135

DB2 update

January 2004

In this issue

DB2 Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38342
From USA: 01144 1635 38342
E-mail: trevore@xephon.com

North American office

Xephon
PO Box 350100
Westminster, CO 80035-0100
USA
Telephone: 303 410 9344

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.

Editor

Trevor Eddolls

Contributions

When Xephon is given copyright, articles published in *DB2 Update* are paid for at the rate of £100 ($160) per 1000 words and £50 ($80) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £20 ($32) 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.

UDB V8 LUW – what is infinite logging?

One of the many new features of the DB2 UDB V8 offering is the ability to specify an ‘infinite’ number of logs. What does this mean, and what does it offer you? What it means I will cover later, but what it offers you is a way to get over the problem of long-running transactions that do not commit, thus causing you a log full problem (and perhaps a sleepless night!).

So what does it mean? Let’s look back at the situation in V7. You could specify from 2 to 128 primary logs and from 0 to 126 secondary logs, with the total number of primary logs plus secondary logs not exceeding 128. With each log having a maximum size of 65,535 4KB pages, the maximum amount of log space was limited to 32GB. In V8, the limits are increased to 2 to 256 primary logs and 0 to 254 secondary logs (with the total number of primary logs plus secondary logs not exceeding 256), with a minimum size of 4 and a maximum of 262,144 4KB pages, and with a total log space of 256GB. This is summarized in Figure 1.

The *Admin Guide* (V7 or V8) gives you the following formulas to calculate the minimum/maximum space requirements for the logs:

Minimum size: (logprimary * (logfilsiz + 2) * 4096) + 8192

	V7	*V8*
Number of primary logs you can specify (min/max)	2/128	2/256
Number of secondary logs you can specify (min/max)	0/126	0/254
Maximum number of logs (primary + secondary)	128	256
Maximum log size (4K pages)	65,535	262,144
Total log space	32GB	256GB

Figure 1: Log sizes

Maximum size:

((logprimary + logsecond) * (logfilsiz + 2) * 4096) + 8192

I ran all the SQL in this article on a Windows 2000 machine running DB2 UDB 8.1 FP1.

So how can you initiate V8 infinite logging? You initiate it by setting the DB CFG parameter userexit to ON and the logsecond parameter to '-1' (you need to invoke the userexit first if you are going to use infinite logging). The commands are:

```
>db2 update db cfg for sample using userexit on
>db2 update db cfg for sample using logsecond –1
```

What this gives you is an active unit of work that can occupy not only active logs but archive logs as well.

As an example, let's define our system with the maximum number of permissible primary and secondary logs and then run a long transaction, which doesn't issue any commits, and see what happens. Then we will enable infinite logging and see what happens.

We will use a test table called EMP in the SAMPLE database, which contains about 1 million rows and was created by copying numerous copies of the EMPLOYEE table into it. How to create the table is not shown in this article – you can use any table of your choice.

First reduce the size of the log file to the minimum permissible and then increase the number of primary logs to the maximum permissible.

```
>db2 update db cfg for sample using logfilsiz 4
>db2 update db cfg for sample using logprimary 256

>db2 connect reset
>db2 connect to sample
```

Check to see that the changes have taken effect:

```
>db2 get db cfg for sample | find /i "Log"
Log file size (4KB)                         (LOGFILSIZ) = 4
Number of primary log files                (LOGPRIMARY) = 256
Number of secondary log files               (LOGSECOND) = Ø
```

```
User exit for logging enabled                (USEREXIT) = OFF
```

We can see that the settings are what we want in order to perform our test. If we now try to update every row in our test table we get the following result:

```
>db2 update emp set salary = salary + 1.Ø
DB21034E The command was processed as an SQL statement because it was
not a valid Command Line Processor command. During SQL processing it
returned:
SQLØ964C  The transaction log for the database is full.  SQLSTATE=57Ø11
```

Switch back to using just three primary logs:

```
>db2 update db cfg for sample using logprimary 3
```

Now invoke infinite logging:

```
>db2 update db cfg for sample using userexit on
>db2 update db cfg for sample using logsecond -1
```

This means that you are using archive logging. If your system uses circular logging, then, before switching on infinite logging, you need to make sure that you have procedures in place to handle the archive logs:

```
>db2 connect reset
>db2 backup db sample to c:\backups
```

And check that the updates have worked:

```
>db2 get db cfg for sample | find /i "Log"

Log file size (4KB)                         (LOGFILSIZ) = 4
Number of primary log files                (LOGPRIMARY) = 3
Number of secondary log files               (LOGSECOND) = -1
User exit for logging enabled                (USEREXIT) = ON
```

Now if we try to update the test table EMP again:

```
>db2 connect to sample
>db2 update emp set salary = salary + 1.Ø
DB2ØØØØI  The SQL command completed successfully.
```

You can see that this time the update command has worked.

I hope I have shown how easy it is to switch on infinite logging and what the benefits are. This doesn't mean that you do not have to manage the logs (just as you did in Version 7 and Version 8 prior to switching on infinite logging), and you should still monitor your system for transactions that do not commit frequently, but it does give you another option to use.

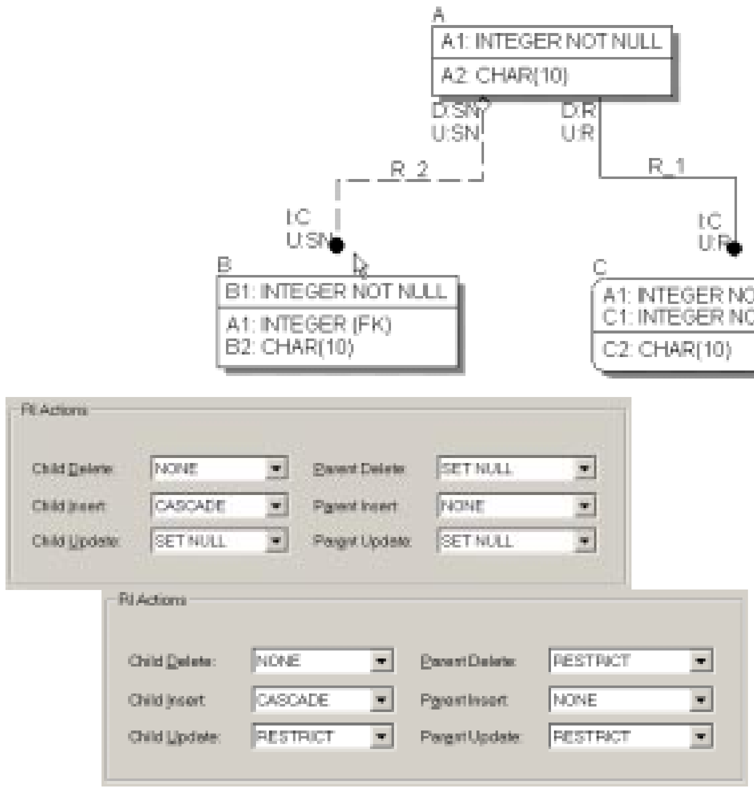
C Leonard
Freelance Consultant (UK)

Fixes for ERwin DB2/UDB trigger templates

One of the best ways to enforce referential integrity rules is through the use of triggers. CA-ERwin supports referential integrity triggers (RI triggers), which are triggers that maintain the integrity between two related tables. For example, if a row in a parent table is inserted, updated, or deleted, an RI trigger tells the DBMS what to do to rows in other tables that have a foreign key value matching the primary key in the row being added, updated, or deleted. ERwin provides a set of default RI trigger templates that you can attach to tables to tell the target server how to enforce referential integrity. You can use ERwin macros to customize the RI trigger templates and override the default code that is generated by ERwin.

This article explains how to correct some problems in ERwin DB2/UDB Trigger Templates (ERwin Version 3.5.2 and higher and DB2/UDB Version 5.0 and higher are prerequisite). The following problems occur during the generation of triggers:

- The SQLSTATE specified in the RAISE_ERROR function does not conform to the rules for an application defined (referential integrity types: CHILD DELETE RESTRICT, CHILD INSERT RESTRICT, CHILD UPDATE RESTRICT, PARENT DELETE RESTRICT, PARENT INSERT RESTRICT, PARENT UPDATE RESTRICT).
- ERwin macro templates do not support transition tables OLD_TABLE and NEW_TABLE (referential integrity types: CHILD INSERT CASCADE, CHILD UPDATE CASCADE).



Child Delete:	NONE	Parent Delete:	SET NULL
Child Insert:	CASCADE	Parent Insert:	NONE
Child Update:	SET NULL	Parent Update:	SET NULL

Child Delete:	NONE	Parent Delete:	RESTRICT
Child Insert:	CASCADE	Parent Insert:	NONE
Child Update:	RESTRICT	Parent Update:	RESTRICT

Figure 1: Trivial model and RI actions for relationships

- In Trigger Update Header there is a missing referencing clause (UPDATE HEADER).

I have made ERwin macros for correcting these problems, which can be used to overwrite the installation macros. I will present a trivial model (see Figure 1) and the whole procedure, which implements this solution.

ERwin installation default DB2/UDB Schema Generation Preview produced the following result:

```
create trigger tD_A after DELETE on A
   REFERENCING OLD AS OLD for each row mode db2sql
    update B
    set B.A1 = NULL
    where B.A1 = old.A1
```

```
 !!

create trigger tD_A2 after DELETE on A
   REFERENCING OLD AS OLD for each row mode db2sql
   WHEN ((select count(*) from C where C.A1 = old.A1) > Ø)
     SIGNAL SQLSTATE '42987' ('Cannot DELETE A because C exists.')
 !!

create trigger tU_A after UPDATE on A
     for each row mode db2sql
   WHEN ((select count(*) from A where A.A1 <> old.A1) > Ø)
     update B
     set B.A1 = NULL
     where B.A1 = old.A1
 !!

create trigger tU_A2 after UPDATE on A
     for each row mode db2sql
   WHEN (((select count(*) from A where A.A1 <> old.A1) > Ø) AND
     ((select count(*) from C where C.A1 = old.A1) > Ø))
       SIGNAL SQLSTATE '42987' ('Cannot UPDATE A because C exists.')
 !!

create trigger tI_B after INSERT on B
   REFERENCING NEW AS NEW for each row mode db2sql
    insert into A (A1)
    select A1 from new
    where  B.A1 is not null and
           not exists (select * from A
                       where new.A1 = A.A1)
 !!

create trigger tU_B after UPDATE on B
     for each row mode db2sql
    update B
    set B.A1 = NULL
    where  not exists (select * from A
                       where new.A1 = A.A1)
 !!

create trigger tI_C after INSERT on C
   REFERENCING NEW AS NEW for each row mode db2sql
    insert into A (A1)
    select A1
    from new
    where
           and
          not exists (select * from A
                      where  new.A1 = A.A1)
 !!
```

```
create trigger tU_C after UPDATE on C
     for each row mode db2sql
   WHEN (((select count(*) from A where new.A1 = A.A1) = Ø))
       SIGNAL SQLSTATE '42987' ('Cannot UPDATE C because A does not
exist.')
 !!
```

During the generation of the triggers, all the problems I mentioned earlier occurred:

- RAISE_ERROR problem has the trigger tD_A2. This trigger will pass the generate phase, but during execution the system generates sqlcode SQL0435N (an invalid SQLSTATE ‘<sqlstate>’ is specified in the function RAISE_ERROR).
- Unsupported transition tables problems have triggers tI_B and tI_C. The tI_B trigger produces error SQL0204N – ‘userid.NEW’ is an undefined name, and the tI_C trigger has a syntax error too (SQL0104N – an unexpected token, ‘exists’, was found following ‘and not’; expected tokens may include: ‘BETWEEN’).
- Update header problems have the following triggers: tU_A, tU_A2, tU_B, and tU_C (SQL0206N – ‘OLD.A1’ is not valid in the context in which it is used).

User override trigger templates to correct errors in the built-in trigger templates are:

```
MY CHILD DELETE RESTRICT
 WHEN ((select count(*) from %Parent where %JoinFKPK(%Old,%Parent," =
"," and")) > Ø)
   SIGNAL SQLSTATE '75ØØ3' ('Cannot DELETE %Child because %Parent
exists.')

MY CHILD INSERT RESTRICT
 WHEN (((select count(*) from %Parent where %JoinFKPK(%New,%Parent," =
"," and")) = Ø)
   %If (%NotnullFK(%New," is not null")) { AND %NotnullFK(," is not
null","new."," and") })
   SIGNAL SQLSTATE '75ØØ1' ('Cannot INSERT %Child because %Parent does
not exist.')

MY CHILD UPDATE RESTRICT
 WHEN (((select count(*) from %Parent where %JoinFKPK(%New,%Parent," =
"," and")) = Ø)
```

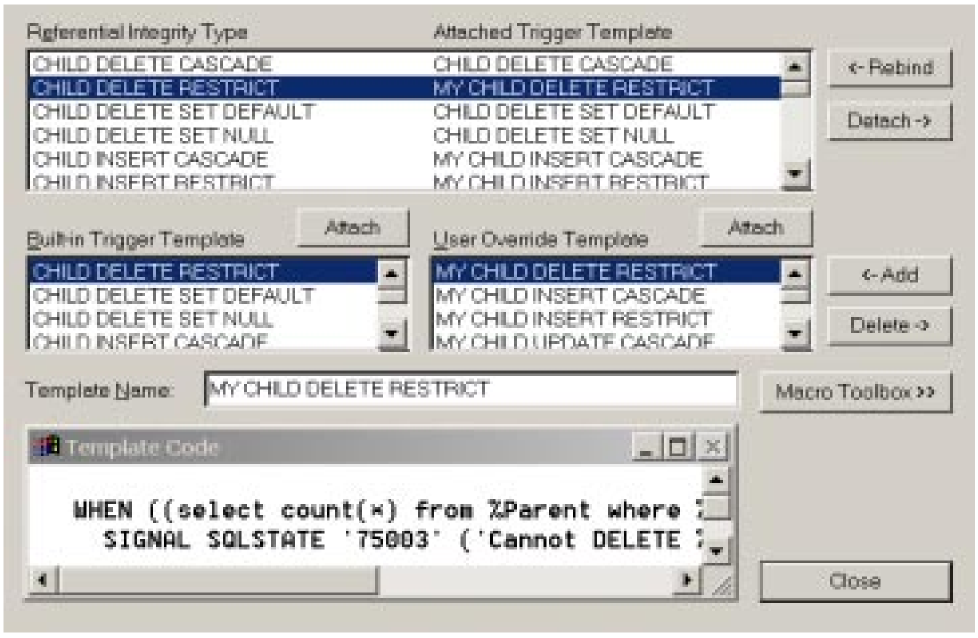
```
   %If (%NotnullFK(%New," is not null",," and")) { AND %NotnullFK(," is
not null","new."," and")})
   SIGNAL SQLSTATE '75002' ('Cannot UPDATE %Child because %Parent does
not exist.')

MY PARENT DELETE RESTRICT
 WHEN ((select count(*) from %Child where %JoinFKPK(%Child,%Old," = ","
and")) > 0)
   SIGNAL SQLSTATE '75003' ('Cannot DELETE %Parent because %Child
exists.')

MY PARENT INSERT RESTRICT
 WHEN ((select count(*) from %Child where %JoinFKPK(%Child,%New," = ","
and")) = 0)
     SIGNAL SQLSTATE '75001' ('Cannot INSERT %Parent because %Child does
not.')
MY PARENT UPDATE RESTRICT
 WHEN (((select count(*) from %Parent where %JoinPKPK(%Old,%New," <> ","
or ")) > 0) AND
   ((select count(*) from %Child where %JoinFKPK(%Child,%Old," = ","
and")) > 0))
     SIGNAL SQLSTATE '75002' ('Cannot UPDATE %Parent because %Child
exists.')

MY UPDATE HEADER
 create trigger t%1Action_%15TableName%TriggerSeq after %Action on
%TableName
  %RefClause REFERENCING OLD AS old NEW AS new for each row mode db2sql

MY CHILD INSERT CASCADE
  %Decl(j,0)
  %=(j,%TriggerSeq)
  %If (%>(%:j,0)) {
     %=(j,%+(%:j,1))
  }
  SIGNAL SQLSTATE'75000' ('Temporary trigger.')
  %DBMSTriggerDelim
  drop trigger t%1Action_%15TableName%:j;
  create trigger t%1Action_%15TableName%:j after %Action on %TableName
     REFERENCING NEW_TABLE AS NEW for each row mode db2sql
      insert into %Parent (%ParentPK(",",))
        select %ChildFK(",",) from %New where 1 = 1
            %NotnullFK(new," is not null"," and "," and") and
            not exists (
              select * from %Parent
                where
                  %JoinFKPK(%New,%Parent," = "," and"))
```

Note: in this trigger template, the default insert header template is changed too.

```
MY CHILD UPDATE CASCADE
  %Decl(j,Ø)
  %=(j,%TriggerSeq)
  %If (%>(%:j,Ø)) {
     %=(j,%+(%:j,1))
  }
  SIGNAL SQLSTATE'75ØØØ' ('Temporary trigger.')
  %DBMSTriggerDelim
  drop trigger t%1Action_%15TableName%:j;
  create trigger t%1Action_%15TableName%:j after %Action on %TableName
     REFERENCING NEW_TABLE AS NEW for each row mode db2sql
      insert into %Parent (%ParentPK(",",))
      select %ChildFK(",",)
        from %New
        where 1 = 1
          %NotnullFK(new," is not null"," and "," and") and
          not exists (
            select * from %Parent
              where
                %JoinFKPK(%New,%Parent," = "," and"))
```

Note: in this trigger template, the default update header template is changed too.

Create an RI type override trigger template in ERwin:



Referential Integrity Type	Attached Trigger Template
CHILD DELETE CASCADE	CHILD DELETE CASCADE
CHILD DELETE RESTRICT	MY CHILD DELETE RESTRICT
CHILD DELETE SET DEFAULT	CHILD DELETE SET DEFAULT
CHILD DELETE SET NULL	CHILD DELETE SET NULL
CHILD INSERT CASCADE	MY CHILD INSERT CASCADE
CHILD INSERT RESTRICT	MY CHILD INSERT RESTRICT

Builtin Trigger Template:
- CHILD DELETE RESTRICT
- CHILD DELETE SET DEFAULT
- CHILD DELETE SET NULL
- CHILD INSERT CASCADE

User Override Template:
- MY CHILD DELETE RESTRICT
- MY CHILD INSERT CASCADE
- MY CHILD INSERT RESTRICT
- MY CHILD UPDATE CASCADE

Template Name: MY CHILD DELETE RESTRICT

```
  WHEN ((select count(*) from %Parent where
    SIGNAL SQLSTATE '75003' ('Cannot DELETE
```

Figure 2: Global Trigger Templates

1 In the physical model, on the *Database* menu, choose *RI Triggers*, and then choose *Global Trigger Templates* (see Figure 2).

2 Type a name for the new template in the *Template Name* box, and then click *Add*.

3 Type the text in the *Template Code* box.

4 Select the RI action you want to attach to the new template in the *Referential Integrity Type* list.

5 Click *Attach* above the *User Override* list.

6 Click *Close*.

Repeat steps 2 to 5 for all the templates in Figure 3.

Note: you must select the RI Type Override check box for triggers during schema generation to generate an RI type override template in the schema.

The ERwin Schema Generation Preview now produces the following results:

```
create trigger tD_A after DELETE on A
   REFERENCING OLD AS OLD for each row mode db2sql
    update B
```

Referential Integrity Type	*Attached Trigger Template*
CHILD DELETE RESTRICT	MY CHILD DELETE RESTRICT
CHILD INSERT CASCADE	MY CHILD INSERT CASCADE
CHILD INSERT RESTRICT	MY CHILD INSERT RESTRICT
CHILD UPDATE CASCADE	MY CHILD UPDATE CASCADE
CHILD UPDATE RESTRICT	MY CHILD UPDATE RESTRICT
PARENT DELETE RESTRICT	MY PARENT DELETE RESTRICT
PARENT INSERT RESTRICT	MY PARENT INSERT RESTRICT
PARENT UPDATE RESTRICT	MY PARENT UPDATE RESTRICT
TRIGGER UPDATE HEADER	MY UPDATE HEADER

Figure 3: User override trigger templates

```
    set B.A1 = NULL
    where B.A1 = old.A1
 !!

create trigger tD_A2 after DELETE on A
   REFERENCING OLD AS OLD for each row mode db2sql
   WHEN ((select count(*) from C where C.A1 = old.A1) > Ø)
     SIGNAL SQLSTATE '75ØØ3' ('Cannot DELETE A because C exists.')
 !!

create trigger tU_A after UPDATE on A
     REFERENCING OLD AS old NEW AS new for each row mode db2sql
   WHEN ((select count(*) from A where A.A1 <> old.A1) > Ø)
     update B
     set B.A1 = NULL
     where B.A1 = old.A1
 !!

create trigger tU_A2 after UPDATE on A
     REFERENCING OLD AS old NEW AS new for each row mode db2sql
   WHEN (((select count(*) from A where A.A1 <> old.A1) > Ø) AND
     ((select count(*) from C where C.A1 = old.A1) > Ø))
       SIGNAL SQLSTATE '75ØØ2' ('Cannot UPDATE A because C exists.')

 !!

create trigger tI_B after INSERT on B
   REFERENCING NEW AS NEW for each row mode db2sql
   SIGNAL SQLSTATE'75ØØØ' ('Temporary trigger.')
   !!

  drop trigger tI_B;
  create trigger tI_B after INSERT on B
     REFERENCING NEW_TABLE AS NEW for each row mode db2sql
      insert into A (A1)
      select A1
      from new
      where 1 = 1  and
            new.A1 is not null and
            not exists (select * from A
                        where  new.A1 = A.A1)
 !!

create trigger tU_B after UPDATE on B
     REFERENCING OLD AS old NEW AS new for each row mode db2sql
    update B
    set  B.A1 = NULL
    where  not exists (select *
                       from A
                       where  new.A1 = A.A1)
 !!

create trigger tI_C after INSERT on C
   REFERENCING NEW AS NEW for each row mode db2sql
   SIGNAL SQLSTATE'75ØØØ' ('Temporary trigger.')
   !!

  drop trigger tI_C;
  create trigger tI_C after INSERT on C
     REFERENCING NEW_TABLE AS NEW for each row mode db2sql
      insert into A (A1)
      select A1
      from new
      where 1 = 1  and
            not exists (select *
                        from A
                        where  new.A1 = A.A1)
 !!

create trigger tU_C after UPDATE on C
     REFERENCING OLD AS old NEW AS new for each row mode db2sql
     WHEN (((select count(*) from A where new.A1 = A.A1) = Ø))
         SIGNAL SQLSTATE '75ØØ2' ('Cannot UPDATE C because A does not
exist.')
 !!
```

Nikola Lazovic
DB2 System Administrator
Postal Savings Bank (Serbia and Monte Negro)

Accessing DB2 using a Web browser and DB2/REXX – part 2

This month we conclude the article that explores getting to mainframe DB2 from a PC using a Web browser such as Internet Explorer and mainframe DB2/REXX.

ADDING SOME INTERACTIVITY

Although this is working dynamically, it’s still pretty static as far as the user is concerned. We’ll now look at getting some input from the user.

In our current version of the REXX, the subsystem is hardcoded – now we'll change things so that the subsystem is input by the user.

The HTML structures that allow this are forms. We have a choice for this – use a pulldown menu with a fixed list of subsystems, or use a text field, which allows the user to input anything. The former method has the advantage of needing no validation, but requires changing whenever a subsystem is added. Let's go with the latter method.

Again, what I'll do is get a form working without DB2, then add DB2 afterwards. This is the HTML for the form:

```
<HTML>
<HEAD>
<TITLE>Just testing</TITLE>
<script>
function checkForm(f)
   {
    if (f.subsys.value == "")
       {
          alert("Subsystem must be filled");
         f.subsys.focus();
         return false;
       }
    if (f.subsys.value.length != 4)
       {
         alert("Subsystem must be four characters long");
         f.subsys.focus();
         return false;
       }
    return true;
   }
</script>
</HEAD>
<H1>Testing REXX</h1>
<FORM METHOD="GET" ACTION="/nucgi/DB2/test3.sh" onsubmit="return
checkForm(this);">
<p>Enter the subsystem name and press enter: <INPUT TYPE=TEXT
NAME=subsys SIZE=6 VALUE="" >
</FORM>
</BODY>
</HTML>
```

The bits to take notice of here are code between the <FORM>... </FORM> tags and between the <script>...</script> tags (case is not important for the tags – the differences are because of my sloppiness).

The <script>..</script> tags contain one javascript routine called checkForm. This validates the form passed to it – it makes sure that the field called subsys is filled, and that the string entered is exactly four characters long. If the conditions are satisfied, it returns true, otherwise it puts up an alert and returns false.

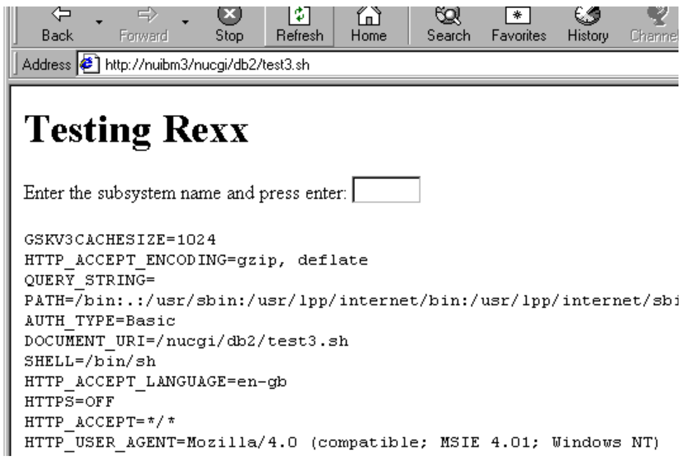
The form (between the <FORM>...</FORM> tags) has one text input field called subsys. When the form is submitted (this will happen when *Enter* is pressed when focus is in the subsys field), Javascript routine checkForm is called to validate the form. If the form is valid (true is returned), the form performs its ACTION. The action is a URL, and so gives the name either of the next page of HTML to go to or of the next CGI script to execute. In this case, the URL is the same script we've come from.

Embedding the HTML in a REXX EXEC, I get:

```
/*                 REXX                                              */
/*    REXX to test REXX/DB2                                          */
/*    Alan Smith September 2ØØ3                                      */
/*********************************************************************/
call write_html_prologue
call write_html_epilogue
exit
write_html_prologue:
say 'Content-type: text/html'
say
say
say '<HTML>'
say '<HEAD>'
say '<TITLE>Just testing</TITLE>'
say '<script>'
say 'function checkForm(f)'
say '   {'
say '    if (f.subsys.value == "")'
say '       {'
say '        alert("Subsystem must be filled");'
say '        f.subsys.focus();'
say '        return false;'
say '       }'
say '    if (f.subsys.value.length != 4)'
say '       {'
say '        alert("Subsystem must be four characters long");'
say '        f.subsys.focus();'
say '        return false;'
say '       }'
say '    return true;'
say '   }'
say '</script>'
say '</HEAD>'
say '<H1>Testing REXX</h1>'
say '<FORM METHOD="GET" ACTION="/nucgi/DB2/test3.sh"'
say 'onsubmit="return checkForm(this);">'
say '<p>Enter the subsystem name and press enter:'
say '<INPUT TYPE=TEXT NAME=subsys SIZE=6 VALUE="">'
say '</FORM>'
say '<pre>'
do i = 1 to __ENVIRONMENT.Ø
    say __ENVIRONMENT.i
end
say '</pre>'
return
write_html_epilogue:
say '</BODY>'
say '</HTML>'
return
```

I've added some lines here to show the environment variables available to the REXX:

```
do i = 1 to __ENVIRONMENT.Ø
    say __ENVIRONMENT.i
end
```



http://nuibm3/nucgi/db2/test3.sh

Testing Rexx

Enter the subsystem name and press enter:

```
GSKV3CACHESIZE=1024
HTTP_ACCEPT_ENCODING=gzip, deflate
QUERY_STRING=
PATH=/bin:.:/usr/sbin:/usr/lpp/internet/bin:/usr/lpp/internet/sbi
AUTH_TYPE=Basic
DOCUMENT_URI=/nucgi/db2/test3.sh
SHELL=/bin/sh
HTTP_ACCEPT_LANGUAGE=en-gb
HTTPS=OFF
HTTP_ACCEPT=*/*
HTTP_USER_AGENT=Mozilla/4.0 (compatible; MSIE 4.01; Windows NT)
```

Figure 1: New page

Key the URL for the program into your browser. In my case it's http://nuibm3/nucgi/DB2/test3.sh. The resultant page looks like Figure 1.

Note the environment variables that are printed out. The one we're interested in is QUERY_STRING – this gives us what was submitted in the form. Because this is the first time into the form, the query string is blank. Click in the input text box and try a few things to make sure the validation is working. Then try inputting a valid string, say DB2T.

You'll notice two things. The URL in the address field of the browser becomes http://nuibm3/nucgi/DB2/test3.sh?subsys=DB2T, and the environment variable holding the query string becomes QUERY_STRING=subsys=DB2T.

So, when we get into the program, after the subsys field has been filled in, we want to get the value of the field and do something with it. We can run down the list of environment variables, find the one for QUERY_STRING, and then parse the string. Although this case is fairly simple – we've only one field, and we wouldn't expect any spaces or other encoded characters – we want a method which will deal with these situations in the general case and not require specific processing each time. In particular, there is no easy way for us to take an encoded character and translate it into an EBCDIC one.

Cgiparse

USS provides us with a routine to parse the query string – cgiparse. The routine looks at QUERY_STRING, splits it into fields, and does the necessary character substitution. Let's look at the case where the user keys 'DB2T' as the value of the field subsys. The query string is set to subsys=DB2T. You can run cgiparse with different parameters:

```
cgiparse –form
FORM_subsys='DB2T'
cgiparse -value subsys
DB2T
```

Now look at what happens if the user keys a string containing a special character. Say the user keys ‘d%12’. The per cent sign is a special character and so the Web browser encodes it when it inserts it into the URL. The URL is http://nuibm3/nucgi/DB2/test3.sh?subsys=d%2512.

The query string is:

```
subsys=d%2512
```

Running cgiparse gets us back the original value:

```
cgiparse -value subsys
d%12
```

The only problem with cgiparse is that it outputs its results to standard output, whereas we want to bring them into our program. Under TSO, we would use OUTTRAP, but that doesn’t work here. The answer is to create a pipe. A pipe is a sort of communication conduit between two processes – one process adds things at the tail of the pipe, while the other reads from the head of the pipe.

This code creates a pipe and stores the file descriptor associated with the head of the pipe in p.1, and that associated with the tail of the pipe in p.2:

```
address syscall 'pipe p.'
```

We then call cgiparse to write the value of the subsys field to the tail-end of the pipe:

```
'cgiparse -value subsys >/dev/fd' || p.2
```

We then close the end of the pipe:

```
address syscall 'close' p.2
```

and read it into a stem:

```
address mvs 'execio * diskr' p.1 '(stem s.)'
```

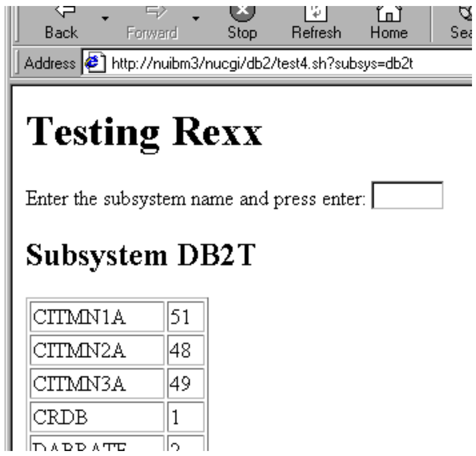
As long as everything went OK, s.1 will contain the value of subsys.

I put this code into routine get_form_values, which puts the value from the form field into the variable subsystem. If the subsystem is blank, this must be the first time in, and I don't do the DB2 bits. Otherwise I use it in the CONNECT statement.

This is the whole thing:

```
/*                 REXX                                */
/*    REXX to test REXX/DB2                            */
/*    Alan Smith September 2003                        */
/*******************************************************/
call write_html_prologue
call get_form_values
if subsystem ¬= '' then
  do
  say '<h2>Subsystem' subsystem '</h2>'
  say '<table border>'
  call do_DB2_stuff
  say '</table>'
  end
call write_html_epilogue
exit
/***********************************************/
/*  do_DB2_stuff                               */
/*  Connect to DB2 and get data                */
/***********************************************/
do_DB2_stuff:
myRC = RXSUBCOM("ADD","DSNREXX","DSNREXX")
ADDRESS DSNREXX
"CONNECT" subsystem
call check_sql_code("connect")
STMT = "SELECT DBNAME,COUNT(*) AS CT FROM SYSIBM.SYSTABLESPACE",
            " GROUP BY DBNAME",
            " ORDER BY DBNAME",
            " WITH UR;"
"EXECSQL DECLARE C1 CURSOR FOR S1"
"EXECSQL PREPARE S1 FROM :STMT"
call check_sql_code("prepare")
"EXECSQL OPEN C1"
call check_sql_code("open cursor")
"EXECSQL FETCH C1 INTO :DBNAME,:DBCOUNT"
do while SQLCODE ¬= 100
  call check_sql_code("fetch")
  say "<TR><TD>" DBNAME "<TD>" DBCOUNT
  "EXECSQL FETCH C1 INTO :DBNAME,:DBCOUNT"
end
"EXECSQL CLOSE C1"
call check_sql_code("close cursor")
"DISCONNECT"
call check_sql_code("disconnect")
myRC = RXSUBCOM("DELETE","DSNREXX","DSNREXX")
```

```
return
/***********************************************/
/*  check_sql_code                             */
/*  check sql code for errors                  */
/***********************************************/
check_sql_code:
if SQLCODE < Ø then
  do
  say '</TABLE>'
  say "<p>sql error " SQLCODE SQLERRMC
  exit 12
  end
return
/***********************************************/
/*  write_html_prologue                        */
/*  write the html code for the start of the   */
/*  page                                       */
/***********************************************/
write_html_prologue:
say 'Content-type: text/html'
say
say
say '<HTML>'
say '<HEAD>'
say '<TITLE>Just testing</TITLE>'
say '<script>'
say 'function checkForm(f)'
say '   {'
say '    if (f.subsys.value == "")'
say '       {'
say '        alert("Subsystem must be filled");'
say '        f.subsys.focus();'
say '        return false;'
say '       }'
say '    if (f.subsys.value.length != 4)'
say '       {'
say '        alert("Subsystem must be four characters long");'
say '        f.subsys.focus();'
say '        return false;'
say '       }'
say '    return true;'
say '   }'
say '</script>'
say '</HEAD>'
say '<H1>Testing REXX</h1>'
say '<FORM METHOD="GET" ACTION="/nucgi/DB2/test4.sh"'
say 'onsubmit="return checkForm(this);">'
say '<p>Enter the subsystem name and press enter:'
say '<INPUT TYPE=TEXT NAME=subsys SIZE=6 VALUE="">'
say '</FORM>'
```



Testing Rexx

Enter the subsystem name and press enter:

Subsystem DB2T

CITMN1A	51
CITMN2A	48
CITMN3A	49
CRDB	1
DARRATE	2

Figure 2: Screenshot of result

```
return
/************************************************/
/*  write_html_epilogue                         */
/*  write the html code for the end of the      */
/*  page                                        */
/************************************************/
write_html_epilogue:
say '</BODY>'
say '</HTML>'
return
/************************************************/
/*  get_form_values                             */
/*  parse the query string to get the form      */
/*  field values                                */
/************************************************/
get_form_values:
address syscall 'pipe p.'
'cgiparse -value subsys >/dev/fd' || p.2
address syscall 'close' p.2
address mvs 'execio * diskr' p.1 '(stem s.)'
if s.Ø = Ø then
  subsystem = ''
else
  parse upper var s.1 subsystem
return
```

Figure 2 is a screenshot of the result:

SUMMARY

In this article I've described a technique of getting to mainframe DB2 from a Web browser using REXX, DB2, and WebSphere http server on a mainframe. This requires little in the way of set-up, and shouldn't require too much learning for the mainframe systems programmer or DBA.

The technique is suitable for a low transaction requirement, such as on a company intranet.

BIBLIOGRAPHY

IBM manuals:

- *z/OS Unix System Services User's Guide* SA22-7801-01
- *z/OS Unix System Services Command Reference* SA22-7802-01
- *Using REXX and z/OS Unix System Services* SA22-7806-01
- *DB2 Universal Database for OS/390 and z/OS Application Programming and SQL Guide* SC26-9933-02

Other:

- Guelich, Gundavaram, and Birznieks, *CGI Programming with Perl*, O'Reilly and Associates, 2000. How to write CGI programs with Perl, but the principles are the same.
- Laurie and Laurie, *Apache – the Definitive Guide*, O'Reilly and Associates, 1999.

Alan Smith
Norwich Union (UK)

Impact of dropping a table

The DRIM REXX procedure shows you which DB2 objects will be affected by the **drop table** statement. The **drop table** statement deletes a table and it deletes the row in the SYSIBM.SYSTABLES catalog table that contains information about the dropped table. It also drops any other objects that depend on the dropped table.

Before dropping a table, check to see what other objects are dependent on it. Drop may have the following effects:

- The column names of the table are dropped from SYSIBM.SYSCOLUMNS.
- If the dropped table has a check constraint, all information in the check column is removed from SYSIBM.SYSCHECKS tables.
- If the dropped table has an identity column, all information about the identity column is removed from SYSIBM.SYSSEQUENCES.
- Any views based on the table are dropped.
- All indexes based on the table are dropped.
- Synonyms for the table are dropped from SYSIBM.SYSSYNONYMS.
- Referential constraints that involve the table are dropped.
- Authorization information kept in the DB2 catalog authorization table is updated to reflect the dropping of the table. Users who were previously authorized to use the table, or views on it, no longer have those privileges.
- Access path statistics and space statistics for the table are deleted from the catalog.
- If the table space containing the table is implicitly created (using CREATE TABLE without the TABLESPACE clause), the table space is also dropped.

- Application plans or packages that involve the use of the table are invalidated.
- The packages of the stored procedures and user-defined functions that involve the use of the table are invalidated.
- If triggers are defined on the table, they are dropped, and the corresponding rows are removed from SYSIBM.SYSTRIGGERS and SYSIBM.SYSPACKAGES.

The tool, DRIM, runs in a TSO environment.

The DRIM procedure shows information about table space, plans, packages, indexes, views, referential integrity, stored procedure, user-defined function, triggers, and check constraints. This table occupied 477 plans, 521 packages, etc.

The drop impact list display looks like:

```
------------------  Drop Impact List Display   ------ Row 1 to 16 of 16
Command ===>                                             Scroll ===> PAGE

SSID DSNN  Tbcreator NADI   Tbname  TLØ57

Show what?: TS - PL -   PA -   IN -  VW -   RI Y   SP Y   UDF Y   TG Y
CC Y
Obj.counts: TS 1 PL 477 PA 521 IN 5  VW 1Ø  RI 3   SP 7   UDF 2   TG 3
CC 1
Select S Sql D Drop
S Type
- Function SYSADM.NASLOV L=PLI Package NPAR.NPAR
- Function SYSADM.SMMDAV L=PLI Package SMMDAV.SMMDAV
- Procedure KE.WUDAV L=SQL Package KEP.SQL39485
- Procedure KE.WUINS L=SQL Package KEP.SQLØ95ØØ
- Procedure SYSPROC.NASPAR L=PLI Package NASPAR.NASPAR
- Procedure SYSPROC.NASPARX L=PLI Package NASPARX.NASPARX
- Procedure SYSPROC.NASPARZ L=PLI Package NASPARZ.NASPARZ
- Procedure SYSPROC.NASPROC L=PLI Package NASPROC.NASPROC
- Procedure SYSPROC.POSPFPO L=PLI Package POSPFPO.POSPFPO
- RI-Child SYSADM.TL173
- RI-Child SYSADM.TL188
- RI-Child SYSADM.TL181
- Trigger NADI.TØØ1D, AFTER DELETE
- Trigger NADI.TØØ1I, AFTER INSERT
- Trigger NADI.TØØ1U, AFTER UPDATE
- Checkname PRO
```

It shows some DB2 objects (option Y in the *Show What?* fields). You can also select the S command to see DDL CREATE statements, or select the D command to generate DROP statements.

The components of DRIM are as follows:

- DRIM – the driver procedure
- DROPP1 – drop impact menu
- DRIMES – message panel
- SDRIM – ISPF skeleton.

DRIM

```
/* REXX */
/* DRIM Drop Impact List Display                              */
/* trace r */
 zpfctl = 'OFF'
 Y=MSG("OFF")
 address ispexec 'vput (zpfctl) profile'
 cur='ttb'
 TOP:
 sel=''; item=''; ccrp=''; ttbp=''
 Title='Drop Impact List Display'
 address ispexec "display panel(DROPP1) cursor("CUR")"
 if rc=8 then Exit
 SUB:
 Call Create_messg
 /* Check input parameters                                    */
 if ccr=' ' & ttb=' 'then do
    cur='ccr'
    zedsmsg = "Enter Catalog fields"
    zedlmsg = "Enter Tbcreator and Tbname"
    address ispexec "setmsg msg(isrzØØ1)"
    signal top
 end
 /* All fields "Y"                                            */
 if zcmd='Y'then do
    ztt='Y';zpl='Y';zpa='Y'; zin='Y';zvw='Y'
    zri='Y';zsp='Y';zudf='Y';ztg='Y';zcc='Y'
    zcmd=''
    Signal top
 end
 /* All fields "N"                                            */
 if zcmd='N'then do
```

```
   ztt='';zpl='';zpa=''; zin='';zvw=''
   zri='';zsp='';zudf='';ztg='';zcc=''
   zcmd=''
   Signal top
end
/* All fields "?"                                    */
if zcmd='?'then do
   ztt='?';zpl='?';zpa='?'; zin='?';zvw='?'
   zri='?';zsp='?';zudf='?';ztg='?';zcc='?'
   zcmd=''
   Signal top
end
/* Explain input parameters                          */
if ztt='?'then do
   cur='ztt'
   ztt=''
   zedsmsg = "TS: Tablespace"
   zedlmsg = "TS: Tablespace"
   address ispexec "setmsg msg(isrzØØ1)"
   Signal top
end
if zpl='?'then do
   cur='zpl'
   zpl=''
   zedsmsg = "PL: Plan"
   zedlmsg = "PL: Plan"
   address ispexec "setmsg msg(isrzØØ1)"
   Signal top
end
if zpa='?'then do
   cur='zpa'
   zpa=''
   zedsmsg = "PA: Package"
   zedlmsg = "PA: Package"
   address ispexec "setmsg msg(isrzØØ1)"
   Signal top
end
if zin='?'then do
   cur='zin'
   zin=''
   zedsmsg = "IN: Index"
   zedlmsg = "IN: Index"
   address ispexec "setmsg msg(isrzØØ1)"
   Signal top
end
if zvw='?'then do
   cur='zvw'
   zvw=''
   zedsmsg = "VW: View"
   zedlmsg = "VW: View"
   address ispexec "setmsg msg(isrzØØ1)"
   Signal top
end
if zri='?'then do
   cur='zri'
   zri=''
   zedsmsg = "Referential Integrity"
   zedlmsg = "RI: Referential Integrity"
   address ispexec "setmsg msg(isrzØØ1)"
   Signal top
end
if zsp='?'then do
   cur='zsp'
   zsp=''
   zedsmsg = "SP: Stored Procedure"
   zedlmsg = "SP: Stored Procedure"
   address ispexec "setmsg msg(isrzØØ1)"
   Signal top
end
if zudf='?'then do
   cur='zudf'
   zudf=''
   zedsmsg = "User Defined Function"
   zedlmsg = "UDF: User Defined Function"
   address ispexec "setmsg msg(isrzØØ1)"
   Signal top
end
if ztg='?'then do
   cur='ztg'
   ztg=''
   zedsmsg = "TG: Trigger"
   zedlmsg = "TG: Trigger"
   address ispexec "setmsg msg(isrzØØ1)"
   Signal top
end
if zcc='?'then do
   cur='zcc'
   zcc=''
   zedsmsg = "CC: Check Constraint"
   zedlmsg = "CC: Check Constraint"
   address ispexec "setmsg msg(isrzØØ1)"
   Signal top
end
HVT1=' '; HVC1=' '; HVI1=' '

/* DSNREXX Language Support                                */
Address TSO "SUBCOM DSNREXX"
IF RC THEN
S_RC = RXSUBCOM(ADD,DSNREXX,DSNREXX)
```

```
/* Numbers of DB2 objects                                  */
SSID = db2
ADDRESS DSNREXX "CONNECT" SSID

IF ccr=ccrp & ttb=ttbp then nop
else do
   messg = "Counter    DB2 Catalog    information"
   messg = time() || " " || messg
   Call Send_messg
   SQLSTMT= "SELECT TYPE,COUNT(*)                                  ",
   "FROM (                                                         ",
   "SELECT SUBSTR(Z.TEXT,1,POSSTR(Z.TEXT,' ')) TYPE,               ",
   "       SUBSTR(Z.TEXT,POSSTR(Z.TEXT,' ')+1) OBJ                 ",
   "FROM (                                                         ",
   "SELECT CHAR(CASE                                               ",
   "        WHEN ROUTINETYPE IS NULL THEN                          ",
   "          'PACKAGE '||STRIP(DCOLLID)                           ",
   "          ||'.'||DNAME                                         ",
   "        WHEN ROUTINETYPE='P' THEN                              ",
   "          'PROCEDURE '||STRIP(SCHEMA)                          ",
   "          ||'.'||STRIP(NAME)||' L='||                          ",
   "          STRIP(LANGUAGE)||                                    ",
   "          ' PACKAGE '||STRIP(COLLID)                           ",
   "          ||'.'||STRIP(EXTERNAL_NAME)                          ",
   "        WHEN ROUTINETYPE='F' THEN                              ",
   "          'FUNCTION '||STRIP(SCHEMA)                           ",
   "          ||'.'||STRIP(NAME)||' L='||                          ",
   "          STRIP(LANGUAGE)||                                    ",
   "          ' PACKAGE '||STRIP(COLLID)                           ",
   "          ||'.'||STRIP(EXTERNAL_NAME)                          ",
   "       END, 6Ø) TEXT                                           ",
   "FROM                                                           ",
   " (SELECT DCOLLID, DNAME                                        ",
   "    FROM SYSIBM.SYSPACKDEP                                     ",
   "   WHERE BQUALIFIER='"ccr"'                                    ",
   "     AND BNAME='"ttb"') P                                      ",
   "LEFT OUTER JOIN                                                ",
   "  (SELECT SCHEMA, NAME, ROUTINETYPE,                           ",
   "          LANGUAGE, COLLID, EXTERNAL_NAME                      ",
   "     FROM SYSIBM.SYSROUTINES) R                                ",
   " ON DCOLLID=COLLID                                             ",
   "AND DNAME=EXTERNAL_NAME                                        ",
   "UNION                                                          ",
   "SELECT CHAR('TRIGGER '||STRIP(SCHEMA)||'.'||STRIP(NAME) ||     ",
   "       CASE TRIGTIME                                           ",
   "        WHEN 'A' THEN ', AFTER '                               ",
   "        WHEN 'B' THEN ', BEFORE '                              ",
   "       END ||                                                  ",
   "       CASE TRIGEVENT                                          ",
   "        WHEN 'I' THEN 'INSERT'                                 ",
   "        WHEN 'U' THEN 'UPDATE'                                      ",
   "        WHEN 'D' THEN 'DELETE'                                      ",
   "       END,6Ø) TEXT                                                 ",
   "FROM SYSIBM.SYSTRIGGERS                                             ",
   "WHERE TBOWNER='"ccr"'                                               ",
   "  AND TBNAME='"ttb"'                                                ",
   "UNION                                                               ",
   "SELECT CHAR('VIEW '||STRIP(DCREATOR)||'.'                           ",
   "       ||STRIP(DNAME),6Ø) TEXT                                      ",
   "FROM SYSIBM.SYSVIEWDEP                                              ",
   "WHERE BCREATOR ='"ccr"'                                             ",
   "  AND BNAME    ='"ttb"'                                             ",
   "  AND BTYPE    = 'T'                                                ",
   "UNION                                                               ",
   "SELECT CHAR('VIEW '||STRIP(Y.DCREATOR)||'.'                         ",
   "       ||STRIP(Y.DNAME),6Ø) TEXT                                    ",
   "FROM SYSIBM.SYSVIEWDEP Y,                                           ",
   "(SELECT DCREATOR, DNAME                                             ",
   "   FROM SYSIBM.SYSVIEWDEP                                           ",
   "   WHERE BCREATOR ='"ccr"'                                          ",
   "     AND BNAME    ='"ttb"'                                          ",
   "     AND BTYPE    = 'T' ) X                                         ",
   " WHERE BCREATOR = X.DCREATOR                                        ",
   "   AND BNAME    = X.DNAME                                           ",
   "UNION                                                               ",
   "SELECT CHAR('INDEX '||STRIP(CREATOR)||'.'                           ",
   "    ||STRIP(NAME)||' UNIQUERULE='||UNIQUERULE,6Ø) TEXT              ",
   "FROM SYSIBM.SYSINDEXES                                              ",
   "WHERE TBCREATOR='"ccr"'                                             ",
   "  AND TBNAME   ='"ttb"'                                             ",
   "UNION                                                               ",
   "SELECT CHAR('RI '||STRIP(R.CREATOR)                                 ",
   "             ||'.'||STRIP(R.TBNAME),6Ø) TEXT                        ",
   "FROM SYSIBM.SYSRELS R,                                              ",
   "     SYSIBM.SYSFOREIGNKEYS F                                        ",
   "WHERE R.REFTBCREATOR='"ccr"'                                        ",
   "  AND R.REFTBNAME='"ttb"'                                           ",
   "  AND R.CREATOR=F.CREATOR                                           ",
   "  AND R.TBNAME=F.TBNAME                                             ",
   "  AND R.RELNAME=F.RELNAME                                           ",
   "UNION                                                               ",
   "SELECT CHAR('RI '|| STRIP(R.REFTBNAME)                              ",
   "        ||'.'||STRIP(R.REFTBCREATOR),6Ø) TEXT                       ",
   "FROM SYSIBM.SYSRELS R,                                              ",
   "     SYSIBM.SYSFOREIGNKEYS F                                        ",
   "WHERE R.REFTBCREATOR='"ccr"'                                        ",
   "  AND R.TBNAME='"ttb"'                                              ",
   "  AND R.CREATOR=F.CREATOR                                           ",
   "  AND R.TBNAME=F.TBNAME                                             ",
   "  AND R.RELNAME=F.RELNAME                                           ",
   "UNION                                                            ",
   "SELECT CHAR('CHECKNAME '||STRIP(CHECKNAME),6Ø) TEXT              ",
   "FROM SYSIBM.SYSCHECKDEP                                          ",
   "WHERE TBOWNER='"ccr"'                                            ",
   "  AND TBNAME='"ttb"'                                             ",
   "UNION                                                            ",
   "SELECT CHAR('PLAN '||STRIP(DNAME),6Ø) TEXT                       ",
   "FROM SYSIBM.SYSPLANDEP                                           ",
   "WHERE BCREATOR='"ccr"'                                           ",
   "  AND BNAME='"ttb"'                                              ",
   "  AND BTYPE='T'  ) Z ) Q                                         ",
   "GROUP BY TYPE                                                    ",
   "WITH UR                                                          ",

   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX "EXECSQL FETCH C1 INTO :HVT1, :HVC1"
   ctt=1;cpl=Ø;cpa=Ø;cin=Ø;cvw=Ø;cri=Ø;csp=Ø;cudf=Ø;ctg=Ø;ccc=Ø;
   do while(sqlcode=Ø)
      IF hvt1='PACKAGE'   then cpa=hvc1
      IF hvt1='PROCEDURE' then csp=hvc1
      IF hvt1='FUNCTION'  then cudf=hvc1
      IF hvt1='TRIGGER'   then ctg=hvc1
      IF hvt1='VIEW'      then cvw=hvc1
      IF hvt1='INDEX'     then cin=hvc1
      IF hvt1='CHECKNAME' then ccc=hvc1
      IF hvt1='RI'        then cri=hvc1
      IF hvt1='PLAN'      then cpl=hvc1
      Address DSNREXX "EXECSQL FETCH C1 INTO :HVT1, :HVC1"
   end
   Address DSNREXX "EXECSQL CLOSE C1"
end

address ispexec 'tbcreate "dlist" names(item)'
if ztt='Y' then do
   messg = "Select     SYSTABLESPACE  information"
   messg = time() || " " || messg
   Call Send_messg
   SQLSTMT= "SELECT CHAR('Tablespace '||STRIP(DBNAME)||'.'           ",
   "         ||STRIP(TSNAME),6Ø)                                      ",
   " FROM SYSIBM.SYSTABLES                                            ",
   " WHERE CREATOR='"ccr"'                                            ",
   "   AND NAME='"ttb"'                                               ",
   "   AND TYPE='T'                                                   ",
   "WITH UR                                                           "
   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1"
   item=hvi1
   Address DSNREXX "EXECSQL CLOSE C1"
   address ispexec 'tbadd "dlist"'
end
if zpl='Y' then do
   messg = "Select     SYSPLANDEP     information"
   messg = time() || " " || messg
   Call Send_messg
   SQLSTMT= "SELECT CHAR('Plan '||STRIP(DNAME),6Ø)                    ",
   " FROM SYSIBM.SYSPLANDEP                                          ",
   " WHERE BCREATOR='"ccr"'                                          ",
   "   AND BNAME='"ttb"'                                             ",
   "   AND BTYPE='T'                                                 ",
   " ORDER BY 1                                                      ",
   "WITH UR                                                          "
   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1"
   do while(sqlcode=Ø)
      item=hvi1
      address ispexec 'tbadd "dlist"'
      Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1"
   end
   Address DSNREXX "EXECSQL CLOSE C1"
end
if zpa='Y' | zsp='Y' | zudf='Y' then do
   messg = "Select     SYSPACKDEP     information"
   messg = time() || " " || messg
   Call Send_messg
   messg = "Select     SYSROUTINES    information"
   messg = time() || " " || messg
   Call Send_messg
   SQLSTMT= "SELECT CHAR(CASE                                          ",
   " WHEN ROUTINETYPE IS NULL THEN                                   ",
   "   'Package '||STRIP(DCOLLID)                                    ",
   "   ||'.'||DNAME                                                  ",
   " WHEN ROUTINETYPE='P' THEN                                       ",
   "   'Procedure '||STRIP(SCHEMA)                                   ",
   "   ||'.'||STRIP(NAME)||' L='||                                   ",
   "   STRIP(LANGUAGE)||                                             ",
   "   ' Package '||STRIP(COLLID)                                    ",
   "   ||'.'||STRIP(EXTERNAL_NAME)                                   ",
   " WHEN ROUTINETYPE='F' THEN                                       ",
   "   'Function '||STRIP(SCHEMA)                                    ",
   "   ||'.'||STRIP(NAME)||' L='||                                   ",
   "   STRIP(LANGUAGE)||                                             ",
   "   ' Package '||STRIP(COLLID)                                    ",
   "   ||'.'||STRIP(EXTERNAL_NAME)                                   ",
   " END, 6Ø) TEXT                                                   ",
   " FROM                                                            ",
   "  (SELECT DCOLLID, DNAME                                         ",
   "     FROM SYSIBM.SYSPACKDEP                                      ",
   "    WHERE BQUALIFIER='"ccr"'                                     ",
   "      AND BNAME='"ttb"') P                                       ",
   " LEFT OUTER JOIN                                                 ",
   "   (SELECT SCHEMA, NAME, ROUTINETYPE,                            ",
   "           LANGUAGE, COLLID, EXTERNAL_NAME                       ",
   "      FROM SYSIBM.SYSROUTINES) R                                 ",
   "  ON DCOLLID=COLLID                                              ",
   " AND DNAME=EXTERNAL_NAME                                         ",
   " ORDER BY 1                                                      ",
   "WITH UR                                                          "
   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1"
   do while(sqlcode=Ø)
      if zsp= 'Y' & word(hvi1,1)='Procedure' then do
         item=hvi1
         address ispexec 'tbadd "dlist"'
      end
      if zpa= 'Y' & word(hvi1,1)='Package' then do
         item=hvi1
         address ispexec 'tbadd "dlist"'
      end
      if zudf='Y' & word(hvi1,1)='Function' then do
         item=hvi1
         address ispexec 'tbadd "dlist"'
      end
      Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1"
   end
   Address DSNREXX "EXECSQL CLOSE C1"
end
if zin='Y' then do
   messg = "Select     SYSINDEXES     information"
   messg = time() || " " || messg
   Call Send_messg
   SQLSTMT= "SELECT char('Index '||strip(creator)||'.'              ",
   " ||strip(NAME)||' Uniquerule='||UNIQUERULE,6Ø)                   ",
   " FROM SYSIBM.SYSINDEXES                                          ",
   " WHERE TBCREATOR='"ccr"'                                         ",
   "   AND TBNAME='"ttb"'                                            ",
   " ORDER BY 1                                                      ",
   "WITH UR                                                          "
   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1"
   do while(sqlcode=Ø)
```

```
      item=hvi1
      address ispexec 'tbadd "dlist"'
      Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1"
   end
   Address DSNREXX "EXECSQL CLOSE C1"
end
if zvw='Y' then do
   messg = "Select     SYSVIEWDEP     information"
   messg = time() || " " || messg
   Call Send_messg
   SQLSTMT= "SELECT char('View '||strip(DCREATOR)||'.'             ",
   " ||strip(DNAME),6Ø)                                             ",
   " FROM SYSIBM.SYSVIEWDEP                                         ",
   " WHERE BCREATOR='"ccr"'                                         ",
   "   AND BNAME='"ttb"'                                            ",
   "   AND BTYPE='T'                                                ",
   " UNION                                                          ",
   " SELECT char('View '||strip(Y.DCREATOR)||'.'                    ",
   "        ||strip(Y.DNAME),6Ø)                                    ",
   " FROM SYSIBM.SYSVIEWDEP Y,                                      ",
   " (SELECT DCREATOR, DNAME                                        ",
   "    FROM SYSIBM.SYSVIEWDEP                                      ",
   "    WHERE BCREATOR ='"ccr"'                                     ",
   "      AND BNAME    ='"ttb"'                                     ",
   "      AND BTYPE    = 'T' ) X                                    ",
   "  WHERE BCREATOR = X.DCREATOR                                   ",
   "    AND BNAME    = X.DNAME                                      ",
   " ORDER BY 1                                                     ",
   "WITH UR                                                         "
   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1"
   do while(sqlcode=Ø)
      item=hvi1
      address ispexec 'tbadd "dlist"'
      Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1"
   end
   Address DSNREXX "EXECSQL CLOSE C1"
end
if zri='Y' then do
   messg = "Select     SYSRELS        information"
   messg = time() || " " || messg
   Call Send_messg
   messg = "Select     SYSFOREIGNKEYS information"
   messg = time() || " " || messg
   Call Send_messg
   SQLSTMT= "SELECT char('RI-Child '||strip(r.creator)             ",
   " ||'.'||strip(R.TBNAME),6Ø)                                     ",
   " FROM SYSIBM.SYSRELS R,                                         ",
   "      SYSIBM.SYSFOREIGNKEYS F                                   ",
   " WHERE R.REFTBCREATOR='"ccr"'                                   ",
   "   AND R.REFTBNAME='"ttb"'                                      ",
   "   AND R.CREATOR=F.CREATOR                                      ",
   "   AND R.TBNAME=F.TBNAME                                        ",
   "   AND R.RELNAME=F.RELNAME                                      ",
   " UNION                                                          ",
   " SELECT char('RI-Father '|| strip(r.reftbcreator)               ",
   " ||'.'||strip(r.reftbname),6Ø)                                  ",
   " FROM SYSIBM.SYSRELS R,                                         ",
   "      SYSIBM.SYSFOREIGNKEYS F                                   ",
   " WHERE R.REFTBCREATOR='"ccr"'                                   ",
   "   AND R.TBNAME='"ttb"'                                         ",
   "   AND R.CREATOR=F.CREATOR                                      ",
   "   AND R.TBNAME=F.TBNAME                                        ",
   "   AND R.RELNAME=F.RELNAME                                      ",
   " ORDER BY 1 DESC                                                ",
   "WITH UR                                                         "
   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1"
   do while(sqlcode=Ø)
      item=hvi1
      address ispexec 'tbadd "dlist"'
      Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1"
   end
   Address DSNREXX "EXECSQL CLOSE C1"
end
if ztg='Y' then do
   messg = "Select     SYSTRIGGERS    information"
   messg = time() || " " || messg
   Call Send_messg
   SQLSTMT= "SELECT CHAR('Trigger '||strip(SCHEMA)                  ",
   "   ||'.'||strip(NAME) ||                                        ",
   "         case TRIGTIME                                          ",
   "          when 'A' then ', AFTER '                              ",
   "          when 'B' then ', BEFORE '                             ",
   "         end ||                                                 ",
   "         case TRIGEVENT                                         ",
   "          when 'I' then 'INSERT'                                ",
   "          when 'U' then 'UPDATE'                                ",
   "          when 'D' then 'DELETE'                                ",
   "         end,6Ø)                                                ",
   " FROM SYSIBM.SYSTRIGGERS                                        ",
   " WHERE TBOWNER = '"ccr"'                                        ",
   "   AND TBNAME='"ttb"'                                           ",
   " ORDER BY 1                                                     ",
   "WITH UR                                                         "
   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1"
   do while(sqlcode=Ø)
      item=hvi1
      address ispexec 'tbadd "dlist"'
      Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1"
   end
   Address DSNREXX "EXECSQL CLOSE C1"
end
if zcc='Y' then do
   messg = "Select     SYSCHECKDEP    information"
   messg = time() || " " || messg
   Call Send_messg
   SQLSTMT= "SELECT char('Checkname '||strip(checkname),6Ø)         ",
   " FROM SYSIBM.SYSCHECKDEP                                        ",
   " WHERE TBOWNER = '"ccr"'                                        ",
   "   AND TBNAME='"ttb"'                                           ",
   " ORDER BY 1                                                     ",
   "WITH UR                                                         "
   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1"
   do while(sqlcode=Ø)
      item=hvi1
      address ispexec 'tbadd "dlist"'
      Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1"
   end
   Address DSNREXX "EXECSQL CLOSE C1"
end
address ispexec 'tbtop "dlist"'
ccrp=ccr
ttbp=ttb
JUMP:
address ispexec 'tbdispl "dlist" panel(DROPP1)'
if rc=8 then do
   address ispexec 'tbend "dlist"'
   address ispexec "tbend "messdb""
   Exit
end
if sel='S' | sel='D' then
   address ispexec 'tbcreate "ilist" names(irow)'
if sel='S' then do
   if word(item,1)='Checkname' then Call Check_S
   if word(item,1)='View'      then Call View_S
   if substr(item,1,2)='RI'    then Call RI_S
   if word(item,1)='Index'     then Call Index_S
end
if sel='D' then do
```

```
   if word(item,1)='Checkname' then Call Check_D
   if word(item,1)='View'      then Call View_D
   if substr(item,1,2)='RI'    then Call RI_S
   if word(item,1)='Index'     then Call Index_D
end
if sel='S' | sel='D' then nop
else do
   address ispexec 'tbend "dlist"'
   address ispexec "tbend "messdb""
   sel=''
   Signal Sub
end

messg = 'Building the DB2 Statements'
messg = time() || " " || messg
Call Send_messg
/* DB2 Statement                          */
date=date()
time=time(c)
user=userid()
tempfile=userid()||'.SQL.DRIM'
address tso
"delete '"tempfile"'"
"free dsname('"tempfile"')"
"free ddname(ispfile)"
"free attrlist(formfile)"
"attrib formfile blksize(8ØØ) lrecl(8Ø) recfm(f b) dsorg(ps)"
"alloc ddname(ispfile) dsname('"tempfile"')",
       "new using (formfile) unit(339Ø) space(1 1) cylinders"
address ispexec
"ftopen"
"ftincl SDRIM"
"ftclose"
zedsmsg = "SQL shown"
zedlmsg = "SQL Statement shown"
"setmsg msg(isrzØØ1)"
"edit dataset('"tempfile"')"
address ispexec 'tbend "ilist"'
ccr=ccrp
ttb=ttbp
sel=''
Signal Jump
Check_S:
 chn=word(item,2)
 SQLSTMT= "SELECT replace(checkcondition,'  ','')                ",
 " FROM SYSIBM.SYSCHECKS                                         ",
 " WHERE TBOWNER = '"ccr"'                                       ",
 "   AND TBNAME='"ttb"'                                          ",
 "   AND CHECKNAME='"chn"'                                       ",
 "WITH UR                                                        "
 Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
 Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
 Address DSNREXX "EXECSQL OPEN C1"
 Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1"
 Address DSNREXX "EXECSQL CLOSE C1"
 irow='ALTER TABLE '||ccr||'.'||ttb
 address ispexec 'tbadd "ilist"'
 irow='      ADD CONSTRAINT '||chn
 address ispexec 'tbadd "ilist"'
 if length(hvi1) > 5Ø
 then do
   irow=''
   irow='      CHECK ('
   do i=1 to length(hvi1)
      irow=irow||substr(hvi1,i,1)
      if length(irow)>5Ø & substr(hvi1,i,1)=' ' then do
         address ispexec 'tbadd "ilist"'
         irow='             '
      end
   end
 end
 else irow='      CHECK ('||hvi1
 address ispexec 'tbadd "ilist"'
 irow='             );'
 address ispexec 'tbadd "ilist"'
 address ispexec 'tbtop "ilist"'
Return
Check_D:
 chn=word(item,2)
 irow='ALTER TABLE '||ccr||'.'||ttb
 address ispexec 'tbadd "ilist"'
 irow='      DROP CONSTRAINT '||chn||';'
 address ispexec 'tbadd "ilist"'
 address ispexec 'tbtop "ilist"'
Return
View_S:
 view=''
 own = word(translate(word(item,2),' ','.'),1)
 tab = word(translate(word(item,2),' ','.'),2)
 SQLSTMT= "SELECT replace(text,'  ','')                              ",
 " FROM SYSIBM.SYSVIEWS                                              ",
 " WHERE CREATOR = '"own"'                                           ",
 "   AND NAME = '"tab"'                                              ",
 " ORDER BY SEQNO                                                    ",
 "WITH UR                                                            "
 Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
 Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
 Address DSNREXX "EXECSQL OPEN C1"
 Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1"
 do while(sqlcode=Ø)
```

```
      view=view||hvi1
      Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1"
   end
   Address DSNREXX "EXECSQL CLOSE C1"
   if length(view) > 5Ø
   then do
     irow=''
     do i=1 to length(view)
        irow=irow||substr(view,i,1)
        if length(irow)>5Ø &,
           (substr(view,i,1)=' ' | substr(view,i,1)=',') then do
           address ispexec 'tbadd "ilist"'
           irow=''
        end
     end
   end
   else irow=view
   address ispexec 'tbadd "ilist"'
   address ispexec 'tbtop "ilist"'
  Return
  View_D:
   own = word(translate(word(item,2),' ','.'),1)
   tab = word(translate(word(item,2),' ','.'),2)
   irow='DROP VIEW '||own||'.'||tab||';'
   address ispexec 'tbadd "ilist"'
   address ispexec 'tbtop "ilist"'
  Return
  RI_S:
   view=''
   own = word(translate(word(item,2),' ','.'),1)
   tab = word(translate(word(item,2),' ','.'),2)
   if word(item,1)='RI-Child' then do
      SQLSTMT= "SELECT 'ALTER TABLE '||STRIP(R.CREATOR)||'.'||      ",
      " STRIP(R.TBNAME)||' FOREIGN KEY '||STRIP(R.RELNAME),         ",
      " F.COLNAME, 'REFERENCES '||STRIP(R.REFTBCREATOR)             ",
      " ||'.'||STRIP(R.REFTBNAME)||                                 ",
      " CASE DELETERULE                                             ",
      "   WHEN 'C' THEN ' ON DELETE CASCADE'                        ",
      "   WHEN 'R' THEN ' ON DELETE RESTRICT'                       ",
      "   WHEN 'S' THEN ' ON DELETE SET NULL'                       ",
      " END                                                         ",
      " FROM SYSIBM.SYSRELS R,                                      ",
      "      SYSIBM.SYSFOREIGNKEYS F                                ",
      " WHERE R.REFTBCREATOR='"ccr"'                                ",
      "   AND R.REFTBNAME='"ttb"'                                   ",
      "   AND R.CREATOR='"own"'                                     ",
      "   AND R.TBNAME='"tab"'                                      ",
      "   AND R.CREATOR=F.CREATOR                                   ",
      "   AND R.TBNAME=F.TBNAME                                     ",
      "   AND R.RELNAME=F.RELNAME                                   ",
      " ORDER BY 1,2,3,F.COLSEQ                                    ",
      "WITH UR                                                     "
   end
   if word(item,1)='RI-Father' then do
      SQLSTMT= "SELECT 'ALTER TABLE '||STRIP(R.CREATOR)||'.'||      ",
      " STRIP(R.TBNAME)||' FOREIGN KEY '||STRIP(R.RELNAME),        ",
      " F.COLNAME, 'REFERENCES '||STRIP(R.REFTBCREATOR)            ",
      " ||'.'||STRIP(R.REFTBNAME)||                                ",
      " CASE DELETERULE                                            ",
      "   WHEN 'C' THEN ' ON DELETE CASCADE'                       ",
      "   WHEN 'R' THEN ' ON DELETE RESTRICT'                      ",
      "   WHEN 'S' THEN ' ON DELETE SET NULL'                      ",
      " END                                                        ",
      " FROM SYSIBM.SYSRELS R,                                     ",
      "      SYSIBM.SYSFOREIGNKEYS F                               ",
      " WHERE R.REFTBCREATOR='"own"'                               ",
      "   AND R.REFTBNAME='"tab"'                                  ",
      "   AND R.CREATOR='"ccr"'                                    ",
      "   AND R.TBNAME='"ttb"'                                     ",
      "   AND R.CREATOR=F.CREATOR                                  ",
      "   AND R.TBNAME=F.TBNAME                                    ",
      "   AND R.RELNAME=F.RELNAME                                  ",
      " ORDER BY 1,2,3,F.COLSEQ                                    ",
      "WITH UR                                                     "
   end
   Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
   Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
   Address DSNREXX "EXECSQL OPEN C1"
   Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1, :HVI2, :HVI3"
   irow=hvi1
   if sel='D' then do
      irow=subword(hvi1,1,3)||' DROP '||subword(hvi1,4)||';'
      address ispexec 'tbadd "ilist"'
      Address DSNREXX "EXECSQL CLOSE C1"
      address ispexec 'tbtop "ilist"'
      Return
   end
   address ispexec 'tbadd "ilist"'
   k=Ø
   do while(sqlcode=Ø)
      if k=Ø
      then irow='      ('||hvi2
      else irow='      ,'||hvi2
      Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1, :HVI2, :HVI3"
      if sqlcode=1ØØ then irow=irow||')'
      address ispexec 'tbadd "ilist"'
      k=1
   end
   irow=hvi3||';'
   address ispexec 'tbadd "ilist"'
```

```
 Address DSNREXX "EXECSQL CLOSE C1"
 address ispexec 'tbtop "ilist"'
Return
Index_S:

 own = word(translate(word(item,2),' ','.'),1)
 inx = word(translate(word(item,2),' ','.'),2)
 SQLSTMT= "SELECT 1,CHAR('CREATE' CONCAT                          ",
 "         CASE UNIQUERULE                                       ",
 "          WHEN 'D' THEN '' ELSE ' UNIQUE'                      ",
 "         END CONCAT ' INDEX ' ||                               ",
 "         STRIP(CREATOR)||'.'||STRIP(NAME)||' ON '||            ",
 "         STRIP(TBCREATOR)||'.'||STRIP(TBNAME),7Ø)              ",
 " FROM SYSIBM.SYSINDEXES                                        ",
 " WHERE CREATOR='"own"'                                         ",
 "   AND NAME='"inx"'                                            ",
 "UNION                                                          ",
 "SELECT 1+COLSEQ,                                               ",
 "        CHAR(CASE                                              ",
 "         WHEN COLSEQ=1                                         ",
 "          THEN '       ( '||COLNAME                            ",
 "          ELSE '       , '||COLNAME                            ",
 "        END CONCAT                                             ",
 "        CASE ORDERING                                          ",
 "         WHEN 'A' THEN ' ASC'                                  ",
 "         WHEN 'D' THEN ' DESC'                                 ",
 "        END CONCAT                                             ",
 "        CASE                                                   ",
 "         WHEN COLCOUNT=COLSEQ                                  ",
 "         THEN ' )'                                             ",
 "         ELSE ''                                               ",
 "        END CONCAT                                             ",
 "        CASE                                                   ",
 "         WHEN CLUSTERING='Y'                                   ",
 "         THEN ' CLUSTER'                                       ",
 "         ELSE ''                                               ",
 "        END,7Ø)                                                ",
 "FROM SYSIBM.SYSINDEXES X,                                      ",
 "      SYSIBM.SYSINDEXPART P,                                   ",
 "      SYSIBM.SYSKEYS K                                         ",
 "WHERE X.CREATOR='"own"'                                        ",
 "  AND X.NAME='"inx"'                                           ",
 "  AND X.CREATOR=P.IXCREATOR                                    ",
 "  AND X.NAME=P.IXNAME                                          ",
 "  AND X.CREATOR=K.IXCREATOR                                    ",
 "  AND X.NAME=K.IXNAME                                          ",
 "UNION ALL                                                      ",
 "SELECT 1ØØ+P.PARTITION*1Ø,                                     ",
 "  CHAR(CASE                                                    ",
 "   WHEN P.PARTITION=1                                          ",
 "   THEN '       ( PART 1 VALUES ('||                           ",
 "        STRIP(T.LIMITKEY)||')'                                 ",
 "   ELSE '       , PART '||STRIP(CHAR(P.PARTITION))||           ",
 "        ' VALUES ('||STRIP(T.LIMITKEY)||')'                    ",
 "  END,7Ø)                                                      ",
 "FROM SYSIBM.SYSINDEXES X,                                      ",
 "     SYSIBM.SYSINDEXPART P,                                    ",
 "     SYSIBM.SYSTABLEPART T                                     ",
 "WHERE CREATOR='"own"'                                          ",
 "  AND X.NAME='"inx"'                                           ",
 "  AND X.CREATOR=P.IXCREATOR                                    ",
 "  AND X.NAME=P.IXNAME                                          ",
 "  AND T.IXCREATOR = X.CREATOR                                  ",
 "  AND T.IXNAME    = X.NAME                                     ",
 "  AND T.PARTITION = P.PARTITION                                ",
 "UNION ALL                                                      ",
 "SELECT 1Ø1+P.PARTITION*1Ø,                                     ",
 "   CHAR('         USING STOGROUP ' ||                          ",
 "       STORNAME,7Ø)                                            ",
 "FROM SYSIBM.SYSINDEXES X,                                      ",
 "     SYSIBM.SYSINDEXPART P                                     ",
 "WHERE CREATOR='"own"'                                          ",
 "  AND NAME='"inx"'                                             ",
 "  AND X.CREATOR=P.IXCREATOR                                    ",
 "  AND X.NAME=P.IXNAME                                          ",
 "UNION ALL                                                      ",
 "SELECT 1Ø2+P.PARTITION*1Ø,                                     ",
 "  '         PRIQTY ' ||STRIP(CHAR(PQTY*4))||                   ",
 "  ' SECQTY '||STRIP(CHAR(SECQTYI*4))                           ",
 "FROM SYSIBM.SYSINDEXES X,                                      ",
 "     SYSIBM.SYSINDEXPART P                                     ",
 "WHERE CREATOR='"own"'                                          ",
 "  AND NAME='"inx"'                                             ",
 "  AND X.CREATOR=P.IXCREATOR                                    ",
 "  AND X.NAME=P.IXNAME                                          ",
 "UNION                                                          ",
 "SELECT 1Ø3+P.PARTITION*1Ø,                                     ",
 "  '         FREEPAGE ' ||STRIP(CHAR(FREEPAGE))||               ",
 "  ' PCTFREE '||STRIP(CHAR(PCTFREE))                            ",
 "FROM SYSIBM.SYSINDEXES X,                                      ",
 "     SYSIBM.SYSINDEXPART P                                     ",
 "WHERE CREATOR='"own"'                                          ",
 "  AND NAME='"inx"'                                             ",
 "  AND X.CREATOR=P.IXCREATOR                                    ",
 "  AND X.NAME=P.IXNAME                                          ",
 "UNION ALL                                                      ",
 "SELECT DISTINCT 1ØØØ, CASE                                     ",
 "       WHEN PARTITION>Ø                                        ",
 "        THEN '       ) BUFFERPOOL '||STRIP(BPOOL)              ",
 "        ELSE '         BUFFERPOOL '||STRIP(BPOOL)              ",
 "       END                                                      ",
 "FROM SYSIBM.SYSINDEXES X,                                       ",
 "     SYSIBM.SYSINDEXPART P                                      ",
 "WHERE CREATOR='"own"'                                           ",
 "  AND NAME='"inx"'                                              ",
 "  AND X.CREATOR=P.IXCREATOR                                     ",
 "  AND X.NAME=P.IXNAME                                           ",
 " ORDER BY 1                                                     ",
 "WITH UR                                                         "
 Address DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
 Address DSNREXX 'EXECSQL PREPARE S1 FROM :SQLSTMT'
 Address DSNREXX "EXECSQL OPEN C1"
 Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1, :HVI2"
 do while(sqlcode=Ø)
    irow=hvi2
    address ispexec 'tbadd "ilist"'
    Address DSNREXX "EXECSQL FETCH C1 INTO :HVI1, :HVI2"
 end
 Address DSNREXX "EXECSQL CLOSE C1"
 address ispexec 'tbtop "ilist"'
Return
Index_D:
 own = word(translate(word(item,2),' ','.'),1)
 inx = word(translate(word(item,2),' ','.'),2)
 irow='DROP INDEX '||own||'.'||inx||';'
 address ispexec 'tbadd "ilist"'
 address ispexec 'tbtop "ilist"'
Return
Create_messg:
 messg = "s"||userid()
 address ispexec "tbcreate "messdb" names(messg) write replace"
Return
Send_messg:
 address ispexec "tbadd " messdb
 address ispexec "control display lock "
 address ispexec "addpop row(13) column(6)"
 address ispexec "tbdispl "messdb" panel(drimes)"
 address ispexec rempop
Return
```

DROPP1

```
)Attr Default(%+_)
   | type(text)   intens(high) caps(on ) color(yellow)
   $ type(output) intens(high) caps(off) color(yellow) hilite(reverse)
   § type(output) intens(high) caps(off) color(white) hilite(reverse)
   # type(text)   intens(high) caps(off) hilite(reverse)
   } type(text)   intens(high) caps(off) color(white)
   [ type( input) intens(high) caps(on ) just(left ) pad('_')
   { type( input) intens(high) caps(on ) just(left ) pad('_')
   ] type( input) intens(high) caps(on ) just(left ) pad('-')
   ^ type(output) intens(low ) caps(off) just(asis ) color(white)
)Body  Expand(//)
%-/-/- $title                   +%-/-/-
%Command ===>_zcmd                                      / /%Scroll
===>_amt +
+
+SSID[db2 + Tbcreator[ccr     +Tbname [ttb               +
+
+Show what?: TS]z+PL]z+  PA]z+  IN]z+ VW]z+  RI]z+  SP]z+  UDF]z+  TG]z+
CC]z+
+Obj.counts: TS^z+PL^z  +PA^z  +IN^z +VW^z  +RI^z  +SP^z  +UDF^z  +TG^z
+CC^z +
+Select|S+Sql|D+Drop
#S#Type
+
)Model
]z^z
+
)Init
  .ZVARS= '(ztt zpl zpa zin zvw zri zsp zudf ztg zcc +
            ctt cpl cpa cin cvw cri csp cudf ctg ccc sel item)'
  &amt = PAGE
  &cmd = ''
  if (&ccr ¬= ' ')
      .attr (ccr) = 'pad(nulls)'
  if (&ttb ¬= ' ')
      .attr (ttb) = 'pad(nulls)'
)Reinit
)Proc
  VPUT (db2, ccr, ttb) PROFILE
  VPUT (ztt, zpl, zpa, zin, zvw, zri, zsp, zudf, ztg, zcc) PROFILE
)End
```

DRIMES

```
)ATTR DEFAULT(%+_)
| TYPE (TEXT)   INTENS(LOW)  COLOR(WHITE)
@ TYPE (TEXT)   INTENS(HIGH) COLOR(RED)   CAPS(OFF) HILITE(REVERSE)
| TYPE (INPUT)  INTENS(NON)  COLOR(GREEN) CAPS(ON)  JUST(LEFT)
# TYPE (OUTPUT) INTENS(LOW)  COLOR(GREEN) CAPS(OFF)
)BODY DEFAULT(%~\) WINDOW(6Ø,8)
!ZCMD +            @ Message display !AMT  |
|------------------------------------------------------------
)MODEL CLEAR(MESSG)
#Z                                        +
)INIT
     .ZVARS = '(MESSG)'
```

```
)REINIT
)PROC
  IF (.PFKEY = PFØ3) &PF3 = EXIT
  IF (&ZCMD=END)
     &COMMAND = CANCEL
)END
```

SDRIM

```
)TBA 72
)CM ------------------------------------------------------------------
)CM Skeleton - Drop Impacat                                         --
)CM ------------------------------------------------------------------
)DOT "ILIST"
 &irow
)ENDDOT
```

Bernard Zver (Bernard.zver@informatika.si)
DBA
Informatika (Slovenia)

UDB V8 LUW – different ways of deleting rows from a table

This article looks at the problem of how you can quickly and easily delete all the rows in a DB2 UDB table. In the following sections I will look at four possible ways of doing this and point out some of the advantages/disadvantages of each method.

I ran all the SQL in this article on a Windows 2000 machine running DB2 UDB 8.1 FP1.

As a starting point for all three methods, I created a table called EMP, which is a copy of the SAMPLE database table EMPLOYEE, and enabled archive logging:

```
>db2 connect to sample;
>db2 create table EMP like EMPLOYEE;
>db2 insert into EMP select * from EMPLOYEE;
>db2 update db cfg for sample using logretain recovery;
>db2 connect reset;
>db2 backup db sample to c:\backups
```

The load command detailed below will work only if you have logretain switched on (as the command uses the **copy yes** parameter). I then loaded this table 16 times into itself (using the commands below), so that I ended up with an EMP table of about 1 million rows:

```
>db2 EXPORT TO C:\temp\emp.txt OF DEL MESSAGES c:\temp\msgs.txt SELECT *
FROM DB2ADMIN.EMP

>db2 load from c:\temp\emp.txt of del messages c:\temp\msgs.txt insert
into db2admin.emp copy yes to c:\backups
```

I also switched on infinite logging as follows:

```
>db2 update db cfg for sample using userexit on;
>db2 update db cfg for sample using logsecond –1
```

Now let’s look at the different ways of deleting the rows from the table.

THE DELETE COMMAND

Using the DELETE command is the most obvious! You simply issue:

```
>db2 delete from emp
```

The problem with this command is that DB2 logs every row that is deleted. This is not a problem for small tables, but if you have a table like ours, with over 1 million rows in it, the command could take 25 minutes to complete and use a lot of log space (when I ran this, I switched on infinite logging, as I kept getting 'SQL0964C The transaction log for the database is full.' messages). It took 132 logs (default size 250 x 4KB) to store all the information about the delete statement.

DROPPING THE TABLE

We could delete the rows by dropping the table. This would save on logging, but means that all grants, views, etc, on the table would also be dropped, and any plans that access the table would be invalidated. You would then have to recreate the table and put back all the grants, views, etc. This could involve a lot of work, and I would use it only for very simple tables (where I already have all the DDL I need!).

THE LOAD COMMAND

One method to delete a large number of rows from a table without affecting the grants, views, etc, based on the table, is the LOAD method.

With this method, you load an empty file into the table you want to empty, which doesn't involve logging the deleted rows and allows you to keep all the authorities you have on the table.

First create an empty text file called *c:\temp\empty.txt*. Then issue the command:

```
>db2 load from c:\temp\empty.txt of del messages c:\temp\msg.txt replace
into db2admin.emp copy yes to c:\backups
```

Note that I specify the **copy yes** parameter. If you don't specify this, you won't be able to use the table space without first doing a back-up.

For me, this command took about two seconds to execute. If you now select from the table you see:

```
>db2 select count(*) from emp

1
-----------
          Ø
```

So we have deleted all the rows in the table. Now try and insert a row into the table (the row is taken from the EMPLOYEE table):

```
>db2 insert into emp
values('ØØØ1Ø','CHRISTINE','I','HAAS','AØØ','3978','Ø1/Ø1/
1965','PRES',18,'F','24/Ø8/1933'",5275Ø.ØØ,1ØØØ.ØØ,422Ø.ØØ)
```

Note that the insert command works without having to take a back-up because we used the **copy yes** parameter in the load statement.

THE ALTER TABLE COMMAND

The best method for deleting a large number of rows from a table without affecting the grants, views, etc, based on the table, is the ALTER TABLE method. You simply issue the command:

```
>db2 alter table emp activate not logged initially with empty table
```

You do not need to have created the table with the **not logged initially** option to use the command.

So what is the best way to delete a large number of rows from a table? If your database uses circular logging, you can't use the **copy yes** parameter of the LOAD command, which means you will have to take an off-line back-up after the LOAD command. You might not be able to do this (because it will lock out other users from using the database/tablespace), so this option won't be of any use to you.

If you are using archive logging, the LOAD option is quicker than using the DELETE command. You could always drop and recreate the table, but that option together with the DELETE command would always be my last choice!

The best option seems to be the ALTER TABLE command – it is quick and simple to use.

C Leonard
Freelance Consultant (UK)

DB2 news

Quest Software has announced Quest Central for Databases 4.0, the first database management solution to combine domain-specific functionality and performance management into a single console for heterogeneous database environments including DB2 (distributed and mainframe), Oracle, and SQL Server.

The product provides built-in expertise and administration capabilities across all three database platforms. Quest Central for Databases minimizes database differences and removes platform barriers, allowing DBAs to cross train to maximize the productivity of existing staff.

For further information contact:
Quest Software, 8001 Irvine Center Drive, Irvine, CA 92618, USA.
Tel: (949) 754 8000.
URL: http://www.quest.com/quest_central/db2/.

* * *

Embarcadero Technologies has announced new powerful administration and navigation enhancements for DB2 z/OS and OS/390 in Version 7.3 of DBArtisan.

The new features offer z/OS and OS/390 users more administration abilities by providing an intuitive interface to execute DB2 utilities. In addition, DB2 z/OS and OS/390 administrators will find improved usability enhancements and support for data sharing in these environments.

DBArtisan 7.3 delivers the industry's broadest cross-platform support, allowing DBAs to administer Oracle, Microsoft SQL Server, IBM DB2 for Windows/Unix/Linux, DB2 z/OS and OS/390, and Sybase databases from a single console.

For further information contact:
Embarcadero Technologies, 425 Market Street, Suite 425, San Francisco, CA 94105, USA.
Tel: (415) 834 3131.
URL: http://www.embarcadero.com/products/dbartisan/index.asp.

* * *

IBM has announced the availability of Migrate Now! for DB2 UDB Version 8.1. This facilitates migration from Oracle, Sybase, Microsoft SQL Server, and other database platforms to DB2 UDB V8.1 at a special price.

For further information contact your local IBM representative.
URL: http://ibm.com/db2/migration.

* * *

Embarcadero has announced Change Manager Version 2.5, its cross-platform change management product.

Version 2.5 incorporates new change-based notification features, comprehensive HTML reporting capabilities designed to facilitate group workflow, and a new-look visual differences window with advanced compare criteria.

Embarcadero Change Manager supports the following database platforms and versions: DB2 for Windows/Unix/Linux, Oracle, Sybase, and Microsoft SQL Server.

For further information contact:
Embarcadero Technologies, 425 Market Street, Suite 425, San Francisco, CA 94105, USA.
Tel: (415) 834 3131.
URL: http://www.embarcadero.com/products/changemanager/index.asp.