular data, especially data stored in a Relational Database. It works on top of **ODBC (Open Data Base Connectivity)** which was the driver for database connectivity since age old days but since ODBC was implemented in C so people from the VB background had some problems in understanding the implementation intricacies. Since JDBC works on top of ODBC we have something called as a JDBC-ODBC bridge to access the database. JDBC helps you to write Java applications that manage mainly three programming activities listed below namely:

a) connecting to a data source, like a database,
b) sending queries and updating statements to the database and
c) retrieving and processing the results received from the database in answer to your query

A database is a collection of related information and a Data Base Management System (DBMS) is the software that provides you with a mechanism to manipulate data in the database. There are lots of DBMS/RDBMS products available to you, for example, MS-Access, MS- SQL Server, Oracle, Sybase and Ingres. The list is never-ending. Each of these RDBMS stores the data in their own format. MS-Access stores the data in .MDB file format whereas MS-SQL Server stores the data in a .DAT file format. Will the database alone be of any use to your client? The answer is NO! It will be a good idea for you to develop a customized application for the client with options to retrieve, add, and modify data at the touch of a key. To accomplish this, you should have a mechanism of making the application understand and work with the file format of the database i.e. .MDB or .DAT files.

SELF ASSESSMENT QUESTIONS – 1

1. JAVA API stands for_____.
2. Give some examples for DBMS products.

3. CONCEPTS INVOLVED IN CONNECTING A JAVA CODE TO A DATABASE

3.1 ODBC

ODBC is the abbreviation for Open Database Connectivity, a standard database access method developed by Microsoft Corporation. The goal of ODBC is to make it possible to access any data from any application, regardless of which database management system (DBMS) is handling the data. ODBC manages this by inserting a middle layer, called a driver, between an application and the DBMS. The purpose of this layer is to translate the queries of the application into commands that the DBMS understands. For this to work, both the application and the DBMS must be ODBC-compliant i.e. the application must be capable of issuing ODBC commands and the DBMS must be capable of responding to them.

3.2 JDBC

JDBC provides a database-programming interface for Java programs. Since the ODBC is written in 'C' language, a Java program cannot directly communicate with an ODBC driver.

JavaSoft created the JDBC-ODBC bridge driver that translates the JDBC API to the ODBC API. It is used with ODBC drivers.

3.3 JDBC Driver Manager

The JDBC driver manager is the backbone of the JDBC architecture. The function of the JDBC driver manager is to connect a Java application to the appropriate driver.

3.4 JDBC-ODBC Bridge

The JDBC-ODBC bridge allows you to use the ODBC driver as JDBC drivers.

3.5 JDBC Application Architecture

JDBC Application architecture has been shown in the figure 11.1.

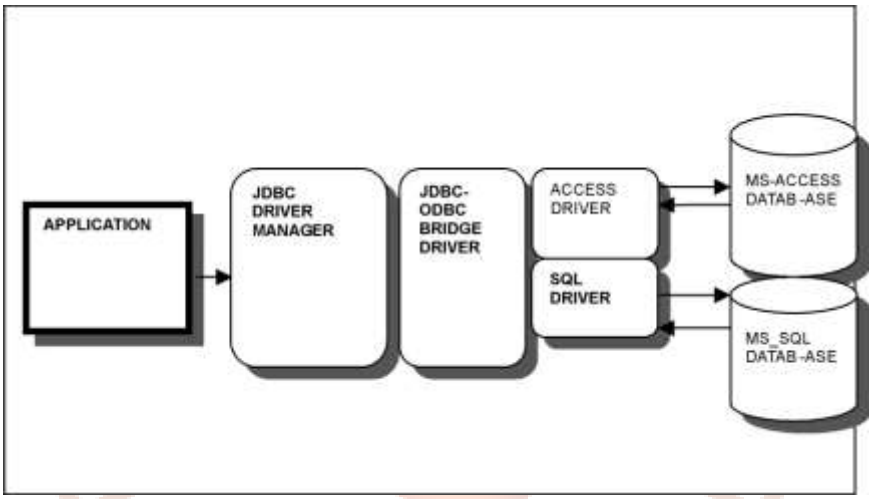

Fig 11.1: JDBC Application Architecture

3.6 Loading the ODBC Driver

To connect to an ODBC driver, follow the below steps.

- Click on the **ODBC Data Source** icon in the Control Panel.
- Click on **Add**.
- Select the **SQL Server** option from the list and click on **Finish.**
- Select a name for the data source and click on the **Next** button.
- Select with **SQL Server authentication using login ID and Password entered by the user.**
- Select the database that you want to use and click on **Next.**
- Click on **Finish.**
- Click on **Test Data Source** to check for proper connectivity and click on **OK.**

4. CONNECTION TO A DATABASE

The ***java.sql*** package contains classes that help in connecting to a database, sending SQL statements to the database, and processing query results.

4.1 The Connection Objects

The **Connection** object represents a connection with a database. You may have several Connection objects in an application that connects to one or more databases.

4.2 Loading the JDBC-ODBC Bridge and Establishing Connection

To establish a connection with a database, you need to register the ODBC- JDBC Driver by calling the ***forName()*** method from the ***Class*** class and then calling the ***getConnection()*** method from the ***DriverManager*** class.

The ***getConnection()*** method of the ***DriverManager*** class attempts to locate the driver that can connect to the database represented by the JDBC URL passed to the ***getConnection()*** method.

4.3 JDBC URL

The JDBC URL is a string that provides a way of identifying a database. A JDBC URL is divided into three parts:

<protocol>:<subprotocol>:<subname>

- **<protocol>** in a JDBC URL is always **jdbc.**
- **<subprotocol>** is the name of the database connectivity mechanism. If the mechanism of retrieving the data is ODBC-JDBC bridge, the subprotocol must be **odbc.**
- **<subname>** is used to identify the database.

Example: JDBC URL

```
String url = "jdbc:odbc:MyDataSource";
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection(url);
```

4.4 Using the Statement Object

You can use the statement object to send simple queries to the database as shown in the sample QueryApp program given in the figure 11.2.

```
import java.sql.*;

public class Program1 {
	public static void main(String arg[]) {
		ResultSet result;
		try {
			Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
			Connection con;
			con = DriverManager.getConnection.
				("jdbc:odbc:MyDataSource", "sa",
				"");
			Statement stat = con.createStatement();
			result = stat.executeQuery
				("select * from publishers");
			while (result.next()) {
				System.out.println(result.getString(2));
			}
		} catch (Exception e) {
		}
	}

}
```

Fig 11.2: Usage of Statement Object

In the above QueryApp example:

- The JDBC-ODBC bridge driver is loaded.
- The Connection object is initialized using the getConnection() method.
- The Statement object is created using the createStatement() method.
- Finally, a simple query is executed using executeQuery() method of the Statement object.

The **Statement** object allows you to execute simple queries. It has the following three methods that can be used for the purpose of querying:

- The ***executeQuery()*** method executes a simple query and returns a single ResultSet object.
- The ***executeUpdate()*** method executes an SQL INSERT, UPDATE or DELETE statement.
- The ***execute()*** method executes an SQL statement that may return multiple results.

4.5 The ResultSet Object

The ResultSet object provides you with methods to access data from the table. Executing a statement usually generates a ResultSet object. It maintains a cursor pointing to its current row of data. Initially the cursor is

positioned before the first row. The ***next()*** method moves the cursor to the next row. You can access data from the ResultSet rows by calling the ***getXXX()*** method where XXX is the data type. The code given in the figure 11.3 queries the database and process the ResultSet.

```
import java.sql.*;

public class Program1 {
	public static void main(String arg[]) {
		ResultSet result;
		try {

			Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
			Connection con;
			con = DriverManager.getConnection("jdbc:odbc:MyDataSource",
"sa",
				"");
			Statement stat = con.createStatement();
			result = stat.executeQuery("select * from publishers");
			while (result.next()) {
				System.out.println(result.getString(2));
			  }
		} catch (Exception e) {
		}
	}
}
```

Fig 11.3: The ResultSet Object

4.6 Using the PreparedStatement Object

You have to develop an application that queries the database according to the search criteria specified by a user. For example, the user supplies the publisher ID and wants to see the details of that publisher.

select * from publishers where pub_id=?

To make it possible, you need to prepare a query statement at runtime with an appropriate value in the where clause.

The **PreparedStatement** object allows you to execute parameterized queries. The **PreparedStatement** object is created using the **prepare Statement()** method of the **Connection** object.

stat=con.prepareStatement ("select * from publishers where pub_id=?");

The ***prepareStatement(),*** method of the Connection object takes an SQL statement as a parameter. The SQL statement can contain placeholders that can be replaced by INPUT parameters at runtime.

The '?' symbols is a placeholder that can be replaced by the INPUT parameters at runtime.

4.7 Passing INPUT Parameters:

Before executing a **PreparedStatement** object, you must set the value of each '?' parameter. This is done by calling an appropriate ***setXXX()*** method, where XXX is the data type of the parameter.

```
stat.setString(1, pid.getText());
ResultSet result=stat.executeQuery();
```

The following code makes use of the **PreparedStatement** object:

```
import java.sql.*;
import javax.swing.*;

import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.*;

public class PreparedQueryApp extends JFrame implements ActionListener {
	JTextField pid;
	JTextField pname;
	JButton query;
	static ResultSet result;
	static Connection con;
	static PreparedStatement stat;

	public PreparedQueryApp() {
		super("The Query Application");
		setLayout(new GridLayout(5, 1));
		pid = new JTextField();
		pname = new JTextField();
		query = new JButton("Query");
		add(new JLabel("Publisher ID:"));
		pid.setSize(1,15);
		add(pid);
		add(new JLabel("Publisher Name"));
		add(pname);
		add(query);
		query.addActionListener(this);


		setSize(750, 500);
	    setVisible(true);
	}
	public static void main(String a[]) {

		PreparedQueryApp obj = new PreparedQueryApp();
		try {
		     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
		       con =  DriverManager.getConnection("jdbc:odbc:MyDataSource",
"sa",
		                "");
		       stat = con.prepareStatement(
		                "select * from publishers where pub_id =?");
		} catch (Exception e) {
		}
```

```
        obj.showRecord(result);
        }


        public void actionPerformed(ActionEvent event) {
                if (event.getSource() == query) {
                        try {

                                stat.setString(1, pid.getText());
                                result = stat.executeQuery();
                                result.next();

                        } catch (Exception e) {
                        }

                        showRecord(result);
                }
        }

        public void showRecord(ResultSet result) {
                try {

                        pid.setText(result.getString(1));
                        pname.setText(result.getString(2));

                } catch (Exception e) {

                }

        }
}
```

In the above example:

- The **PreparedStatement** object is created using the ***prepare Statement()*** method.
- The parameters of the **PreparedStatement** object are initialized when the user clicks on the **Query** button.
- The query is then executed using the ***executeQuery()*** method and the result is displayed in the corresponding controls.

SELF ASSESSMENT QUESTIONS – 2

3. <protocol> in a JDBC URL is always_____.
4. <subprotocol> in JDBC URL must be_____.
5. _____is used to identify the database in the JDBC URL.
6. A_____is a collection of related information and a_____is the software that provides you with a mechanism to manipulate data in the database.
7. MS-Access stores the data in_____file format whereas MS-SQLServer stores the data in a_____file format.

5. TRANSACTION MANAGEMENT

Very often there is a need to group multiple statements into a single unit. If one of the statements fail, then the changes done by the previous statements in the group should be reverted. Also, the changes done by the statements should be made available to other applications only when the entire set of statements are executed. This is can be achieved by using Transaction Management.

Transaction refers to creating a boundary or demarcation around a set of SQL statements so that they are treated as a single operation. "Commit" is a term that refers to saving the changes to the DB. "Rollback" refers to the term of undoing the changes done by the SQL statements. A transaction results in the changes being committed or rolled back depending on the success of all the statements within the transaction.

A transaction follows the ACID Property. They are:

Atomic (A): All the actions or statements within the transaction are executed or none of them is executed. There is no case where changes done by a few are retained and the remaining are discarded.

Consistency(C): The data is shown in a consistent fashion to all other applications or codes.

Isolation (I): The statements are protected from the impact of other statements or transactions that are being executed.

Durability (D): The transaction maintains logs. As a result, during a failure it is possible to restore the system into a stable state using these logs.

For a transaction to begin a connection object is called with 'auto commit' set to False. A transaction ends with with a call to commit() or rollback() statement.

The code below demonstrates a hypothetical case where the student and marks details are entered into a system. If there are any errors in inserting the marks, then the student record also should not be persisted. As shown in the code below, initially, the 'auto commit' is set to false. The related statements are grouped within a try block. If all the statements are successful, then the commit() statement is executed. If an error occurs, then the rollback() statement is executed. Finally, the resources are returned.

```
import java.sql.*;

class JavaTransaction {
	public static void main(String args[]) throws Exception {
		Connection con =
DriverManager.getConnection("jdbc:odbc:MyDataSource", "sa",
			"");
		con.setAutoCommit(false);
		Statement stmt = con.createStatement();

		try {
			stmt.executeUpdate("insert into student values(1,'Sam','MCA
Sem3')");
			stmt.executeUpdate("insert into marks
values('DCA7102','Programming in Java',90)");
			con.commit();
		} catch (Exception e) {
			con.rollback();
		} finally {
			stmt.close();
			con.close();
		}
	}
}
```

Thus the transaction ensures that the all the changes are reflected in the DB if successful. Alternaltively, it ensures no changes are reflected even if error occurs.

6. TERMINAL QUESTIONS

1. What is JDBC?
2. Draw and explain the JDBC Application Architecture?
3. How will you load the ODBC driver?
4. Explain transaction management in JDBC with example.

7. ANSWERS

Self-Assessment Questions

1. Application Program Interface
2. Oracle, MySql etc
3. jdbc
4. odbc
5. <subname>
6. database and DBMS
7. .MDB and .DAT

Terminal Questions

1. JDBC stands for Java Data Base Connectivity. It works on top of ODBC (Open Data Base Connectivity). (Refer Section 2)
2. (Refer section 3)
3. Click on the ODBC Data Source icon in the Control Panel and follow the instructions. (Refer Section 4)
4. Refer to section 4.