

Learning Outcome: Programming and Software Development.

Assessment Standard: Query a database via an application package and a programming language.

Database connectivity allows for the manipulation of a database file via a Java class file.

The **JdbcOdbcDriver** has to be loaded in order to establish a successful connection between the Java programming language and the database program.

A connection between the java compiler and a database requires the registering of the database file with an alias as a data source, thereby allowing the 'DriverManager' connection to successfully establish a connection.

The Microsoft Access program will be used as the connecting database

There are several steps that need to be followed in order to read and manipulate contents of a database table in a java class.

Remember all files related to reading and writing must be stored in the same folder.

Step 1. Create a table in the database program.

Step 2. Register the database file as a data source via the control panel.

Step 3. Write a class to include the following:

- Establish the successful loading of drivers.
- Establish a connection to the data source.
- Execute queries using the ResultSet method.
- Execute file maintenance.

Step 4. Write a User class to link the class written in step 3, to display data using the required queries and to invoke file maintenance.

The necessary code to instantiate an object with the data source name is required.

We will use an example to illustrate the steps above. Create a folder called LearnerDetails.

Step 1.

Create the following file in Microsoft Access with filename **LearnerDB**.

Save the file in the LearnerDetails folder.

The fields are as follows:

Admission Number, Surname and Name: String

Fields - Age: Numeric field.

Select the Admission Number as the **primary key**

field. Save the table as **LearnerTbl**.

Select the Datasheet view to populate the fields in the table. You may choose your own data.

Step 2:

Register the LearnerDB file as a **data source** using an alias called "**LearnerFile**".

Do not attempt to create a data source with the LearnerTbl opened. A message "invalid path" will appear.

Step 3:

Java class DB used to establish a successful connection of drivers, data source, ResultSet method and updateTbl methods.

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

public class DB
{
    Connection conn;
    DB (String name) // Parameterised
    constructor to receive the data source
    {
        try // search for the required drivers
        {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"
); System.out.println("Driver successfully
loaded"); }
            catch (ClassNotFoundException e)
            {
                System.out.println("Unable to connect"); }
            try // search for the required data source
            {
                conn =
                DriverManager.getConnection("jdbc:odbc:"+na
me); System.out.println("Connection to db
"+name+" successfully established"); }
            catch (Exception e)
            { System.out.println("Unable to connect"); }
        }
        ResultSet queryTbl(String sqlStmt)throws
SQLException
        { // query method to handle queries received
        via 'sqlStmt'.
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(sqlStmt);
            return rs; }
        void updateTbl(String update)throws
SQLException
        // method to handle maintenance
        {
            Statement stmt = conn.createStatement();
            stmt.executeUpdate(update);
            stmt.close(); } }
}
```

Step 4:

Write a class UseDB to instantiate an object using the parameterised constructor in the DB class.

Use the class to display the contents of the table LearnerTbl.

```
import java.sql.*;

public class UseDB
{
    DB myLearner = new DB("LearnerFile"); //
Sends the data source to the
constructor in the DB class

    UseDB()throws SQLException
    {
        try
        {
            displayLearnerTbl("Our School
Learners"); // heading
        }
        catch(Exception e )
        {
```

```
System.out.println("Cannot find contents!!!
Please check connection"); } }
}
```

```
public static void main (String [] args) throws
SQLException
{
```

```
new UseDB(); // calls the constructor method }
```

```
Void displayLearnerTbl(String heading) throws
SQLException
{
```

```
throws SQLException
{
    try
    {
```

```
ResultSet rs = myLearner.queryTbl("SELECT *
from LearnerTbl");
```

```
System.out.println("\t\t"+heading);
System.out.println("\t\t-----
-----");
```

```
System.out.println("Adm No"+spaces("Adm
No",12)
```

```
+ "Surname"+spaces("Surname",20)
```

```
+ "Name"+spaces("Name",15) + "Age
```

```
No"+spaces("Age",12));
```

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

```
while (rs.next())
{
```

```
String a = rs.getString("AdmNo");
```

```
String s = rs.getString("Surname");
```

```
String n =
```

```
rs.getString("Name");
```

```
int age = rs.getInt("Age");
```

```
System.out.println(a+spaces(a,12)
```

```
+s+spaces(s,20)
```

```
+n+spaces(n,15)
```

```
+age+spaces(" "+age,12));
```

```
// numeric fields must be converted to
```

String before the spaces method is called. The actual parameters must be of

```
type (String,int) }
```

```
catch (SQLException e)
{
```

```
System.out.println(e.getMessage()); } }
```

// The spaces method ensures neat spacing in columns.

// It returns the (column width) - (length of the field).

```
String spaces(String s, int w)
{
```

```
String sendSpace = "";
```

```
for (int i = 0; i <= w - s.length(); i++)
{
```

```
SendSpace = sendSpace + " ";
```

```
}
return sendSpace; }
```

