April 2004

In this issue

3 SQL execution
12 Implementing a DB2 federated database system
19 Invoking DSNUTILS DB2 stored procedure via REXX
29 Spiffing up SPUFI (Version 2)
48 DB2 news

© Xehon Inc 2004
Subscriptions and back-issues
A year's subscription to DB2 Update, comprising twelve monthly issues, costs $380.00 in the USA and Canada; £255.00 in the UK; £261.00 in Europe; £267.00 in Australasia and Japan; and £265.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 2000 issue, are available separately to subscribers for $33.75 (£22.50) each including postage.

Disclaimer
Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, and other contents of this journal before making any use of it.

Contributions
When Xephon is given copyright, articles published in DB2 Update are paid for at the rate of $160 (£100 outside North America) per 1000 words and $80 (£50) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of $32 (£20) per 100 lines. To find out more about contributing an article, without any obligation, please download a copy of our Notes for Contributors from www.xephon.com/nfc.

© Xephon Inc 2004. All rights reserved. None of the text in this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior permission of the copyright owner. Subscribers are free to copy any code reproduced in this publication for use in their own installations, but may not sell such code or incorporate it in any commercial product. No part of this publication may be used for any form of advertising, sales promotion, or publicity without the written permission of the publisher. Copying permits are available from Xephon in the form of pressure-sensitive labels, for application to individual copies. A pack of 240 labels costs $36 (£24), giving a cost per copy of 15 cents (10 pence). To order, contact Xephon at any of the addresses above.

Printed in England.
SQL execution

The REXXX macro `@SQL` enables you to execute SQL statements directly from your TSO edit session. For SQL execution, program DSNTEP will be used.

Place a PDS member with the SQL statement(s) in TSO edit mode and run `@SQL`.

SQL statements end with a `;`.

On the panel that you see, you can choose from the following three selections:

1. SQL will be executed on your screen.
2. SQL will be executed in the background (class=A).
3. SQL-execution in the evening (class=M).

Place an ‘S’ in front of the selection of your choice and press Enter.

Option 3 enables you to start SQL execution in the background at a given time (for instance to execute heavy SQL statements after the on-line window). When you want to use this option you’ll have to automatically start the batch class at the desired time. We start jobclass M at 19:00 hours and stop it at 06:00 hours.

When you want to use another jobclass at your site, you can change this in REXXX program `@SQL`.

The following programs/panels are used:

- `@SQL` – the macro that has to be started.
- `SQLR01` – subprogram for on-line execution.
- `SQLP01` – selection panel.
- `SQLP02` – result panel.
- `SQLH01` – help panel for `SQLP01`.
- `SQLPOPUP` – message panel.
• SQLPARM – parameter member.

The following variables must be changed in the INIT section of @SQL:
• SQLVDB2L='loadlib DSNTEP program'.
• SQLVDB2S='DB2 subsystem id'.
• SQLVPARM='name of PDS lib where member SQLPARM can be found'.

@SQL REXX
/* REXX */
"Isredit macro"
"Isredit (cursor) = user_state"
/* * * MAIN line * * */
Main:
If Init() Then Do
  Call Check
  Call Popup
  Call Selection
End
Call EndProg
Exit
/* * * Initialization * * */
/* */
Init:
/* THE VALUE FOR THE FOLLOWING 3 VARIABLES MUST BE CHANGED */
SQLVDB2L='loadlib DSNTEP program'
SQLVDB2S='DB2 subsystem id'
SQLVPARM='name of PDS·lib where member SQLPARM can be found'
"Isexec vput (SQLVDB2S SQLVDB2L) profile"
SQLSEL=""
YES=(1=1); NO=(1=0); IN_rval=YES
X = outtrap(O newline.)
"ALLOCATE FI(PARMFIL) DA(’”SQLParm”(SQLPARM)’) SHR"
lrc=rc
X = outtrap("OFF")
If lrc -- 0 Then Do
  "EXECIO * DISKR PARMFIL (FINIS STEM LINES."
  ecc = lrc
  "FREE FI(PARMFIL)"
If ecc -- 0 Then Do
  "ISPEXEC TBCREATE SQLTAB NAMES(DESCR) KEYS(KD) NOWRITE"
If lrc -- 8 Then Do
  "ISPEXEC TBCLOSE SQLTAB"
"ISPEXEC TBCREATE SQLTAB NAMES(DESCR) KEYS(KD) NOWRITE"
End
If lrc = Ø Then Do
Do IN_i = 1 To lines.Ø
   record = lines.IN_i
   Parse Var record KD DESCR
   "ISPEXEC TBADD SQLTAB"
End
"ISPEXEC TBTOP SQLTAB"
End; Else Do
   Call Message "Error in table definition.", "WAIT"
   IN_rval = NO
End
End; Else Do
   Call Message "Error while reading parameter file.", "WAIT"
   IN_rval = NO
End
End; Else Do
   Call Message "Error during allocation parameter file.", "WAIT"
   IN_rval = NO
End
Return IN_rval

/* Check SQL-member */
Check:
   "Isredit seek // 1":
   If rc = Ø then;
      Do;
         ZMSG000S = 'MEMBER CONTAINS JCL'
         ZMSG000L = 'Only SQL-statements can be executed with @SQL'
         "ISPEXEC SETMSG MSG(ISPZ000)"
         Exit;
      End;
   Return
   /* Show Pop-up panel */
Popup:
   "ISPEXEC ADDPOP ROW(5) COLUMN(40)"
   READY? = NO
   Do While READY? = NO
      "ISPEXEC TBDISPL SQLTAB PANEL(SQLP01)"
      If rc > Ø Then Do
         Exit
      End; Else Do
      If sqlsel<>""
         Then Do
            READY? = YES
         End
      End
   End
End
"ISPEXEC REMPOP"
"ISPEXEC TBCLOSE SQLTAB"

Return YES

/* ----------------------------------------------- *
*                                             *
* Read user input                             *
* ----------------------------------------------- */

Selection:
Select
When KD='1' then call online
When KD='2' then do
  JCLASS='A'
  ANSWER='now'
  Call Add_jcl
  Call Submit
End;
When KD='3' then do
  JCLASS='M'
  ANSWER='tonight'
  Call Add_jcl
  Call Submit
End;
Otherwise Call Stop_REXX
End;

Return

/* ----------------------------------------------- *
* Add JCL for batch execution                    *
* ----------------------------------------------- */

Add_jcl:
ACCOUNT = ''
Select
When length(USERID()) = 6 then JOBNAME= USERID()'QQ'
When length(USERID()) = 7 then JOBNAME= USERID()'Q'
Otherwise exit(9)
End
"Isredit (dsn)=dataset";
"Isredit (mem)=member ";
"Isredit line_before .zfirst = ' '"
"Isredit label .zfirst=.a";
mbrline: /*'JOBNAME' JOB ('ACCOUNT'),'
"Isredit line_before .a = (mbrline)"
mbrline: /*'userid()''
"Isredit line_before .a = (mbrline)"
mbrline: /*CLASS='JCLASS',
"Isredit line_before .a = (mbrline)"
mbrline: /* NOTIFY='userid()',
"Isredit line_before .a = (mbrline)"
mbrline: /* MSGLEVEL=(1,1),
"Isredit line_before .a = (mbrline)"
mbrline: /* MSGCLASS=X
"Isredit line_before .a = (mbrline)"
mbrline: //** *****************************************|
"Isredit line before .a = (mbrline)"
  mbrline= '// Execute SQL in batch 'SQLVDB2S 'Jobclass='JCLASS
"Isredit line before .a = (mbrline)"
  mbrline= ' ******************************************************************************
"Isredit line before .a = (mbrline)"
  mbrline= '//SQL EXEC PGM=IKJEFT01.DYNAMNBR=20'
"Isredit line before .a = (mbrline)"
  mbrline= '//SYSTSPRT DD SYSOUT=*'
"Isredit line before .a = (mbrline)"
  mbrline= '//SYSPRINT DD SYSOUT=*'
"Isredit line before .a = (mbrline)"
  mbrline= '//SYSDBOUT DD SYSOUT=*'
"Isredit line before .a = (mbrline)"
  mbrline= '//SYSSIN DD *'
"Isredit line before .a = (mbrline)"
  mbrline= 'DSN SYSTEM('SQLVDB2S')'
"Isredit line before .a = (mbrline)"
  mbrline= 'RUN PROGRAM(DSNTEP2) PLAN(DSNTEP2) -'
"Isredit line before .a = (mbrline)"
  mbrline= 'SUBMIT,:'
"Isredit line before .a = (mbrline)"
  mbrline= 'LIB('''SQLVDB2L''')'
"Isredit line before .a = (mbrline)"
  mbrline= '/*'
"Isredit line before .a = (mbrline)"
  mbrline= 'SYSSIN DD *'
"Isredit line before .a = (mbrline)"
Return:
*---------------------------*
* Submit batchjob*
*---------------------------*/

Submit:
  "Isredit change all p'< p'>" 
  "Isredit reset"
  "Isredit submit"
  "Isredit delete .zfirst .a"
  "Isredit cursor = .zfirst 1"
  ZMSG000S = 'SQL executed 'ANSWER
  ZMSG000L = 'Job 'JOBNAME' will be executed 'ANSWER' jobclass='JCLASS
  "ISPEXEC SITMSG MSG(ISPZ000)"
Return:
*---------------------------*
* Execute online*
*---------------------------*/

Online:
  text = 'SQL will be executed online'
  text = Center(Strip(text),50)
  "ISPEXEC CONTROL DISPLAY LOCK"
  "ISPEXEC ADDPPOP ROW(9) COLUMN(14)"
  "ISPEXEC DISPLAY PANEL(SQLPOPUP)"
  "ISPEXEC REMPOP"

Call SQLR01
Return;
/* ............................................................ */
/* Stop Procedure ............................................ */
/* ............................................................ */

stop_REXX:
ZMSG000S = 'Procedure stopped'
ZMSG000L = 'You did not make any choice; nothing happened'
"ISPEXEC SETMSG MSG(ISPZ000)"

Exit;
Message:
msg = Arg(1); parms = Arg(2)
ZMSG000S = 'Error'
ZMSG000L = MSG
"ISPEXEC SETMSG MSG(ISPZ000)"
If Find(parms,"FATAL") > Ø Then Exit

Return
EndProg:
"Isredit user_state = (cursor)"
Return Ø

SQLR01

/* rexx */
trace o
dropbuf
"Isredit (maxline) = linenum .zlast"
"Isredit (cursor) = user state"
"Ispexec vget (SQLVDB2S SQLVDB2L) profile"
erc=rc
If erc>4 then do
  zmsg0001=
zmsg000s='Error 'erc' reading DB2 parameter'
  "ISPEXEC SETMSG MSG(ISPZ000)"
  Return erc
End

Do i=1 to maxline
  "Isredit (mbrline) = line "i
  parse var mbrline
  queue mbrline
End

Call alloctmp
"Execio" Queued() "diskw sysin (finis)"
erc=rc
If erc>4 then do
  zmsg0001 = 'RC: 'erc' while copying SQL-statement'
  zmsg000s = 'ERROR SQL EXECUTION'
  "ISPEXEC SETMSG MSG(ISPZ000)"
Exit erc

© 2004. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.
End
X-outtrap(OUTLINE.)
push 'END'
push "RUN PROGRAM(DSNTEP2) LIB('SQLVDB2L')"
"DSN SYSTEM('SQLVDB2S')"
erc=rc
X-outtrap("OFF")
Select
When erc>4 then do
  zmsg0001='Error in SQL-statement'
  zmsg000s='ERROR IN SQL'
  "ISPEXEC SETMSG MSG(ISPZ000)"
End
When erc=-2871 then do
  zmsg0001='WORKFILE too small to contain SQL-result'
  zmsg000s='OUTPUT INCOMPLETE'
  "ISPEXEC SETMSG MSG(ISPZ000)"
End
Otherwise nop
End
Call edittmp
Exit erc
/* ************************************************** */
alloctmp:
  X-outtrap(OUTLINE.)
  "ALLOC FI(SYSIN) TRACKS SPACE(5) LRECL(80) REUSE"
  "ALLOC FI(SYSPPUNCH) TRACKS SPACE(1) LRECL(80) REUSE"
  If rc<>0 then call Error_alloc
  "ALLOC FI(SYSPPRINT) SPACE(2,2) TRACKS BLKSIZE(0) LRECL(255) REUSE",
    "RECFM(F B) DSORG(PS)"
  If rc<>0 then call Error_alloc
  X-outtrap("OFF")
  "ISPEXEC LMINIT DATAID(DID1) DDNAME(SYSPPRINT)"
Return
edittmp:
  "ISPEXEC EDIT DATAID("DID1") panel(SQLPAR)"
  "ISPEXEC LMFREE DDNAME(SYSPPRINT)"
return
Error_alloc:
  ZMSG000S='ALLOCATION ERROR'
  ZMSG000L='Error while allocating workfile.'
  "ISPEXEC SETMSG MSG(ISPZ000)"
Exit(B)

SQLP01

)ATTR
  _ TYPE(INPUT) INTENS(HIGH)
@ TYPE(OUTPUT) INTENS(HIGH) SKIP(ON) CAPS(OFF) COLOR(WHITE)
% TYPE(TEXT) INTENS(LOW) SKIP(ON) COLOR(RED) HILITE(REVERSE)
```
~ TYPE(TEXT) INTENS(LOW) SKIP(ON) COLOR(TURQ)
# TYPE(TEXT) INTENS(LOW) SKIP(ON) COLOR(BLUE)
)BODY SMSG(MSG) LMSG(MSG) WINDOW(40,8)
~
~CMD:_ZCMD
~
~<ENTER> = Execute
~
~<PF1> = HELP-information.
~
~
)MODEL
~
Z~ @KD@DESCR
~
)
INIT
~.ZVARS = 'SQLSE)
~.CURSOR = SQLSE
~.HELP = SQLH01
~)

PROC
~&PF = .PFKEY
~IF (&PF = ' ')
~&PF = ENTER
~
~)

)END

SQLP02
~
~)
ATTR
~.TYPE(INPUT) CAPS(OFF) INTENS(HIGH) FORMAT(&MIXED)
~.TYPE(TEXT) INTENS(LOW)
~)
BODY WIDTH(&ZWIDTH) EXPAND(/)
~)

%SQL OUTPUT ------------------------------------------/~/----------------------------------------
~
~%COMMAND ----->_ZCMD
~
~----------------------> Z
~
~%SCROLL
~
~)
INIT
~&P = N
~.ZVARS = 'ZSCBR'
~)
PROC
~)

)END

SQLH01
~
~)
ATTR
~.TYPE(INPUT) INTENS(HIGH)
~.TYPE(OUTPUT) INTENS(HIGH) SKIP(ON) CAPS(OFF) COLOR(WHITE)
~.TYPE(TEXT) INTENS(LOW) SKIP(ON) COLOR(White)
~.TYPE(TEXT) INTENS(LOW) SKIP(ON) COLOR(TURQ)
~.TYPE(TEXT) INTENS(LOW) SKIP(ON) COLOR(BLUE)
~)

)BODY SMSG(MSG) LMSG(MSG) WINDOW(50,9)
~
~Put "S" before your choice and press <ENTER>~
~
~
~# 1: SQL will be executed in foreground
~
~# 2: SQL will be executed in background
~
~
```

© 2004. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.
Implementing a DB2 federated database system

A DB2 federated system is a distributed computing system that consists of:

- A DB2 server, called a federated server.
- A local database, called a federated database, which is logically within a federated server, plus multiple data sources to which the federated server sends queries and from which it retrieves data. These different types of data source are located on multiple platforms at different locations.

Each data source consists of an instance of a relational database management system plus the database or databases that the instance supports. The data sources in a DB2 federated system can include...
Oracle instances, Sybase, MS SQL Server, Informix, or instances of any members of the DB2 family.

The data sources are semi-autonomous. For example, the federated server can send queries to Oracle data sources at the same time that Oracle applications can access these data sources. A DB2 federated system does not monopolize or restrict access to Oracle or other data sources (beyond integrity and locking constraints).

To end users and client applications in the federated server, the data sources appear as a single collective database. To obtain data from data sources, they submit queries in DB2 SQL to the federated server. The federated server then distributes the queries to the appropriate data sources. The federated server also provides access plans for optimizing the queries (in some cases, these plans call for processing the queries at the federated server rather than at the data source). Finally, the federated server collects the requested data, performs and finalizes processing (joining, sorting, aggregating), and the result set is passed to the users and applications.

WHY DO WE NEED THE SYSTEM?

Figure 1 shows a comparative analysis of a federated system with a conventional system.

WRAPPERS AND WRAPPER MODULES

A wrapper is the mechanism by which the federated server communicates with, and retrieves data from, a data source. To implement a wrapper, the server uses routines stored in a library called a wrapper module. These routines allow the server to perform operations such as connecting to a data source and retrieving data from it iteratively.

There are default wrappers as follows:

- A wrapper with the default name of DRDA is used for all DB2 family data sources.
- A wrapper with the default name of SQLNET is used for all Oracle data sources supported by Oracle's SQL*Net client software (for Oracle's Version 7).
<table>
<thead>
<tr>
<th>DB2 federated system</th>
<th>Conventional system</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>For the user it requires single connection to query multiple databases in different location.</td>
<td>It requires multiple connections to each database.</td>
<td>It requires direct access for the conventional system.</td>
</tr>
<tr>
<td>User can join tables from two or more databases of same type or different type.</td>
<td>User cannot join tables from multiple databases.</td>
<td>Federated system can join tables from Oracle and DB2 or any other RDBMS databases.</td>
</tr>
<tr>
<td>For the user it appears as a single collective database.</td>
<td>It appears as multiple databases.</td>
<td></td>
</tr>
<tr>
<td>From a DB2 client a user can not only query DB2 family of products but also other databases like Oracle, Sybase, or Informix in DB2’s own SQL.</td>
<td>From a DB2 client a user can access only the DB2 family of products.</td>
<td></td>
</tr>
<tr>
<td>In a special mode called pass-through mode, the user can use the Data source's own SQL dialect to retrieve data from the data sources.</td>
<td>User needs to use only DB2’s own SQL dialect.</td>
<td></td>
</tr>
<tr>
<td>Connection overhead is less.</td>
<td>Connection overhead is greater.</td>
<td></td>
</tr>
<tr>
<td>Better performance and availability to query multiple databases.</td>
<td>Performance is an issue to access multiple databases.</td>
<td></td>
</tr>
<tr>
<td>Depending upon the query and the optimizer’s cost model the query can be evaluated partly at the data sources and partly at the federated system.</td>
<td>Full query will be evaluated at the data sources so fully dependent on data sources CPU and I/O capability and other considerations.</td>
<td></td>
</tr>
<tr>
<td>The cost of transmitting data or messages between a federated server and data sources is taken into consideration while calculating the total cost to arrive at an optimal access path.</td>
<td>The cost of transmitting data is not taken into account so there could be a network traffic problem while transferring data between data sources and the DB2 client machine.</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 1: Comparison*
• A wrapper with the default name of NET8 is used for all Oracle data sources supported by Oracle’s Net8 client software (for Oracle’s Version 8 and 9).
• A wrapper with the default name INFORMIX is used for Informix data sources.
• A wrapper with the default name DJXMSSQL3 in Windows and MSSQLLODBC3 for Unix environment for MS SQL Server.
• A wrapper with the default name CTLIB or DBLIB can be used for Sybase data sources.

INTRODUCTION TO SERVER DEFINITIONS

In defining a data source to the federated database, the DBA supplies a name for the data source as well as information that pertains to the data source. This information includes the type and version of the RDBMS of which the data source is an instance, and the RDBMS’s name for the data source. It also includes metadata that is specific to the RDBMS. For example, a DB2 family data source can have multiple databases, and the definition of such a data source must specify which database the federated server can connect to. In contrast, an Oracle data source has one database, and the federated server can connect to the database without needing to know its name. Thus, the name is not included in the federated server’s definition of the data source.

The name and information that the DBA supplies are collectively called a server definition. This term reflects the fact that data sources answer requests for data and are therefore servers in their own right. Other terms reflect this fact also. For example, some of the information within a server definition is stored as values of parameters called server options. Thus, the name for a data source instance is stored as a value of a server option called NODE. For a DB2 family data source, the name of the database to which the federated server connects is stored as a value of a server option called DBNAME.

Factors affecting the performance of the system and values supplied in server definitions are:
• I/O ratio
• CPU ratio
• Collating sequence
• Plan hints.

I/O ratio is the relative speed of the I/O system at the data source and the federated server. CPU ratio is the relative speed of the CPU at the data source and the federated server. Collating sequence, if it's the same both at the federated server and data source, will improve the performance. Plan hints are statement fragments, which give valuable information to the data source’s optimizer in choosing the access path.

USER MAPPINGS AND USER OPTIONS
The federated server can send the distributed request of an authorized user or application to a data source under either of these conditions:

• The user or application uses the same user ID for both the federated database and the data source. In addition, if the data source requires a password, the user or application uses the same password for the federated database and the data source.

• The user's or application's authorization to access the federated database differs in some way from the user's or application's authorization to access the data source. In addition, when the user or application requests access to the data source, the federated database authorization is changed to the data source authorization, so that the access can be granted. This change can occur only if a defined association, called a user mapping, exists between the two authorizations.

DATA TYPE MAPPINGS
For the federated server to retrieve data from columns of data source tables and views, the columns’ data types at the data source must map to the corresponding data types that are already defined to the federated database. DB2 supplies default mappings for most kinds of data types. For example, the Oracle type FLOAT maps by default to the DB2 type DOUBLE, and the DB2 Universal Database for OS/390 type DATE maps by default to the DB2 type DATE. There are no mappings for the
data types that DB2 federated servers do not support, such as LONG VARCHAR, LONG VARGRAPHIC, DATALINK, large object (LOB) types, and user-defined types.

When values from a data source column are returned, they conform fully to the DB2 type in the type mapping that applies to the column. If this mapping is a default, the values also conform fully to the data source type in the mapping. For example, when an Oracle table with a FLOAT column is defined to the federated database, the default mapping of Oracle FLOAT to DB2 DOUBLE will, unless it has been overridden, automatically apply to that column. Consequently, the values returned from the column will conform fully to both FLOAT and DOUBLE.

FUNCTION MAPPINGS

For the federated server to recognize a data source function, there needs to be a mapping between this function and a corresponding DB2 function that already exists at the server. DB2 supplies default mappings between existing built-in data source functions and built-in DB2 functions.

NICKNAMES

When a client application submits a distributed request to the federated server, the server parcels out the request to the appropriate data sources. The request does not need to specify these data sources. Instead, it references data source tables and views by identifiers, called nicknames, which map to the tables’ and views’ names at the data source. The mappings obviate the need to qualify the nicknames by data source names. The locations of the tables and views are transparent to the client application.

Nicknames are not alternative names for tables and views in the same way that aliases are; they are pointers by which the federated server references these objects.

When a nickname is created for a table or view, the catalog is populated with metadata that the optimizer can use to facilitate access to the table or view. For example, the catalog is supplied with the names of the DB2
data types to which the data types of the table’s or view’s columns map. If the nickname is for a table with an index, the catalog is also supplied with information related to the index; for example, the name of each column in the index key.

INDEX SPECIFICATIONS
When a nickname is created for a data source table, the federated server supplies the catalog with information about any indexes that a data source table has. The optimizer uses this information to facilitate retrieval of the table’s data. If the table has no indexes, the user can nevertheless supply information that an index definition typically contains; for example, which column or columns in the table to search in order to find information quickly. The user would do this by running a CREATE INDEX statement that contains the information and references the table’s nickname.

If there is a change in the data sources table statistics, it needs to be reflected in the federated system. So there should be synchronization between the table data at the source and at the federated system. Dropping the nicknames and recreating it can achieve this. Alternatively, the statistics can be changed manually.

COMPENSATION
Compensation is the processing of SQL statements for RDBMSs that do not support those statements. Each type of RDBMS supports a subset of the international standard of SQL. In addition, some types support SQL constructs that are outside any standard. The totality of SQL that a type of RDBMS supports is called an SQL dialect. If an SQL construct is found in DB2’s SQL dialect, but not in a data source’s dialect, the federated server can implement this construct on behalf of the data source.

PASS-THROUGH
Users can use the pass-through function to communicate with data sources in the data sources’ own SQL dialect. In pass-through, users can submit not only queries, but also DML and DDL statements.
There are certain restrictions on using pass-through. For example, in a pass-through session, a cursor cannot be opened directly against a data source object.

When using pass-through the following information applies to all data sources:

- If a PREPARE statement is used to dynamically prepare a statement in a pass-through session, the latter statement cannot be processed unless it is referenced in an EXECUTE statement in the same session.
- If a COMMIT or ROLLBACK command is issued during a pass-through session, this command will complete the current Unit Of Work (UOW).
- Users and applications can use pass-through to write to data sources – for example, to insert, update, and delete table rows. Note that a cursor cannot be opened directly against a data source object in a pass-through session (SQLSTATE 25000).
- An application can have several SET PASSTHRU statements in effect at the same time to different data sources. Although the

<table>
<thead>
<tr>
<th>Pass-through session</th>
<th>Going direct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cursor cannot be opened directly against a data sources.</td>
<td>Cursor can be opened.</td>
</tr>
<tr>
<td>It does not support stored procedure call.</td>
<td>It does support stored procedure calls.</td>
</tr>
<tr>
<td>There is no need for conversion of SQL dialect because it uses the data source's own SQL dialect.</td>
<td>It uses DB2's SQL to query data sources and there is a performance issue because the federated system has to compensate the SQL dialect for the data source's own dialect.</td>
</tr>
<tr>
<td>User can use DML statements like insert, update, and delete.</td>
<td>User can use only select statement.</td>
</tr>
<tr>
<td>LOBs are not supported.</td>
<td>LOBs are supported</td>
</tr>
<tr>
<td>User can create tables/views in remote databases.</td>
<td>User cannot create tables/views in remote databases.</td>
</tr>
</tbody>
</table>

*Figure 2: Statistics comparison*
application might have issued multiple SET PASSTHRU statements, the pass-through sessions are not truly nested. The federated server will not pass through one data source to access another. Rather, the server accesses each data source directly.

- If multiple pass-through sessions are open at the same time, each unit of work within each session must be concluded with a COMMIT or ROLLBACK statement. The sessions can then be ended in one operation with the SET PASSTHRU statement and its RESET option.
- It is not possible to pass through to more than one data source at a time.
- Pass-through does not support stored procedure calls.

Figure 2 compares statistics between pass-through and going direct.

Sujit K Mishra  
Senior Software Engineer  
IBM Global Services (India)  
© Xephon 2004

Invoking DSNUTILS DB2 stored procedure via REXX

This article will discuss and explain what stored procedures are and their general purpose.

The focus of the discussion will be on DB2 DSNUTILS stored procedure and how it is invoked from a REXX environment accessing a DB2 OS/390 environment. The JCL driver and the actual REXX code will be given.

All the necessary and essential components for the invocation and execution of DSNUTILS stored procedure will be discussed, including the Work Load Manager (WLM) in an OS/390 environment, the Recoverable Resource Services (RRS), Recoverable Resource Manager Services Attachment Facility (RRSAF), and the necessary external security interfaces such as RACF or TSS.
This article will not discuss DB2-managed Stored Procedure Address Spaces (SPAS) as an environment to run stored procedures because IBM has stated that SPAS will not support newly-created stored procedures in DB2 OS/390 V8.

At the end of this article we will discuss in a high-level view how the DSNUTILS stored procedure is used by ERP applications such as PeopleSoft HR or Financials. Both PeopleSoft applications, HR and Financials, use DSNUTILS stored procedure in the same way if they are implemented on DB2 OS/390.

STORED PROCEDURES GENERAL PURPOSE

Stored procedures are application programs that can be written in high-level languages such as COBOL, C, PL/I, Assembler, REXX, and Java.

Stored procedures can be written as subroutines or as main programs. They can contain static or dynamic SQL.

Java stored procedures can use static SQL for Java (SQLJ) or dynamic SQL for Java (JDBC).

These stored procedures subroutines can be written in different languages as mentioned above and invoked from a driver program written in yet another language. If program development occurs in the OS/390 environment, it requires Language Environment libraries (LE/370) to establish a common run-time environment for all these languages.

Stored procedures can be invoked and executed locally like any other subroutine or can be invoked locally from a client workstation and executed remotely on a server.

The invoking program can invoke several stored procedures in one Unit Of Recovery (UOR) with two-phase commit processing.

Stored procedures achieve their best objectives and are most suited for implementation in a client/server application, where a single invocation of EXEC SQL CALL can save several individual remote SQL calls and its associated formatting messages. This will significantly reduce network traffic overhead from the client to the server and thus improve performance.

If you find difficulty in building the syntax of stored procedures you
should use DB2 Stored Procedure Builder, which is an application that runs on various versions of Windows OS. For that purpose you can consult an IBM Red Book called Cross-Platform DB2 Stored Procedures: Building and Debugging, SG24-5485-01, or check the Web for the IBM DB2 Development Centre on the DB2 Developer Domain site.

DSNUTILS ENVIRONMENT

DSNUTILS stored procedure is one of several stored procedures that IBM supplied in DB2 OS390 Version 5 via APAR PQ15684 and which later on became part of the base of DB2 Version 6 and Version 7.

During installation of DB2 Version 7 or during migration from 5 to 6 to 7 or directly from 5 to 7, when the system programmer or the environmental DBA runs the DB2 installation CLIST, a job called DSNUTILS will be generated. This is the job that creates the IBM canned stored procedures, including DSNUTILS. We strongly advise the reader to locate that job and study it, particularly when it relates to DSNUTILS stored procedure.

DSNUTILS stored procedure is used by an application automatically to invoke DB2 utilities such as image copy, reorg, RUNSTATS, or quiesce from the body of its calling program without the need to worry about JCL, dataset allocation, or utility statements, etc. This JCL and dataset management is all done dynamically by DSNUTILS stored procedure.

All the programmer is required to do is to invoke the DSNUTILS stored procedure with EXEC SQL CALL DSNUTILS (<bunch of parameters>) END-EXEC, supplying values for the relevant bunch of parameters.

About 43 parameters are involved in the DSNUTILS call. Some of these parameters are relevant to some utilities and some are relevant to other utilities. The irrelevant parameters are initialized to the empty string "".

Behind the scenes the DSNUTILS stored procedure invokes the famous DSNUTILS program to execute the desired utility.

The output generated by the desired utility is written to the SYSIBM.SYSPRINT global temporary table. The invoking program can declare a cursor on this SYSIBM.SYSPRINT table and opens it like this:
EXEC SQL DECLARE SYSPRINT CURSOR WITH RETURN FOR
SELECT * FROM SYSCMY.SYSPRINT
ORDER BY SEQNO;

EXEC SQL OPEN SYSPRINT;

The invoking program then fetches the output from the SYSPRINT cursor row-by-row in the traditional way.

The DB2 Utility Guide and Reference manual explains the meaning and how to set up the various parameters for the call.

DSNUTILS stored procedure supports the LISTDEF and TEMPLATE features of DB2 Version 7. But one should be aware that since this stored procedure allows only a single utility invocation, caution should be exercised in using the TEMPLATE feature for dynamic allocation.

Actually the original intent for DSNUTILS stored procedure was to run via the IBM Control Centre from the workstation, but since then it has been developed for other stand-alone uses by applications, as we shall see with PeopleSoft.

The DSNUTILS stored procedure must run in a Work Load Manager (WLM) environment that enables multiple address spaces. The WLM is an MVS facility providing a solution to managing, balancing, and distributing MVS computing resources that are concurrently demanded by various competing subsystems such as CICS, DB2, etc.

Consequently the systems programmer should set up the application environment and the service policy for DSNUTILS in WLM.

The WLM environment name in the WLM config file should match the name of the environment in the CREATE STORE PROCEDURE statement of the DSNUTILS or execution will fail.

The WLM procedure created for DSNUTILS should state NUMTCB=1 and there should be no limit on starting server address spaces for the subsystem instance.

Stored procedures in general, including DSNUTILS, can be used to provide an additional layer of security. The privilege to create or execute a stored procedure must be explicitly granted or revoked regardless of the security requirements of the underlying base tables.

Stored procedures also have to satisfy external security interfaces such...
as RACF or TSS. The SECURITY parameter in the CREATE STORE PROCEDURE statement of the procedure should be given careful consideration.

In addition, one should model on the following sample permissions of RACF to authorize a user to create a stored procedure in a non-data sharing environment:

PERMIT <db2 subsystem name>.WLMENV.<name of the wlm env> CLASS(DSNR) ID(<user id>) ACCESS(READ)

The following is a sample of the necessary TSS permissions for the stored procedure:

TSS ADD(<user profile name>) DB2(DSNR.)
TSS PERMIT(<user id>) DB2(DSNR.<db2 subsystem>.WLMENV.<wlm env>)

The Recoverable Resource Services (RRS) is an OS/390 facility that coordinates two-phase commit processing between various resources within the MVS system. DB2 supports these services, consequently any application that accesses DB2 resources like DSNUTILS needs RRS to be installed on OS/390. It needs it up and running at the time of the execution of the DSNUTILS stored procedure.

In addition, the Recoverable Resource Services Attachment Facility (RRSAF) is needed for any stored procedure that uses the WLM-established address space.

The user ID associated with the WLM-established stored procedures address space must be able to read the ssnm.RRSAF profile and be associated with it. That user ID must also be able to run this DB2 RRSAF attachment facility.

Since we are invoking DSNUTILS stored procedure via REXX, it goes without saying that the REXX language support must be installed on OS/390. This is done by the MVS systems programmer. The DBA needs only to BIND the relevant DBRMIs into packages that DB2 uses. These packages are placed in different collections, namely DSNREXX, DSNREXCS, DSNREXRS, DSNREXRR, and DSNREXUR. These collections reflect the various isolation levels of DB2. In order to achieve the BINDing, the DBA should modify then submit the installation job DSNTIJRX, which can be found in the SDSNSAMP library that is supplied by IBM.
HOW PEOPLESOFT APPLICATIONS USE DSNUTILS IN AN OS/390 ENVIRONMENT

We said before that the original intent for DSNUTILS stored procedure was to run via the IBM Control Centre from the workstation, but, since then, DSNUTILS stored procedure has evolved for use by various applications. One of these applications is PeopleSoft, which uses it to invoke a DB2 utility called RUNSTATS.

One practical usage for invoking DSNUTILS stored procedure from REXX as discussed in this article is in troubleshooting of PeopleSoft applications. Let us discuss how PeopleSoft invokes DSNUTILS stored procedure.

There is one kind of PeopleSoft application programs called Application Engine (AE) that runs in the OS/390 environment in Unix System Services to perform all sorts of concurrent batch activities for PeopleSoft.

PeopleSoft AE programs invoke a proprietary function from within their code called %UpdateStats. This PeopleSoft %UpdateStats invoked function will resolve into an EXEC SQL CALL DSNUTILS (<bunch of parameters>) END-EXEC. One of the parameters involved would be the name of the required DB2 utility, which in this case is RUNSTATS, and other parameters would be the name of the database and tablespace that AE requires to run the DB2 RUNSTATS utility against.

In order for the %UpdateStats to be successful, the entire environmental configurable components discussed above for DSNUTILS have to be installed, configured, and running. Also the %UpdateStats has to be enabled within the PeopleSoft system itself, but that is beyond the scope of this article.

The purpose for which AE invokes the DB2 RUNSTATS utility is a classic one. Fresh statistics for DB2, particularly in a dynamic SQL environment such as a PeopleSoft environment, will help the DB2 optimizer to decide and chose a better access path to access the underlying DB2 tables. A better access path will improve application performance and its throughput.

Like any other application, one may experience a performance problem in PeopleSoft that relates to the RUNSTATS utility.

One may not be sure where the problem resides. Is the problem in the
application program or in the infrastructure environment?

In order to answer that question one can invoke DSNUTILS using this REXX program as explained in this article, outside PeopleSoft, in order to validate that all the components necessary for the execution of DSNUTILS are set up and functioning well. Only then can we exclude the infrastructure environment from our troubleshooting effort as a potential problem.

SAMPLE JCL TO RUN A REXX PROGRAM THAT CALLS DSNUTILS STORED PROCEDURE OUTSIDE PEOPLESOFT

//USERIDA JOB .CLASS-Z
//STEP01 EXEC PGM=IKJEFI01
//** browse the sdsnload library to verify that rxsubcom is there
//STPLIB DD DISP=SHR,DSN=SYS1.XXX.SDSNLOAD
//SYSEXEC DD DISP=SHR,DSN=<library contains the rexx logic>
//SYSPRINT DD SYSOUT-*
//SYSPSPLUNCH DD SYSOUT-*
//SYSSDUMP DD SYSOUT-*
//SYSTSIN DD *
//DSNUTILS

SAMPLE OF THE REXX PROGRAM (CALLED CONVENIENTLY DSNUTILS) THAT INVOKES DSNUTILS STORED PROCEDURE

/* - x/  
/* THIS IS A SAMPLE REXX PROGRAM. INVOKING DSNUTILS */
/* - x/  
/* FIRST CHECK TO SEE WHETHER THE REXX/DB2 COMMAND ENVIRONMENT IS */
/* AVAILABLE. IF IT IS NOT, THEN ESTABLISH IT. */
/* THEN CONNECT TO THE APPROPRIATE DATABASE (SUBSYSTEM) */
ADDRESS TSO "SUBCOM DSNREXX"
/*THE PREVIOUS COMMAND IS TO CHECK WHETHER THE REXX/DB2 COMMAND 
ENVIRONMENT IS AVAILABLE TO YOU. IF IT IS NOT AVAILABLE THEN 
YOU NEED TO ISSUE THE NEXT COMMAND, IE RXSUBCOM, TO ESTABLISH 
THE ENVIRONMENT. AFTER YOU ESTABLISH YOUR ENVIRONMENT YOU CAN 
CONNECT TO YOUR DESIRED DB2 SUBSYSTEM */
IF RC THEN
   DO
      S_RC =RXSUBCOM('ADD ','DSNREXX','DSNREXX')
      IF S_RC=0 THEN
          RXSTATUS = 'ADDED'
      ELSE RXSTATUS = 'ISNOTADDED'
   END
ADDRESS 'DSNREXX'
SSID='YOUR DB2SUBSYSTEM THAT YOU WANT TO ACCESS'
"CONNECT" SSID
IF SQLCODE <> "0" THEN
   DO
        SAY "FAILED TO CONNECT TO THE DATABASE"
        SAY SQLCODE
        EXIT 8
END
closing comment

/* ASSIGN VALUES TO YOUR VARIABLES */
WWUTILITY_ID = "RUNS01"
WWRESTART = "NO"
WWUTSTMT = "RUNSTATS TABLESPACE <DATABASE>,<TABLESPACE>"
WWUTSTMT = WWUTSTMT || "TABLE(<CREATOR>.TABLE_X)
COLUMN(ALL)"
SAY "WWUTSTMT :" WWUTSTMT
WWRETCODE = 0
WWUTILITY_NAME = "RUNSTATS TABLESPACE"
WWRECDSN = ""
WWRECDEV = ""
WWRECSPACE = 10
WWDISCDSN = ""
WWDISCDEV = ""
WWDISCSPACE = 10
WWPNCHDSN = ""
WWPNCHDEV = ""
WWPNCHSPACE = 10
WWCOPYDSN1 = ""
WWCOPYDEV1 = ""
WWCOPYSPACE1 = 10
WWCOPYDSN2 = ""
WWCOPYDEV2 = ""
WWCOPYSPACE2 = 10
WWRCPYDSN1 = ""
WWRCPYDEV1 = ""
WWRCPYSPACE1 = 10
WWRCPYDSN2 = ""
WWRCPYDEV2 = ""
WWRCPYSPACE2 = 10
WWWORKDSN1 = ""
WWWORKDEV1 = ""
WWWORKSPACE1 = 10
WWWORKDSN2 = ""
WWWORKDEV2 = ""
WWWORKSPACE2 = 10
WWMAPDSN = ""
WWMAPDEV = ""
WWMAPSPACE = 10
WWERRDSN = ""
WWERREDEV = ""
WWERRSPACE = 10
EXEC SQL 

"CALL DSNUTILS(
    "WWUTILITY_ID",
    "WWRESTART",
    "WWUTSTMT",
    "WWRETCODE",
    "WWUTILITY_NAME",
    "WWRECDSN",
    "WWRECDENV",
    "WWRECSPACE",
    "WWDISCDSN",
    "WWDISCDENV",
    "WWDISCSPACE",
    "WWPNCHDSN",
    "WWPNCHDEV",
    "WWPNCHSPACE",
    "WWCOPYDSN1",
    "WWCOPYDEV1",
    "WWCOPYSPACE1",
    "WWCOPYDSN2",
    "WWCOPYDEV2",
    "WWCOPYSPACE2",
    "WWRPCYDSN1",
    "WWRPCYDEV1",
    "WWRPCYSPACE1",
    "WWRPCYDSN2",
    "WWRPCYDEV2",
    "WWRPCYSPACE2",
    "WWWORKDSN1",
    "WWWORKDEV1",
    "WWWORKSPACE1",
    "WWWORKDSN2",
    "WWWORKDEV2",
    "WWWORKSPACE2",
    "WWWAPDSN",
    "WWWAPDEV",
    "WWWAPSPACE",
    "WWERRDSN",
    "WWERRDEV",
    "WWERRSPACE",
    "WWFILTRDSN",
    "WWFILTRDEV",
    "WWFILTRSPACE"
)"

/* READING FROM CURSOR SYSPRINT */

SAY
SAY "HERE ARE THE UTILITY MESSAGES :"
SAY "-------------------------------------"
SAY SQLSTMO1 = "SELECT * FROM SYIBM.SYSPRINT WITH UR"
"EXECSQL DECLARE C1 CURSOR FOR S1"
CALL CHECKSQLCODE
"EXECSQL PREPARE S1 INTO :SQLDA01 FROM :SQLSTM01"
CALL CHECKSQLCODE
"EXECSQL OPEN C1"
CALL CHECKSQLCODE
S1_SQLCODE = 0

/* REMEMBER THAT THE SQL/REXX INTERFACE AUTOMATICALLY PROVIDES THE
FIELDS OF THE SQLDA FOR EACH UNIQUE DESCRIPTOR NAME. SO ALL YOU NEED TO
DO IS TO INTERROGATE ITS VARIABLES */.
DO UNTIL(S1_SQLCODE <> 0)
"EXECSQL FETCH C1 USING DESCRIPTOR :SQLDA01"
CALL CHECKSQLCODE
S1_SQLCODE = SQLCODE
IF S1_SQLCODE = 0 THEN DO
SAY SQLDA01.2.SQLDATA
END
"EXECSQL CLOSE C1"
CALL CHECKSQLCODE

/* DISCONNECT FROM YOUR DATABASE */
ADDRESS 'DSNREXX '
"DISCONNECT" '<YOUR DB2 SUBSYSTEM>'
ADDRESS DSNREXX "DISCONNECT"
/* remove the host command environment */
IF RXSTATUS = 'ADDED' THEN
S_RC = RXSUBCOM('DELETE' , 'DSNREXX', 'DSNREXX')
EXIT 0

CHECKSQLCODE:

/* REMEMBER THAT IN REXX SQL COMMUNICATIONS AREA (SQLCA) IS
AUTOMATICALLY INCLUDED BY THE SQL/REXX INTERFACE. SO ALL YOU NEED TO DO
IS TO INTERROGATE ITS VARIABLES */.
IF(SQLCODE <> 0) THEN
DO
SAY "SQLCODE = " SQLCODE
SAY "SQLERRMC = " SQLERRMC
SAY "SQLERRP = " SQLERRP
SAY "SQLERRD1 = " SQLERRD.1
SAY "SQLERRD2 = " SQLERRD.2
SAY "SQLERRD3 = " SQLERRD.3
SAY "SQLERRD4 = " SQLERRD.4
SAY "SQLERRD5 = " SQLERRD.5
SAY "SQLERRD6 = " SQLERRD.6
EXIT 8
END
RETURN

Nicola Nur
Senior DBA (Canada)
Spiffing up SPUFI (Version 2)

*DB2 Update*, issue 132 (October 2003) had an article about invoking SPUFI from EDIT mode on some SQL. Various requests have been made for improvements, and here is the result.

**NEW FEATURES**

New features include:

- Individual users want different options, so ESQL allows the user to specify options via run-time parameters. These options include most of the fields that you can normally specify on the SPUFI panels, plus some functions specially for ESQL.

- A detailed HELP panel documents ESQL functionality, including the actual values of all default options as they are tailored in ESQL EXEC.

- ESQL can generate a batch job to process SQL using the DSNTEP2 program and the SQL script is included inline.

- A title box can be automatically added at the start of the SQL input to identify the subsystem, date/time, SQL source dataset, and which lines are being executed.

- SQL scripts downloaded from other platforms are often variable-length files. Therefore, ESQL now accepts SQL input with any record length and either fixed or variable record format. That is converted to an SQL script with 80-byte fixed-length records for SPUFI (or DSNTEP2).

- Any lines of SQL input longer than 80 bytes are truncated to 80 bytes before being passed to SPUFI (or DSNTEP2) to run. Alternatively, there is a new option to REFORMAT an SQL script before running it. This option first removes any line numbers. Next, any lines with SQL text longer than 72 bytes, or a comment continuing past column 80, will be split into multiple lines.

- SPUFI will dynamically change the DCB of an existing output file.
if the LRECL and RECFM values don’t match what’s specified on the SPUFI panel. EXSPUFI EXEC will now preserve the DCB of a pre-allocated output file by checking it and passing the existing values to SPUFI.

- Some problems using IBM’s standard CLISTs to invoke SPUFI are avoided by not using them. SPUFI is now invoked directly in EXSPUFI instead.

IMPLEMENTATION

The improvements are so extensive that all parts of the first version need to be replaced, and a HELP panel and skeleton have been added. Various tailoring of the code to your local standards is required. If you have already created an ISPF command to invoke SPUFI using ESQL (as described in the first article), it will continue to work OK with this updated code.

ESQLHELP PANEL

This needs no changes.

)PANEL KEYLIST(ISRHLP2,ISR)
)ATTR DEFAULT(!+_
  TYPE(PT)
  $ TYPE(NT)
  $ TYPE(ET)
  # TYPE(CT)
  ~ TYPE(TEXT) COLOR(BLUE)
  ^ TYPE(OUTPUT) COLOR(RED) CAPS(OFF)
  £ TYPE(CHAR) INTENS(HIGH) COLOR(TURQ)
  \ TYPE(CHAR) INTENS(LOW) COLOR(GREEN)
  / AREA(DYNAMIC)
  % AREA(SCRL) EXTEND(ON)
)BODY
#HELP$--------------------------! ESQL Command
$--------------------------#HELP
$Command ==>_ZCMD
$Version
2.0
%INFO
%
)AREA INFO
$FUNCTION :$ESQL runs SQL, either from an online EDIT (invoking SPUFI), or by
$ creating a TSO BATCH job (invoking DSNTEP2) and submitting it.
$ It can also be used to invoke SPUFI from an ISPF command, from any
$ ISPF panel, as is described at the end of this HELP.
$ -INVOCATION:$The ESQ\textsuperscript{L} command can be called from the command line of
$ any ISPF EDIT or VIEW to execute all lines of the SQL data, or to
$ execute only the lines specified by$CC$line commands.
$ like the following:
$  
$ Command -->$ESQL (ssid) (other parameters)
$  \textbackslash L2,SH $ SELECT col1, col2
$  000003 FROM table1
$  \textbackslash L4,SH $ ORDER BY col1 ;
$  000005 SELECT * FROM SYSIBM.SYSTABLES;
$  
$ The above example would run only the SELECT from table1.
$  
$ The SQL data does not need to be saved to disk before running ESQ\textsuperscript{L}.
$ -SQL INPUT :$The input dataset can have any record length and either
$ fixed or
$ variable record format. ESQ\textsuperscript{L} will copy the selected lines of SQL
$ to an 80-byte fixed-length file for input to SPUFI, or added as
$ in-stream SYSIN input in a BATCH job. Any long lines of SQL text
$ will be truncated at 80 bytes if ESQ\textsuperscript{L} is run with the parameter
$ #FORM(NO)$or they will be reformatted into multiple lines if it is
$ run with#FORM(YES)$parameter.
$ -REFORMATTED SQL:
$  
$ The reformatting of SQL starts by removing any line numbers.
$ Next any SQL text too long for SPUFI or DSNTEP2 will be split into
$ multiple lines. Then SQL can be up to column 72, and comments can
$ be up to column 80 for SPUFI or to column 72 for DSNTEP2.
$  
$ For the best readability, the splitting is done at a blank, comma
$ or right-bracket that is not inside a quoted character string,
$ (delimited by$"$or$'quot$es). Any lines thus split are marked
$ with#<REFORM$in columns 73 - 80.
$  
$ Quoted character strings are preserved as follows:
$ a) If the original SQL text is less than 73 bytes, any character
$ string on that line may be incomplete and continue onto the
$ next line, where the continuation would start in column 1.
$ Such a line would be left unchanged.
$ b) If the text is longer than 72 bytes, any character string on
$ that line must be complete. That line would be split into
$ multiple lines, with any long character strings continued onto
$ the next line, as described in a) above.
$ -SPUFI DATASETS:
If you are running the SQL via SPUFI and you don't specify dataset names in INDSN and OUTDSN parameters it uses default names like:
SPUFI input: ^INDSN
SPUFI output: ^ODSN
If the output dataset already exists its DCB will be left as it is, unless you use the LRECL or RECFM parameters to change it. If the dataset doesn't already exist it will be created with default DCB or according to any LRECL or RECFM parameters specified.

- PARAMETERS:

$If any of the following are used they must be the first parameter
! Parm    Description
# ssid   $DB2 sub-system id. If not supplied - it uses the default: def@ssid-
# HELP   $display this HELP panel
# ?      $display this HELP panel

$The following can be entered in any parameter position
! Parm    Description
# BATCH  $run the SQL via a batch job

$For SPUFI the following can be entered in any order, in format parm( value )
! Parm    Default    Possible Values    Description
# ALLOC  ^def@liba=any clist or exec
#        $SPUFI library allocation routine name
# SCR    ^def@scrn=New. Ontop
#        $logical SPUFI screen
# PAN    ^def@panl=Yes. No
#        $show SPUFI panel?
# TITLE  ^def@title=Yes. No
#        $title in input identifying SQL source
# FORM   ^def@form=Yes. No
#        $reformat SQL input into 1st 72 bytes
# ED     ^def@ed=Yes. No
#        $edit input file before SPUFI invoked
# INDSN  ~fully qualified name
#        $SPUFI input SQL dataset name
# OUTDSN ~fully qualified name
#        $SPUFI output report dataset name
# LRECL  ^def@len ~80 - 32768
#        $SPUFI output dataset LRECL
# RECFM  ^def@fmt ~F, FB, VBA, VB, VBA
#        $SPUFI output dataset RECFM
# UNIT   ^def@uni=any type of DASD unit
#        $SPUFI output dataset UNIT
# BR     ^def@br=Yes. No
#        $browse SPUFI output?
# LOC    ~any defined location
#        $SPUFI connect to DB2 location via DDF
# ISO    ^def@iso=CS, RR
#        $SPUFI isolation
# MAXNO  ^def@mno ~greater than 1
#        $max numeric chars in SELECT output
# MAXCH  ^def@mch ~greater than 0
#        $max character field length in SELECT
# MAXSEL ^def@max ~greater than 0
#        $maximum rows of data in SELECT output
# COL    ^def@col=Names, Labels, Any, Both
#        $SPUFI output column headings
# COM    ^def@com=Yes. No
#        $auto-commit for SPUFI

$For BATCH the following can be entered in any order, in
format$parm(value)

<table>
<thead>
<tr>
<th>Parm</th>
<th>Default</th>
<th>Possible Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TITLE</td>
<td>^bat@title-Yes, No</td>
<td></td>
<td>$title in input identifying SQL source</td>
</tr>
<tr>
<td>FORM</td>
<td>^bat@form-Yes, No</td>
<td></td>
<td>$reformat SQL input into 1st 72 bytes</td>
</tr>
<tr>
<td>ED</td>
<td>^def@ejc1-Yes, No</td>
<td></td>
<td>$EDIT batch JCL before job submission</td>
</tr>
<tr>
<td>SUB</td>
<td>^def@sub-Yes, No</td>
<td></td>
<td>$auto-submit batch JCL</td>
</tr>
</tbody>
</table>

$Examples:

!a)$ESQ1
   $Run SQL on the default DB2 sub-system via SPUFI, using the default
   $value of all parameters as shown above.

!b)$ESQ1 DB2A FORM(Y) ED(Y) MAXSEL(5000)
   $Run SQL on DB2A sub-system, reformat the SQL into the first 72 bytes
   $of the the SPUFI input file and invoke an EDIT of that file before
   $SPUFI runs, and show a maximum of 5000 rows for any SELECT.

!c)$ESQ1 LRECL(3200) RECFM(FB) UNIT(DISK) MAXCH(80)
   $Run SQL on the default DB2 sub-system, creating a SPUFI output file
   $with RECFM=FB,LRECL=3200 and a maximum of 80 bytes shown for any
   $selected column of type CHARACTER.

!d)$ESQ1 SCR(NW) PAN(YES)
   $Run SQL on the default DB2 sub-system, invoking SPUFI in a new split
   $screen and displaying the SPUFI panel - allowing the user to change
   $any parameters in SPUFI before running the SQL.

!e)$ESQ1 DB2E BATCH FORM(YES) ED(N)
   $Run SQL on DB2E sub-system via a batch job which executes DSNTEP2,
   $and reformat the SQL into the first 72 bytes for input, but do not
   $EDIT the batch job's JCL before submitting it.

-Invoke SPUFI via an ISPF Command:
   $You can invoke the standard SPUFI by entering$SPUFI$or$SPUFI ssid$on
   $the command line of any ISPF panel. Then the SPUFI primary panel
   $will be shown, ready for input.
   $To be able to do that, add the following command to one of your
   $active command tables (preferably a User or Site table, if you
   $have them).
   $   #Verb = SPUFI
   $   #Trunc = 0
   $   #Action = SELECT CMD(%ESQ1 &amp;ZPARM) NEWAPPL
   $   #Description = 'Invoke SPUFI'
   $
   )INIT
   VGET (DEF@SSID, DEF@SCRN, DEF@LIBA, DEF@PANL, DEF@TITL, DEF@FORM, DEF@ED, DEF@LEN, DEF@FMT, DEF@UNI, DEF@BR, DEF@ISO, DEF@MNO,
DSNESP01 PANEL

This is not the full panel definition. Start with a copy of the IBM panel and apply the following additions/updates:

1. Insert this at the start of the )ATTR section (before IBM's comments):

```plaintext
/****************************
/* This is a standard IBM panel with modifications to enable SPUFI to*
/* be run directly from an EXEC or CLIST, without the panel displayed*/
/****************************
/*
/* a) When a dataset name is specified in SPUFIDSN (Profile pool), it*
/* has come from REXX EXEC EXSPUFI. Current field values will be *
/* saved, replaced by values for immediate execution of the dsname*
/* from SPUFIDSN, and <ENTER> key simulated (to invoke the SQL *
/* processing without displaying the panel). On return to this *
/* panel the original field values are written back to the pool. *
/* and <END> key simulated (to exit without displaying the panel).*
/* Therefore, the user does not see this panel at all. *
/*
/* Note that the above actions only occur if SPUFIDSN holds a name*
/* from EXSPUFI; otherwise it functions as usual. Edit macro ESQL*
/* invokes EXSPUFI and other execs/clists could potentially do the*
/* same. *
```
2 Put your company name in the title to show it is a modified panel. Change the second line of the )BODY section as follows (the rest of the section is unchanged):

% SPUFI for Spiffy Computer Co. SSID: &DSNEOV01

3 Insert this at the end of the )INIT section (the rest of the section is unchanged):

*/
/* SAVE & RLSSTORE VALUES TO RUN SPUFI WITHOUT DISPLAYING THIS PANEL*/
/*---------------------------------------------------------------*/
VGET (SPUFIDSN) PROFILE
IF (&SPUFIDSN = &Z) /* dsn from EXSPUFI exec? Ron */
&EXSPUFI = Y /* if dsnname found Ron */
/* Input Dataset Name */
&OLDESV15 = &DSNESV15 /* invoked by EXSPUFI Ron */
&DSNESV15 = &SPUFDNSN /* */
&SPUFDNSN = &Z /* save Dataset Name Ron */
&SPUFDNSN = &Z /* remove dsnname from pool Ron */
VPUT (SPUFDNSN) PROFILE /* */
/* Output Dataset Name */
&OLDESV16 = &DSNESV16 /* */
VGET (SPUFIOUT) PROFILE /* */
/* */
IF (&SPUFIOUT = &Z) /* new out-dsname supplied Ron */
&OUTDSN = TRUNC(&SPUFIOUT,',' ) /* out-dsname Ron */
&OUTCOND = 'DEVICE' or blank /* 'COND' or blank Ron */
&DSNESV16 = &OUTDSN /* */
&DSNESV16 = &OUTDSN /* */
&SPUFIOUT = &Z /* remove dsnname from pool Ron */
VPUT (SPUFIOUT) PROFILE /* */
/* */
/* Change Defaults? */
&OLDESVIA = &DSNESVIA /* save Change Defaults Ron */
&DSNESVIA = NO /* */
/* */
/* Edit Input Dataset? */
&OLDESV17 = &DSNESV17 /* save Edit Input Ron */
&DSNESV17 = NO /* */
/* */
/* Execute SQL? */
&OLDESV18 = &DSNESV18 /* save Execute SQL Ron */
&DSNESV18 = YES /* */
/* */
/* Autocommit? */
&OLDESVID = &DSNESVID /* save Autocommit Ron */
VGET (SPUFICOM) PROFILE /* */
IF (&SPUFICOM = &Z) /* value supplied Ron */
&DSNESVID = &SPUFICOM /* */

&SPUFICOM = &Z
VPUT (SPUFICOM) PROFILE
/* remove value from pool Ron */
ELSE
&DSNESV1D = YES
/* Ron */
/*-- Browse Output Dataset? --*/
&OLDENV19 = &DSNESV19
VGET (SPUFIBRO) PROFILE
/* value supplied Ron */
IF (&SPUFIBRO = &Z)
&DSNESV19 = &SPUFIBRO
&SPUFIBRO = &Z
VPUT (SPUFIBRO) PROFILE
/* remove value from pool Ron */
ELSE
&DSNESV19 = YES
/* Ron */
/*-- Connect Location --*/
&OLDENV1 = &DSNESV1V
VGET (SPUIFLOC) PROFILE
/* value supplied Ron */
IF (&SPUIFLOC = &Z)
&DSNESV1 = &SPUIFLOC
&SPUIFLOC = &Z
VPUT (SPUIFLOC) PROFILE
/* remove value from pool Ron */
ELSE
&DSNESV1 = &Z
/* Ron */
/*-- Isolation Level --*/
&OLDENV2F = &DSNESV2F
VGET (SPUIISO) PROFILE
/* value supplied Ron */
IF (&SPUIISO = &Z)
&DSNESV2F = &SPUIISO
&SPUIISO = &Z
VPUT (SPUIISO) PROFILE
/* remove value from pool Ron */
ELSE
&DSNESV2F = CS
/* Ron */
/*-- Max SELECT lines --*/
&OLDENV2D = &DSNESV2D
VGET (SPUFIMAX) PROFILE
/* value supplied Ron */
IF (&SPUFIMAX = &Z)
&DSNESV2D = &SPUFIMAX
&SPUFIMAX = &Z
VPUT (SPUFIMAX) PROFILE
/* remove value from pool Ron */
ELSE
&DSNESV2D = 2500
/* Ron */
/*-- Output LRECL --*/
&OLDENV2C = &DSNESV2C
VGET (SPUFILN) PROFILE
/* value supplied Ron */
IF (&SPUFILN = &Z)
&DSNESV2C = &SPUFILN
&SPUFILN = &Z
VPUT (SPUFILN) PROFILE
/* remove value from pool Ron */
ELSE
&DSNESV2C = 4092
/* Ron */
/*-- Output BLKSIZE --*/
&OLDENV21 = &DSNESV21
/* save Output BLKSIZE Ron */
VGET (SPUFIBLK) PROFILE
/*
IF (&SPUFIBLK = &Z)
&DSNESV21 = &SPUFIBLK
&SPUFIBLK = &Z
VPUT (SPUFIBLK) PROFILE /* remove value from pool Ron */
ELSE
&DSNESV21 = 4096 /*
/*- Output RECFM --*/ /* Ron */
&OLDESV22 = &DSNESV22 /* save Output RECFM Ron */
VGET (SPUFIFMT) PROFILE /*
IF (&SPUFIFMT = &Z)
&DSNESV22 = &SPUFIFMT /*
&SPUFIFMT = &Z /*
VPUT (SPUFIFMT) PROFILE /* remove value from pool Ron */
ELSE
&DSNESV22 = VB /*
/*- Output UNIT --*/ /* Ron */
&OLDESV2E = &DSNESV2E /* save Output UNIT Ron */
VGET (SPUFIUNI) PROFILE /*
IF (&SPUFIUNI = &Z)
&DSNESV2E = &SPUFIUNI /*
&SPUFIUNI = &Z /*
VPUT (SPUFIUNI) PROFILE /* remove value from pool Ron */
ELSE
&DSNESV2E = SYSDA /*
/*- Maximum numeric characters --*/ /* Ron */
&OLDESV24 = &DSNESV24 /* save max num chars Ron */
VGET (SPUFIMNO) PROFILE /*
IF (&SPUFIMNO = &Z)
&DSNESV24 = &SPUFIMNO /*
&SPUFIMNO = &Z /*
VPUT (SPUFIMNO) PROFILE /* remove value from pool Ron */
ELSE
&DSNESV24 = 33 /*
/*- Maximum character field --*/ /* Ron */
&OLDESV25 = &DSNESV25 /* save max chars Ron */
VGET (SPUFIMCH) PROFILE /*
IF (&SPUFIMCH = &Z)
&DSNESV25 = &SPUFIMCH /*
&SPUFIMCH = &Z /*
VPUT (SPUFIMCH) PROFILE /* remove value from pool Ron */
ELSE
&DSNESV25 = 256 /*
/*- Column Heading --*/ /* Ron */
&OLDESV26 = &DSNESV26 /* save column heading Ron */
VGET (SPUFICOL) PROFILE /*
IF (&SPUFICOL = &Z)
&DSNESV26 = &SPUFICOL /*
&SPUFICOL = &Z /*
VPUT (SPUFICOL) PROFILE /* remove value from pool Ron */
ELSE
/*
4 Change the end of the )PROC section to the following; note that all the altered lines have the comment "** Ron */" and the rest are unchanged:

IF (&DSNESV18 = 'YES') /* EXECUTION REQUESTED */
VER(&DSNESV15,NONBLANK,MSG=DSNE350) /* IS INPUT DS SPECIFIED ? */
VER(&DSNESV15,DSNAMEQ) /* SYNTAX CHECKING 'OTHER' DS */
VER(&DSNESV16,NONBLANK,MSG=DSNE359) /* IS OUT DATA SET SPECIFIED? */
VER(&DSNESV16,DSNAMEQ) /* SYNTAX CHECKING OUTPUT DS */
IF (&DSNESV16 = SYSSOUT) /* IF SYSSOUT SPECIFIED */
.MSG = DSNE360 /* WE DONT SUPPORT IT */
IF (&DSNESV18 = 'NO') /* EXECUTION NOT REQUESTED */
IF (&DSNESV17 = 'YES') /* EDIT OPTION SELECTED */
VER(&DSNESV15,NONBLANK,MSG=DSNE350) /* IS INPUT DS SPECIFIED ? */
VER(&DSNESV15,DSNAMEQ) /* SYNTAX CHECKING 'OTHER' DS */
IF (&DSNESV19 = 'YES') /* BROWSE OPTIONS SELECTED */
VER(&DSNESV16,NONBLANK,MSG=DSNE359) /* IS OUT DATA SET SPECIFIED? */
VER(&DSNESV16,DSNAMEQ) /* SYNTAX CHECKING OUTPUT DS */

5 At the end of the )PROC section add the following immediately before the )END:

IF (&EXSPUFI = Y & .RESP = END) /* restore Dataset Name */
&DSNESV15 = &OLDESV15 /* restore Dataset Name */
Modify the job card to your local standards. Note that the jobname is the TSO userid plus one character, and that character changes each time a job is generated.

You should also change the STEPLIB libraries, and possibly the PLAN name.

ESQLJOB SKELETON
ESQL MACRO

Check all the default values at the start of the EXEC and update them as desired:

- Set a default DB2 sub-system ID.

- An output dataset name can be generated and definitely used (‘YES’), or used on the condition that no previous name was defined (‘COND’), or the previously used name should be used (‘NO’).

- As supplied here, some SPUFI fields are set to non-standard values (MAX CHAR FIELD = 256, MAX SELECT LINES = 2500, ISOLATION = ‘CS’).

- Check liby_alloc = ‘DB2LIBS’; change it to the name of your own library allocation routine if you don’t want to use DB2LIBS.

Check section ‘CREATE_SPUFIDSN’ where it creates the name for the SPUFI input dataset. In this code it will be called ‘userid.SPUFIN’ or ‘tsoprefix.userid.SPUFIN’, conforming to IBM standards. However, if the user has set TSO PROFILE NOPREFIX, it gets a value from RACF and that may not be valid for your company. (It may also generate a name for the SPUFI output dataset, and that will be based on the name for the SPUFI input dataset.)
ESQL: Execute SQL using SPUFI (online) or DSNTEP2 (batch)

Version 2.0

a) If invoked from EDIT (for ONLINE use):
   ie enter 'ESQL' or 'ESQL ssid' on command line, and
   select the lines of SQL via 'CC' line commands
   The SQL will be copied to a file (reformatted into the
   first 72 bytes if necessary), for input to SPUFI.
   Then that SQL will be invoked immediately and the SPUFI
   output file shown, returning directly to EDIT via PF3.
   Thus, the user does not see the normal SPUFI panel.

b) If invoked from EDIT (for BATCH use):
   ie enter 'ESQL BATCH' or 'ESQL ssid BATCH', and select
   the lines of SQL via 'CC' line commands
   The SQL will be processed exactly as above, except a
   batch job will be submitted to run DSNTEP2.

c) If invoked via command:
   ie enter 'SPUFI' or 'SPUFI ssid' on command line, where
   that is defined as: SELECT CMD(ESQL &ZPARM) NEWAPPL
   in an active ISPF command table.
   Then the normal SPUFI panel (DSNESPO1) will be shown.

Externa:

EXSPUFI REXX .. call DB2LIBS, then invoke SPUFI
DB2LIBS REXX .. allocate DB2 libraries for SPUFI
ESQJOB Skel .. JCL for batch SQL job (uses DSNTEP2)
ESQLHELP Panel .. HELP panel explaining use of ESQL
DSNESPO1 Panel .. modified version of IBM-supplied panel

Written: 2003/06/05 Last Updated: 2003/11/26 by Ron Brown
def@mch = '256' /* maximum character field */
def@col = 'NAMES' /* column headings */
def@iso = 'CS' /* SPUFI isolation? 'CS' or 'RR' */
def@max = '2500' /* maximum SELECT lines */
def@com = 'YES' /* autocommit? 'YES' or 'NO' */
def@liba = 'DB2LIBS' /* routine to alloc SPUFI libs */
def@ejcl = 'YES' /* edit batch JCL before submit? */
def@sub = 'N0' /* auto-submit of batch JCL? */
dset = '' /* name of SPUFI input dsn (if SQL copied into it)*/
/*-- get any input parameters --*/
Address ISREDIT "MACR0(parms) NOPROCESS"
macro rc = rc
If macro_rc > 0 Then Arg parms /* if NOT running from Edit/View */
Else Address ISPEXEC "CONTROL DISPLAY SAVE"
Call INPUT_PARMS /* get any user-specified parms */
/*-- create input file for SPUFI or DSNTEP2 --*/
If macro_rc = 0 Then /* running from Edit/View */
  Call GET_SQL /* get specified SQL statements */
/*-- run the SQL in the input file --*/
If result = 0 Then Do
  If esql_mode = 'ONLINE' Then Do
    If macro_rc = 0 Then /* if running from Edit/View */
      Call CREATE_SPUFIDSN /* write SQL into SPUFI input file */
      Call RUN_EXSPUFI /* SPUFI online */
    End
  End
  If esql_mode = 'BATCH' Then
    Call SUBMIT_BATCH_JOB /* DSNTEP2 batch job */
  End
  If macro_rc = 0 Then /* running from Edit/View */
    Address ISPEXEC "CONTROL DISPLAY RESTORE"
Return /*----------------------> END OF MAIN PROGRAM <-----------------------*/
/*-- get user parms + defaults to set input parms */
/*----------------------*/

INPUT_PARMS:
/* now we can see what parms we received */
Parse Upper Value ' 'parms With parml . ,
  1 'ALLOC(' libry_alloc '),' /* DB2 library alloc */
  1 'EDJCL(' edit_jcl '),' /* EDIT batch JCL */
  1 'SUB(' submit_jcl '),' /* SUBMIT batch JCL */
  1 'SCR(' spufi_screen '),' /* logical SPUFI screen */
  1 'PAN(' spufi_pan '),' /* show SPUFI panel? */
  1 'PNL(' spufi_pnl '),' /* show SPUFI panel? */
  1 'TITLE(' sql_title '),' /* title in SQL input */
  1 'INDSN(' spufidsn '),' /* SPUFI input dsname */
  1 'OUTDSN(' spufiout '),' /* SPUFI output dsname */
  1 'FORM(' format '),' /* reformat input data? */
  1 'REFORM(' reformat '),' /* reformat input data? */
  1 'LRECL(' spufi_len '),' /* SPUFI output LRECL */
  1 'RECFM(' spufi_fmt '),' /* SPUFI output RECFM */
  1 'UNIT(' spufi_uni '),' /* SPUFI output UNIT */
ED(' spufi_ed '), /* edit input file? */
BR(' spufi_br '), /* browse SPUIF output? */
LOC(' spufi_loc '), /* connect location */
ISO(' spufi_iso '), /* SPUIF isolation */
MAXSEL(' spufi_max '), /* max SELECT lines */
MAXNO(' spufi_mno '), /* max numeric field */
MAXCH(' spufi_mch '), /* max char field */
COL(' spufi_col '), /* column heading */
COM(' spufi_com '), /* auto-commit */
/*-- Show HELP panel --*/
If parm1 = 'HELP' | parm1 = '?' Then Do
  Address ISPEXEC "VPUT (", /* pass default values to HELP panel*/
    "def@ssid, def@scrn, def@liba, def@panl, def@titl, def@form,
    "def@ed, def@len, def@fmt, def@uni, def@br, def@iso, def@mno,
    "def@mch, def@max, def@col, def@com, def@ejcl, def@sub) SHARED"
  Address ISPEXEC "SELECT PGM(ISPTUTOR) PARM(ESQLHELP)"
Exit
End
/*-- DB2 ssid --*/
If Length(parm1) <> 4 | Pos('(',parm1) > 0
  Then ssid = def@ssid /* user specified no ssid */
Else ssid = parm1
/*-- title in SQL input --*/
If sql_title = '' Then sql_title = def@titl
If Pos(sql_title,'NO') = 1 Then sql_title = 'NO'
Else sql_title = 'YES'
/*-- reformat input into first 72 bytes?... FORM(x) or REFORM(x) --*/
If format <> '' Then reformat = format
If reformat = '' Then reformat = def@form
If Pos(reformat,'NO') = 1 Then reformat = 'NO'
Else reformat = 'YES'
/*-- submitting a batch job? --*/
If Pos(' BATCH ',Translate('parms ')) = 0 Then
esql_mode = 'ONLINE'
Else Do
esql_mode = 'BATCH'
/*-- edit batch JCL ... EDJCL(x) or ED(x) --*/
If edit_jcl = '' Then edit_jcl = spufi_ed
If edit_jcl = '' Then edit_jcl = def@ejcl
If Pos(edit_jcl,'NO') = 1 Then edit_jcl = 'NO'
Else edit_jcl = 'YES'
/*-- submit batch JCL ... SUB(x) --*/
If submit_jcl = '' Then submit_jcl = def@sub
If Pos(submit_jcl,'NO') = 1 Then submit_jcl = 'NO'
Else submit_jcl = 'YES'
/*-- create message for EDIT of batch job --*/
If edit_jcl = 'YES' Then Do
  If submit_jcl = 'YES' Then
    ZEDLMSG = ' "***** This JCL will be automatically submitted "
  If submit_jcl = 'NO' Then
ZEDLMSG = '***** This JCL will NOT be',
'automatically submitted *****'
Address ISPEEXEC "SETMSG MSG(ISRZ001)"
End

/*-- create message for invalid batch parameter combination --*/
If edit_jcl = 'NO' & submit_jcl = 'NO' Then Do
    ZEDLMSG = '***** ESQL parameters: BATCH ED(NO) SUB(NO) not',
        'accepted *****'
Address ISPEEXEC "SETMSG MSG(ISRZ001)"
Exit 8
End

/*-- end of BATCH parameters --*/
Return 0
End

/*-- routine to allocate libraries --*/
If libry_alloc = '' Then /*'NO' means already allocated */
    libry_alloc = def@liba

/*-- SPUFI logical screen --*/
If spufi_screen = '' Then spufi_screen = def@scrn
If Pos(spufi_screen,'NEW') = 1 Then spufi_screen = 'NEW'
Else spufi_screen = 'ONTOP'

/*-- show main SPUFI panel? --*/
If spufi_pan <> '' Then spufi_pan = spufi_pan
Else If spufi_pan = '' Then spufi_pan = def@pan1
If Pos(spufi_pan,'YES') = 1 Then spufi_pan = 'YES'
Else spufi_pan = '' /*'NO' is default */

/*-- SPUFI input dataset name --*/
If spufdsn <> '' Then
    spufdsn = Strip(spufdsn,'B','"') /* remove any quotes*/

/*-- SPUFI output dataset name --*/
If spufiout <> '' & Left(spufiout,1) <> "" Then
    spufiout = """spufiout"""" /* ensure dsname has quotes*/

/*-- SPUFI output LRECL, RECFM & BLKSIZE --*/
If spufi_len <> '' | spufi_fmt <> '' Then Do
    If spufi_len = '' Then spufi_len = def@len
    If spufi_fmt = '' Then spufi_fmt = def@fmt
    Select
        When spufi_fmt = 'VB' | spufi_fmt = 'VBA' Then
            spufi_blk = 32760
        When spufi_fmt = 'V' Then
            spufi_blk = spufi_len + 4
        When spufi_fmt = 'FB' | spufi_fmt = 'FBA' Then
            spufi_blk = 32760 - (32760 // spufi_len)
        When spufi_fmt = 'F' Then
            spufi_blk = spufi_len
    End
End
Else spufi_blk = ''

/*-- SPUFI output UNIT --*/
If spufi_uni = '' Then spufi_uni = def@uni
If spufi_uni = 'SYSDA' Then spufi_uni = '' /* default in EXSPUFU */
/*-- edit generated SPUFI input file? --*/
If spufi_ed = '' Then spufi_ed = def@ed
If Pos(spufi_ed,'YES') = 1 Then spufi_ed = 'YES'
Else spufi_ed = 'NO'
/*-- browse SPUFI output? --*/
If spufi_br = '' Then spufi_br = def@br
If Pos(spufi_br,'NO') = 1 Then spufi_br = 'NO'
Else spufi_br = 'YES'
/*-- SPUFI isolation --*/
If spufi_iso = '' Then spufi_iso = def@iso
If Pos(spufi_iso,'RR') = 1 Then spufi_iso = 'RR'
Else spufi_iso = /* 'CS' is default in EXSPUFI */
/*-- maximum SELECT lines --*/
If spufi_max = '' Then spufi_max = def@max
If spufi_max = '2500' Then spufi_max = '' /* default in EXSPUFI */
/*-- maximum numeric characters --*/
If spufi_mno = '' Then spufi_mno = def@mno
If spufi_mno = '33' Then spufi_mno = '' /* default in EXSPUFI */
/*-- maximum character field --*/
If spufi_mch = '' Then spufi_mch = def@mch
If spufi_mch = '256' Then spufi_mch = '' /* default in EXSPUFI */
/*-- column headings --*/
If spufi_col = '' Then spufi_col = def@col
If Pos(spufi_col,'ANY') = 1 Then spufi_col = 'ANY'
If Pos(spufi_col,'BOTH') = 1 Then spufi_col = 'BOTH'
If Pos(spufi_col,'LABEL') = 1 Then spufi_col = 'LABEL'
If Pos(spufi_col,'NAMES') = 1 Then spufi_col = '' /* default */
/*-- autocommit the SQL? --*/
If spufi_com = '' Then spufi_com = def@com
If Pos(spufi_com,'NO') = 1 Then spufi_com = 'NO'
Else spufi_com = '' /* 'YES' is default in EXSPUFI */
Return Ø
/* put the specified SQL into REXX array: input */
/*-----------------------------------------------*/
GET_SQL:
/*-- get CC range into variables fl & ll --*/
Address ISREDIT "PROCESS RANGE C "
proc_rc = rc
Select
When proc_rc = Ø then Do
Address ISREDIT "((fl) = LINENUM .ZFRANGE"
Address ISREDIT "((ll) = LINENUM .ZLRANGE"
If fl = ll
Then sel_lines = 'line' Format(fl)
Else sel_lines = 'lines' Format(fl) 'to' Format(ll)
End
When proc_rc = 4 then Do /* no range specified, take whole d'set*/
Address ISREDIT "((fl) = LINENUM .ZF"
Address ISREDIT "((ll) = LINENUM .ZL"
/* sel_lines = 'all' Format(ll) 'lines' */

sel_lines = 'all lines'
End

Otherwise  /* eg rc=16 for incomplete or conflicting line cmds */
    Address ISPEXEC "SETMSG MSG(ISRZ002)"
    Exit proc_rc
End

/*-- check DCB of SQL input dataset --*/
Address ISREDIT "((fml, fm2) = RECFM"
Address ISREDIT "((lrecl) = LRECL"
Address ISREDIT "((num) = AUTONUM" /* auto-numbering: ON, OFF */
If fml < 'F' & fml < 'V' Then Do
    ZEDLMSG = 'This data is RECFM='fml||Strip(fm2),',
        'LRECL='Strip(lrecl,'L','0')', but only'.
        'RECFM-F,FB,V or VB is accepted'
    Address ISPEXEC "SETMSG MSG(ISRZ001)"
    Exit 8
End

/*-- create title lines for start of input --*/
If sq1_title = 'YES' Then Do
    /*-- what dataset are we in? --*/
    Address ISREDIT "((ds1) = DATASET"
    Address ISREDIT "((lm1) = MEMBER"
    If lm1 <> ' ' Then ds1 = ds1('lm1')'
    If spufi_loc <> ' ' Then loc_text = Left(' , Location' spufi_loc,25)
        Else loc_text = Copies(' ',.25)
    If esql_mode = 'BATCH' Then Do
        input.1 = Copies('--',.36)
        input.2 = ' -- Sub-system' ssid loc_text,
            ' at' Time('N') 'on' Date()
        input.3 = ' -- DSNTEP2: run' sel_lines 'of' ds1
            /* input.4 will get a reformatting summary */
        input.5 = Copies('--',.36)
        input.6 = ' ' If spufi_loc <> ' ' Then Do
            input.7 = ' CONNECT TO' spufi_loc;'
            input.8 = ' '
            j = 8
    End
    Else j = 6
End

If esql_mode = 'ONLINE' Then Do
    input.1 = ' -- Sub-system' ssid loc_text ' .
        ' at' Time('N') 'on' Date()
    input.2 = ' -- SPUFI: run' sel_lines 'of' ds1
        /* input.3 will get an input (& reformatting) summary */
    input.4 = Copies('----------+',.8)
    input.5 = ' ' 
    j = 5
End
End
Else \( j = 0 \)  /* no title lines at start of input */
/*-- write the selected SQL into input. array --*/
input_summary = '--- ' l1 - fl + 1 'lines input to ESQL'
/*-- no reformating of SQL --*/
If reformat = 'NO' Then Do i = fl to l1
  j = j + 1
  Address ISREDIT "(line) = LINE "i
  input.j = Left(line,80)
End
/*-- reformating the SQL --*/
If reformat <> 'NO' Then Do  /* ie. reformat = 'YES' */
  cont_quote = '
  lines_split = 0
  /* comments can go up to column 72 (batch) or 80 (SPUFI) */
  If esql_mode = 'ONLINE' Then
    end_comment = 80
  Else  /* esql_mode = 'BATCH' */
    end_comment = 72
  /* process each line of SQL input in turn */
  Do i = fl to l1
    j = j + 1
    Address ISREDIT "(line) = LINE "i
    line = Strip(line,'T')  /* remove any trailing blanks */
    line_leng = Length(line)
    If fml = 'V' | (fml = 'F' & line_leng = lrec1) Then
      Call REMOVE_LINE_NUMBER
    If line = '' Then Do  /* blank line */
      input.j = ''
      Iterate i
      End
    line_leng = Length(line)
    Call FIND_COMMENT
    If result > 0 Then Return 16  /* invalid SQL */
  Select
    When line_leng < 73 & cs = 0 Then Do  /* no comment */
      sql_text = line
      ich = IN_STRING(73,'START')
      cont_quote = quote  /* = ' or " if unfinished string */
      input.j = line
    End
    When line_leng < 81 & cs > 0 & cs < 74 Then Do  /* comment*/
      cont_quote = ''
      input.j = line
    End

Editor's note: this article will be concluded in the next issue.

Ron Brown (ron_brown@hotmail.com)
Principal Consultant
Triton Consulting (Germany)

© Xephon 2004
.com Solutions has announced Version 1.73 Enterprise Edition of its CGlScripter software.

The product runs on MacOS X, Windows, Solaris, and Linux, and the new version now has support for Sybase and DB2 database servers.

CGlScripter instantly writes Perl CGI scripts for MySQL, Oracle, Access, SQL Server, Sybase, and DB2 databases.

CGlScripter now supports generating Perl CGI scripts for database servers enhancing developer productivity for heterogeneous database installations.

For further information contact: .com Solutions, 3984 Washington Blvd #334, Fremont, CA 94538, USA. Tel: (510) 490 3482.
URL: http://www.cgiscripter.net/products/cgiscripter/index.html.

***

Embarcadero has announced that they are partnering with IBM to extend their data access product, tcACCESS, with IBM’s DB2 Information Integrator product. The joint data access solution will provide integrated real-time access to diverse data such as IMS, ADABAS, DATACOM, IDMS and more, regardless of where it resides.

tcACCESS enables the joining of multiple disparate data sources from both mainframe and non-mainframe platforms to deliver logical views of information to the requestor. It enables any PC or Web application to select data from any IBM mainframe data source, and it enables any IBM mainframe application to access any non-mainframe data source.

For further information contact: GT Software, 1314 Spring Street NW, Atlanta, GA 30309-2810, USA. Tel: (404) 253 1300.