DO NOT MAKE THE ACCESS FILE PASSWORD PROTECTED.

[**Manipulating an Access database from Java without ODBC**](http://stackoverflow.com/questions/21955256/manipulating-an-access-database-from-java-without-odbc)

[](http://engine.adzerk.net/r?e=&s=XP4T0OI6o8mm3E9W4-bc1L9Nu8k)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| up vote20down vote[favorite](http://stackoverflow.com/questions/21955256/manipulating-an-access-database-from-java-without-odbc)  **13** | I want to manipulate a Microsoft Access database (.accdb or .mdb file) from my Java project. I don't want to use the JDBC-ODBC Bridge and the Access ODBC driver from Microsoft because:   * the JDBC-ODBC Bridge has been removed from Java SE 8 and is not supported (ref: [here](http://docs.oracle.com/javase/7/docs/technotes/guides/jdbc/bridge.html)), * the JDBC-ODBC Bridge does not work properly with the Access ODBC driver when text includes Unicode characters with code points above U+00FF (ref: [here](http://stackoverflow.com/a/20982205/2144390)), so such a setup would not be able to handle characters such as Greek, Russian, Chinese, Arabic, etc., * the Access ODBC driver from Microsoft only works in Windows, and * there are separate 32-bit and 64-bit versions of the Access Database Engine (and ODBC driver) which can be a nuisance for deployment.   I have seen other answers mentioning a JDBC driver for Access databases named [UCanAccess](http://ucanaccess.sourceforge.net/site.html). How can I set up my Java project to use this approach?  (Answers suggesting better ways of working with Access databases from Java would also be most welcome.)  [java](http://stackoverflow.com/questions/tagged/java) [eclipse](http://stackoverflow.com/questions/tagged/eclipse) [ms-access](http://stackoverflow.com/questions/tagged/ms-access) [netbeans-7](http://stackoverflow.com/questions/tagged/netbeans-7) [ucanaccess](http://stackoverflow.com/questions/tagged/ucanaccess" \o "show questions tagged 'ucanaccess')   |  |  |  | | --- | --- | --- | | [share](http://stackoverflow.com/q/21955256)|[improve this question](http://stackoverflow.com/posts/21955256/edit) | [edited Mar 12 at 15:07](http://stackoverflow.com/posts/21955256/revisions) | asked Feb 22 at 13:55  [[http://i.stack.imgur.com/6QjmY.jpg?s=32&g=1](http://stackoverflow.com/users/2144390/gord-thompson)](http://stackoverflow.com/users/2144390/gord-thompson)  [Gord Thompson](http://stackoverflow.com/users/2144390/gord-thompson) **30.9k**61647 | |
|  | |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | |  |  | | Gord I would like to get in touch with you. Email address, is it possible? Cheers –  [bonCodigo](http://stackoverflow.com/users/1389394/boncodigo" \o "8782 reputation) [Mar 18 at 15:28](http://stackoverflow.com/questions/21955256/manipulating-an-access-database-from-java-without-odbc#comment34203104_21955256) |   add a comment |

**1 Answer**

[active](http://stackoverflow.com/questions/21955256/manipulating-an-access-database-from-java-without-odbc?answertab=active#tab-top)[oldest](http://stackoverflow.com/questions/21955256/manipulating-an-access-database-from-java-without-odbc?answertab=oldest#tab-top)[votes](http://stackoverflow.com/questions/21955256/manipulating-an-access-database-from-java-without-odbc?answertab=votes#tab-top)

|  |  |
| --- | --- |
| up vote26down voteaccepted | [UCanAccess](http://ucanaccess.sourceforge.net/site.html) is a pure Java JDBC driver that allows us to read from and write to Access databases without using ODBC. It uses two other packages, [Jackcess](http://jackcess.sourceforge.net/) and [HSQLDB](http://hsqldb.org/), to perform these tasks. The following is a brief overview of how to get it set up.  **Add the JARs to your project**  As mentioned above, UCanAccess requires Jackcess and HSQLDB. Jackcess in turn has its own[dependencies](http://jackcess.sourceforge.net/dependencies.html). So to use UCanAccess you will need to include the following components:  [UCanAccess](http://ucanaccess.sourceforge.net/site.html) (ucanaccess-2.x.x.jar) [HSQLDB](http://hsqldb.org/) (hsqldb.jar, version 2.2.5 or newer) [Jackcess](http://jackcess.sourceforge.net/) (jackcess-2.x.x.jar) [commons-lang](http://commons.apache.org/proper/commons-lang/) (commons-lang-2.4.jar, or newer) [commons-logging](http://commons.apache.org/proper/commons-logging/) (commons-logging-1.0.4.jar, or newer)  Fortunately, UCanAccess includes all of the required JAR files in its distribution file. When you unzip it you will see something like  ucanaccess-2.0.4.jar  lib/commons-lang-2.6.jar  lib/commons-logging-1.1.3.jar  lib/hsqldb.jar  lib/jackcess-2.0.3.jar  All you need to do is add all five (5) JARs to your project.  **Eclipse:** Right-click the project in Package Explorer and choose Build Path > Configure Build Path.... Click the "Add External JARs..." button to add each of the five (5) JARs. When you are finished your Java Build Path should look something like this  eclBuildPath.png  **NetBeans:** Expand the tree view for your project, right-click the "Libraries" folder and choose "Add JAR/Folder...", then browse to the JAR file.  nbAddJar.png  After adding all five (5) JAR files the "Libraries" folder should look something like this:  nbLibraries.png  **You're set!**  Now "U Can Access" data in .accdb and .mdb files using code like this  // assumes...  // import java.sql.\*;  Connection conn=DriverManager.getConnection(  "jdbc:ucanaccess://C:/\_\_tmp/test/zzz.accdb");  Statement s = conn.createStatement();  ResultSet rs = s.executeQuery("SELECT [LastName] FROM [Clients]");  while (rs.next()) {  System.out.println(rs.getString(1));  }  **Disclosure**  I have no involvement in or affiliation with the UCanAccess project. I just use it. |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Getting Started**

The distribution comes with both a number of Junit Test Cases (package *net.ucanaccess.test*) and a simple java example class (*net.ucanaccess.example.Example*) which illustrate how UCanAccess may be used.

**Getting UCanAccess connection**

*Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");  
Connection conn=DriverManager.getConnection("jdbc:ucanaccess://<mdb or accdb file path>",user, password);*   
for example:   
*Connection conn=DriverManager.getConnection("jdbc:ucanaccess://c:/pippo.mdb");*

**Driver Properties**

**Memory**: set if hsqldb will work only in memory. Default=true.

**Newdatabaseversion**: UCanAccess create a new access file in the specified version if this parameter is setted and the file specified in the jdbc url does not exist.   
Valid values for this parameter are: V2000, V2003, V2007, V2010.

**Lockmdb**: if lockmdb is enabled (lockmdb=true), Ucanaccess locks the mdb file to prevent access from other processes. Default=false.  
  
**Showschema**: if true, catalog "PUBLIC" and schema "PUBLIC" and others hsqldb schemas are shown in the DatabaseMetadata implementation. Default=false. Using *showschema=true* is raccomanded with Squirrel.

**InactivityTimeout** (since UcanAccess 0.1.1): it is used only with the default memory mode (Memory=true). If there are no active connections for the inactivityTimeout period (in minutes) hsqldb will be temporarily shutted down and also the filesystem resources will be released. It will be restarted at the next connection. Setting InactivityTimeout=0 can be used for avoiding this behavior. Default=2 minutes.

**SingleConnection**(since UcanAccess 1.0.0): it is used for for etl job, scheduled tasks or "one-shot" use of UCanAccess with only one connection open. All resources (memory and filesystem) will be released at the closing of the connection. Default=false.

**Encrypt**(since UcanaAccess 1.0.4): it allows hsqldb files encryption. You can use it in conjunction with Memory=false. Default=false.

**Sysschema**(since UcanaAccess 1.0.4):If true the msaccess system tables will be loaded in the "sys" readonly schema. Default=false.

**Ignorecase**: Disables (ignorecase = true) or enables (ignorecase = false) the case sensitivity of text comparison. Default=true.

for example:   
*Connection conn=DriverManager.getConnection("jdbc:ucanaccess://c:/pippo.mdb;lockmdb=true;ignorecase=true");*

**Remap (since UCanAccess-2.0.2)**:It allows to remap the paths to one or more external linked databases.

The syntax is:   
*jdbc:ucanaccess://C:/db/<main db>;remap=<original db1 path>|<new db1 path>&<original i-db path>|<new i-db path>* e.g.,  
*jdbc:ucanaccess://C:/db/main.mdb;remap=c:\db\linkee1.mdb|C:\pluto\linkee1.mdb&c:\db\linkee2.mdb|C:\pluto\linkee2.mdb.*   
Original path and new path must be separated by '|', different pairs must be separated by '&' .

**KeepMirror (since UCanAccess-2.0.2)** to keep the mirror hsqldb database after the VM ends and so to reuse it in the following VM processes.  
It forces *memory=false*.  
In other words, using this parameter, the time-expensive process of creating and populating the database hsqldb (with memory=false), is executed only once.

*jdbc:ucanaccess://C:/db/main.mdb;keepMirror=C:/db/mirrorName*  
Reusing the same keepMirror setting (e.g. keepMirror=C:/db/mirrorName) at the next VM process execution allows you to drammatically reduce the time of first connection.

It should be used only with very large databases (e.g., 1GB with many binary OLE objects) and when:

* You have to change the default memory=true setting because you can't allocate sufficient heap space (Xmx VM paramenter).
* UCanaccess takes too much time to establish the first connection (because it's populating the mirror hsqldb).

Pay attention! If the access database is modified by a different program (so not using UCanAccess) after the hsqldb creation, UCanAccess recreates and repopulates the whole mirror hsqldb for avoiding unchecked misalignments.   
  
**ColumnOrder (since UCanAccess-2.0.9)** to use, in your SQL, the "display" order ("data" order is still the default).

*jdbc:ucanaccess://c:/db/cico.mdb;COLUMNORDER=DISPLAY.*

Once a column order has been setted on the first connection to a given database, the column order setting will be the same for all the following connections to that database, in the whole VM life. 

**JackcessOpener (since UCanAccess-0.0.2)**: in order to use *Jackcess Encrypt* extension, you can specify a class that implements the*net.ucanaccess.jdbc.JackcessOpenerInterface* interface (in that case you need additional jar files in your classpath: a version of jackcess-encrypt.jar and all related dependencies).  
The following example shows a custom integration with *Jackcess Encrypt* for opening a Money (.mny) file:

***Example for UCanAccess2. Notice that you must use UCanAccess 2.x.x with jackcess-encrypt-2.x.x and all related dependecies*** *package yourPackage.example;  
  
import java.io.File;  
import net.ucanaccess.jdbc.JackcessOpenerInterface;  
//imports from Jackcess Encrypt  
import com.healthmarketscience.jackcess.CryptCodecProvider;  
import com.healthmarketscience.jackcess.Database;  
import com.healthmarketscience.jackcess.DatabaseBuilder;  
  
import net.ucanaccess.jdbc.JackcessOpenerInterface;  
  
public class CryptCodecOpener implements JackcessOpenerInterface {  
  public Database open(File fl,String pwd) throws IOException {  
   DatabaseBuilder dbd =new DatabaseBuilder(fl);  
   dbd.setAutoSync(false);  
   dbd.setCodecProvider(new CryptCodecProvider(pwd));  
   dbd.setReadOnly(false);  
   return dbd.open();  
     
  }  
  //Notice that the parameter setting****autosync =true****is recommended with UCanAccess for performance reasons.   
  //UCanAccess flushes the updates to disk at transaction end.   
  //For more details about autosync parameter (and related tradeoff), see the Jackcess documentation.   
}*

*Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");  
Connection conn=  
DriverManager.getConnection("jdbc:ucanaccess:///opt/prova1.mny;jackcessOpener=yourPackage.example.CryptCodecOpener", "sa", pwd);  
...*

***For old UCanAccess versions (0.x.x,1.x.x) JackcessOpenerInterface should be implemented in a different way, because of the jackcess API changes from jackcess1 to jackcess2.*** *package yourPackage.example;  
  
import java.io.File;  
import java.io.IOException;  
//imports from Jackcess Encrypt  
import com.healthmarketscience.jackcess.CryptCodecProvider;  
import com.healthmarketscience.jackcess.Database;  
  
import net.ucanaccess.jdbc.JackcessOpenerInterface;  
  
public class CryptCodecOpener implements JackcessOpenerInterface {  
  public Database open(File fl,String pwd) throws IOException {  
   return Database.open(fl, true, true, null, null, new CryptCodecProvider(pwd));  
  }  
}  
  
...*

**Identifiers escaping**

You should escaping identifiers including a blank space using square brackets or backtick character (`).  
Also you shold escaping identifiers that equal a SQL keyword (e.g. SELECT \* FROM [SELECT] or SELECT \* FROM `SELECT`).  
SQL Keywords are: ALL, AND,ANY, AS, AT, AVG, BETWEEN, BOTH, BY, CALL, CASE, CAST, COALESCE, CONSTRAINT, CORRESPONDING, CONVERT, COUNT, CREATE, CROSS, DEFAULT, DISTINCT,DO, DROP, ELSE, EVERY, EXISTS, EXCEPT, FOR, FROM, FULL, GRANT, GROUP, HAVING, IN, INNER , INTERSECT, INTO, IS, JOIN, LEFT, LEADING, LIKE, MAX , MIN, NATURAL, NOT, NULLIF, ON, ORDER, OR, OUTER, PRIMARY, REFERENCES, RIGHT, SELECT, SET, SOME, STDDEV\_POP, STDDEV\_SAMP, SUM, TABLE, THEN, TO, TRAILING, TRIGGER, UNION, UNIQUE, USING, USER, VALUES, VAR\_POP, VAR\_SAMP, WHEN, WHERE, WITH, END.

**Examples**

**Creating table**

*Statement st =conn.createStatement(); st.execute("CREATE TABLE example1 (id COUNTER PRIMARY KEY,descr text(400), number numeric(12,3), date0 datetime) ");  
...st.execute("create table dkey(c counter , number numeric(23,5) , PRIMARY KEY (C,NUMBER) ");  
...st.execute("create table dtrx(c text , number numeric(23,5) , unique (C,NUMBER)) ");*  
Also Ucanaccess supports create table as () syntax:  
*st.executeUpdate("CREATE TABLE copy as (select \* from example1) WITH DATA ");*

**Executing queries**

The following example shows a **full outer join:**   
*st =conn.createStatement();  
ResultSet rs=st.executeQuery("SELECT \* FROM example3 full outer join example4 on (example3.id=example4.id)");*

**Using access functions**

*ResultSet rs=st.executeQuery("SELECT IIf(descr='Show must go off','tizio','caio&sempronio'&'&Marco Amadei'&' '&now()& RTRIM(' I''m proud of you ')) from example1");*  
  
//Aggregate functions:  
*ResultSet rs=st.executeQuery(*"select id, DCount('\*','t234','1=1') from t234");

More complete examples of access functions implementation are shown in the *net.ucanaccess.test.FunctionsTest* junit test suite. The following functions are already implemented:   
ASC, ATN, CBOOL, CCUR, CDATE, CDBL, CDEC, CINT, CLONG, CSIGN, CSTR, CVAR, DATEADD, DATEDIFF, DATEPART, DATE, DATESERIAL, DATEVALUE, FIX, FORMAT, IIF, INSTR, INSTRREV, ISDATE, ISNUMERIC, INT, IsNull, LEN, MID, MONTHNAME, NOW, NZ, PARTITION, SIGN, SPACE, SQR, STR,, STRING, STRCOMP, STRCONV, STRREVERSE, SWITCH, RND, TIME, TIMESERIAL, VAL, WEEKDAY, WEEKDAYNAME;   
Aggregate and Domain Functions: FIRST, LAST, DCOUNT, DAVG, DSUM, DMAX, DMIN, DFIRST, DLAST, DLOOKUP.   
Financial Functions(since UCanAccess 2.0.7.1): PMT, NPER, IPMT, PPMT, RATE, PV, FV, DDB, SYD, SLN.   
Also you can use the following functions from the hsqldb implementation:   
COS, SIN, LTRIM, RTRIM, UCASE, LCASE;   
Aggregate Functions: COUNT, AVG, SUM, MAX, MIN, STDEV, STDEVP, VAR, VARP.

**User defined functions**

Step 1: implementation of a new function:

*public class XXX{  
...   
 @FunctionType(functionName="justconcat",argumentTypes={AccessType.TEXT,AccessType.TEXT},returnType=AccessType.TEXT)   
  public static String concat(String s1,String s2){   
  return s1+" >>>>"+s2;   
 } …   
}*

Step 2: registration:

*UcanaccessConnection uc=(UcanaccessConnection)this.ucaConn; uc.addFunctions(XXX.class);*

Step 3: trying your new fuction:

*ResultSet rs=st.executeQuery("SELECT justConcat(descr,''&now()) from example1");*

**Using criteria in like clause**

*ResultSet rs=st.executeQuery("select descr from example2 where descr like 'P%'");   
//access \* jolly   
rs=st.executeQuery("select descr from example2 where descr like 'P\*'");   
//number and interval patterns   
rs=st.executeQuery("select descr from example2 where descr like 'P[A-F]###'");   
//number pattern   
rs=st.executeQuery("select descr from example2 where descr like 'C#V##'");*

**Limit and offset SQL pagination**

*rs=st.executeQuery("SELECT \* FROM example2 order by id desc limit 5 offset 1");*  
*rs=st.executeQuery("SELECT TOP 10 \* FROM example2 order by id");*

**Performing transactions**

*conn.setAutoCommit(false);   
st =this.ucaConn.createStatement();  
st.executeUpdate("update example4 set descr='Lugo di Romagna'");   
st.execute("insert into example4 (ID, descr) values(5,'DALLAS')");   
conn.commit();*

**Using updatable ResultSet**

*PreparedStatement ps = super.ucanaccess.prepareStatement( "SELECT \* FROM T1", ResultSet.TYPE\_FORWARD\_ONLY, ResultSet.CONCUR\_UPDATABLE, ResultSet.CLOSE\_CURSORS\_AT\_COMMIT);  
rs = ps.executeQuery();  
rs.next();   
rs.updateString(2, "show must go off");   
rs.updateRow();*

**Working with date type**

*st.execute("INSERT INTO example1 (descr,number,date0) VALUES( \"Show must go up and down\",-113.55446,#11/22/2003 10:42:58 PM#)");*

**Working with OLE type**

*InputStream isDB = rs.getBinaryStream('OleColumn');  
File fl = new File("CopyElisaArt.JPG");  
OutputStream outFile = new FileOutputStream(fl);  
ba = new byte[4096];   
while ((len = isDB.read(ba)) != -1) {   
outFile.write(ba, 0, len); }   
out.flush();   
out.close();   
System.out.println("CopyElisaArt.JPG was created in " + fl.getAbsolutePath());*

**About ' and " escaping**

Both ' and " are valid string delimeters.  
So the following SQL statements are correct:  
*UPDATE TABLE1 SET SURN='D''Ambrogio';(value saved=D'Ambrogio)*  
*UPDATE TABLE1 SET SURN="""D'Ambrogio""";(value saved="D'Ambrogio")*

**Working with Complex Types**

*...  
PreparedStatement ps=conn.prepareStatement("INSERT INTO TABLE1(ID , MEMO\_DATA , APPEND\_MEMO\_DATA , MULTI\_VALUE\_DATA , ATTACH\_DATA) VALUES (?,?,?,?,?)");  
ps.setString(1, "row12");  
ps.setString(2,"ciao");  
ps.setString(3,"to version");  
SingleValue[] svs=new SingleValue[]{new SingleValue("ccc16"),new SingleValue("ccc24") };  
//Array of net.ucanaccess.complex.SingleValue  
  
ps.setObject(4,svs);  
Attachment[] atcs=new Attachment[]{new Attachment(null,"ccc.txt","txt","ddddd ddd".getBytes(), new Date(),null),  
new Attachment(null,"ccczz.txt","txt","ddddd zzddd".getBytes(), new Date(),null) };  
//Array of net.ucanaccess.complex.Attachment  
  
ps.setObject(5,atcs);  
ps.execute();  
...  
  
  
ps=conn.prepareStatement("UPDATE TABLE1 SET APPEND\_MEMO\_DATA='THE CAT' ");  
//APPEND\_MEMO\_DATA is an append only column: a new net.ucanaccess.complex.Version will be added to the version array  
  
ps.execute();  
...  
  
  
ps=super.ucanaccess.prepareStatement("UPDATE TABLE1 SET ATTACH\_DATA=? ");  
Attachment[] atc;  
ps.setObject(1,atc=new Attachment[]{new Attachment(null,"cccsss.cvs","cvs","ddddd ;sssssssssssssssssssddd".getBytes(), new Date(),null) });  
ps.execute();  
//you can't use = operator here: use the Equals or the EqualsIgnoreOrder functions instead ps=conn.prepareStatement("select \* from Table1 where Equals(ATTACH\_DATA,?);  
ps.setObject(1,atc);  
ResultSet rs=ps.executeQuery();  
while(rs.next()){  
  System.out.println(rs.getObject("ATTACH\_DATA"));  
//rs.getObject("ATTACH\_DATA") will return an array of net.ucanaccess.complex.Attachment  
}  
...*

**Filtering on Complex Type Columns (feature available since UCanAccess 2.0.9)**

***The equality operator (=) can't be used on complex type columns, use the following functions instead.   
  
  
-Equals. It returns true if the two specified arrays of SingleValue, Attachment or Version are equal to one another, false otherwise.  
In other words they are equal if all corresponding pairs of elements in the two arrays are equal.*** *//true for all records whose MULTI\_VALUE\_COLUMN value equals [value1,value2]  
...  
PreparedStatement ps=conn.prepareStatement("select \* from TABLE1 WHERE****Equals(MULTI\_VALUE\_COLUMN,?)****");  
ps.setObject(1, SingleValue.multipleValue("value1","value2",));  
ResultSet rs=ps.executeQuery();  
...****-EqualsIgnoreOrder. It returns true if the two specified arrays of SingleValue, Attachment or Version are equal to one another regardless the order, false otherwise.****//true for all records whose MULTI\_VALUE\_COLUMN value equals [value1,value2] or [value2,value1]  
...  
PreparedStatement ps=conn.prepareStatement("select \* from TABLE1 WHERE****EqualsIgnoreOrder(MULTI\_VALUE\_COLUMN,?)****");  
ps.setObject(1, SingleValue.multipleValue("value1","value2",));  
ResultSet rs=ps.executeQuery();  
...****-Contains. It returns true if the Array of SingleValue, Attachment or Version(column value) passed as first argument contains  
all the element of the Array passed as second argument.*** *//true for all records whose MULTI\_VALUE\_COLUMN value contains both value1 and value2  
...  
PreparedStatement ps=conn.prepareStatement("select \* from TABLE1 WHERE****Contains(MULTI\_VALUE\_COLUMN,?)****");  
ps.setObject(1, SingleValue.multipleValue("value1","value3"));  
ResultSet rs=ps.executeQuery();  
...*