

Recital Universal JDBC Driver

Universal JDBC Driver

Recital Corporation,
100 Cummings Center, Suite 318J
Beverly, MA 01915

Recital may have patents and/or patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.
COPYRIGHT ©1988-2006 Recital Corporation. All rights reserved. All Recital products are trademarks or registered trademarks of Recital Corporation, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

Last Updated August, 2006

Index

Installation and Data Source Configuration	
Windows	4
Linux/UNIX	5
Integrity/Alpha OpenVMS	6
Standard Imports	7
Connection URL Format	7
Bridging to Other File Formats	
Connecting to C-ISAM Files	10
Connecting to RMS Files	14
Accessing a Resultset from a Stored Procedure	22

INSTALLATION

The Recital Universal JDBC Driver is available for Windows (2000/XP/2003), Alpha/Integrity OpenVMS and Linux/UNIX. The Recital Universal JDBC Driver requires a running, correctly installed and licensed Recital Database Server.

Java 2 version 1.4 or above is required.

If you have been sent a license for the Recital Database Server, please follow the instructions on the fax or email for installation.

Windows (2000/XP/2003)

Classpath

The following reference must be added to the CLASSPATH environment variable or specified in the classpath parameter to Java commands. The reference should be changed if the JDBC Driver was not installed in the default installation directory.

C:\Program Files\Recital\Universal JDBC Driver\RecitalJDBC.jar

JDBC_TEST

The jdbc_test class is included to allow the JDBC Driver installation to be tested. The jdbc_test.java source code is also included for information purposes.

NOTE: The Sun Java SDK requires the <SDK path>/bin directory to be included in the system PATH, or the full path to be specified on commands, e.g.

To compile:

Rem Change the specified paths to match your installation

```
C:\j2sdk1.4.2_05\bin\javac -classpath "C:\Program Files\Recital\Universal JDBC Driver\RecitalJDBC.jar"  
jdbc_test.java
```

To run:

Rem Change the specified paths to match your installation

```
C:\j2sdk1.4.2_05\bin\java -classpath "C:\Program Files\Recital\Universal JDBC Driver\RecitalJDBC.jar";.  
jdbc_test
```

Linux/UNIX

The Recital Database Server and Recital Client Drivers both include the Recital Universal JDBC Driver. Distributions are available as tar archives, or as rpm files (Linux only). Follow the distribution installation instructions to install the files.

Classpath

The file *RecitalJDBC.jar* needs to be moved or linked to the Java lib directory, which needs to be included in the CLASSPATH.

JDBC_TEST

The jdbc_test class is included to allow the Recital Universal JDBC Driver installation to be tested. The jdbc_test.java source code is also included for information purposes.

To run the Recital JDBC test program:

1. Look for addyourusername and addyourpassword in jdbc_test.java and change them to the correct values for your environment.
2. Compile the test program:
javac jdbc_test.java
3. Run the applet:
java jdbc_test

Alpha/Integrity OpenVMS Installation

The Recital Database Server save set includes the Recital Classes as both individual classes and as a Jar archive, *RecitalJDBC.jar*. These can be installed where required.

Classpath

The appropriate Recital classes or Recital Jar file path should be added to CLASSPATH after installation.

JDBC_TEST

The jdbc_test java file is included to allow the Recital Universal JDBC Driver installation to be tested.

To run the Recital JDBC test program:

1. Look for the username and password details in jdbc_test.java and change them to the correct values for your environment.
2. Compile the test program:
`javac jdbc_test.java`
3. Run the applet:
`java jdbc_test`

Standard Imports required

```
import java.sql.*;
import java.io.*;
import java.net.URL;
import java.math.BigDecimal;
import Recital.sql.*;
```

Connection URL format

Standard format:

```
String url = "jdbc:Recital:" +
    "SERVERNAME=servername;" +
    "DIRECTORY=directory;" +
    "USERNAME=username;" +
    "PASSWORD=password";
```

Parameter	Description
<i>servername</i>	The IP address of the Database Server. '?' connects to the local machine.
<i>directory</i>	The startup directory on the server. NOTE: For Windows paths the '\' should be doubled e.g. C:\Program Files\Recital\UAS\data\southwind
<i>username</i>	Database Server Login username.
<i>password</i>	Database Server Login password.

Alternative Standard format:

```
String url = "jdbc:Recital:" +
    "SERVERNAME=servername;" +
    "DATABASE=database;" +
    "USERNAME=username;" +
    "PASSWORD=password";
```

Parameter	Description
<i>servername</i>	The IP address of the Database Server. '?' connects to the local machine.
<i>database</i>	The database on the server. Databases in Recital are implemented as directories containing files that correspond to the tables in the database. The directory is a sub-directory of the Recital data directory. The environment variable / symbol DB_DATADIR points to the current Recital data directory. The database catalog can contain information about a table's associated index files (single .ndx files and tagged .dbx files). It also contains path information, which allows tables in other directories to be accessed.
<i>username</i>	Database Server Login username.
<i>password</i>	Database Server Login password.

Extended format:

The extended format provides logging and encryption capabilities. If *ENCRYPTION* is set to *true*, the username and password information is transmitted in DES3 encrypted format. The extended format also provides gateway connectivity to the following data sources. Please note this may require the purchase of additional license options. Details of gateway availability for each platform are available from the Recital web site <http://www.recital.com/products.htm>.

ODBC data sources

Oracle

Informix

Ingres

Remote Recital

MySQL

PostgreSQL

String url =

```
"jdbc:Recital:" +  
"SERVERNAME= servername;" +  
"DIRECTORY=directory;" +  
"USERNAME= username;" +  
"PASSWORD= password;" +  
"LOGGING=logging;" +  
"LOGFILE=logfile;" +  
"ENCRYPTION=true/false;" +  
"GATEWAY=type@node:dbms_username/dbms_password-database.protocol";
```

or

String url =

```
"jdbc:Recital:" +  
"SERVERNAME= servername;" +  
"DIRECTORY=directory;" +  
"USERNAME= username;" +  
"PASSWORD= password;" +  
"LOGGING=logging;" +  
"LOGFILE=logfile;" +  
"ENCRYPTION=true/false;" +  
"GATEWAY=odbc:datasource";
```

Parameter	Description
servername	The IP address of the Database Server. “?” connects to the local machine.
directory	The startup directory on the server. To access tables in other directories and associate single index files with their table, a sysodbc.ini file is required. Please see the Database Server Installation and Configuration Guide for information on sysodbc.ini files.
username	Database Server Login username.
password	Database Server Login password.
logging	Set to ‘true’ or ‘false’ to turn logging on or off.
logfile	The name of the file to log to.
gateway	Gateway definition.
datasource	ODBC datasource (DSN).
type	Gateway type: ‘ora’; ‘inf’; ‘ing’; ‘odb’; ‘rec’; ‘mys’; ‘pos’

<i>node</i>	The IP address (or hostname) of the data server.
<i>dbms_username</i>	The username for the data source, e.g. if connecting to Oracle, this must be the name of a valid Oracle user.
<i>dbms_password</i>	The password for the <i>dbms_username</i> above.
<i>database</i>	The database to connect to.
<i>protocol</i>	The network protocol, decnet (DECNET) or tcpip (TCP/IP). If the protocol is not specified, TCP/IP is assumed on Unix/Linux and DECNET on OpenVMS.

NOTE: If *ENCRYPTION* is set to *true*, the corresponding Database Server must be configured to expect encrypted username and password information. Please see the Database Server documentation for more information.

CONNECTING TO C-ISAM FILES

The Recital Universal JDBC Driver can access Informix compliant C-ISAM files.

Data access is achieved through a C-ISAM Bridge. This requires the creation of a Bridge file and an empty Recital table that has the same structure as the C-ISAM file.

NOTE: Users of Recital Terminal Developer on UNIX or Linux can create the Recital table and the Bridge file through the CREATE and CREATE BRIDGE work surfaces. Please see the online documentation (help) in Unix Developer for instructions on using these work surfaces.

Creating the Recital Table

Create a Recital table with the same structure as the C-ISAM file. The fields/columns in the structure file must exactly match the data type and length of those in the C-ISAM file. The Recital table will have one byte more in total record length due to the Recital record deletion marker.

To create the table, use the SQL CREATE TABLE command. The table should be given a '.str' file extension (rather than the default '.dbf') to signify that this is a structure file only.

Please see the end of this section for information on matching Informix and Recital data types. Sample code for table and Bridge creation is given at the end of the Connecting to RMS files section.

Creating the Bridge File

The Bridge File can be created in two ways: by using an 'ini' file, or by the SQL CREATE BRIDGE command.

Maximums Widths

The following maximum widths apply to the bridge elements:

Element	Maximum Width in Characters	Description
Type	10	Bridge type: CISAM
External	80	External file name
Metadata	80	Recital 'structure' table name
Alias	10	Alias name

'ini' file

Firstly, an 'ini' file should be created on the server in the data directory where the C-ISAM file is held. The ini file has the following structure:

```
[bridge]
bridgetype=<bridgetype>
externalname=<name of the C-ISAM file>
databasename=<name of the Recital structure table>
alias=<the name to use to access your file>
```

e.g. cisamdemo.ini

```
[bridge]
bridgetype=CISAM
externalname=cisam.dat
databasename=cisamstru.str
alias=cisamdemo
```

NOTE: There should be no white space either side of the '=' signs.

The Bridge file can now be created from the ini file. This can be given a '.dbf' file extension (rather than the default '.brg') so that it can be accessed like a normal Recital table. The SQL EXECUTE IMMEDIATE command is used to run the Recital/4GL CREATE BRIDGE <.brg filename> | (<expC>) FROM command:

e.g.

execute immediate create bridge cisamdemo.dbf from cisamdemo

CREATE BRIDGE (SQL)

The CREATE BRIDGE SQL command defines and creates the bridge in one step:

e.g.

```
CREATE BRIDGE cisamdemo.dbf
TYPE "CISAM"
EXTERNAL "cisamdemo.dat"
METADATA "cisamdemo.str"
ALIAS "cisamdemo"
```

//or

```
CREATE BRIDGE cisamdemo.dbf
AS "TYPE=CISAM;EXTERNAL=cisamdemo.dat;METADATA=cisamdemo.str;ALIAS=cisamdemo"
```

Using the Bridge

The Bridge can now be used. To access the C-ISAM file, use the 'alias' specified in the Bridge definition.

e.g.

```
select * from cisamdemo
```

Data Types

Informix	Recital
Byte	Numeric
Char	Character
Character	Character
Date	Date
Datetime	Character
Decimal	Numeric
Double Precision	Float
Float	Real
16 Bit Integer	Short
Integer	Numeric
Interval	Character
32 Bit Long	Integer
Money	Numeric
Numeric	Numeric
Real	Numeric
Smallfloat	Numeric
Smallint	Numeric
Text	Unsupported
Varchar	Character

C-ISAM Error Messages

The following errors relate to the use of the Recital CISAM Bridge and can be received as an 'errno <expN>' on Recital error messages:

ERRNO()	Error Description
100	Duplicate record
101	File not open
102	Invalid argument
103	Invalid key description
104	Out of file descriptors
105	Invalid ISAM file format
106	Exclusive lock required
107	Record claimed by another user
108	Key already exists
109	Primary key may not be used
110	Beginning or end of file reached
111	No match was found
112	There is no "current" established
113	Entire file locked by another user
114	File name too long
115	Cannot create lock file
116	Memory allocation request failed
117	Bad custom collating
118	Duplicate primary key allowed
119	Invalid transaction identifier
120	Exclusively locked in a transaction
121	Internal error in journaling
122	Object not locked

CONNECTING TO RMS FILES

The Recital Universal JDBC Driver can access the following fixed length RMS File types:

- RMS Sequential
- RMS Indexed Sequential
- RMS Relative

Data access is achieved through an RMS Bridge. This requires the creation of a Bridge file and an empty Recital table that has the same structure as the RMS file.

NOTE: Users of Recital Terminal Developer for OpenVMS can create the Recital table and the Bridge file through the CREATE and CREATE BRIDGE work surfaces. Please see the online documentation (help) in OpenVMS Developer for instructions on using these work surfaces.

Creating the Recital Table

Create a Recital table with the same structure as the RMS file. The fields/columns in the structure file must exactly match the data type and length of those in the RMS file. The Recital table will have one byte more in total record length due to the Recital record deletion marker. To create the table, use the SQL CREATE TABLE command. The table should be given a '.str' file extension (rather than the default '.dbf') to signify that this is a structure file only.

Please see the end of this section for information on accessing VAX COBOL data types.

Creating the Bridge File

The Bridge File can be created in two ways: by using an 'ini' file, or by the SQL CREATE BRIDGE command.

Maximums Widths

The following maximum widths apply to the bridge elements:

Element	Maximum Width in Characters	Description
Type	10	Bridge type: RMSSEQ, RMSIDX, RMSREL
External	80	External file name
Metadata	80	Recital 'structure' table name
Alias	10	Alias name
Index	50	Index key or filename

CREATE BRIDGE (SQL)

The CREATE BRIDGE SQL command defines and creates the bridge in one step:

e.g.

```
CREATE BRIDGE rmsseqdemo.dbf
TYPE "RMSSEQ"
EXTERNAL "rmsseq.dat"
METADATA "rmsseqdemo.str"
ALIAS "rmsseqdemo"
```

or

```
CREATE BRIDGE rmsseqdemo.dbf
AS "type=RMSSEQ;external=rmsseq.dat;metadata=rmsseqdemo.str;alias=rmsseqdemo"
```

For RMS Indexed Sequential files, the RMS index keys to be used can also be included in the bridge definition. Up to 7 different keys may be specified:

e.g.

```
CREATE BRIDGE rmsidxdemo.dbf
TYPE "RMSIDX"
EXTERNAL "rmsidx.dat"
METADATA "rmsidxdemo.str"
ALIAS "rmsidxdemo"
INDEX "acc_prefix+acc_no,acc_prefix+str(ord_total)"
```

or

```
CREATE BRIDGE rmsidxdemo.dbf
AS "type=RMSIDX;external=rmsidx.dat;metadata=rmsidxdemo.str;alias=rmsidxdemo;;
indexkey1=acc_prefix+acc_no;indexkey2=acc_prefix+str(ord_total)"
```

For RMS Sequential and RMS Relative files, up to 7 Recital single indexes can be built and associated with the bridge.

e.g.

```
CREATE BRIDGE rmsreldemo.dbf
TYPE "RMSREL"
EXTERNAL "rmsrel.dat"
METADATA "rmsreldemo.str"
ALIAS "rmsreldemo"
INDEX "ind1.ndx,ind2.ndx,ind3.ndx"
```

or

```
CREATE BRIDGE rmsreldemo.dbf
AS "type=RMSREL;external=rmsrel.dat;metadata=rmsreldemo.str;alias=rmsreldemo;;
indexkey1=ind1.ndx;indexkey2=ind2.ndx,indexkey3=ind3.ndx"
```

CREATE BRIDGE FROM <ini>

Firstly, an 'ini' file should be created on the server in the data directory where the external data file is held. The ini file has the following contents:

```
[bridge]
bridgetype=<bridgetype>
externalname=<name of the external data file>
databasename=<name of the Recital structure table>
alias=<the name to use to access your file>
indexkey1=<optional RMS index key or Recital index filename>
indexkey2=<optional RMS index key or Recital index filename>
indexkey3=<optional RMS index key or Recital index filename>
indexkey4=<optional RMS index key or Recital index filename>
indexkey5=<optional RMS index key or Recital index filename>
indexkey6=<optional RMS index key or Recital index filename>
indexkey7=<optional RMS index key or Recital index filename>
```

e.g. rmsreldemo.ini

```
[bridge]
bridgetype=RMSREL
externalname=rmsrel.dat
databasename=rmsreldemo.str
alias= rmsreldemo
indexkey1=ind1.ndx
indexkey2=ind2.ndx
```

e.g. rmsidxdemo.ini

```
[bridge]
bridgetype=RMSIDX
externalname=rmsidx.dat
databasename=rmsdemo.str
alias=rmsidxdemo
indexkey1=acc_prefix+acc_no
indexkey2=acc_prefix
```

NOTE: The SQL CREATE BRIDGE or 4GL CREATE BRIDGE FROM <ini> command needs to be reissued to add in details of newly built Recital indexes.

Then the CREATE BRIDGE command should be issued:

execute immediate create bridge rmsidxdemo.dbf from rmsidxdemo

Using the Bridge

The Bridge can now be used. To access the RMS file, use the 'alias' specified in the Bridge definition.

e.g.

Select * from rmsseqdemo

Accessing VAX COBOL Data Types

The following table provides details of the COBOL data types that can be directly accessed by RECITAL using the RECITAL RMS Bridge.

COBOL Picture Clause	COBOL Usage Clause	RECITAL Data type	Storage in bytes
PIC 9(n) [n <=18]	USAGE IS DISPLAY	(N)umeric	n
PIC 9(n) [n <=4]	USAGE IS COMP	(S)hort	2
PIC 9(n) [5 <=n <=9]	USAGE IS COMP	(I)nteger	4
PIC 9(n) [10 <=n <=18]	USAGE IS COMP	(Q)uad	8
PIC S9(n) [n <=4]	USAGE IS COMP	(S)hort	2
PIC S9(m) [5 <=n <=9]	USAGE IS COMP	(I)nteger	4
PIC S9(n) [10 <=n <=18]	USAGE IS COMP	(Q)uad	8
	USAGE IS COMP	(I)nteger	4
	USAGE IS POINTER	(I)nteger	4
	USAGE IS COMP-1	(R)eal	4
	USAGE IS COMP-2	(F)loat	8
PIC S9(n) [n <=18]	USAGE IS COMP-3	(P)acked	*1
PIC 9(n) [n <=18]	USAGE IS COMP-3	(P)acked	*1
PIC X(n) [n <=254]	USAGE IS DISPLAY	(C)haracter	n
PIC A(n) [n <=254]	USAGE IS DISPLAY	(C)haracter	n
PIC 9(n)V9(s)	USAGE IS DISPLAY	(S)hort	2
PIC S9(n)V9(s) [(n+s) <=4]	USAGE IS COMP	(S)hort	2
PIC S9(n)V9(s) [5<=(n+s)<=9]	USAGE IS COMP	(I)nteger	4
PIC S9(n)V9(s) [10<=(n+s)<=18]	USAGE IS COMP	(Q)uad	8
PIC 9(n)V9(s) [n <=18]	USAGE IS COMP-3	(P)acked	*1
PIC S9(n)V9(s) [n <=18]	USAGE IS COMP-3	(P)acked	*1
PIC S9(n) [n <=18]	USAGE IS DISPLAY	--not currently supported--	
PIC S9(n) [n <=18]	USAGE IS DISPLAY SIGN IS TRAILING	--not currently supported--	
PIC S9(n) [n <=18]	USAGE IS DISPLAY SIGN IS LEADING	--not currently supported--	
PIC S9(n) [n <=18]	USAGE IS DISPLAY SIGN IS TRAILING SEPARATE	--not currently supported--	

PIC S9(n) [n <=18]	USAGE IS DISPLAY SIGN IS LEADING SEPARATE	--not currently supported--	
PIC S9(n)V9(s) [(n+s) <=18]	USAGE IS DISPLAY SIGN IS TRAILING	--not currently supported--	
PIC S9(n)V9(s) [(n+s) <=18]	USAGE IS DISPLAY SIGN IS TRAILING	--not currently supported--	
PIC S9(n)V9(s) [(n+s) <=18]	USAGE IS DISPLAY SIGN IS TRAILING SEPARATE	--not currently supported--	
PIC S9(n)V9(s) [(n+s) <=18]	USAGE IS DISPLAY SIGN IS LEADING SEPARATE	--not currently supported--	

NOTE:

The storage occupied by packed decimal data types is calculated as follows:

if (n+s) is odd then storage = ((n+s)+1)/2
else storage = ((n+s)+2)/2

When defining the “width” for binary data types, this value denotes the output display width. The storage occupied by the data type is as specified above.

When defining the number of decimal places for binary data types, this value represents the “scale” of the value. When the field is referenced, RECITAL scales it down by successive divisions of 10, as specified by “scale”, and evaluates all arithmetic in double precision floating point. When fields of this type are updated, then the result to be stored in the field is again re-scaled.

Example Java Program for JDBC RMS Access

```
/*#=====
 *# Copyright (C) 2006 Recital Corporation Inc.
 *# As an unpublished licensed proprietary work.
 *# All rights reserved worldwide.
 *#=====
 * MODULE : rms_test.java
 * PURPOSE : Recital Universal JDBC Driver RMS file access
 * AUTHOR : Recital Corporation
 * DATE : Aug-2006
 *=====*/

//-----
//-- Standard imports required by Recital JDBC Driver
//-----
import java.sql.*;
import java.io.*;
import java.net.URL;
import java.math.BigDecimal;
import Recital.sql.*;

public class rms_test {

    public static void main(String argv[]) {
        int i;
        ResultSet rs;
        ResultSetMetaData rsmd;

        try {
            //-----
            //-- Load the Client Driver for the
            //-- Recital Database Server
            //-----
            new RecitalDriver();
            //-----
            //-- Build the connection URL:
            //-----
            String url = "jdbc:Recital:" +
                "SERVERNAME=servername;" +
                "DATABASE=jdbc_test;" +
                "USERNAME=username;" +
                "PASSWORD=password";
            //-----
            //-- Connect
            //-----
            Connection con = DriverManager.getConnection(url);
            //-----
            //-- Create a statement on the connection
            //-----
            Statement stmt = con.createStatement();
            //-----
            //-- Query info from the driver
```

```

//-----
DatabaseMetaData dbmd = con.getMetaData();
System.out.println(dbmd.getDriverVersion() + "\n" +
                    dbmd.getDriverName());

//-----
//-- Create table to define the metadata of the RMS table on the server
//-----
System.out.println("Create Metadata file rmsdemo.str.");
stmt.execute("create table rmsdemo.str " +
             "(ord_no char(8), " +
             "ord_date date, " +
             "name char(20), " +
             "street char(25), " +
             "city char(25), " +
             "state char(13), " +
             "goods char(40), " +
             "order_value num(11,2), " +
             "rec_date date, " +
             "paid_date date, " +
             "paid_value num(11,2), " +
             "acc_start date, " +
             "acc_end date, " +
             "acc_prefix char(3), " +
             "acc_no char(5), " +
             "batch_no char(4)) ");

//-----
//-- Create Bridge file
//-----
System.out.println("Create Bridge File rmsdemo.dbf.");
stmt.execute("create bridge rmsdemo.dbf " +
             "type 'RMSIDX'+
             "external 'rmsidx.dat'" + // Name of the RMS data file.
             "metadata 'rmsdemo.str'" + // Name of the Metadata file.
             "alias 'rmsdemo'" +
             "index 'acc_prefix+acc_no'");

//-----
//-- Query for data
//-----
System.out.println("Select rows from rmsdemo.");
rs = stmt.executeQuery("SELECT * from rmsdemo");
rsmd = rs.getMetaData();
int nr_cols = rsmd.getColumnCount();
while (rs.next()) {
    for (i = 1; i <= nr_cols; i++) {
        System.out.println("rs Column[" + i + "] : " +
                           rsmd.getColumnName(i) + "<" +
                           rsmd.getColumnTypeName(i) + "> = " +
                           rs.getString(i));
    }
    System.out.println("Got results:");
}

//-----
//-- Release the statement
//-----

```

```

        stmt.close();
        //-----
        //-- Disconnect from the server
        //-----
        con.close();
    } catch (Exception e) {
        System.out.flush();
        System.err.flush();
        DriverManager.println("Recital JDBC driver exception: " + e.getMessage());
        e.printStackTrace();
    }
    try {
        System.out.println("Press any key to continue...");
        System.in.read();
    } catch (IOException ie) {
        ;
    }
}
}
}

```

Accessing a Resultset from a Stored Procedure

Stored procedures and user-defined functions are collections of SQL statements and optional control-of-flow statements written in the Recital 4GL (compatible with VFP) stored under a name and saved in a **Database**. Both stored procedures and user-defined functions are just-in-time compiled by the Recital database engine.

Stored Procedures can return a Resultset using the 4GL SETRESULTSET() function. The 'recital' command is used to execute the 4GL Stored Procedure and return the Resultset to the client application for processing. Resultset that are returned from Stored Procedures are read-only.

In this example, the getexamplecursor.prg stored procedure is in the 'southwind' database on the server. It accepts a parameter and runs a query, saving the results into a cursor. This cursor is then returned as a Resultset using the SETRESULTSET() 4GL function.

```
// Stored Procedure getexamplecursor.prg
lparameters lcAccountNo
exec sql
select * from example
where account_no = lcAccountNo
into cursor curExample;
return setresultset("curExample")
// end of getexamplecursor.prg
```

The Java program establishes the JDBC connection, calls the Stored Procedure and displays the data.

```
import java.sql.*;
import java.io.*;
import java.net.URL;
import Recital.sql.*;

public class rs_example {

    public static void main(String[] args) {

        int i;
        ResultSet rs;
        ResultSetMetaData rsmd;
        String s;

        System.out.println("Recital resultset example program started.");
        try {
            new RecitalDriver();
            String url = "jdbc:Recital:" +
                "SERVERNAME=?" +
                "DATABASE=southwind;" +
                "USERNAME=?" +
                "PASSWORD=?";
            Connection con = DriverManager.getConnection(url);
            Statement stmt = con.createStatement();

            rs = stmt.executeQuery("recital getexamplecursor('0001')");
            rsmd = rs.getMetaData();
```

```

int nr_cols = rsmd.getColumnCount();
while (rs.next()) {
    for (i = 1; i <= nr_cols; i++) {
        s = rs.getString(i);
        System.out.println(rsmd.getColumnName(i) + " (" + rsmd.getColumnTypeName(i) + ") = " + s);
    }
    System.out.println("***** Next Record *****");
}
System.out.println("End of results:");
stmt.close();
con.close();
} catch (Exception e) {
    System.out.flush();
    System.err.flush();
    DriverManager.println("Driver exception: " + e.getMessage());
    e.printStackTrace();
}
try {
    System.out.println("Press any key to continue...");
    System.in.read();
} catch (IOException ie) {
    ;
}
}
}

```