In this issue

3  Improving DB2 UDB back-up times
6  DB2 consistency tokens
27  DB2 plan and package query
46  DB2 UDB 8.1 – inserting into a view
49  DB2 news

© Xophon plc 2003
Subscriptions and back-issues
A year’s subscription to DB2 Update, comprising twelve monthly issues, costs £255.00 in the UK; $380.00 in the USA and Canada; £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 1999 issue, are available separately to subscribers for £22.50 ($33.75) each including postage.

DB2 Update on-line
Code from DB2 Update, and complete issues in Acrobat PDF format, can be downloaded from our Web site at http://www.xephon.com/db2; you will need to supply a word from the printed issue.

© Xephon plc 2003. 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.
Improving DB2 UDB back-up times

The DB2 UDB back-up database utility is used to back up a database and is a relatively simple utility with few parameters. With the nightly/weekly batch window becoming ever smaller, every second that you can save counts. But do these parameters make a big difference to the speed of the back-up? The following article looks at the speed of taking a full off-line DB2 UDB back-up using DB2 UDB 7.2 FP7 running on a Windows 2000 system, and whether the optional parameters make a difference to the time taken. I will look at only full off-line back-ups.

As a reminder, the back-up command is:

```
>db2 backup db <db-alias> to <drive>
WITH <num-buffers> BUFFERS BUFFER <buffer-size> PARALLELISM <n>
```

The only required parameters are db-alias (the database alias of the database you want to back up) and the drive (where the back-up should go). The optional parameters are num-buffers, which is the number of buffers that DB2 can use to perform the back-up; buffer-size, which, as the Command Reference manual says, is “...the size, in 4KB pages, of the buffer used when building the back-up image” (minimum 8, maximum 16384); and parallelism, which is the number of tablespaces that can be read in parallel by the back-up utility. The maximum number of buffers you can specify seems to depend on the buffer size, and the total amount of memory available for UTIL_HEAP_SZ (see below), whereas the maximum buffer size is fixed at 16,384 4KB pages. If we issue the back-up command ‘out of the box’, then the default values are: number of buffers 2, buffer size 1024, and parallelism 1. The value for buffer size (if you do not specify it) is the value of the database manager configuration parameter BACKBUFSZ.

I will not look at using tapes as the back-up medium, but I will look at backing up the database to the same drive as the database resides on, and also backing up to a different drive.

To work with a table of a meaningful size, I created an EMPLOYEE2
table of 4,194,304 rows in the SAMPLE database based on the EMPLOYEE table. This table was just under 6GB in size. The absolute table size/back-up times are not important, what we are looking for are the relative times.

As I will only be backing up 1 tablespace (userspace1), I will keep the default value for parallelism of 1.

One other variable in the back-up equation is the database configuration parameter UTIL_HEAP_SZ (default 5000 4KB pages and maximum value 524,288 4KB pages). The Command Reference manual says that this parameter should “be at least as high as the number of buffers * buffer size”. If you don’t increase the value of UTIL_HEAP_SZ to the recommended value, then you might get an SQL2009C error message when you try to take a back-up. I increased the size of UTIL_HEAP_SZ to the maximum value of 524,288, as shown below:

```sql
>db2 update db cfg for sample using UTIL_HEAP_SZ 524288
>db2 get db cfg for sample | find "UTIL"
```

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>2,048</td>
<td>4,096</td>
<td>8,192</td>
<td>12,288</td>
<td>16,384</td>
<td>32,768</td>
<td>65,536</td>
<td>131,072</td>
<td>262,144</td>
<td>524,288</td>
</tr>
<tr>
<td>2048</td>
<td>4,096</td>
<td>8,192</td>
<td>16,384</td>
<td>24,576</td>
<td>32,768</td>
<td>65,536</td>
<td>131,072</td>
<td>262,144</td>
<td>524,288</td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>8,192</td>
<td>16,384</td>
<td>32,768</td>
<td>49,152</td>
<td>65,536</td>
<td>131,072</td>
<td>262,144</td>
<td>524,288</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8192</td>
<td>16,384</td>
<td>32,768</td>
<td>65,536</td>
<td>98,304</td>
<td>131,072</td>
<td>262,144</td>
<td>524,288</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16384</td>
<td>32,768</td>
<td>65,536</td>
<td>131,072</td>
<td>196,608</td>
<td>262,144</td>
<td>524,288</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 1: Buffers*

Figure 1 is a matrix of all possible values for buffer size (vertical) and number of buffers (horizontal) and the corresponding UTIL_HEAP_SZ value. Don’t forget that the maximum buffer size allowed is 16,384.

There seems to be a slight overhead in the UTIL_HEAP_SZ allocation, so if you try to use 128 buffers with a size of 4096 and specify a UTIL_HEAP_SZ of 524,288, you will get the SQL2009C error message (this is why the figure at the intersection of such rows/columns is in italics).
To check on how long a back-up took, I used the list history command (>db2 list history backup all for sample).

First, let’s look and see the times (shown in minutes:seconds) we get for specifying different drives (C and D) for the back-up (remembering that the database exists on the C drive, and that the D drive is an external drive).

Command:

>db2 backup db sample to <drive>

Time for C drive 5:47. Time for D drive 4:16.

You can see that the time for the ‘out of the box’ command shows a 26% improvement if you back up to a different drive from the one on which the database resides (as you would expect!). So, let’s work with backing up to the D drive and try every allowable combination of number of buffers and buffer size. The resulting database back-up times (shown in minutes:seconds) are shown in Figure 2 (number of buffers is the horizontal axis and the buffer size is the vertical axis).

<table>
<thead>
<tr>
<th>Buffer Size</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>4:16</td>
<td>4:02</td>
<td>4:03</td>
<td>3:57</td>
<td>4:02</td>
<td>4:00</td>
<td>4:03</td>
<td>4:03</td>
<td>4:02</td>
<td>x</td>
</tr>
<tr>
<td>2048</td>
<td>4:05</td>
<td>4:11</td>
<td>4:05</td>
<td>4:02</td>
<td>4:01</td>
<td>4:05</td>
<td>4:08</td>
<td>4:08</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>4096</td>
<td>4:47</td>
<td>4:13</td>
<td>4:09</td>
<td>4:05</td>
<td>4:10</td>
<td>4:03</td>
<td>4:14</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>8192</td>
<td>4:32</td>
<td>4:33</td>
<td>4:33</td>
<td>5:33</td>
<td>4:33</td>
<td>4:34</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>16384</td>
<td>5:01</td>
<td>4:59</td>
<td>5:04</td>
<td>5:12</td>
<td>5:16</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Figure 2: Buffers and back-up times

Figure 2 shows us that in this particular situation/environment the best back-up time is achieved by specifying 12 buffers with a size of 1024, which decreases the back-up time by 7% from the ‘out of the box’ figures. This might not seem like a large decrease, but, as I said in the introduction, every second counts in the batch
window. Also, I am not saying that you should change your back-up commands to use these values. What I wanted to show was that there was a benefit in using values other than the defaults. You would have to go through a similar exercise to the one above to see what the optimum values are for your particular set-up.

Does the UTIL_HEAP_SZ value matter? I ran the ‘out of the box’ back-up command with UTIL_HEAP_SZ values of 50,000 and 100,000 and the difference in back-up times was negligible. It therefore seems that there is no gain in over-allocating the UTIL_HEAP_SZ value (but you need to specify the minimum value!), and, of course, its value also depends on how much memory you have available on your machine for DB2.

This has by no means been a scientific study, but I wanted to show that it is worth experimenting with the num-buffers and buffer-size parameters of the BACK-UP command to see whether you can improve your back-up times.

C Leonard
Freelance Consultant (UK) © Xephon 2003

DB2 consistency tokens

THE PROBLEM
A common problem I have encountered at several sites is the need to understand the specific format of the DB2 default CONTOKEN (consistency token). This value is used in numerous places:

• SYSPACKAGE DB2 table
• DBRM module
• LOAD module.

When DB2 runs a program, it examines the CONTOKEN stamped
in the module, and compares it with the SYSPACKAGE table – if they match the program works, otherwise we get the 805 SQL message.

Although we have useful fields in SYSPACKAGE we can refer to, such as the PDS the DBRM was bound from and the PCTTIMESTAMP (precompile timestamp), there is no exact method to derive a timestamp format from the CONTOKEN.

Secondly, at most sites there is usually some sort of REXX routine that converts this value using cumbersome counting techniques to approximate the date.

SOLUTION

Examining the DB2 manuals, the CONTOKEN is described as ‘internal format’ or ‘enhanced storeclock’ value, and hence gives us little information about decoding this value. So, after playing with various timestamps and creating storeclock values from them, the solution appeared.

The actual CONTOKEN is a storeclock value, shifted to millisecond rather than nanosecond precision.

So now we have a means to decode a CONTOKEN (and, of course, we can now create one if the urge takes us!).

Rather than giving a complete system to ensure -805 prediction, I have created a couple of programs that, using standard macros and copybooks, give usable functions to enable a bespoke solution if required.

THE PROGRAMS

The programs are:

- FUTTOKEN – this returns a DB2 timestamp from either a real hexadecimal CONTOKEN or a character representation of a CONTOKEN.
- FUTDBRM – this expects a DBRM PDS and member as
input, where the DBRM is mapped by DB2 macro DSNXDBRM. It calls FUTTOKEN and hence returns the CONTOKEN in hexadecimal format, a space, and the timestamp. Also this programs gives a good example of SVC99 processing.

The programs can be called either from JCL, inter-program linkage, or as a REXX function. For a REXX function, FUTTOKEN has to be statically link edited into FUTDBRM, hence I have statically bound it for everything.

FUTTOKEN JCL call example:

```plaintext
//PROG EXEC PROG=FUTTOKEN, PARM='CTOKEN=16CFB583DF01226'
//STEPLIB DD DISP=SHR, DSN=<load library>
//SYSOUT DD SYSOUT=A <- where the output is written
```

Alternatively the parm can be ‘HTOKEN=xxxxxxxx’, where xxxxxxxxxx is a real hexidecimal notation of the CONTOKEN.

FUTTOKEN REXX call example:

```plaintext
/* REXX */
LOAD_HLQ = 'my.hlq'
address ispexec
   "libdef isplib dataset id('" || LOAD_HLQ || ".LOADLIB")"

TVAL1 = '16CFB583DF01226'
TVAL = 'CTOKEN=' || TVAL1

/* Call the main program routine */
my_timestamp = FUTTOKEN(TVAL)
say my_timestamp
```

FUTDBRM JCL call example:

```plaintext
//PROG EXEC PROG=FUTDBRM, PARM='my.dbrm.lib(member)'
//STEPLIB DD DISP=SHR, DSN=<load library>
//SYSOUT DD SYSOUT=A <- where the output is written
```

FUTDBRM REXX call example:

```plaintext
/* REXX */
LOAD_HLQ = 'my.hlq'
address ispexec
   "libdef isplib dataset id('" || LOAD_HLQ || ".LOADLIB")"

my_return_val = FUTDBRM(my.dbrm.lib(member))
```
my_contoken = WORD(my_return_val,1)
my_timestamp = WORD(my_return_val,2)

Assembly decks:

// ASSEM EXEC PGM=ASMAØ, PARM='OBJECT, NODECK'
// SYSLIB DD DISP=SHR, DSN=SYS1. MACLIB
//        DD DISP=SHR, DSN=<DB2_HLQ>. SDSNMACS
// SYSUT1 DD UNIT=SYSDA, SPACE=(CYL,(5,5), RLSE)
// SYSPRINT DD SYSPRINT=*  
// SYSLIN DD DSN=&&LINK, DSN=(NEW, PASS), SPACE=(CYL,(5,5)), 
// DCB=(RECFM=FB, LRECL=80, BLKSIZE=400)
// SYSLIN DD DSN=/<my HLQ>. SOURCE(program)
// SYSPUNCH DD DUMMY
// SYSPRINT DD SYSPRINT=*  
// IF (ASSEM.RC < 8) THEN
// LINKEDIT EXEC PGM=IEWL, PARM='MAP, XREF, LIST, AC(Ø)'  
// SYSUT1 DD UNIT=SYSDA, SPACE=(CYL,(5,5), RLSE)
// SYSLIB DD DSN=/<my HLQ>. LOADLIB
// SYSLIN DD DSN=/<my HLQ>. LOADLIB(program)
// SYSPRINT DD SYSPRINT=*  
//         ENDIF  

where FUTTOKEN is assembled first, followed by FUTDBRM.

FUTTOKEN

********************************************************************************
* Name: FUTTOKEN (DB2 consistency token descrambler)                          *
* Version: V1.Ø                                                                *
* Author: P. Lenart (Futurex Computing International Limited)                 *
* Purpose: This program is passed the consistency token via:                  *
*          JCL, REXX as a function or program call depending on the           *
*          input parameter.                                                  *
*          It has the DB2 CONTOKEN as input, with the output returned          *
*          as a 26 character timestamp.                                       *
********************************************************************************

* Parameters used:                                                           *
*   R1 - Address of parm list                                               *
*   H, CLØØ - means JCL call - hence output written to SYSPRINT             *
*   F, F    - means inter program call and returned in ADDRESS             *
*   Map    - mapped by EFPL - as a REXX function                            *
* Parameters sent in R15                                                    *
*   Ø - Good return                                                        *
*   >Ø- Errors.                                                            *

© 2003. Reproduction prohibited. Please inform Xephon of any infringement. 9
* Calls: Nothing

* Equates and DSECTs

RØ EQU Ø
R1 EQU 1
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15

* Register redefinitions

BASE1 EQU R12 * Program base register
EFPLPTR EQU R9 * Parameter base
EVALPTR EQU R1Ø * Parameter base
ARGPTR EQU R11 * Parameter base
BALREG EQU R8 * Branch register

* DSECT - for REXX interface

IRXEFPL
IRXEVALB
IRXARGTB

* Control section

FUTTOKEN CSECT
FUTTOKEN AMODE 31 * State we can address in 31bit
FUTTOKEN RMODE 24 * but wish to reside in 24bit
SAVE (14,12) * Save the registers
USING FUTTOKEN,15 * and use a temporary base
BALR 5,Ø * Get the present addressing mode
LTR 5,5 * POS=24, NEG=31
BM STRTØ * Addressing mode okay
LR 0,5 * Save reg5
LA 5,*+10 * Calculate branch displacement
O 5,BIT31 * set 31 bit on
BSM 0,5 * set AMODE=31
LR 5,Ø * restore Reg 5
STRTØ  DS  ØH
  BALR  5, Ø       * Use Reg 5 for base *
STRT1  DS  ØH
  DROP  R15       * Drop the temporary base *
  USING *, 5      * and establish addressability *
  LR  BASE1, 5    * Load base reg from our 'mode' *
  S  BASE1, OFFSET * Establish from start of prog *
  O  BASE1, BIT31 * Set 31 bit on for second base *
  DROP  5         * and drop reg5 *
  USING FUTTOKEN, BASE1
B  START       * Branch around declarations *
DS  ØD        * Eyecatcher on boundary *
XXPROG  DC  CL8 'FUTTOKEN'     * - Name of our program *
  DC  C' '       * - space *
  DC  CL8 'SYSDATE'     * - And date of assemble *
  DC  C' '       * - space *
  DC  CL8 'SYSTIME'     * - And time of assemble *
DS  ØF
OFFSET  DC  A(STRT1-FUTTOKEN) * Offset for base calculation *
FOURK  DC  F' 4096'      * This is what 4K looks like *
BIT31  DC  X' 80000000'  * 31bit constant *
START  DS  ØH
  ST  R13, SAVEA+4  * Store @ of passed savearea *
  LR  R2, R13      * keep R13 *
  LA  R13, SAVEA    * Load @ of our savearea *
  ST  R13, 8(R2)    * and store it in our savearea *
B  C00000
***********************************************************************
* All addressing is now in order - Real code starts here              *
***********************************************************************
* Start of main code                                                  *
***********************************************************************
C00000  DS  ØH
  LR  EFPLPTR, R1       * Save this register *
  USING EFPL, EFPLPTR   * And use DSECT to map it *
***********************************************************************
* Check for format of parameter list to decide how to process         *
***********************************************************************
L  R1, Ø(EFPLPTR)      * Check for REXX call *
LTR  R1, R1           * Is this zero ? *
BZ  C00200            * It's a REXX call *
L  R2, Ø(R1)          * Address pointer *
CLC  CFUTCALL, Ø(R2)  * Check for literal string *
BE  C00100            * Therefore inter program call *
LH  R1, Ø(R1)         * Get length of JCL call *
LTR  R1, R1           * Is this zero ? *
BZ  ERR010            * No so give up *
***********************************************************************
* This is a JCL type call - so address the parm list accordingly      *
***********************************************************************

C00010  EQU *
SR  R1, R1    * Clear register  *
ST  R1, CALLTYPE  * And set call type  *
L   EFPLPTR, Ø(EFPLPTR)  * Address inbound JCL value  *
LA  R2, 2(EFPLPTR)  * Set location pointer  *
LH  R1, Ø(EFPLPTR)  * And the length of it  *
B   C01000  * And call common routine  *

***********************************************************************
* This is a inter program linkage, hence address accordingly          *
***********************************************************************

C00100  EQU *
LA  R0, 1    * Clear register  *
ST  R0, CALLTYPE  * And set call type  *
L   R3, 4(R1)  * Load pointer to argument  *
LA  R2, 2(R3)  * Set location pointer  *
LH  R1, Ø(R3)  * And length of argument  *
B   C01000  * And call common routine  *

***********************************************************************
* This is a REXX function call hence address accordingly              *
***********************************************************************

C00200  EQU *
LA  R1, 2    * Clear register  *
ST  R1, CALLTYPE  * And set call type  *
ICM  ARGPTR, B'1111', EFPLARG  * Address the arguments *
USING ARGTABLE_ENTRY, ARGPTR  * And map them  *
ICM  EVALPTR, B'1111', EFPLEVAL  * Address return area *
L   EVALPTR, Ø(EVALPTR)  * And load the pointer  *
USING EVALBLOCK, EVALPTR  * And map it  *
SPACE
C   ARGPTR, =X'FFFFFFFFFFFFFFFF'  * Null value ? *
BE  ERR010  * Error  *
ICM  R1, B'1111', ARGTABLE_ARGSTRING_LENGTH  * Arg length *
ICM  R2, B'1111', ARGTABLE_ARGSTRING_PTR  * Arg location *

***********************************************************************
* After deciding which call type this is - the following is true      *
*    R1 - is the length of the argument passed                        *
*    R2 - is the actual location of the argument                      *
***********************************************************************

C01000  EQU *
CLC  Ø(L'CTOKENH, R2), CTOKENH  * Is it a HEX token ? *
BE  C01100  * Yes - so call routine *
CLC  Ø(L'CTOKENC, R2), CTOKENC  * Must be CHAR representation *
BNE  ERR010  * If not then error *

***********************************************************************
* Character of hex values - make real HEX string section              *
***********************************************************************

LA  R15, L'CTOKENC+L'WSTOKEN  * Get the length we expect *
CR  R1, R15  * And check it is correct *
BNE  ERR010  * No - invalid parm *
SPACE
LA R2, L'CTOKENC(R2) * Point to actual value *
MVC WSTOKEN, Ø(R2) * Keep the token *
LA R5, WSTOKEN * Address of input variable *
LA R6, WSTOKENH * Address of output variable *
LA R7, L'WSTOKENH * Length of input variable *

C01010 EQU *
ICM R1, B'0001', Ø(R5) * Load first character *
SLL R1, 28 * Remove first word *
SRL R1, 28 * And back *
TM Ø(R5), B'11110000' * Is it numeric ? *
BC 1, C01020 * If yes - then jump over *
AH R1, =H'9' * Otherwise make numeric *

C01020 EQU *
SLL R1, 4 * Treat as first 4 bits of byte *
ICM R2, B'0001', 1(R5) * Load first character *
SLL R2, 28 * Remove first word *
SRL R2, 28 * And back *
TM 1(R5), B'11110000' * Is it numeric *
BC 1, C01030 * If yes then jump over *
AH R2, =H'9' * Otherwise make numeric *

C01030 EQU *
XR R1, R2 * Create the hex byte *
STCM R1, B'0001', Ø(R6) * And save to output *
LA R6, 1(R6) * Increment *
LA R5, 2(R5) * Increment *
BCT R7, C01010 * Iterate *
B C02000 * Call the token deriver *

***********************************************************************
* Is it in hexadecimal format already section *
***********************************************************************
C01100 EQU *
LA R15, L'CTOKENH+L'WSTOKENH * Get the length we expect *
CR R1, R15 * Check it is right length *
BNE ERR010 * Error if invalid *
SPACE
LA R2, L'CTOKENH(R2) * Point to actual value *
MVC WSTOKENH, Ø(R2) * And save as appropriate *
B C02000 * Derive the token *

***********************************************************************
* CONTOKEN to real DB2 timestamp section *
***********************************************************************
C02000 EQU *
LA R5, WSTOKEN * Address the hex value *
SPACE
ICM R4, B'1111', WSTOKENH * Get the token for the shift *
ICM R5, B'1111', WSTOKENH+4 * and second portion *
SLDL R4, 3 * Shift for STCK value *
SPACE
STCM R4, B'1111', TODCOND * Save for conversion *
STCM R5, B'1111', TODCOND+4 * And this portion
SPACE
STCKCONV STCKVAL=TODCOND, CONVVAL=TODAREA, DATETYPE=DDMMYYYY

LA R5, TODAREA
* Address of input variable *
LA R6, CLKDONE
* Address of output variable *
LA R7, 16
* Length of input variable *
BAL BALREG, SHEX2CHR
* And call HEX -> CHAR routine *

MVC MSG010, CSPACE
* Clear output line & write TS *
MVC MSG010V1(4), CCLKDYY
MVC MSG010V1+4(1), =C'.'
MVC MSG010V1+5(2), CCLKDMM
MVC MSG010V1+7(1), =C'.'
MVC MSG010V1+8(2), CCLKDDD
MVC MSG010V1+10(1), =C'.'
MVC MSG010V1+11(2), CCLKTHH
MVC MSG010V1+13(1), =C'.'
MVC MSG010V1+14(2), CCLKTMM
MVC MSG010V1+16(1), =C'.'
MVC MSG010V1+17(2), CCLKTSS
MVC MSG010V1+19(1), =C'.'
MVC MSG010V1+20(6), CCLKTTT

SR R1, R1
* Clear the register *
ST R1, RETCODE
* so setting the return code *
MVC MSGOUT, MSG010
* And populating return message *

******************************************************************************
* Return routine                                                            *
******************************************************************************
C900000 EQU *
L R1, CALLTYPE
* Load the type of call *
SLL R1, 2
* Prepare for branch table *
B BRTAB1(R1)
* And branch for return *

BRTAB1 B C90100
* Return is JCL call *
B C90200
* Return is interprogram *
B C90300
* Return is REXX function *

******************************************************************************
* Return routine for JCL call                                               *
******************************************************************************
C90100 EQU *
OPEN (FUTOUT, OUTPUT), MODE=31
* Open output file *
PUT FUTOUT, MSGOUT
* Write the return string *
CLOSE FUTOUT
* Close file and end *
B C99999
* Finish *

******************************************************************************
* Return routine for Inter program link                                    *
******************************************************************************
C90200 EQU *
L R1, Ø(EFPLPTR)
* Re-address the parm list *
L R2, 8(R1)
* Address the return area *

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.
MVC Ø(L' MSGOUT, R2), MSGOUT * Write the return string *
B C99999 * Finish *

***********************************************************************
* Return to REXX routine *
***********************************************************************
C90300 EQU *
  LA R1, L' MSGOUT * Load the length of return *
  ST R1, EVALBLOCK_EVLLEN * Populate the return block *
  MVC EVALBLOCK_EVDATA(L' MSGOUT), MSGOUT * and Message *
  SR R1, R1 * Override the return code *
  ST R1, RETCODE * And store *

C99999 EQU *
  L R15, RETCODE * Clear return code *
  L R13, SAVEA+4 * Reload save area *
  RETURN (14, 12), RC=(15) * And end *

***********************************************************************
* Subroutines *
***********************************************************************

* HEX to CHARACTER Subroutine *
* expects: R5 - @ of Hex string to convert *
* R6 - @ of Output field *
* R7 - Length of hex string *
***********************************************************************
SHEX2CHR EQU *
  MVC HEXWORK, Ø(R5)
  TR HEXWORK, TRTABLE1
  MVC Ø(1, R6), HEXWORK
  MVC HEXWORK, Ø(R5)
  TR HEXWORK, TRTABLE2
  MVC 1(1, R6), HEXWORK
  LA R5, 1(R5)
  LA R6, 2(R6)
  BCT R7, SHEX2CHR
  BR BALREG

***********************************************************************
* Error routines *
***********************************************************************
ERR010 EQU *
  MVC MSGOUT, MSGEØØ1 * Populate error message *
  LA R15, 4 * Set return code *
  ST R15, RETCODE * Set return code *
  B C900000 * And finish the return process *

***********************************************************************
* Working Storage *
***********************************************************************
DS ØD
SAVEA DS 18A
SAVBAL DS F
RETCODE DS F
CALLTYPE DS F
WSTOKENH DS CL8
WSTOKEN DS CL16
***********************************************************************
* Constants                                                           *
***********************************************************************
CSPACE DC CL133' ' 
CTOKENC DC C'CTOKEN='
CTOKENH DC C'HGRAPH='
CFUTCALL DC CL8'FUTCALL'
***********************************************************************
* Storeclock manipulation fields                                     *
***********************************************************************
DS ØD
TODAREA DS 4F
TODCOND DS D
*
CCLKDONE DS ØCL32
CCLKTHH DS CL2
CCLKTMM DS CL2
CCLKTSS DS CL2
CCLKTTT DS CL6
CCLKTRR DS CL4
*
CCLKDDD DS CL2
CCLKDMM DS CL2
CCLKDYY DS CL4
CCD2 DS CL8
***********************************************************************
* Messages                                                            *
***********************************************************************
MSGOUT DS CL8Ø
MSGO1Ø DC CL8Ø' '
      ORG MSO1Ø
MSGO1ØV1 DS CL26
      ORG
MSGEØØ1 DC CL8Ø'ERROR - expecting CTOKEN= or HTOKEN='
***********************************************************************
* TRANSLATE TABLE HEX - CHAR                                         *
***********************************************************************
TRTABLE1 DC C'0000000000000000'
      DC C'1111111111111111'
      DC C'2222222222222222'
      DC C'3333333333333333'
      DC C'4444444444444444'
      DC C'5555555555555555'
      DC C'6666666666666666'
      DC C'7777777777777777'
      DC C'8888888888888888'
FUTDBRM

***********************************************************************
* Name: FUTDBRM (DBRM examination)                                 *
* Version: V1.0                                                      *
* Author: P. Lenart (Futurex Computing International Limited)       *
*                                                                *
* Purpose: This program is passed the consistency token via:        *
*          JCL, REXX as a function or program call depending on the *
*          input parameter.                                          *
*          It has the dsname(member) as input with the output return *
*          as a 8 byte consistency token, and the derived timestamp  *
*          from FUTTOKEN.                                            *
***********************************************************************

***********************************************************************
* Parameters used:                                                   *
*      R1 - Address of parm list                                      *
*        H, CL8Ø - means JCL call - hence output written to SYSPRINT*
*        F, F    - means inter program call and returned in ADDRESS *
*        Map    - mapped by EFPL - as a REXX function                *
*      Ø - Good return                                               *
*      >Ø- Errors.                                                   *
* Calls: Nothing                                                    *
***********************************************************************

***********************************************************************
* Equates and DSECTs                                                 *
***********************************************************************

RØ   EQU Ø
R1   EQU 1
R2   EQU 2
R3   EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15

***********************************************************************
* Register redefinitions                                              *
***********************************************************************
BASE1 EQU R12                 * Program base register          *
EFPLPTR EQU R9                  * Parameter base                     *
EVALPTR EQU R10                 * Parameter base                     *
ARGPTR EQU R11                 * Parameter base                     *
BALREG EQU R8                  * Branch register                      *

***********************************************************************
DSECT - for REXX interface                                           *

***********************************************************************

FUTDBRM CSECT
FUTDBRM AMODE 31          * State we can address in 31bit     *
FUTDBRM RMODE 24          * but wish to reside in 24bit  *
SAVE (14,12)              * Save the registers             *
USING FUTDBRM,15          * and use a temporary base    *
BALR 5,0                  * Get the present addressing mode *
LTR 5,5                   * POS=24, NEG=31               *
BM STRTØ                  * Addressing mode okay           *
LR 0,5                    * Save reg5                     *
LA 5,*+10                 * Calculate branch displacement *
O 5,BIT31                 * set 31 bit on                 *
BSM 0,5                   * set AMODE=31                 *
LR 5,0                    * restore Reg 5                 *
STRTØ DS ØH               * Use Reg 5 for base           *
STRT1 DS ØH               *
DROP R15                  * Drop the temporary base       *
USING *5                  * and establish addressibility *
LR BASE1,5                * Load base reg from our 'mode' *
S BASE1,OFFSET            * Establish from start of prog  *
O BASE1,BIT31 * Set 31 bit on for second base *
DROP 5 * and drop reg5 *
USING FUTDBRM, BASE1
B START * Branch around declarations *
DS ØD * Eyecatcher on boundary *
XXPROG DC CL8'FUTDBRM' * - Name of our program *
DC C' ' * - space *
DC CL8'&SYSDATE' * - And date of assemble *
DC C' ' * - space *
DC CL8'&SYSTIME' * - And time of assemble *
DS ØF
OFFSET DC A(STRT1-FUTDBRM) * Offset for base calculation *
FOURK DC F'4096' * This is what 4K looks like *
BIT31 DC X'80000000' * 31bit constant *
START DS ØH
ST R13,SAVEA+4 * Store @ of passed savearea *
LR R2,R13 * keep R13 *
LA R13,SAVEA * Load @ of our savearea *
ST R13,8(R2) * and store it in our savearea *
B C000000

------------------------------------------------------------------------
* All addressing is now in order - Real code starts here               *
------------------------------------------------------------------------
* Start of main code                                                   *
------------------------------------------------------------------------
C00000 DS ØH
LR EFPLPTR,R1 * Save this register *
USING EFPL,EFPLPTR * And use DSECT to map it *
------------------------------------------------------------------------
* Check for format of parameter list to decide how to process          *
------------------------------------------------------------------------
L R1,Ø(EFPLPTR) * Check for REXX call *
LTR R1,R1 * Is this zero ? *
BZ C00200 * It's a REXX call *
CLC CFUTCALL,Ø(R1) * Check for literal string *
BE C00100 * Okay - its inter program call *
LH R1,Ø(R1) * Get length of JCL call *
LTR R1,R1 * Is this zero ? *
BZ ERR010 * No so give up *

------------------------------------------------------------------------
* This is a JCL type call - so address the parm list accordingly        *
------------------------------------------------------------------------
C0010 EQU *
SR R1,R1 * Clear register *
ST R1,CALCTYPE * And set call type *
L EFPLPTR,Ø(EFPLPTR) * Address inbound JCL value *
LA R2,2(EFPLPTR) * Set location pointer *
LH R1,Ø(EFPLPTR) * And the length of it *
B C01000 * And call common routine *
* This is an inter program linkage, hence address accordingly *

**C0010** EQU *
  LA R0,1  * Clear register *
  ST R0,CALLTYPE  * And set call type *
  L R3,4(R1)  * Load pointer to argument *
  LA R2,2(R3)  * Set location pointer *
  LH R1,Ø(R3)  * And length of argument *
  B C01000  * And call common routine *

***********************************************************************

* This is a REXX function call hence address accordingly *

**C0020** EQU *
  LA R1,2  * Clear register *
  ST R1,CALLTYPE  * And set call type *
  ICM ARGPTR,B'1111',EFPLARG  * Address the arguments *
  USING ARGTABLE_ENTRY,ARGPTR  * And map them *
  ICM EVALPTR,B'1111',EFPLEVAL  * Address return area *
  L EVALPTR,Ø(EVALPTR)  * And load the pointer *
  USING EVALBLOCK,EVALPTR  * And map it *

C ARGPTR,=X'FFFFFFFFFFFFFFFF'  * Null value ? *
BE ERR010  * Error *
ICM R1,B'1111',ARGTABLE_ARGSTRING_LENGTH  * Arg length *
ICM R2,B'1111',ARGTABLE_ARGSTRING_PTR  * Arg location *

***********************************************************************

* Common routine for parm manipulation. We now know: *
* R1 - points to length of argument *
* R2 - points to the actual argument *

**C0100** EQU *
  MVC S99MEM,CSPACE  * Clear the member name *
  MVC S99DSN,CSPACE  * Clear the dataset name *
  LA R3,99DSN  * Load output address *
  LA R4,L'99DSN(R3)  * And the maximum length *

**C0101** EQU *
  CLC Ø(1,R2),=C'('  * End of PDS name ? *
  BE C01020  * Yes - look for member *
  CLC Ø(1,R2),CSPACE  * End of DSNAM ? *
  BE ERR010  * Yes - no member error *
  MVC Ø(1,R3),Ø(R2)  * Move byte *
  LA R2,1(R2)  * Increment *
  LA R3,1(R3)  * Increment *
  CR R3,R4  * Check we are not at max *
  BH C01015  * Jump out if we are *
  BCT R1,C01010  * And iterate *

**C0105** EQU *
  CLC Ø(1,R2),=C'('  * Max length routine *
  BNE ERR010  * No - must be > 44 bytes *
LA R1,44 * Otherwise assume 44 bytes*
STCM R1,B'ØØ11',S99DSNL * And store as length*
B C01025 * Look for member*
SPACE

C01020 EQU *
LA R4,S99DSN * Derive the DSNAME start point*
SR R3,R4 * And subtract end address*
STCM R3,B'ØØ11',S99DSNL * Hence store the length*
SPACE

C01025 EQU *
LA R2,1(R2) * Past the (*
BCTR R1,RØ * Exclude the ( for length*
LA R3,S99MEM * Load output address*
LA R4,L'S99MEM(R3) * And maximum length*
SPACE

C01030 EQU *
CLC Ø(1,R2),=C')' * End of member ?*
BE C01040 * Yes*
CLC Ø(1,R2),CSPACE * No ending bracket ?*
BE C01040 * Yes*
MVC Ø(1,R3),Ø(R2) * Move byte*
LA R2,1(R2) * Increment*
LA R3,1(R3) * Increment*
CR R3,R4 * Check we are not at max*
BH C01040 * Jump out if we are*
BCT R1,C01030 * And iterate*

C01040 EQU *
LA R1,S99MEM * Load address of start*
SR R3,R1 * And subtract end address*
STCM R3,B'ØØ11',S99MEML * Hence giving length of member*
***********************************************************************
* Allocate the library to check for the member*
***********************************************************************

C01100 EQU *
OI S99RBPTR,X'8Ø' * Set end of list indicator*
OI S99TUP3,X'8Ø' * And here also*
LA R4,6 * Length of DDNAME*
STCM R4,B'ØØ11',S99DDNL * And save*
MVC S99DDN,=C'INDCBM' * Give name of DDNAME*
LA R4,1 * 1 - Allocate*
STCM R4,B'0011',S99RBVRB * And set as verb for SVC99*
LA R1,S99RBPTR * Address the parameter list*
SVC 99 * And issue the SVC call*
SR R1,R1 * Set pointer for error array*
LTR R15,R15 * Test for good call*
BNZ ESV99 * Error*
***********************************************************************
* Does the member exist ?*
***********************************************************************
OPEN INDCBM,MODE=31 * Open the allocated dataset*
FIND INDCBM,S99MEM,D * And find the member*

LR R5, R15 * Save return code *
CLOSE INDCB * Close the ddname *
OI S99TUP2, X'80' * Set end of list indicator *
LA R4, 2 * 2 - Deallocate *
STCM R4, B'001', S99RBVRB * And set as verb for SVC99 *
LA R1, S99RBPTR * Address the parameter list *
SVC 99 * And issue the SVC call *
LA R1, 2 * Set pointer for error array *
LTR R15, R15 * Test for good call *
BNZ ESVC99 * Error *
SPACE

LR R5, R5 * Test the member find return code *
BNZ ERR040 * No member found *

***********************************************************************
* Allocate for member retrieval                                       *
***********************************************************************
NI S99TUP2, X'7F' * Set continue flag *
NI S99TUP3, X'7F' * Set continue flag *
OI S99TUP4, X'80' * Set end of list indicator *
LA R4, 5 * Load length of DDNAME *
STCM R4, B'001', S99DDNL * Store *
MVC S99DDN, =C'INDCB' * And populate *
LA R4, 1 * 1 - Allocate *
STCM R4, B'001', S99RBVRB * And set as verb for SVC99 *
LA R1, S99RBPTR * Address the parameter list *
SVC 99 * And issue the SVC call *
LA R1, 1 * Set pointer for error array *
LTR R15, R15 * Test for good call *
BNZ ESVC99 * Error *
SPACE

OPEN INDCB * Open for member record retrieval *
GET INDCB, INREC * Read the DBRM header record *
LA R5, INREC * Load address of input *
USING DBRMHEAD, R5 * And use the DSECT to map it *
SPACE

CLC DBRMHID, CDBRM * Check it's a DBRM module *
BNE ERR020 * No eyecatcher - so error *
MVC TOKENH, DBRMTIMS * Save consistency token (internal) *
DROP R5 * Drop the DSECT *

***********************************************************************
* Deallocate                                                          *
***********************************************************************
C012ØØ EQU *
CLOSE INDCB * Close the ddname *
LA R4, 2 * 2 - Deallocate *
STCM R4, B'001', S99RBVRB * And set as verb for SVC99 *
LA R1, S99RBPTR * Address the parameter list *
OI S99TUP2, X'80' * Set end of list indicator *
SVC 99 * And issue the SVC call *
LA R1, 1 * Set error array pointer *

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.
LTR R15,R15  * Test for good call  *
BNZ ESVC99  * Error  *

* Call FUTTOKEN to translate the TOKEN *

LA R1,FUTPARMS  * Load parameter list  *
call FUTTOKEN  * Can statically call routine  *
ST R15,RETCODE  * Save return code  *
LTR R15,R15  * Good return ?  *
BNZ ERRØ3Ø  * No error  *

* Format return string ie CONTOKEN <SPACE> Timestamp representation *

SR R1,R1  * Clear the register  *
ST R1,RETCODE  * so setting the return code  *
MVC MSGØ1Ø,CSPACE  * Clear the message area  *
MVC MSGØ1ØV1,TOKENH  * Populate the CONTOKEN  *
MVC MSGØ1ØV2,INREC  * And the timestamp returned  *
MVC MSGOUT,MSGØ1Ø  * And populate return area  *

* Return routine *

C9ØØØØ EQU *
L R1,CALLTYPE  * Load the type of call  *
SLL R1,2  * Prepare for branch table  *
B BRTAB1(R1)  * And branch for return  *

BRTAB1 B C9Ø1ØØ  * Return is JCL call  *
B C9Ø2ØØ  * Return is interprogram  *
B C9Ø3ØØ  * Return is REXX function  *

* Return routine for JCL call *

C9Ø1ØØ EQU *
OPEN (FUTOUT, OUTPUT), MODE=31  * Open JCL dd card  *
PUT FUTOUT, MSGOUT  * Write the message  *
CLOSE FUTOUT  * Close  *
B C99999  * And end  *

* Return routine for Inter program link *

C9Ø2ØØ EQU *
L R1,Ø(EFPLPTR)  * Re-address the parm list  *
L R2,8(R1)  * Address the return area  *
MVC Ø(L' MSGOUT, R2), MSGOUT  * Write the return string  *
B C99999  * Finish  *

* Return to REXX routine *

C9Ø3ØØ EQU *
LA R1, L' MSGOUT  * Load the length of return  *
ST R1, EVALBLOCK_EVLEN * Populate the return block *
MVC EVALBLOCK_EVDATA(L'MSGOUT), MSGOUT * and Message *
SR R1, R1 * Override the return code *
ST R1, RETCODE * And store *
C99999 EQU *
L R15, RETCODE * Clear return code *
L R13, SAVEA+4 * Reload save area *
RETURN (14, 12), RC=(15) * And end *

*****************************************************************************
* Subroutines                                                           *
*****************************************************************************
*****************************************************************************
* Error routines                                                       *
*****************************************************************************
*****************************************************************************
ERRØ1Ø EQU *
MVC MSGOUT, MSGØØ1 * Populate error message *
LA R15, 4 * Set return code *
ST R15, RETCODE * And save *
B C9ØØØ0 * And end *
ERRØ2Ø EQU *
MVC MSGOUT, MSGØØ4 * Populate error message *
LA R15, 4 * Set return code *
ST R15, RETCODE * And save *
B C9ØØØ0 * And end *
ERRØ3Ø EQU *
MVC MSGOUT, MSGØØ5 * Populate error message *
LA R15, 4 * Set return code *
ST R15, RETCODE * And save *
B C9ØØØ0 * And end *
ERRØ4Ø EQU *
MVC MSGOUT, MSGØØ3 *
LA R15, 4 * Set return code *
ST R15, RETCODE * Set return code *
B C9ØØØ0 *

******************************************************************************
* SVC99 error routines                                                 *
******************************************************************************
******************************************************************************
ESVC99 EQU *
MVC MSGOUT, MSGØØ2 * Set header message *
SLL R1, 2 * Adjust error array pointer *
B BRTAB2(R1) * To branch into table *
BRTAB2 B ESVC991 * 0 - allocate error *
B ESVC992 * 4 - deallocate error *
B ESVC993 * 8 - Member allocate error *
ESVC991 EQU *
MVC MSGØØ2V, MSGØØ21 * Set error descriptor *
B ESVC999 * And end *
ESVC992 EQU *
MVC MSGØØ2V, MSGØØ22 * Set error descriptor *
B ESVC999 * And end *
ESVC993 EQU *

MVC MSGEØØ2V, MSGEØØ23 * Set error descriptor *
B ESVC999 * And end *

ESVC999 EQU *

LA R15,4 * Set return code *
ST R15,RETCODE * And store *
B C9ØØØØ * Finish *

***********************************************************************

* Working Storage *
***********************************************************************

DS ØD
SAVEA DS 18A
SAVBAL DS F
RETCODE DS F
CALLTYPE DS F

***********************************************************************

* SVC99 parms *
***********************************************************************

S99RBPTR DC A(S99RB) * Address of SVC99 request block *

S99RB DS ØF * SVC99 request block *
DC AL1(20) * Length of request block *

S99RKBVRB DS AL1 * SVC99 - verb *
S99RBFLG DS AL2 * SVC99 - flags *
S99RBERRC DS AL2 * SVC99 - error code *
S99RBINF DS AL2 * SVC99 - info code *
S99RBTUP DC A(S99TUPL) * SVC99 - pointer to text units *

DS AL4 * SVC99 - Extension Block *

S99RBFL2 DS AL4 * SVC99 - APF only flags *

S99TUPL DS ØF * SVC99 - Text pointer list *
S99TU1 DC A(S99TU1) * SVC99 - Pointer to DDNAME *
S99TU2 DC A(S99TU2) * SVC99 - Pointer to DSNAME *
S99TU3 DC A(S99TU3) * SVC99 - Pointer to DISPOSITION *
S99TU4 DC A(S99TU4) * SVC99 - Pointer to MEMBER *

S99DDNL DS AL2 * SVC99 - DDNAME descriptor *
DC AL2(1) * SVC99 - Key 1 *
DC AL2(1) * SVC99 - Number of entries *

S99DDN DS CL8 * SVC99 - DDNAME *

S99TU2 DS ØF * SVC99 - DSNAME descriptor *
DC AL2(2) * SVC99 - Key 2 *
DC AL2(1) * SVC99 - Number of entries *

S99DSNL DS AL2 * SVC99 - Length of DSNAME *
S99DSN DS CL44 * SVC99 - DSNAME *

S99TU3 DS ØF * SVC99 - MEMBER descriptor *
DC AL2(4) * SVC99 - Key 4 *
DC AL2(1) * SVC99 - Number of entries *
DC AL2(1) * SVC99 - Length of DISP *
DC  AL1(8)  *  SVC99  -  DISP=SHR  *
S99TU4  DS  0F  *  SVC99  -  Disposition descriptor  *
DC  AL2(3)  *  SVC99  -  Key 3  *
DC  AL2(1)  *  SVC99  -  Number of entries  *
S99MEML  DS  AL2  *  SVC99  -  Length of MEMBER  *
S99MEM  DS  CL8  *  SVC99  -  MEMBER  *
***********************************************************************
* Constants                                                          *
***********************************************************************
CSPACE   DC    CL133' '
CFUTCALL DC    CL8'FUTCALL'
CDBRM    DC    CL4'DBRM'
***********************************************************************
* Interprogram linkage                                                *
***********************************************************************
FUTPARMS DC    AL4(WLPARM1)
WLPARM1  DC    AL4(CFUTCALL)
WLPARM2  DC    AL4(TOKENP)
WLPARM3  DC    AL4(INREC)
DS  ØH
TOKENP  DC    AL2(L'TOKENP1+L'TOKENH)
TOKENP1  DC    C'HTOKEN='
TOKENH   DS    CL8
DS  ØH
INREC   DS    CL8Ø
***********************************************************************
* Messages                                                            *
***********************************************************************
MSGOUT  DS    CL8Ø
MSG010  DC    CL8Ø'
  ORG  MSG010
MSG010V1 DS    CL8
  DS  C
MSG010V2 DS    CL26
  ORG  MSGOUT+25
MSGE002V DS    CL20
  ORG
MSGE001 DC    CL8Ø'ERROR - expecting dsname(member)'
MSGE002 DC    CL8Ø'ERROR - SVC99 error for'
MSGE0021 DC    CL20'Allocate Dsname'
MSGE0022 DC    CL20'Deallocate'
MSGE0023 DC    CL20'Allocate Member'
MSGE003 DC    CL8Ø'ERROR - SVC99 cannot find the member'
MSGE004 DC    CL8Ø'ERROR - Not a DBRM'
MSGE005 DC    CL8Ø'ERROR - Bad return from FUTTOKEN'
***********************************************************************
* DCB and literal pool                                                *
***********************************************************************
FUTOUT  DCB  DSORG=PS, RECFM=FB, LRECL=80, DDNAME=SYSOUT, MACRF=PM, x DCBE=FUTOUTE
DB2 plan and package query

The following REXXX programs give you some useful information regarding DB2 plans, packages, tables, and relationships between each other. All programs use DB2 REXXX support. Before executing the programs, you should create two indexes to increase the performance of queries on catalog tables. Panel definitions, programs, DDL statements, and sample output are shown below.

PDCQ PANEL

+ type(text)   intens(low)   color(yellow)   hilite(reverse)
? type(text)   intens(low)   color(yellow)   hilite(reverse)
# type(input)  intens(high)  color(yellow)   pad(#)
[ type(input)  intens(low)   color(yellow)]
] type(output) intens(high)  color(pink)

+ Option :#o +
+ Ø1. List of packages for a plan
+ Ø2. List of plans for a package
Ø3. List of tables which are used by a package
Ø4. List of packages which are used by a table

PF3 : return

PROC
  VER(&o, nonblank, range, Ø1, Ø9)
END

PDCQ001 PANEL

PROC
  ATTR
    £ TYPE(TEXT) INTENS(HIGH) HILITE(REVERSE)
    + TYPE(TEXT) INTENS(HIGH) SKIP(ON)
    # TYPE(INPUT) INTENS(LOW) COLOR(YELLOW) PAD(#)
    Q TYPE(OUTPUT) INTENS(HIGH) COLOR(YELLOW)
    $ TYPE(OUTPUT) INTENS(HIGH) COLOR(YELLOW) HILITE(REVERSE)
  BODY
    £ List of tables which are used by a package +
    +
    + DB2 subsystem id.....: #d+ D, T, E, V, R, W, P
    +
    + Collection id.......: #colname +
    +
    + Package name........: #pckname +
    +
    +
    +
    +
    +
    +
    +
MSG : $msg
PF3 : Return
init
proc
    if (&wsys = 'TX')
        ver(&d, nonblank, list, D, T, E, V, R, G)
    if (&wsys = 'PX')
        ver(&d, nonblank, list, P)
    if (&wsys = 'PW')
        ver(&d, nonblank, list, W)
END

PDCQ001A PANELS

)panel
)attr
% type(text) intens(high)
£ type(text) intens(high) hilite(reverse)
+ type(text) intens(low) skip(on)
# type(input) intens(high) caps(on) just(left)
] type(output) intens(high) color(blue)
" type(input) intens(low) pad(#)
* type(output) intens(low) color(yellow)
)body expand(/)
% £ List of tables which are used by a package +
+ "r+
+ Dbid: *d+ Coll. id : *colname + Package name : *pckname +
+ Total number of sql stmt: *st +
+ Commit (yes - no) : *wcom+
+ % Creator Table name Select Insert Delete Update Declare
% ========= =========== = ====== ====== ====== ====== ====== ====== ====== ====== ======
)model
]tbtext
)init
)end

PDCQ002 PANELS

)panel
)attr
£ type(text) intens(high) hilite(reverse)
+ type(text) intens(high) skip(on)
# type(input) intens(low) color(yellow) pad(#)
Q type(output) intens(high) color(yellow)
$ type(output) intens(high) color(yellow) hilite(reverse)
List of packages which are used by a table

DB2 subsystem id....: #d+ D, T, E, V, R, W, P

Table creator.......: #creator +

Table name..........: #tbname +

MSG : $msg +
PF3 : Return
init
proc
if (&wsys = 'TX')
   ver(&d, nonblank, list, D, T, E, V, R, G)
if (&wsys = 'PX')
   ver(&d, nonblank, list, P)
if (&wsys = 'PW')
   ver(&d, nonblank, list, W)

PDCQ002A PANELS

panel
attr
% type(text) intens(high)
£ type(text) intens(high) hilite(reverse)
+ type(text) intens(low) skip(on)
# type(input) intens(high) caps(on) just(left)
] type(output) intens(high) color(blue)
* type(input) intens(low) pad(#)
* type(output) intens(low) color(yellow)
) body expand(/)
%
   List of packages which are used by tables
+ "r+
+ Dbid: *d+ Creator : *creator + Table name: *tbname +
+ Total number of sql stmt: *st1 +
% coll.id Package Select Insert Delete Update Declare Commit
% ================== ======== ====== ====== ====== ====== ======= ======
)model
+]tbtext
+
)init
)end

PDCQ006 PANELS

)PANEL
)ATTR
£ TYPE(TEXT) INTENS(HIGH) HILITE(REVERSE)
+ TYPE(TEXT) INTENS(HIGH) SKIP(ON)
# TYPE(INPUT) INTENS(LOW) COLOR(YELLOW) PAD(#)
Q TYPE(OUTPUT) INTENS(HIGH) COLOR(YELLOW)
$ TYPE(OUTPUT) INTENS(HIGH) COLOR(YELLOW) HILITE(REVERSE)
)BODY
+
+ £ List all packages for a plan + User : Qwuser
+
+ DB2 subsystem id....: #d+ D, T, V, E, V, W, P
+
+ Plan name............: #plnname +
+
+ MSG : $msg +
+ PF3 : RETURN
)INIT
)proc
IF (&wsys = 'TX')
  VER(&d, nonblank, list, D, T, E, V)
IF (&wsys = 'PX')
  VER(&d, nonblank, list, P)
IF (&wsys = 'PW')
  VER(&d, nonblank, list, W)
} END

PDCQ006A PANELS
)
| panel             |
| attr             |
| % type(text) intens(high) |
| + type(text) intens(low) skip(on) |
| # type(input) intens(high) caps(on) just(left) |
| ] type(output) intens(high) color(blue) |
| * type(input) intens(low) pad(#) |
| * type(output) intens(low) color(red) |
| body expand(/) |
| % List all packages for a plan |
| + |
| + DBID: *d+ Plan name : *plname + #r+ |
| + |
| % collection id package name |
| % =================== ============ |
| ) model |
| + |collid + |packnam + |
| ) init |
| .cursor=r |
| ) end |

PDCQ007 PANELS
)
| PANEL             |
| ATTR             |
| £ TYPE(TEXT) INTENS(HIGH) HILITE(REVERSE) |
| + TYPE(TEXT) INTENS(HIGH) SKIP(ON) |
| # TYPE(INPUT) INTENS(LOW) COLOR(YELLOW) PAD(#) |
| Q TYPE(OUTPUT) INTENS(HIGH) COLOR(YELLOW) |
| $ TYPE(OUTPUT) INTENS(HIGH) COLOR(YELLOW) HILITE(REVERSE) |
| BODY             |
| + £ List all plans for a package + User : Qwuser |
| + |
| + DB2 subsystem id....: #d+ D, T, V, E, V, W, P |
| + |
+   Package name........: #pckname +
+   +
+   +
+   +
+   +
+   +
+   +
+   +
+   +
+   +
+   +
+   PF3 : RETURN
+INIT
)proc
   IF (&wsys = 'TX')
      VER(&d, nonblank, list, D, T, E, V)
   IF (&wsys = 'PX')
      VER(&d, nonblank, list, P)
   IF (&wsys = 'PW')
      VER(&d, nonblank, list, W)
)END

PDCQ007A PANELS
)
panel
)attr
% type(text) intens(high)
+ type(text) intens(low) skip(on)
# type(input) intens(high) caps(on) just(left)
] type(output) intens(high) color(blue)
" type(input) intens(low) pad(#)
* type(output) intens(low) color(red)
)body expand(/)
% List all plans for a package
+ "r+
+ DBID: *d+ package name : *pckname +
+  % Plan names for the package
% ================
)model
  ]plannam +
)init
)end
RDCQ (MENU PROGRAM)

/* rexx                          */
/*/                         */
//** main                        */
//**                            */
$ispexec libdef ispplib dataset id('xxxxxxxx.pd0.dcq')$
start:
o = ' ';
$ispexec display panel(pdcq)$
if rc ¬= Ø then return
select
  when(o = '1') then $ex 'sk0psyp.pd0.dcq(rdcq006)'$
  when(o = '2') then $ex 'sk0psyp.pd0.dcq(rdcq007)'$
  when(o = '3') then $ex 'sk0psyp.pd0.dcq(rdcq001)'$
  when(o = '4') then $ex 'sk0psyp.pd0.dcq(rdcq002)'$
  otherwise nop
end
signal start

RDCQ001 REXX

/* rexx                          */
/*/                             */
//** main                         */
//**                              */
$ispexec libdef ispplib dataset id('xxxxxxxx.pd0.dcq')$

wuser = sysvar(sysuid)
wsys = mvsvar('SYSNAME')
wstart:
  address tso
  r = ' ';
  $ispexec display panel(pdcq001)$
  if rc ¬= Ø then return
  ssid = 'DB' || d || '0'
  ws_pack_name = pckname
  wrc = '0'
  msg = '
  call select_pack_dep
  if wrc = '1' then signal wstart
  call select_pack_stmt
  address ispexec
  call create_table
  $tbtop wtb$
  $tbdisp wtb panel(pdcq001a)$
  if rc ¬= Ø then do
    $tbend wtb$
    signal wstart
$tbend wtb$
signal wstart

/***********************************************************/
/*       find tables which are used by given package      */
/***********************************************************/
select_pack_dep:
  'subcom dsnrexx'
if rc then
  s_rc = rxsubcom('ADD', 'DSNREXX', 'DSNREXX')
address dsnrexx
connect ssid
if rc ≠ 0 then return
sqlstmt =,
  'select bqualifier, bname ',
  'from sysibm.syspackdep where dname = ',
  '''' || pckname || '''',
  ' and dcollid = ',
  '''' || colname || '''',
  ' and btype = ' || 'T',
  ' order by 1, 2 ' 'execsql declare c1 cursor for s1'
if sqlcode ≠ 0 then call sql_error
'execsql prepare s1 into :outsqlda from :sqlstmt'
if sqlcode ≠ 0 then call sql_error
'execsql open c1'
if sqlcode ≠ 0 then call sql_error
'execsql fetch c1 using descriptor :outsqlda'
if sqlcode < 0 then call sql_error
if sqlcode = 100 then do
  msg = 'Package or collection id are not valid'
  wrc = '1'
  return
end
tb = 0
do while(sqlcode = 0)
  tb = tb + 1
  wtbcre.tb = outsqlda.1.sqldata
  wtbname.tb = outsqlda.2.sqldata
  'execsql fetch c1 using descriptor :outsqlda'
end
if sqlcode < 0 then call sql_error
'execsql close c1'
if sqlcode < 0 then call sql_error
return

/***********************************************************/
/*       find sql statements which are used by the package */
/***********************************************************/
select_pack_stmt:
  'subcom dsnrexx'
if rc then
s_rc = rxsubcom('ADD', 'DSNREXX', 'DSNREXX')
address dsnrexx
connect ssid
if rc ≠ Ø then return
sqlstmt =
'select seqno, stmnto, stmnt ' ,
'from sysibm.syspackstmt where collid = ',
'''' || pckname || '''' ,
' and name = '''' || pckname || '''' ,
' order by seqno, stmnto ' ,
'execsql declare c1 cursor for s1'
if sqlcode ≠ Ø then call sql_error
'execsql prepare s1 into :outsqlda from :sqlstmt'
if sqlcode ≠ Ø then call sql_error
'execsql open C1'
if sqlcode ≠ Ø then call sql_error
'execsql fetch c1 using descriptor :outsqlda'
if sqlcode ≠ Ø then call sql_error
wstno1 = outsqlda.2.sqldata
st = 1
wstmt.st = ''
do while(sqlcode = Ø)
  wstno = outsqlda.2.sqldata
  if wstno1 ≠ wstno then do
    st = st + 1
    wstno1 = outsqlda.2.sqldata
    wstmt.st = ''
  end
  wstmt.st = wstmt.st || outsqlda.3.sqldata
  'execsql fetch c1 using descriptor :outsqlda'
end
if sqlcode < Ø then call sql_error
'execsql close c1'
if sqlcode < Ø then call sql_error
s_rc = rxsubcom('DELETE', 'DSNREXX', 'DSNREXX')
wcom = 'NO'
do tt = 1 to tb
  wtps.tt = ''
  wtpi.tt = ''
  wtpd.tt = ''
  wtpu.tt = ''
  wtpc.tt = ''
do ss = 1 to st
  if word(substr(wstmt.ss, 9, 15), 1) = ' COMMIT',
    & wcom = 'NO' then wcom = 'YES'
  ff = find(wstmt.ss, wtbname.tt)
  if ff > Ø then do
    stmtx = word(substr(wstmt.ss, 9, 15), 1)
`select
   when(stmtx = 'SELECT')  then wtps.tt = 'X'
   when(stmtx = 'INSERT')  then wtpi.tt = 'X'
   when(stmtx = 'DELETE')  then wtpd.tt = 'X'
   when(stmtx = 'UPDATE')  then wtpu.tt = 'X'
   when(stmtx = 'DECLARE') then wtpc.tt = 'X'
   otherwise
end end end return
`
wsys = mvsvar('SYSNAME')

wstart:
  address tso
  r = ''
  $ispexec display panel(pdcq002)$
  if rc ≠ 0 then return
  ssid = 'DB' || d || '0'
  ws_pack_name = pckname
  wrc = '0'
  msg = ''
  call select_pack_stmt
  if wrc = '1' then signal wstart
  address ispexec
  call create_table
  $tbtop  wtb$
  $tbdispl  wtb panel(pdcq002a)$
  if rc ≠ 0 then do
    $tbend  wtb$
    signal wstart
  end
  $tbend  wtb$
  signal wstart

/***************************************************************/
/*       find sql statements which are used by the package    */
/***************************************************************/
select_pack_stmt:
  'subcom dsnrexx'
  if rc then
    s_rc = rxsubcom('ADD','DSNREXX','DSNREXX')
    address dsnrexx
    connect ssid
    if rc ≠ 0 then return
  sqlstmt=,
  'select a.dcollid,a.dname,','b.seqno,b.stmtno,b.stmt ','
  'from sysibm.syspackdep a,','sysibm.syspackstmt b','
  'where a.bqualifier = ' || '''' || creator || '''','
  'and a.bname = ' || '''' || tbname  || '''','
  'and a.dcollid = b.collid ','
  'and a.dname = b.name ','
  'and stmtno > 0','
  'order by 1,2,3,4 '
  'execsql declare c1 cursor for s1'
  if sqlcode ≠ 0 then call sql_error
  'execsql prepare s1 into :outsqlda from :sqlstmt'
  if sqlcode ≠ 0 then call sql_error
  'execsql open C1'
  if sqlcode ≠ 0 then call sql_error
  'execsql fetch c1 using descriptor :outsqlda'
if sqlcode < 0 then call sql_error
if sqlcode = 100
   then do
      msg = 'Table does not exist or has no valid package'
      wrc = '1'
      return
   end
wcol1 = outsqlda.1.sql data
wpac1 = outsqlda.2.sql data
wstn1 = outsqlda.4.sql data
pg = 1
wcol1id.pg = outsqlda.1.sql data
wpcname.pg = outsqlda.2.sql data
st = 1
wstmt.st = ''
st1 = Ø
   do while(sqlcode = Ø)
      if wcol1 ¬= outsqlda.1.sql data
         then do
            call prep_array
            pg = pg + 1
            st1 = st1 + st
            st = 1
            wcol1id.pg = outsqlda.1.sql data
            wpcname.pg = outsqlda.2.sql data
            wcol1 = outsqlda.1.sql data
            wpac1 = outsqlda.2.sql data
            wstn1 = outsqlda.4.sql data
            wstmt.st = ''
         end
      if wpac1 ¬= outsqlda.2.sql data
         then do
            call prep_array
            pg = pg + 1
            st1 = st1 + st
            st = 1
            wcol1id.pg = outsqlda.1.sql data
            wpcname.pg = outsqlda.2.sql data
            wcol1 = outsqlda.1.sql data
            wpac1 = outsqlda.2.sql data
            wstn1 = outsqlda.4.sql data
            wstmt.st = ''
         end
      if wstn1 ¬= outsqlda.4.sql data
         then do
            st = st + 1
            wstno1 = outsqlda.4.sql data
            wstmt.st = ''
         end
      wstmt.st = wstmt.st || outsqlda.5.sql data
'execsql fetch c1 using descriptor :outsqlda'
end
st1 = st1 + st
call prep_array
s_rc = rxsubcom('DELETE','DSNREXX','DSNREXX')
return
/*******************************************************************************/
/* prep_array */
/*******************************************************************************/
prep_array:
  wtps.pg = ' ';
  wtpi.pg = ' ';
  wtpd.pg = ' ';
  wtpu.pg = ' ';
  wtpc.pg = ' ';
  wtpo.pg = ' ';
do ss = 1 to st
  ff = find(wstmt.ss,tbname)
  if ff > Ø then do
    stmtx = word(substr(wstmt.ss,9,15),1)
    select
      when(stmtx = 'SELECT') then wtps.pg = 'X'
      when(stmtx = 'INSERT') then wtpi.pg = 'X'
      when(stmtx = 'DELETE') then wtpd.pg = 'X'
      when(stmtx = 'UPDATE') then wtpu.pg = 'X'
      when(stmtx = 'DECLARE') then wtpc.pg = 'X'
      when(stmtx = 'COMMIT') then wtpo.pg = 'X'
    otherwise
    end
  end
return
/*******************************************************************************/
/* create table */
/*******************************************************************************/
create_table:
  $tbcreate  wtb names(tbtext)
  nowrite $
do pp = 1 to pg
  tbtext = substr(wcollid.pp,1,18)
  tbtext = tbtext || ' ' || substr(wpcname.pp,1,8)
  tbtext = tbtext || ' ' || wtps.pp
  tbtext = tbtext || ' ' || wtpi.pp
  tbtext = tbtext || ' ' || wtpd.pp
  tbtext = tbtext || ' ' || wtpu.pp
  tbtext = tbtext || ' ' || wtpc.pp
  tbtext = tbtext || ' ' || wtpo.pp
  $tbadd  wtb save(tbtext)$
end
return
/** sql_error */
sql_error:
    say 'SQL error.............'
say sqlcode sqlda
say sqlstmt
    'execsql close c1'
exit

RDCQ006

/*******************************************************************************/
/* main */
/*******************************************************************************/
bndpln:
   $ispexec libdef isplib dataset id('xxxxxxxx.pdØ.dcq')$
   wuser = sysvar(sysuid)
   wsys = mvsvar('SYSNAME')
   wstart:
      address tso
      hsts = Ø
      ix = Ø
      r ='
      $ispexec display panel(pdcqØØ6)$
      if rc ¬= Ø then return
      dbid = d
      ssid = 'db' || d || 'Ø'
      ws_plan_name = plnname
      wrc = Ø
      call select_plan
      if wrc > Ø then return
      address ispexec
      call create_table
      $tbtop wdb2plan$
      $tbdispl wdb2plan panel(pdcqØØ6a)$
      $tbend wdb2plan$
      signal wstart
/*******************************************************************************/
/* prepare sql statement and execute */
/*******************************************************************************/
select_plan:
   hsts = Ø
   'subcom dsnrexx'
   if rc then
      s_rc = rxsubcom('ADD','DSNREXX','DSNREXX')
      address dsnrexx
      connect ssid
if rc ≠ Ø then return
sqlstmt =
' select collid, name ' ' from sysibm.syspacklist',
' where planname = ' || ws_plan_name || ' ' order by collid, name ' 'execsql declare c1 cursor for s1'
if sqlcode ≠ Ø then call sql_error
'execsql prepare s1 into :outsqlda from :sqlstmt'
if sqlcode ≠ Ø then call sql_error
'execsql open c1'
if sqlcode ≠ Ø then call sql_error
'execsql fetch c1 using descriptor :outsqlda'
if sqlcode < Ø then call sql_error
  do while(sqlcode = Ø)
    hsts = hsts + 1
    cname.hsts = outsqlda.1.sqldata
    pname.hsts = outsqlda.2.sqldata
    'execsql fetch c1 using descriptor :outsqlda'
  end
if sqlcode < Ø then call sql_error
'execsql close c1'
if sqlcode < Ø then call sql_error
return
/***************************************************************/
/* create table                                               */
/***************************************************************/
create_table:
$tbcreate wdb2plan names(packnam)
nowrite $
do i = 1 to hsts
collid = cname.i
packnam = pname.i
$tbadd wdb2plan save(collid packnam)$
end
return
/***************************************************************/
/*/ sql_error                                                  */
/***************************************************************/
sql_error:
  say 'SQL error.............'
say sqlcode sqlda
say sqlstmt
'execsql close c1'
exit

RDCQ007
/* rexx */
fndpln:

$ispexec libdef ispllib dataset id('xxxxxxxx.pdO.dcq')$

wuser = sysvar(sysuid)

wsys = mvsvar('SYSNAME')

wstart:

  address tso
  hsts = Ø
  r = '

$ispexec display panel(pdcqØØ7)$

if rc ¬= Ø then return

ssid = 'db' || d || 'Ø'

ws_pack_name = word(pckname,1)

call select_plan

address ispexec

call create_table

$tbtop wdb2plan$

$tbdispl wdb2plan panel(pdcqØØ7a)$

$tbend wdb2plan$

signal wstart

/**************************************************************/
/*       prepare and execute sql stmt                          */
/***************************************************************/

select_plan:

sqlstmt=,

'select planname ',

' from sysibm.syspacklist ',

' where name = ''' || ws_pack_name || '''

'order by planname ',

'subcom dsnrexx'

if rc then

  s_rc = rxsubcom('ADD','DSNREXX','DSNREXX')

address dsnrexx

connect ssid

if rc ¬= Ø then exit

'execsql declare c1 cursor for s1'

if sqlcode ¬= Ø then call sql_error

'execsql prepare s1 into :outsqlda from :sqlstmt'

if sqlcode ¬= Ø then call sql_error

'execsql open C1'

if sqlcode ¬= Ø then call sql_error

'execsql fetch c1 using descriptor :outsqlda'

if sqlcode < Ø then call sql_error

  do while(sqlcode = Ø)

    hsts = hsts + 1
    pname.hsts = outsqlda.1.sqldata

    'execsql fetch c1 using descriptor :outsqlda'
  end
return
/***************************************************************
/* create table                                              */
/***************************************************************
create_table:
 $tbcreate  wdb2plan names(plannam)
  nowrite $ 
  do i = 1 to hsts
   plannam = pname.i 
   $tbadd  wdb2plan  save(plannam)$
  end 
return
return
/***************************************************************
/*    sql_error                                               */
/***************************************************************
sql_error:
  say 'SQL error.............'
  say sqlcode sqlda
  say sqlstmt
  'execsql close c1 '
  exit

INDEX DDLS

CREATE INDEX SYSBM.DSNKSX02
  ON SYSBM.SYSPACKSTM
  ( NAME ASC ,
    COLLID ASC ,
    SEQNO ASC )
  USING STOGROUP SYSDEFLT
    PRIQTY 21600
    SECQTY 1000
    ERASE NO
  FREEPAGE 0
  PCTFREE 10
  GBPCACHE CHANGED
  BUFFERPOOL BP0
  CLOSE NO
  DEFINE YES

CREATE INDEX SYSBM.DSNKDX04
  ON SYSBM.SYSPACKDEP
  ( DNAME ASC ,
    DCOLLID ASC )
  USING STOGROUP SYSDEFLT
    PRIQTY 16800
    SECQTY 480
    ERASE NO
SAMPLE OUTPUT

RDCQ

DB2 Plan and Package query

Option : 

01. List of packages for a plan
02. List of plans for a package
03. List of tables which are used by a package
04. List of packages which are used by a table

PF3 : return

RDCQ001

List of tables which are used by a package Row 1 to 4 of 4
Dbid: T Coll. id : ADDR001 Package name : ADDR001

Total number of sql stmt: 26
Commit (yes - no) : NO

Creator Table name Select Insert Delete Update Declare
= ======== = = = = = = = == = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
TXXX INDV X
TXXX CUSTOMER X X
TXXX CUSTOMER_RL X X
TXXX CUSTOMER_ADDR X

RDCQ002

List of packages which are used by tables Row 1 to 6 of 6
Dbid: T Creator : TXXX Table name: BHHTTRS
DB2 UDB 8.1 – inserting into a view

This article discusses two of the new options available for inserting rows into a view in DB2 UDB V8.1. These new options
overcome two of the restrictions in previous releases – namely not being able to insert into a view that is a union of tables, and not being able to insert into a view that is a join of one or many tables.

In DB2 UDB V8.1 you can insert into a view that is a union of two/many tables if you code a check statement on the underlying tables.

Moving on to the second case, there has always been a restriction of not being able to insert into a view that is a join of two/many tables. Consider the following scenario – you have two table tab1 and tab2:

```sql
CREATE TABLE tab1 (empno INT, comm1 CHAR(10))
CREATE TABLE tab2 (empno INT, comm2 CHAR(10))
```

with a row in each:

```sql
INSERT INTO tab1 VALUES(1,'Helen')
INSERT INTO tab2 VALUES(1,'LOC1')
```

and a view (tabv) based on these two tables defined as follows:

```sql
CREATE VIEW tabv AS SELECT tab1.empno, comm1, comm2 FROM tab1, tab2 WHERE tab1.empno=tab2.empno
```

Now if you select from the view:

```sql
SELECT * FROM tabv
```

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>COMM1</th>
<th>COMM2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Helen</td>
<td>LOC1</td>
</tr>
</tbody>
</table>

you can see that the SELECT works. But if you try and insert into the view:

```sql
INSERT INTO tabv VALUES(2,'Sharon','LOC2')
```

you get an SQL0150N message, which says that the requested operation is not permitted.

So, how do we overcome this restriction? We need to use another new DB2 UDB V8.1 concept, namely that of ‘instead of triggers’. The ‘instead of triggers’ supplements the previously
available BEFORE and AFTER triggers, but what it uniquely offers you is the possibility to update the base tables of a view without going through the view.

So let’s create an instead of trigger (hm) as shown below:

```sql
CREATE TRIGGER hm
INSTEAD OF INSERT ON tabv
REFERENCING NEW AS N
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
    INSERT INTO tab1 values (empno,comm1);
    INSERT INTO tab2 values (empno,comm2);
END
```

This trigger intercepts the insert into the view (tabv) and splits it up into separate inserts into the two individual tables (tab1 and tab2). So now if we try to insert into the view again:

```sql
insert into tabv values(2,'Sharon','LOC2')
```

we get the successful execution message back. And if we select from the tables, we see that the insert statement has worked:

```sql
select * from tab1
EMPNO      COMM1
---------- ----------
1 Helen
2 Sharon

select * from tab2
EMPNO      COMM2
---------- ----------
1 LOC1
2 LOC2
```

So we now have a method of inserting a row into a view which is made up of a join of two tables. This is a welcome addition to the ever expanding SQL query family.

---

*C Leonard
Freelance Consultant (UK) © Xephon 2003*
IBM has announced DB2 Information Integrator for Content V8.2, promising on-demand access to critical business information.

DB2 Information Integrator for Content V8.2 (formerly IBM Enterprise Information Portal) connects to many information sources, including DB2 or other ODBC and JDBC relational databases, file systems, and Lotus Domino databases. It also provides information mining, with expanded language support and updated functions, that supports sites with Intelligent Miner for Text on AIX and NT.

Specific new features include workflow enhancements that exploit WebSphere MQ Workflow, and Lotus Extended Search 4.0.

For further information contact your local IBM representative.

* * *

IBM has announced Version 8.1 of its DB2 OLAP Server for z/OS for analysing and transforming data on zSeries or S/390 systems without having to move data to other platforms. Based on Hyperion Essbase 6.5.1, it adds functionality, scalability, performance, usability, and administration enhancements.

A logical cube can now be defined to include, in lower portions of the cube, cells whose values physically reside in a relational database, combining mass data scalability with the data analysis.

Operations such as calculation, data load, and export can now be parallelized by running them in multi-threaded mode and improvements via zFS deliver performance improvements for load and calculation processing and simplify data set maintenance.

Integration server improvements include support for hybrid analysis, multiple data source connections, formula verification, and z/OS and OS/390 specific enhancements such as support for multiple ports and the ability for users in a group to share metadata. OLAP Miner lets users discover anomalies, special segments, and notable fluctuations in critical business data. A more comprehensive installation and administration guide streamlines the installation process. Client installation can now be done over the network and a code-page conversion utility is included to speed installation.

For security, users can now change their RACF password through the Application Manager interface or the Spreadsheet interface.

Admin enhancements include improvements to production workflow and productivity. Several new templates and scripts are shipped, which can be customized to automate OLAP processes. Users can capture and route statistics and other data to STDOUT and STDERR, and the new MEMCHECK utility can be used to protect from memory related failures.

For further information contact your local IBM representative.

* * *