How to manipulate a Microsoft Access Database via JDBC

This will teach you how to connect to a Microsoft Access database. Once you are connected, you may run any SQL statement that is allowable on Access, such as:

- A **SELECT** statement to retrieve data
- An **INSERT** statement to add data
- A **DELETE** statement to remove data
- A **CREATE TABLE** statement to build a new table
- A **DROP TABLE** statement to destroy a table

Steps to take:

There are three things we need to do to manipulate a MS Access database:
1) Set up Java to understand ODBC (Load Driver).
2) Get a connection to our MS Access Database (Establishing Connection).
3) Run a SQL statement (Creating and executing SQL Statement).

1) First we need to set up Java to understand how to communicate with an ODBC data source:

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

2) After we set up the DriverManager, we need to get a Connection
There are two ways to get a connection from your Microsoft Access Database:

1. Get a connection by accessing the Database Directly
2. The simpler way, but may not work on all systems!
3. Set the Access Database up as an ODBC DSN and get a connection through that
4. A little more complex, but will work on any system, and will work even if you don't already have a Microsoft Access Database!

3) Once you have gained access to the Database (been granted a connection), you are ready to try:

- Running a SQL Statement on your Access Database

Step 1) Set up your DriverManager to understand ODBC data sources

The first thing we must do in order to manipulate data in the database is to be granted a connection to the database. This connection, referenced in the Java language as an Object of type **java.sql.Connection**, is handed out by the **DriverManager**. We tell the DriverManager what type of driver to use to handle the connections to databases, and from there, ask it to give us a connection to a particular database of that type.

For this tutorial, we are interested in accessing a Microsoft Access database. Microsoft has developed a data access method called **ODBC**, and MS Access databases understand this method. We cannot make a connection directly to an ODBC data source from Java, but Sun has provided a **bridge** from JDBC to ODBC. This bridge gives the DriverManager the understanding of how to communicate with an ODBC (ie a MS Access) data source.

So the first thing we'll do is set up our DriverManager and let it know that we want to communicate with ODBC data sources via the JDBC:ODBC bridge. We do this by calling the static forName() method of the Class class. Here is an entire program that accomplishes what we're after:

```
class Test
{
    public static void main(String[] args)
   {
       try
      {
         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      }
     catch (Exception e)
     {
       System.out.println("Error: " + e);
     }
  }
}
//save this code into a file called Test.java and compile it
```

Notice the TRY-CATCH block. The forName() method might throw a *ClassNotFoundException*. This really can't happen with the JDBC:ODBC bridge, since it's built in to the Java API, but we still have to catch it. If you compile and run this code, it's pretty boring. In fact, if it produces any output, then that means that you've encountered an error! But it shows how to get your DriverManager set.

We're now ready to try and get a connection to our specific database so we can start to run SQL statements on it!

Step 2 method 1) Get a connection by direct access

One way to get a connection is to go directly after the MS Access database file. This can be a quick and easy way to do things, but I have seen this not work on some windows machines. Don't ask me why - I just know that it works sometimes and it doesn't others...
Here is a complete sample program getting a connection to a MS Access database on my hard drive at **C: mdbTEST.mdb**. This sample includes the lines required to set the DriverManager up for ODBC data sources:

```
import java.sql.*;
class Test
{
 public static void main(String[] args)
 {
   try
  {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
// set this to a MS Access DB you have on your machine
String filename = "d:/java/mdbTEST.mdb";
String database = "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=";
database+= filename.trim() + ";DriverID=22;READONLY=true}"; // add on to the end
// now we can get the connection from the DriverManager
Connection con = DriverManager.getConnection( database ,"","");
 }
 catch (Exception e) {
System.out.println("Error: " + e);
 }
 }
}
//save this code into a file called Test.java and compile it
```

Notice that this time I imported the **java.sql** package - this gives us usage of the java.sql.Connection object.

The line that we are interested in here is the line

```
Connection con = DriverManager.getConnection( database ,"","");
```

What we are trying to do is get a **Connection** object (named *con*) to be built for us by the DriverManager. The variable *database* is the URL to the ODBC data source, and the two sets of empty quotes ("","") indicate that we are not using a username or password.

In order to have this program run successfully, you have to have an MS Access database located at *filename* location. Edit this line of code and set it to a valid MS Access database on your machine. If you do not already have an MS Access database, please jump down to Set the Access Database up as an ODBC DSN section, which shows how to create an empty MS Access database.

If you do have a MS Access database, and this is working correctly, then you're ready to Run an SQL Statement!

Step 2 method 2) Set up a DSN and get a connection through that

BACK TO TOP

Microsoft has provided a method to build a quick Jet-Engine database on your computer without the need for any specific database software (it comes standard with Windows). Using this method, we can even create a blank Microsoft Access database without having MS Access installed!
As we learned earlier, MS Access data bases can be connected to via ODBC. Instead of accessing the database directly, we can access it via a Data Source Name (DSN). Here's how to set up a DSN on your system:

1. Open Windows' ODBC Data Source Administrator as follows:
2.
   - In Windows 95, 98, or NT, choose Start > Settings > Control Panel, then double-click the ODBC Data Sources icon. Depending on your system, the icon could also be called ODBC or 32bit ODBC.
   - In Windows 2000, choose Start > Settings > Control Panel > Administrative Tools > Data Sources.
3. In the ODBC Data Source Administrator dialog box, click the System DSN tab.
4. Click Add to add a new DSN to the list.
5. Scroll down and select the Microsoft Access (.MDB) driver
6. Type in the name "mdbTEST" (no quotes, but leave the cases the same) for the Data Source Name
7. Click CREATE and select a file to save the database to (I chose "d:\java\mdbTEST.mdb") - this creates a new blank MS Access database!
8. Click "ok" all the way out

Now our data source is done! Here's a complete program showing how to access your new DSN data source:

```
import java.sql.*;
public class Test
{
    public static void main(String[] args)
   {
     // change this to whatever your DSN is
    String dataSourceName = "mdbTEST";
    String dbURL = "jdbc:odbc:" + dataSourceName;
    try {
```

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection(dbURL, "","");
        }
     catch (Exception err)
    {
        System.out.println( "Error: " + err );
     }
  }
}
//save this code into a file called Test.java and compile it
```

As stated in the code, modify the variable *dataSourceName* to whatever you named your DSN in step 5 from above.

If this complies and runs successfully, it should produce no output. If you get an error, something isn't set up right - give it another shot!

Once this is working correctly, then you're ready to Run an SQL Statement!

Step 3) Running a SQL Statement on your Access Database

BACK TO TOP

Once you have your connection, you can manipulate data within the database. In order to run a SQL query, you need to do 2 things:

1. Create a **Statement** from the connection you have made
2.
3. Get a **ResultSet** by executing a query (your insert/delete/etc. statement) on that statement

Now lets learn how to make a **statement**, execute a query and display a the **ResultSet** from that query.

Refer to the following complete program for an understanding of these concepts (details follow):

*This code assumes that you have used the DSN method (Step 2 method 2) to create a DSN named **mdbTest**. If you have not, you'll need to modify this code to work for a direct connection as explained in Step 2 method 1.*

```
import java.sql.*;
public class Test
{
 public static void main(String[] args)
 {
 try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
/* the next 3 lines are Step 2 method 2 from above - you could use the direct
access method (Step 2 method 1) istead if you wanted */
String dataSourceName = "mdbTEST";
String dbURL = "jdbc:odbc:" + dataSourceName;
Connection con = DriverManager.getConnection(dbURL, "","");
// try and create a java.sql.Statement so we can run queries
Statement s = con.createStatement();
s.execute("create table TEST12345 ( column_name integer )"); // create a table
s.execute("insert into TEST12345 values(1)"); // insert some data into the table
s.execute("select column_name from TEST12345"); // select the data from the table
ResultSet rs = s.getResultSet(); // get any ResultSet that came from our query
if (rs != null) // if rs == null, then there is no ResultSet to view
while ( rs.next() ) // this will step through our data row-by-row
{
/* the next line will get the first column in our current row's ResultSet
 as a String ( getString( columnNumber) ) and output it to the screen */
System.out.println("Data from column_name: " + rs.getString(1) );
```

```
}
s.execute("drop table TEST12345");
s.close(); // close the Statement to let the database know we're done with it
con.close(); // close the Connection to let the database know we're done with it
 }
 catch (Exception err) {
System.out.println("ERROR: " + err);
 }
 }
}
```
//save this code into a file called **Test.java** and compile it

If this program compiles and runs successfully, you should see some pretty boring output:

```
Data from column_name: 1
```

While that may not seem like much, let's take a quick look at what we've accomplished in the code.

1. First, we set the DriverManager to understand ODBC data sources.
2. Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
3. Then, we got a connection via the DSN as per Step 2 method 2:
4. String dataSourceName = "mdbTEST";
5. String dbURL = "jdbc:odbc:" + dataSourceName;
6. Connection con = DriverManager.getConnection(dbURL, "","");
7. We could have used the direct method instead to get our connection.
8. Next, we created a java.sql.Statement Object so we could run some queries:
9. Statement s = con.createStatement();
10. Then came the exciting stuff - we ran some queries and made some changes!
11. s.execute("create table TEST12345 (column_name integer)"); // create a table
12. s.execute("insert into TEST12345 values(1)"); // insert some data into the table
13. s.execute("select column_name from TEST12345"); // select the data from the table
14. The next part might be a little strange - when we ran our **select** query (see above), it produced a java.sql.ResultSet. A ResultSet is a Java object that contains the resulting data from the query that was run - in this case, all the data from the column **column_name** in the table **TEST12345**.
15. ResultSet rs = s.getResultSet(); // get any ResultSet that came from our query
16. if (rs != null) // if rs == null, then there is no ResultSet to view
17. while (rs.next()) // this will step through our data row-by-row
18. {
19. /* the next line will get the first column in our current row's ResultSet
20. as a String (getString(columnNumber)) and output it to the screen */
21. System.out.println("Data from column_name: " + rs.getString(1));
22. }

    As you can see, if the ResultSet object **rs** equals null, then we just skip by the entire **while** loop. But since we should have some data in there, we do this *while (rs.next())* bit.

    What that means is: ***while there is still data to be had in this result set, loop through this block of code and do something with the current row in the result set, then move on to the next row.***

    What we're doing is looping through the result set, and for every row grabbing the first column of data and printing it to the screen. We are using the method provided in the result set called getString(int columnNumber) to get the data from the first column in our result set as as **String** object, and then we're just printing it out via *System.out.println*.

    We know that the data in our ResultSet is of type String, since we just built the table a couple of lines before. There are other getXXX methods provided by ResultSet, like getInt() and getFloat(), depending on what type of data you are trying to get out of the ResultSet. Please refer to the JSDK API for a full description of the ResultSet methods.

23. After that we just cleaned up our database by dropping (completely removing) the newly created table:

24. s.execute("drop table TEST12345");
25. Lastly, we need to close the Statement and Connection objects. This tells the database that we are done using them and that the database can free those resources up for someone else to use. **It is very important to close your connections - failure to do so can over time crash your database!** While this isn't too important with a MS Access database, the same rules apply for any data base (like Oracle, MS SQL, etc.)
26. s.close(); // close the Statement to let the database know we're done with it
27. con.close(); // close the Connection to let the database know we're done with it