

RETURN TO MAIN MENU

Recital: Using RMS Files

Using RMS Files

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

INTRODUCTION	1
CREATING THE RECITAL TABLE	2
CREATING THE BRIDGE	3
USING THE BRIDGE	6
COBOL DATA TYPES	7
JAVA/JDBC Example	9

INTRODUCTION

Recital Terminal Developer and the Recital Database and Mirage Application Servers for OpenVMS support access to 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.

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 or the Recital Terminal Developer CREATE workspace. 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 document for information on accessing COBOL data types.

Creating the Bridge File

In Recital Terminal Developer, the Bridge File can be created using the CREATE BRIDGE worksurface.

For Server clients, the Bridge File can be created with the SQL CREATE BRIDGE or 4GL CREATE BRIDGE FROM <ini> commands.

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.

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

or

```
exec sql
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.

```
exec sql
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

```
exec sql
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.

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

or

```
exec sql
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: Recital Terminal Developer users can use the MODIFY BRIDGE to add in details of newly built Recital indexes. In client/server environments the SQL CREATE BRIDGE or 4GL CREATE BRIDGE FROM <ini> command needs to be reissued.

Then the CREATE BRIDGE command should be issued:

```
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

e.g.

- use rmsseqdemo
- edit

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 <=18]	USAGE IS COMP-3	(P)acked	Variable
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(n)[5 <=n <=9]	USAGE IS COMP	(I)nteger	4
PIC S9(n)[10 <=n <=18]	USAGE IS COMP	(Q)uad	8
PIC S9(n)[10 <=n <=18]	USAGE IS INDEX	(I)nteger	4
PIC S9(n)[10 <=n <=18]	USAGE IS POINTER	(I)nteger	4
PIC S9(n)[10 <=n <=18]	USAGE IS COMP-1	(R)eal	4
PIC S9(n)[10 <=n <=18]	USAGE IS COMP-2	(F)loat	8
PIC S9(n)[n <=18]	USAGE IS COMP-3	(P)acked	Variable
PIC 9(n)[n <=18]	USAGE IS COMP-3	(P)acked	Variable
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	Variable
PIC S9(n)V9(s)[n <=18]	USAGE IS COMP-3	(P)acked	Variable
PIC S9(n)[n <=18]	USAGE IS DISPLAY	not supported	
PIC S9(n)[n <=18]	USAGE IS DISPLAY SIGN IS TRAILING	not supported	
PIC S9(n)[n <=18]	USAGE IS DISPLAY SIGN IS LEADING	not supported	
PIC S9(n)[n <=18]	USAGE IS DISPLAY SIGN IS TRAILING SEPARATE	not supported	
PIC S9(n)[n <=18]	USAGE IS DISPLAY SIGN IS LEADING SEPARATE	not supported	
PIC S9(n)V9(s)[(n+s) <=18]	USAGE IS DISPLAY SIGN IS TRAILING	not supported	
PIC S9(n)V9(s)[(n+s) <=18]	USAGE IS DISPLAY SIGN IS TRAILING	not supported	
PIC S9(n)V9(s)[(n+s) <=18]	USAGE IS DISPLAY SIGN IS TRAILING SEPARATE	not supported	
PIC S9(n)V9(s)[(n+s) <=18]	USAGE IS DISPLAY SIGN IS LEADING SEPARATE	not supported	

NOTE:

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

if $(n+s)$ is odd then $\text{storage} = ((n+s)+1)/2$
else $\text{storage} = ((n+s)+2)/2$

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) {
        ;
    }
}
}

```