111

DB2

January 2002

In this issue

update

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 1997 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

Disclaimer

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

Contributions

When Xephon is given copyright, articles published in *DB2 Update* are paid for at the rate of £170 ($260) per 1000 words and £100 ($160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 ($80) per 100 lines. In addition, there is a flat fee of £30 ($50) per article. 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.

Printed in England.

Time dimension table

The time dimension table occupies a special place in every data warehouse because almost every data warehouse fact table is time dependent. This table lets us compare days and weeks across separate years, lets us find month ends, and lets us label our private seasons and fiscal periods. The big reason for this time table is to remove every last vestige of calendar knowledge from our actual applications. We drive calendar navigation through the time dimension table, not through hard-coded logic.

Here is a recommended time dimension table:

```
CREATE TABLE owner.TIME_DIMENSION
     (ID_TIME              INTEGER
     ,FULL_DATE            DATE
     ,YEAR                 INTEGER
     ,QUARTER              INTEGER
     ,YEAR_MONTH           VARCHAR(6)
     ,MONTH                INTEGER
     ,MONTH_NAME           CHAR(9)
     ,MONTH_ABBREV         CHAR(3)
     ,WEEK_OF_YEAR         INTEGER
     ,DAY_OF_MONTH         INTEGER
     ,DAY_NAME             CHAR(1Ø)
     ,DAY_OF_WEEK          INTEGER
     ,DAY_OF_YEAR          INTEGER
     ,FIRST_DAY_WEEK       DATE
     ,DAY_ABBREV           CHAR(3)
     ,WORKING_DAY          VARCHAR(1)            NOT NULL
     ,LAST_DAY_MONTH       VARCHAR(1)            NOT NULL
     ,QUARTER_YEAR         VARCHAR(7)
     ,FISCAL_MONTH         INTEGER
     ,FISCAL_QUARTER       INTEGER
     ,FISCAL_YEAR          INTEGER
     ) IN DATABASE DSNDBØ4 ;

Id_time             (surrogate values, 1..N)
Full_date           (date value, 2ØØ1-Ø8-3Ø)
Year                (four digit year, 2ØØ1)
Quarter             (quarter, 1..4)
Year_month          (year+month, 2ØØ1Ø8)
Month               (month number, 1..12)
Month_name          (January, etc.)
Month_abbrev        (Jan, Feb, etc.)
```

```
Week_of_year       (week number in year, 1..53)
Day_of_month       (day number in month, 1..31)
Day_name           (day name in week, Monday, etc.)
Day_of_week        (day number in week, 1..7)
Day_of_year        (day number in year, 1..366)
First_day_week     (full date in first day of  week)
Day_abbrev         (Mon, Tue, etc.)
Working_day        ( "n" or "y")
Last_day_month     (last day in month - "y" or "n")
Quarter_year       (quarter in year, 1Q2001, etc.)
Fiscal_month       (your private fiscal month)
Fiscal_quarter     (your private fiscal quarter)
Fiscal_year        (your private fiscal year)
```

The following SQL statement runs in a DB2 Universal Database environment and generates test data for Time_Dimension table for the year 2002 – the start date is 2002-01-01 and the end date is 2002-12-31:

```
WITH TIME_DIMENSION (ID_TIME, FULL_DATE, YEAR, QUARTER, YEAR_MONTH,
MONTH,
     MONTH_NAME, MONTH_ABBREV, WEEK_OF_YEAR, DAY_OF_MONTH, DAY_NAME,
     DAY_OF_WEEK, DAY_OF_YEAR, FIRST_DAY_WEEK, DAY_ABBREV, WORKING_DAY,
     LAST_DAY_MONTH, QUARTER_YEAR, FISCAL_MONTH, FISCAL_QUARTER,
     FISCAL_YEAR ) AS
( SELECT 1,
     DATE('2002-01-01'),                              /* start date
     YEAR(DATE('2002-01-01')),
     QUARTER(DATE('2002-01-01')),
     RTRIM(CHAR(YEAR(DATE('2002-01-01'))))  CONCAT
     SUBSTR(CHAR((DATE('2002-01-01'))),6,2),
     MONTH(DATE('2002-01-01')),
     MONTHNAME(DATE('2002-01-01')),
     SUBSTR(MONTHNAME(DATE('2002-01-01')),1,3),
     WEEK(DATE('2002-01-01')),
     DAY(DATE('2002-01-01')),
     DAYNAME(DATE('2002-01-01')),
     DAYOFWEEK(DATE('2002-01-01')),
     DAYOFYEAR(DATE('2002-01-01')),
     CASE DAYOFWEEK(DATE('2002-01-01'))
       WHEN 1 THEN DATE('2002-01-01') - 6 DAYS
       WHEN 2 THEN DATE('2002-01-01')
       WHEN 3 THEN DATE('2002-01-01') - 1 DAY
       WHEN 4 THEN DATE('2002-01-01') - 2 DAYS
       WHEN 5 THEN DATE('2002-01-01') - 3 DAYS
       WHEN 6 THEN DATE('2002-01-01') - 4 DAYS
       WHEN 7 THEN DATE('2002-01-01') - 5 DAYS
     END,
```

```
   SUBSTR(DAYNAME(DATE('2002-01-01')),1,3),
   CASE DAYOFWEEK(DATE('2002-01-01'))
     WHEN 1 THEN 'n'
     WHEN 7 THEN 'n'
     ELSE 'y'
   END,
   CASE MONTH(DATE('2002-01-01'))-MONTH(DATE('2002-01-01')+1 DAY)
     WHEN 0 THEN 'n'
     ELSE 'y'
   END,
   RTRIM(CHAR(QUARTER(DATE('2002-01-01')))) CONCAT 'Q'
   CONCAT CHAR(YEAR(DATE('2002-01-01'))),
   MONTH(DATE('2002-01-01') + 3 MONTHS),
   QUARTER(DATE('2002-01-01') + 3 MONTHS),
   YEAR(DATE('2002-01-01') + 3 MONTHS)
FROM SYSIBM.SYSDUMMY1
UNION ALL
SELECT ID_TIME + 1,
   FULL_DATE + 1 DAY,
   YEAR(FULL_DATE + 1 DAY),
   QUARTER(FULL_DATE + 1 DAY),
   RTRIM(CHAR(YEAR(DATE(FULL_DATE + 1 DAY)))) CONCAT
   SUBSTR(CHAR((DATE(FULL_DATE + 1 DAY))),6,2),
   MONTH(DATE(FULL_DATE + 1 DAY)),
   MONTHNAME(DATE(FULL_DATE + 1 DAY)),
   SUBSTR(MONTHNAME(DATE(FULL_DATE + 1 DAY)),1,3),
   WEEK(DATE(FULL_DATE + 1 DAY)),
   DAY(DATE(FULL_DATE + 1 DAY)),
   DAYNAME(DATE(FULL_DATE + 1 DAY)),
   DAYOFWEEK(DATE(FULL_DATE + 1 DAY)),
   DAYOFYEAR(DATE(FULL_DATE + 1 DAY)),
   CASE DAYOFWEEK(DATE(FULL_DATE + 1 DAY))
     WHEN 1 THEN DATE(FULL_DATE - 5 DAYS)
     WHEN 2 THEN DATE(FULL_DATE + 1 DAY)
     WHEN 3 THEN FULL_DATE
     WHEN 4 THEN DATE(FULL_DATE - 1 DAY)
     WHEN 5 THEN DATE(FULL_DATE - 2 DAYS)
     WHEN 6 THEN DATE(FULL_DATE - 3 DAYS)
     WHEN 7 THEN DATE(FULL_DATE - 4 DAYS)
   END,
   SUBSTR(DAYNAME(DATE(FULL_DATE + 1 DAY)),1,3),
   CASE DAYOFWEEK(DATE(FULL_DATE + 1 DAY))
     WHEN 1 THEN 'n'
     WHEN 7 THEN 'n'
     ELSE 'y'
   END,
   CASE MONTH(DATE(FULL_DATE + 1 DAY)) -
        MONTH(DATE(FULL_DATE + 2 DAYS))
```

Figure 1: Some example Time_Dimension table report rows

ID TIME	FULL DATE	YEAR	QUARTER	YEAR MONTH	MONTH	MONTH NAME	MONTH ABBREV	WEEK OF YEAR	DAY OF MONTH	DAY NAME	DAY OF WEEK	DAY OF YEAR	FIRST DAY WEEK	DAY ABBREV	WORKING DAY	LAST DAY MONTH	QUARTER YEAR	FISCAL MONTH	FISCAL QUARTER	FISCAL YEAR
1	2002-01-01	2002	1	200201	1	January	Jan	1	1	Tuesday	3	1	2001-12-31	Tue	y	n	1Q2002	4	2	2002
2	2002-01-02	2002	1	200201	1	January	Jan	1	2	Wednesday	4	2	2001-12-31	Wed	y	n	1Q2002	4	2	2002
3	2002-01-03	2002	1	200201	1	January	Jan	1	3	Thursday	5	3	2001-12-31	Thu	y	n	1Q2002	4	2	2002
4	2002-01-04	2002	1	200201	1	January	Jan	1	4	Friday	6	4	2001-12-31	Fri	y	n	1Q2002	4	2	2002
5	2002-01-05	2002	1	200201	1	January	Jan	1	5	Saturday	7	5	2001-12-31	Sat	n	n	1Q2002	4	2	2002
6	2002-01-06	2002	1	200201	1	January	Jan	2	6	Sunday	1	6	2001-12-31	Sun	n	n	1Q2002	4	2	2002
7	2002-01-07	2002	1	200201	1	January	Jan	2	7	Monday	2	7	2002-01-07	Mon	y	n	1Q2002	4	2	2002
8	2002-01-08	2002	1	200201	1	January	Jan	2	8	Tuesday	3	8	2002-01-07	Tue	y	n	1Q2002	4	2	2002
9	2002-01-09	2002	1	200201	1	January	Jan	2	9	Wednesday	4	9	2002-01-07	Wed	y	n	1Q2002	4	2	2002
10	2002-01-10	2002	1	200201	1	January	Jan	2	10	Thursday	5	10	2002-01-07	Thu	y	n	1Q2002	4	2	2002
11	2002-01-11	2002	1	200201	1	January	Jan	2	11	Friday	6	11	2002-01-07	Fri	y	n	1Q2002	4	2	2002
12	2002-01-12	2002	1	200201	1	January	Jan	2	12	Saturday	7	12	2002-01-07	Sat	n	n	1Q2002	4	2	2002
13	2002-01-13	2002	1	200201	1	January	Jan	3	13	Sunday	1	13	2002-01-07	Sun	n	n	1Q2002	4	2	2002
14	2002-01-14	2002	1	200201	1	January	Jan	3	14	Monday	2	14	2002-01-14	Mon	y	n	1Q2002	4	2	2002
15	2002-01-15	2002	1	200201	1	January	Jan	3	15	Tuesday	3	15	2002-01-14	Tue	y	n	1Q2002	4	2	2002
16	2002-01-16	2002	1	200201	1	January	Jan	3	16	Wednesday	4	16	2002-01-14	Wed	y	n	1Q2002	4	2	2002
17	2002-01-17	2002	1	200201	1	January	Jan	3	17	Thursday	5	17	2002-01-14	Thu	y	n	1Q2002	4	2	2002
.																				
.																				
.																				

```
       WHEN Ø THEN 'n'
       ELSE 'y'
     END,
     RTRIM(CHAR(QUARTER(DATE(FULL_DATE + 1 DAY)))) CONCAT 'Q'
     CONCAT CHAR(YEAR(DATE(FULL_DATE + 1 DAY))),
     MONTH(DATE(FULL_DATE + 1 DAY) + 3 MONTHS),
     QUARTER(DATE(FULL_DATE + 1 DAY) + 3 MONTHS),
     YEAR(DATE(FULL_DATE + 1 DAY) + 3 MONTHS)
  FROM TIME_DIMENSION
  WHERE FULL_DATE < DATE('2002-12-31'))                  /* end date
SELECT ID_TIME, FULL_DATE,
     YEAR, QUARTER,
     YEAR_MONTH, MONTH,
     UCASE(SUBSTR(MONTH_NAME,1,1)) CONCAT SUBSTR(MONTH_NAME,2,8)
MONTH_NAME,
     UCASE(SUBSTR(MONTH_ABBREV,1,1)) CONCAT SUBSTR(MONTH_ABBREV,2,2)
MONTH_ABBREV,
     WEEK_OF_YEAR,
     DAY_OF_MONTH,
     UCASE(SUBSTR(DAY_NAME,1,1)) CONCAT SUBSTR(DAY_NAME,2,9) DAY_NAME,
     DAY_OF_WEEK,
     DAY_OF_YEAR,
     FIRST_DAY_WEEK,
     UCASE(SUBSTR(DAY_ABBREV,1,1)) CONCAT SUBSTR(DAY_ABBREV,2,2)
DAY_ABBREV,
     WORKING_DAY,
     LAST_DAY_MONTH,
     QUARTER_YEAR,
     FISCAL_MONTH,
     FISCAL_QUARTER,
     FISCAL_YEAR
FROM TIME_DIMENSION
```

Test data must be exported to a mainframe, if your Time_Dimension table and warehouse run on a mainframe. Figure 1: some example rows for a Time_Dimension table report:

Bernard Zver
DBA
Informatika (Slovenia)

DB2 joins explained

INTRODUCTION

There has always been a need to join data from DB2 tables. In the past this may have involved only a couple of tables, but nowadays it involves joining across many tables. DB2 has always offered various join methods, which the experienced DBA can use to extract correct data. However, for the inexperienced DBA, this can all seem rather daunting! This article explains the different types of join available in DB2 (the examples were written in DB2 7.2 on NT, but are applicable to DB2 on all platforms).

THE FOUR TYPES OF SUPPORTED JOIN

DB2 supports four types of join – inner join, left outer join, right outer join, and full outer join. Each of these join types is described and shown in an example.

To demonstrate how each of the joins works, create the following two tables: taba contains a list of product ids (ID) and the number in stock (NUM), and tabb contains a list of product ids (ID) and their description (DESC) irrespective of whether they are in stock or not.

```
create table taba (id int, num int);
create table tabb (id int, desc char(7));
–
insert into taba values(1,10);
insert into taba values(2,20);
insert into taba values(5,50);
```

Populate these tables with the values shown:

```
insert into tabb values(1,'knife');
insert into tabb values(2,'fork');
insert into tabb values(3,'spoon');
insert into tabb values(4,'chair');
insert into tabb values(5,'table');
```

Select from both of these tables:

```
>db2 select * from taba
```

```
          ID          NUM
          --------    --------
           1          10
           2          20
           5          50
           6          60

  4 record(s) selected.
            >db2 select * from tabb

          ID  DESC
          -- --------
           1 knife
           2 fork
           3 spoon
           4 chair
           5 table

  5 record(s) selected.
```

When you join two tables, you need a join column found in both tables, which in our case is the ID column.

Now we will run each type of join listed above, and see what we get.

Inner join

An inner join produces a table containing only rows from both the left and right tables where the joining column has matching values.

```
>db2 select * from taba a join tabb b on a.id = b.id

          ID          NUM         ID  DESC
       ------       -----       ---  -----
          1          10           1 knife
          2          20           2 fork
          5          50           5 table

  3 record(s) selected.
```

This is equivalent to running:

```
>db2 select * from taba a, tabb b where  a.id = b.id

          ID          NUM         ID  DESC
       ------       -----       ---  -----
          1          10           1 knife
          2          20           2 fork
```

```
          5          50           5 table

  3 record(s) selected.
```

If we want to see a list of all products, whether they are in stock or not, and a description of them, then this output is not what is required. So let's now look at the three outer joins available to us.

Left outer join

```
>db2 select * from taba a left outer join tabb b on a.id = b.id

          ID          NUM         ID  DESC
       ------       -----       ---  -----
          1          10           1 knife
          2          20           2 fork
          5          50           5 table
          6          60           - -

  4 record(s) selected.
```

When you specify LEFT OUTER JOIN, DB2 returns rows where the join column values match as well as rows in the table on the left, where the joining column value on the left does not have a matching value in the table on the right. In the above example, the ID value 6 exists in the table on the left of the LEFT OUTER JOIN expression (taba), but not in the table on the right (tabb), and that is why it was included. As an ID of 6 does not have a DESC value, it is shown as a null value (-). Note that there isn't a row with an ID=4 because this ID was present only in the right-hand table (tabb) and we were doing a LEFT OUTER JOIN. So this isn't really what we want either.

Right outer join

```
>db2 select * from taba a right outer join tabb b on a.id = b.id

          ID          NUM         ID  DESC
       ------       -----       ---  -----
          1          10           1 knife
          2          20           2 fork
          -           -           3 spoon
          -           -           4 chair
          5          50           5 table

  5 record(s) selected.
```

The RIGHT OUTER JOIN is similar to the left outer join, but works instead on the table on the right of the expression. DB2 returns rows where the join column values match as well as rows in the table on the right, where the joining column value on the right does not have a matching value in the table on the left. In the above example, the ID values 3 and 4 exist on the table on the right of the RIGHT OUTER JOIN expression (tabb), but not in the table on the left (taba), and that is why they were included. The missing values are nulls (-). This time we are missing the row for ID=6 from the left-hand table (taba). So this isn't quite right either.

What happens if we put tabb on the left of the RIGHT OUTER JOIN expression as shown below:

```
>db2 select * from tabb b right outer join taba a on a.id = b.id

          ID  DESC              ID          NUM
         --- ------            ---         -----
           1 knife              1           1Ø
           2 fork               2           2Ø
           5 table              5           5Ø
           - -                  6           6Ø

  4 record(s) selected.
```

Is this equivalent to a left outer join, with taba on the left and tabb on the right? Well not quite – you still get back four rows, but note the order of the ID columns.

```
>db2 select * from taba a left outer join tabb b on a.id = b.id

          ID          NUM         ID   DESC
          --          ---         --   ------
           1          1Ø           1   knife
           2          2Ø           2   fork
           5          5Ø           5   table
           6          6Ø           -   -

  4 record(s) selected.
```

You might not want to see both ID columns, so specify the column names in the select statement; but which table do you get the ID column from, taba or tabb? Let's go back to the original way we coded the query, and run the query twice, once specifying a.id and then again specifying b.id.

```
>db2 select a.id,num,desc from taba a right outer join tabb b on a.id =
b.id

          ID        NUM  DESC
          --        ---  ----
          1          10 knife
          2          20 fork
          -           - spoon
          -           - chair
          5          50 table

  5 record(s) selected.

>db2 select b.id,num,desc from taba a right outer join tabb b on a.id =
b.id

          ID        NUM  DESC
          --        ---  ----
          1          10 knife
          2          20 fork
          3           - spoon
          4           - chair
          5          50 table

  5 record(s) selected.
```

As you can see, there is no difference in what is returned. Does this mean that it doesn't matter whether we specify a.id or b.id? No – definitely not! To show what I mean, look at the example of putting tabb on the left of the right outer join expression. This is shown below:

```
>db2 select a.id,num,desc from tabb b right outer join taba a on a.id =
b.id

          ID        NUM  DESC
          --        ---  ----
          1          10 knife
          2          20 fork
          5          50 table
          6          60 -

  4 record(s) selected.

>db2 select b.id,num,desc from tabb b right outer join taba a on a.id =
b.id

          ID        NUM  DESC
          --        ---  ----
          1          10 knife
```

```
          2          20 fork
          5          50 table
          -          60 -

  4 record(s) selected.
```

You can see the difference between coding a.id and b.id.

Full outer join

Now let's get back to our original problem. We have tried a LEFT OUTER JOIN and a RIGHT OUTER JOIN, but both do not give us back the information we want. So, our last option is the FULL OUTER JOIN.

```
>db2 select * from taba a full outer join tabb b on a.id = b.id

         ID          NUM         ID DESC
        ----         ---         -- ----
          1          10           1 knife
          2          20           2 fork
          -           -           3 spoon
          -           -           4 chair
          5          50           5 table
          6          60           - -

  6 record(s) selected.
```

The FULL OUTER JOIN returns rows which match in the joining column, as well as rows from both the left and right tables which do not have a match in the other table (ie rows from taba which do not have a matching value in the join column in taba and similarly rows from tabb which do not have a matching value in the join column in tabb).

I don't really want to see the ID column twice in the output – I know it's a common column (because I joined on it). What I really want is to see just three columns, say NUM,ID,DESC. But how do I code that? I could try:

```
>db2 select num,a.id,desc from taba a full outer join tabb b on a.id =
b.id

         NUM         ID  DESC
         ---         --  ----
         10           1 knife
```

```
10            2 fork
 -            - spoon
 -            - chair
50            5 table
60            6 -

  6 record(s) selected.
```

But I am missing some of the ID values from tabb. So let's try:

```
>db2 select num,b.id,desc from taba a full outer join tabb b on a.id =
b.id

         NUM         ID  DESC
         ---         --  ----
         10           1 knife
         20           2 fork
          -           3 spoon
          -           4 chair
         50           5 table
         60           - -

  6 record(s) selected.
```

Now I am missing IDs from taba. Neither of the two sets of output seems to give us quite what we thought we would get back. All the information is there, but it is perhaps difficult to read. This is where the COALESCE function comes into play.

```
>db2 select num,coalesce(a.id,b.id) as id,desc from taba a full outer
join tabb b on a.id = b.id

         NUM         ID  DESC
         ---         --  ----
         10           1 knife
         20           2 fork
          -           3 spoon
          -           4 chair
         50           5 table
         60           6 -

  6 record(s) selected.
```

As you can see, this gives us exactly what we want.

We have looked at the way LEFT OUTER JOINS and RIGHT OUTER JOINS work, and why it is important to decide which tables should go on the left and the right of the expressions. We have looked

at the FULL OUTER JOIN, and seen the use of the COALESCE function to tidy up our output.

Explain output

It is always useful to see how the DB2 optimizer re-writes the SQL that we enter. Run the following command in the command centre:

```
>db2 select * from taba a left outer join tabb b on a.id = b.id;
```

Looking at the access plan and the optimized SQL:

```
SELECT Q2.ID AS "ID", Q2.NUM AS "NUM", Q1.ID AS "ID", Q1.DESC AS "DESC"
FROM DB2ADMIN.TABB AS Q1 RIGHT OUTER JOIN DB2ADMIN.TABA AS Q2 ON (Q2.ID
= Q1.ID)
```

and doing the same for:

```
>db2 select * from taba a right outer join tabb b on a.id = b.id;
```

gives:

```
SELECT Q2.ID AS "ID", Q2.NUM AS "NUM", Q1.ID AS "ID", Q1.DESC AS "DESC"
FROM DB2ADMIN.TABB AS Q1 LEFT OUTER JOIN DB2ADMIN.TABA AS Q2 ON (Q2.ID =
Q1.ID)
```

The optimizer seems to have re-written both sets of SQL to use the other type of join. Something to keep in mind when looking at the explain output from an outer join query.

JOINING MORE THAN TWO TABLES

It's all very well joining two test tables, but most real life situations involve joining more than two tables. The theory is simple – you just expand the SQL statement to include more join statements; but the practice is a little harder, so let's look at an example consisting of four tables:

- taba – contains product id (ID) and number in stock (NUM).
- tabb – contains product id (ID) and a description (DESC).
- tabc – contains product id (ID) and the cost (COST).
- tabd – contains product id (ID) and the size (SIZE).

This gives us:

```
select * from taba

          ID          NUM
          --          ---
          1           10
          2           20
          5           50

  3 record(s) selected.
      select * from tabb

         ID DESC
         -- ----
          1 knife
          2 fork
          3 spoon
          4 chair
          5 table

  5 record(s) selected.
      select * from tabc

         ID          COST
         ---        -----
          1          100
          2          200
          5          500

  3 record(s) selected.
      select * from tabd

         ID SIZE
         -- ----
          1 A
          3 C

  2 record(s) selected.
```

We want to get a list of all product IDs, and description, cost, and size wherever the information exists.

If you run:

```
select coalesce(a.id,b.id,c.id,d.id) as id,num,desc,cost,size
from
(((taba a
left outer join tabb b on a.id = b.id)
left outer join tabc c on a.id = c.id)
```

```
left outer join tabd d on a.id = d.id)
```

then taba is joined with tabb, the result is joined with tabc and the result of that is joined with tabd. Is this what we want? Well, yes, it is, but the joins have to be the correct ones.

```
         ID          NUM DESC           COST  SIZE
         --          --- ----           ----- ----
          1           10 knife            100 A
          2           20 fork             200 -
          5           50 table            500 -

  3 record(s) selected.
```

The above is clearly not what is required. Let's try and break down the problem, to see if we can determine what is wrong with the statement.

Let us run some queries on taba and tabb:

```
>db2 select coalesce(a.id,b.id) as id,num,desc from (taba a left outer
join tabb b on a.id = b.id)

         ID          NUM  DESC
         --          ---  ----
          1           10 knife
          2           20 fork
          5           50 table

  3 record(s) selected.

>db2 select coalesce(a.id,b.id) as id,num,desc from (taba a right outer
join tabb b on a.id = b.id)

         ID          NUM  DESC
         --          ---  ----
          1           10 knife
          2           20 fork
          3            - spoon
          4            - chair
          5           50 table

  5 record(s) selected.

>db2 select coalesce(a.id,b.id) as id,num,desc from (tabb b left outer
join taba a on a.id = b.id)

         ID          NUM  DESC
         --          ---  ----
          1           10 knife
          2           20 fork
```

```
          3           - spoon
          4           - chair
          5          50 table

  5 record(s) selected.

>db2 select coalesce(a.id,b.id) as id,num,desc from (tabb b full outer
join taba a on a.id = b.id)

         ID          NUM  DESC
         --          ---  ----
          1          10 knife
          2          20 fork
          5          50 table
          3           - spoon
          4           - chair

  5 record(s) selected.
```

As you can see, there is no point is just coding left outer joins, as a left outer join between taba and tabb will not return all the rows required. We could code a right outer join between taba and tabb, but we want to avoid right outer joins, so let's code a left outer join between tabb and taba. Then we can left outer join the result of tabb/taba with tabc and tabd all with left outer joins, as the table with the most information should always be on the left of a left outer join.

We therefore get:

```
>db2 select coalesce(a.id,b.id,c.id,d.id) as id,num,desc,cost,size from
(((tabb b left outer join taba a on a.id = b.id)
          left outer join tabc c on b.id = c.id)
          left outer join tabd d on b.id = d.id)

         ID          NUM DESC          COST SIZE
         --          --- ----          ---- ----
          1          10 knife           100 A
          2          20 fork            200 -
          3           - spoon             - C
          4           - chair             - -
          5          50 table           500 -


  5 record(s) selected.
```

The brackets are evaluated from left to right (and not lowest level to highest). This means that we work our way down the SQL statement, evaluating each join as we go down the file.

We couldn't have used a full outer join (instead of the left outer join), because it would result in an extra row in the result set:

```
ID          NUM DESC          COST SIZE
--          --- ----          ---- ----
 1          10 knife          100 A
 2          20 fork           200 -
 5          50 table          500 -
 3           - spoon            - -
 4           - chair            - -
 3           - -                - C

  5 record(s) selected.
```

Does the order of 'on' values matter? Once again – oh yes! It doesn't matter if you code 'on a.id = b.id' or 'on b.id=a.id' for a single join statement, but what does matter in a nested join statement is which letter comes first. Look at the examples below:

```
>db2 select coalesce(a.id,b.id,c.id,d.id) as id,num,desc,cost,size from
(((tabb b left outer join taba a on a.id = b.id)
          left outer join tabc c on b.id = c.id)
          left outer join tabd d on b.id = d.id)

ID          NUM DESC          COST SIZE
--          --- ----          ---- ----
 1          10 knife          100 A
 2          20 fork           200 -
 3           - spoon            - C
 4           - chair            - -
 5          50 table          500 -
```

Now let's just replace two 'b' values with 'a' values:

```
>db2 select coalesce(a.id,b.id,c.id,d.id) as id,num,desc,cost,size from
(((tabb b left outer join taba a on a.id = b.id)
               left outer join tabc c on a.id = c.id)
               left outer join tabd d on a.id = d.id)

ID          NUM DESC          COST SIZE
--          --- ----          ---- ----
 1          10 knife          100 A
 2          20 fork           200 -
 5          50 table          500 -
 3           - spoon            - -
 4           - chair            - -
```

You can see the difference in the results. The correct order for a left outer join is shown in Figure 1.

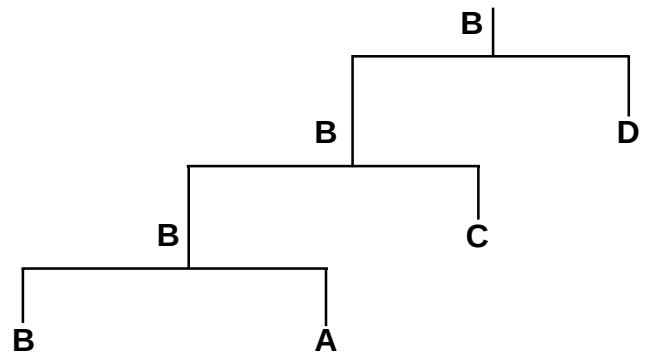

Figure 1: Order for left outer join

CONCLUSION

I hope I have shown that there is nothing to fear in using outer joins as long as you fully understand the underlying table structures and their contents and what you are trying to achieve (ie don't switch table names without thinking of the consequences first etc).

C Leonard
Freelance Consultant (UK)

A formatting routine for DSNTIAUL unload files – part 2

This month we conclude the code that formats DSNTIAUL unload files.

```
READ_SQL:

/*****************************************************************/
/*  Interpret the load statement                                 */
/*****************************************************************/

    xx = outtrap(xx.)
    'FREE   F(SYSPUNCH)'
    "ALLOC  F(SYSPUNCH) DS('"pchout"')  SHR "
    xx =outtrap(off)
    lrc  = rc
    if   lrc >0  then  signal load_tab_end
    xx = outtrap(xx.)
```

```
    'FREE   F(SYSREC00)'
    "ALLOC  F(SYSREC00) DS('"unlout"')  SHR "
    xx =outtrap(off)
    lrc  = rc
    if   lrc >0  then  signal load_tab_end1

   'EXECIO * DISKR SYSPUNCH 1 ( STEM PUN. FINIS'

    do i = 1 to pun.0
       if strip(pun.i,'B') = '(' then leave
    end
    f = 0
    j = i + 1

    do i = j to pun.0
       if strip(pun.i,'B') = ')' then leave
       f = f + 1
       parse var pun.i $name.f . pos.f rest .
       if right(pos.f,1) = ':'
          then do
               pos.f = strip(pos.f,'T',':')
               bis   = strip(rest,'T',')')
               len   = bis - pos.f + 1
               end
       i = i + 1
       parse var pun.i typ.f '(' len.f ')'  .
       typ.f = strip(typ.f,'B')
       len.f = strip(len.f,'B')
       if len.f = ''
          then do
               parse var pun.i typ.f len.f  .
               len.f = strip(len.f,'B',')')
               typ.f = strip(typ.f,'B','(')
               end
       if typ.f = 'DECIMAL'
          then len.f = len
       if typ.f = 'FLOAT'
          then len_float.f = len.f
       i = i + 1
       if pos('NULLIF',pun.i) > 0
          then parse var pun.i . nullpos.f ')' .
          else do
               nullpos.f = 0
               i = i - 1
               end
    end

/*********************************************************************/
/*  Format the unload file                                           */
/*********************************************************************/
```

```
'EXECIO 1 DISKR SYSREC00 'recnr' ( STEM REC.'
ret = rc
do i = recnr by 1 while ret = 0
   if i > recnr + recanz -1 then leave
   queue ' '
   queue '-'unlout'- Record 'i
   queue ' '
   do j = 1 to f
      select
       /*********************************************/
       /*  Format type  DECIMAL                     */
       /*********************************************/
        when typ.j = 'DECIMAL'
         then do
              if nullpos.j > 0
                 then do
                      /* field is nullable */
                      if substr(rec.1,nullpos.j,1) = '?'
                       /* handling of the null indicator */
                       then inhalt = '-?-'
                       else inhalt = unpack()
                      end
                 else inhalt = unpack()
              end
       /*********************************************/
       /*  Format type  INTEGER                     */
       /*********************************************/
        when typ.j = 'INTEGER'
         then do
              len.j = 4
              if nullpos.j > 0
                 then do
                      /* field is nullable */
                      if substr(rec.1,nullpos.j,1) = '?'
                       /* handling of the null indicator */
                       then inhalt = '-?-'
                       else do
                            inhalt = substr(rec.1,pos.j,len.j)
                            if inhalt = x2c(FFFFFFFF)
                               then inhalt = '*?*'
                               else do
                               if inhalt < x2c(7FFFFFFF)
                                  then inhalt = c2d(inhalt)
                                  else do
                                       r1 = c2x(inhalt)
            r1 = translate(r1,'0123456789ABCDEF','FEDCBA9876543210')
                                       r1 = x2c(r1)
                                       inhalt = c2d(r1) + 1
                                       inhalt = inhalt * (-1)
```

```
                                end
                             end
                       end
                  end
             else do
                  inhalt = substr(rec.1,pos.j,len.j)
                  if inhalt = x2c(FFFFFFFF)
                     then inhalt = '*?*'
                     else do
                           if inhalt < x2c(7FFFFFFF)
                              /* value is positive */
                              then inhalt = c2d(inhalt)
                              /* value is negative */
                              else do
                                   r1 = c2x(inhalt)
        r1 = translate(r1,'Ø123456789ABCDEF','FEDCBA9876543210')
                                   r1 = x2c(r1)
                                   inhalt = c2d(r1) + 1
                                   inhalt = inhalt * (-1)
                                   end
                        end
                  end
        end
/***********************************************/
/*  Format type  SMALLINT                      */
/***********************************************/
 when typ.j = 'SMALLINT'
  then do
       len.j = 2
       if nullpos.j > Ø
          then do
               /* field is nullable */
               if substr(rec.1,nullpos.j,1) = '?'
                /* handling of the null indicator */
                then inhalt = '-?-'
                else do
                     inhalt = substr(rec.1,pos.j,len.j)
                     if inhalt = x2c(FFFF)
                        then inhalt = '*?*'
                        else do
                        if inhalt < x2c(7FFF)
                           /* value is positive */
                           then inhalt = c2d(inhalt)
                           /* value is negative */
                           else do
                                r1 = c2x(inhalt)
        r1 = translate(r1,'Ø123456789ABCDEF','FEDCBA9876543210')
                                r1 = x2c(r1)
                                inhalt = c2d(r1) + 1
                                inhalt = inhalt * (-1)
```

```
                                end
                             end
                       end
                  end
             else do
                  inhalt = substr(rec.1,pos.j,len.j)
                  if inhalt = x2c(FFFF)
                     then inhalt = '*?*'
                     else do
                           if inhalt < x2c(7FFF)
                              then inhalt = c2d(inhalt)
                              else do
                                   r1 = c2x(inhalt)
        r1 = translate(r1,'Ø123456789ABCDEF','FEDCBA9876543210')
                                   r1 = x2c(r1)
                                   inhalt = c2d(r1) + 1
                                   inhalt = inhalt * (-1)
                                   end
                        end
                  end
        end
/***********************************************/
/*  Format type  VARCHAR                       */
/***********************************************/
 when typ.j = 'VARCHAR'
  then do
       len.j = c2d(substr(rec.1,pos.j,2))
       posx  = pos.j + 2
       if nullpos.j > Ø
          then do
               /* field is nullable */
               if substr(rec.1,nullpos.j,1) = '?'
                /* handling of the null indicator */
                then inhalt = '-?-'
                else inhalt = substr(rec.1,posx,len.j)
               end
          else inhalt = substr(rec.1,posx,len.j)
       end
/***********************************************/
/*  Format type  FLOAT                         */
/***********************************************/
 when typ.j = 'FLOAT'
  then do
       if len_float.j < 22 then len.j = 4
                           else len.j = 8
       if nullpos.j > Ø
          then do
               /* field is nullable */
               if substr(rec.1,nullpos.j,1) = '?'
                /* handling of the null indicator */
```

```
                 then inhalt = '-?-'
                 else inhalt = substr(rec.1,pos.j,len.j)
              end
           else inhalt = substr(rec.1,pos.j,len.j)
        if inhalt ¬= '-?-'
           then do
                exp = c2d(left(inhalt,1))
                if exp > 127
                   then do
                        vz = -1
                        vz1 = '-'
                        exp = exp - 128
                        end
                   else do
                        vz = 1
                        vz1 = ''
                        end
                exp = exp - 65
                man = substr(inhalt,2)
                man = c2x(man)
                man = strip(man,'T','Ø')
                inhalt = Ø
                inhex  = ''
                l_man = length(man)
                do ee = 1 to l_man
                   manp = substr(man,ee,1)
                   inhex = inhex||manp
                   manp= x2d(manp)
                   exp1 = exp-ee+1
                   if exp1 = Ø then inhex = inhex','
                   inhalt = inhalt+manp*16**(exp1)
                   end
                if inhex = '' then inhex = Ø
                inhex  = '('vz1||inhex' E 'exp')hex'
                inhalt = inhalt * vz'   'inhex
                end
       end
/*************************************************/
/*  Format type  CHAR or TIMESTAMP               */
/*************************************************/
 otherwise do
       if nullpos.j > Ø
          then do
               /* field is nullable */
               if substr(rec.1,nullpos.j,1) = '?'
                /* handling of the null indicator */
                then inhalt = '-?-'
                else inhalt = substr(rec.1,pos.j,len.j)
               end
          else inhalt = substr(rec.1,pos.j,len.j)
```

```
                    end
          end
         interpret $name.j'.i = INHALT'
         interpret 'queue left("'$name.j'",20) " = "'$name.j'.i'
         "EXECIO" queued() "DISKW OUT "
         end
      'EXECIO 1 DISKR SYSREC00 'i+1' ( STEM REC.'
      ret = rc
      end
   rec.0 = i - 1
   'EXECIO 0 DISKR SYSREC00 ( FINIS'

LOAD_TAB_END2:
    'FREE   F(SYSREC00)'
LOAD_TAB_END1:
    'FREE   F(SYSPUNCH)'
LOAD_TAB_END:
return

UNPACK:
/*********************************************************/
/* Buildin function to unpack DECIMAL data format        */
/*********************************************************/
  rech = c2x(substr(rec.1,pos.j,len.j))
  wert = left(rech,2*len.j-1)
  if right(rech,1) = 'D'
     then rech = wert * (-1)
     else rech = wert * 1
return rech
```

Roman Hawlitschek
System Programmer (Germany)

Utility to generate DSN1COPY jobs using the REXX DB2 interface

DSN1COPY is an IBM utility that is very useful in recovering dropped tablespaces. When a tablespace is dropped, all references to it are removed from the catalog tables, including the SYSCOPY entries. However, if the DDL of the table, before it was dropped, is available, then, using DSN1COPY, the tablespace can be easily recovered.

Another usage of the DSN1COPY utility is to migrate data from one subsystem to another. Using the OBIDXLAT and the RESET options, we can easily overlay a DB2 dataset in any subsystem using an image copy from another subsystem, thereby 'migrating' the data from the source to the target. The alternative is to unload the data from the table or from the image copies and then load it into the target subsystem.

At our site, we use DSN1COPY predominantly to refresh a stress test region with production data and our discussion will be with reference to that. Please note that DSN1COPY is not compatible with image copies taken using the CONCURRENT option.

The advantages of using DSN1COPY will be very obvious in that it reduces the number of jobs by half. It is faster because it directly moves the data without any intermediate storage as would be required in an unload and load sequence. This saves on tape and also uses fewer resources. However, a distinct disadvantage is plugging in the correct OBIDs for various database objects. This utility greatly relieves this difficult task. There is also the potential of overwriting the wrong dataset if one is not extremely careful when running DSN1COPY.

The utility presented here uses a panel for convenience to capture the input data and also perform most data validations. The source and target DB2 subsystem IDs are the first inputs on the panel. The list of databases and tablespaces that need to be refreshed is to be given as input in a dataset. This information can be given in a very flexible format. It can be just a list of database names, one on each line; or it can be a database name and a tablespace name on each line separated by one or more spaces. Or it can also be a mixture of the two. If the database name and tablespace name option is used, then the list needs to be sorted to make use of a special feature built into the utility. This feature can combine the JCL generated for the tablespaces in the same database into one job.

The ICDATE and ICTIME range of values to be used to pick the image copy for the refresh must be carefully chosen. The utility will always choose the most recent local FULL image copy on TAPE. The account information and job card for the generated jobs can also be modified on the input panel. The output dataset to hold the generated JCL has to be specified next. Also required are two more input datasets

containing OBID information in a specific format. The query to generate the information in these datasets is also provided below:

```
SELECT  A.DBNAME, B.TSNAME, B.NAME,
                A.DBID, A.PSID, B.OBID, B.CREATOR
   FROM SYSIBM.SYSTABLESPACE A
              , SYSIBM.SYSTABLES     B
 WHERE  A.DBNAME LIKE    'XX%'
      AND  A.NAME     = B.TSNAME
      AND  A.DBID     = B.DBID
     ORDER BY 1,2
  ;
```

At our installation, the production and the test systems are on different machines; hence this approach. If both the systems were on the same machine, we could query the database and get the information we needed. It is advisable to keep a separate OBID dataset for each database environment because it will perform faster. For example, if you had two sets of databases on subsystem DB21 with the table qualifiers (creators) as TEST1 and TEST2, it would be beneficial to have a separate OBID dataset for each.

Finally, the VCAT name and the CREATOR (table qualifier) for the target database is to be provided. All the inputs stated above are mandatory and the datasets must exist, except the output dataset for the DSN1CPY JCL. If this dataset exists, it will be deleted without any warning and rewritten.

DSN1COPY requires that the source and target tablespaces be identical with respect to the number of partitions, and the segment size. The object names may be different, but they can be translated by the OBIDXLAT option. The input image copy must be a full image copy. It is important to remember that the tablespace needs to be stopped before DSN1COPY can work on it.

The utility generates steps to start all the required tablespaces in UT (utility only) mode and then to STOP the same after successful completion. This will prevent the tablespaces from going into STOP Pending mode, which could happen if there is any activity at the time of issuing the STOP request. DSN1COPY updates only the underlying VSAM datasets and it is important to rebuild all the indexes after successful completion of the DSN1COPY steps. The necessary steps

to perform the index rebuild are also generated by the utility in the same JCL. A REPORT member is also generated in the same dataset, which gives details of the utility execution.

Because of the very sensitive nature of the DSN1COPY utility, it is a good idea to reject the production system as a valid target subsystem ID. This can be easily done by modifying the panel definition.

The utility has two REXX programs, DCOPY and DCOPYCAL, and two panels, DSCPY and RCVR51.

Sample input dataset:

```
DATABSØ1 TBLSPCØ1
DATABSØ1 TBLSPCØ2
DATABSØ1 TBLSPCØ3
DATABSØ2
DATABSØ3 TBLSPCØ6
```

Sample OBID dataset:

```
DATABSØ1  TBLSPCØ1 TABLE_NAMEØ1              261       2      47  PROD
DATABSØ1 TBLSPCØ2  TABLE_NAMEØ2              261       4      48  PROD
DATABSØ1 TBLSPCØ3  TABLE_NAMEØ3              261       6      49  PROD
DATABSØ1 TBLSPCØ4  TABLE_NAMEØ4              261       8      5Ø  PROD
```

Sample panel input:

```
DSN1CPY JOBS FROM IMAGE COPIES
   System:  PROD           DSN1CPY
 *******

   Source SSID : PROD           Target SSID : TEST
   DBNAME/TSNAME List PDS : PREF.USERID.DBLST
   Edit this dataset (Y/N): Y

   ICDATE and ICTIME values of SYSCOPY records
    After  which to recover : Ø1Ø718 (yymmdd)   : 19ØØØØ (hhmmss)
    Before which to recover : Ø1Ø718 (yymmdd)   : 215ØØ (hhmmss)

   //XXXXXXXX JOB (427AR968)        ,'DSN1CPY',
   // MSGLEVEL=(1,1),REGION=ØM,CLASS=Q,MSGCLASS=T,
   // TIME=144Ø,NOTIFY=&SYSUID

   Dataset for DSN1CPY JCLs: PREF.USERID.DSN1CPY.DØ1Ø719
   Dataset for Source OBIDs: PREF.USERID.DBXX.SUCCESS.OBID
   Dataset for Target OBIDs: PREF.USERID.DBXX.SUCCESS.OBID

   Give for Target DATABASE VCATNAME : HRDMDBT3    CREATOR : HRDMST
```

DCOPY

```
/* rexx */
/*********************************************************************/
/*                                                                   */
/*     Panel driven tool for generating DSN1COPY jobs                */
/*     using image copies on TAPE.                                   */
/*                                                                   */
/*   Invocation: TSO DCOPY                                           */
/*                                                                   */
/*   Uses panels in HRDBA.$$PAJKJ.PANELS                             */
/*                                                                   */
/*  Sitespecific info is marked as *** SITESPEC***                   */
/*********************************************************************/
trace o
clear
address tso "ispexec  vget (zsysid zacctnum)"
p_sysid=zsysid
if p_sysid = 'PROD' then
   MC = 'PROD'
else
   MC = 'TEST'
pref =strip(sysvar(syspref))
cd = date(U)
us_date = substr(cd,7,2)||substr(cd,1,2)||substr(cd,4,2)
ZWINTTL = 'DSN1COPY JOBS FROM IMAGE COPIES'
cur_date = date(S)-1
cur_date = substr(cur_date,3)
ICDTE1 = cur_date
ICDTE2 = cur_date
ICT1 = '190000'
ICT2 = '215900'
EDFLG= 'Y'
ACCOUNT = "("||:/cctnum||")"
/*** SITESPEC ***/
JCLCRD2 = "    MSGLEVEL=(1,1),REGION=0M,CLASS=A,MSGCLASS=X,"
JCLCRD3 = "    TIME=1440,NOTIFY=&SYSUID"
OUTUNL = pref||'.'||userid()||'.DSN1CPY.D'||us_date
BEGIN:
/*** SITESPEC ***/
"ISPEXEC LIBDEF ISPPLIB DATASET ID('PREFIX.USERID.PANELS')"
"ISPEXEC ADDPOP ROW(2) COLUMN(10)"
"ISPEXEC DISPLAY PANEL(DSCPY)"
"ISPEXEC REMPOP ALL "
clear
upper SSID; upper TSID
Upper DBLST
Upper TGTDSN
Upper SRCDSN
```

```
upper EDFLG
upper ICDATE
upper ICT1; upper ICT2; upper ACCTNUM; upper ACCTNUL; upper OUTUNL
upper VCATNAM
upper TCRETOR
if PF3 = 'EXIT' then
  exit(Ø)
CHK_DSN:
DBLST = strip(DBLST,Leading,"'")
z = outtrap("zlst.","*")
x = SYSDSN("'"DBLST"'")
z = outtrap("OFF")
if x = 'OK' then
   nop
else
do
   ERRMSG = 'Dataset not found ...'
   ERRCOL = DBLST
   "ISPEXEC ADDPOP ROW(4) COLUMN(6)"
   "ISPEXEC DISPLAY PANEL(RCVR51)"
   "ISPEXEC REMPOP ALL "
   signal BEGIN
end
CHK_TGTDSN:
TGTDSN = strip(TGTDSN,Leading,"'")
z = outtrap("zlst.","*")
x = SYSDSN("'"TGTDSN"'")
z = outtrap("OFF")
if x = 'OK' then
   nop
else
do
   ERRMSG = 'Dataset not found ...'
   ERRCOL = TGTDSN
   "ISPEXEC ADDPOP ROW(4) COLUMN(6)"
   "ISPEXEC DISPLAY PANEL(RCVR51)"
   "ISPEXEC REMPOP ALL "
   signal BEGIN
end
CHK_SRCDSN:
SRCDSN = strip(SRCDSN,Leading,"'")
z = outtrap("zlst.","*")
x = SYSDSN("'"SRCDSN"'")
z = outtrap("OFF")
if x = 'OK' then
   nop
else
do
   ERRMSG = 'Dataset not found ...'
```

```
   ERRCOL = SRCDSN
   "ISPEXEC ADDPOP ROW(4) COLUMN(6)"
   "ISPEXEC DISPLAY PANEL(RCVR51)"
   "ISPEXEC REMPOP ALL "
   signal BEGIN
end
CHK_ICDTE1:
D_yy = substr(ICDTE1,1,2)
D_mm = substr(ICDTE1,3,2)
D_dd = substr(ICDTE1,5,2)
if (D_mm > 12) | (D_dd > 31) | (D_dd < 1) | (D_mm < 1)  then
do
   ERRMSG = 'Invalid date specified '
   ERRCOL = ICDTE1
   "ISPEXEC ADDPOP ROW(4) COLUMN(6)"
   "ISPEXEC DISPLAY PANEL(RCVR51)"
   "ISPEXEC REMPOP ALL "
    signal BEGIN
end
cur_date = date(S)
cur_date = substr(cur_date,3)
if ICDTE1 > cur_date then
do
   ERRMSG = 'Date must be < =  'cur_date
   ERRCOL = ICDTE1
   "ISPEXEC ADDPOP ROW(4) COLUMN(12)"
   "ISPEXEC DISPLAY PANEL(RCVR51)"
   "ISPEXEC REMPOP ALL "
    signal BEGIN
end
CHK_ICDTE2:
D_yy2 = substr(ICDTE2,1,2)
D_mm2 = substr(ICDTE2,3,2)
D_dd2 = substr(ICDTE2,5,2)
if (D_mm2 > 12) | (D_dd2 > 31) | (D_dd2 < 1) | (D_mm2 < 1)  then
do
   ERRMSG = 'Invalid date specified '
   ERRCOL = ICDTE2
   "ISPEXEC ADDPOP ROW(4) COLUMN(6)"
   "ISPEXEC DISPLAY PANEL(RCVR51)"
   "ISPEXEC REMPOP ALL "
    signal BEGIN
end
cur_date = date(S)
cur_date = substr(cur_date,3)
if ICDTE2 > cur_date then
do
   ERRMSG = 'Date must be < =  'cur_date
   ERRCOL = ICDTE2
```

```
   "ISPEXEC ADDPOP ROW(4) COLUMN(12)"
   "ISPEXEC DISPLAY PANEL(RCVR51)"
   "ISPEXEC REMPOP ALL "
    signal BEGIN
end
CHK_ICDS:
if (D_yy <> D_yy2) | (D_mm <> D_mm2) | (D_dd > D_dd2) then
do
   ERRMSG = 'ICDATES incorrect  '
   ERRCOL = ICDTE1||' > '||ICDTE2
   "ISPEXEC ADDPOP ROW(4) COLUMN(6)"
   "ISPEXEC DISPLAY PANEL(NRCVR51)"
   "ISPEXEC REMPOP ALL "
    signal BEGIN
end
CHK_ICT1:
T_hh = substr(ICT1,1,2)
T_mm = substr(ICT1,3,2)
T_ss = substr(ICT1,5,2)
if (T_hh > 23) | (T_mm > 59) | (T_ss > 59) then
do
   ERRMSG = 'Invalid time specified '
   ERRCOL = ICT1
   "ISPEXEC ADDPOP ROW(4) COLUMN(6)"
   "ISPEXEC DISPLAY PANEL(RCVR51)"
   "ISPEXEC REMPOP ALL "
    signal BEGIN
end
CHK_ICT2:
T_hh2 = substr(ICT2,1,2)
T_mm2 = substr(ICT2,3,2)
T_ss2 = substr(ICT2,5,2)
if ((T_hh2 > 23) | (T_mm2 > 59) | (T_ss2 > 59))  then
do
   ERRMSG = 'Invalid time specified '
   ERRCOL = ICT2
   "ISPEXEC ADDPOP ROW(4) COLUMN(6)"
   "ISPEXEC DISPLAY PANEL(RCVR51)"
   "ISPEXEC REMPOP ALL "
    signal BEGIN
end
Signal CHKSSID
CHK_ICTS:
if (T_hh2 < T_hh) | (T_hh2 = T_hh & T_mm2 < T_mm) then
do
   ERRMSG = 'ICTIMES incorrect  '
   ERRCOL = ICT1||' < 'ICT2
   "ISPEXEC ADDPOP ROW(4) COLUMN(6)"
   "ISPEXEC DISPLAY PANEL(RCVR51)"
```

```
   "ISPEXEC REMPOP ALL "
    signal BEGIN
end
CHKSSID:
address tso "ispexec  vget (zsysid)"
p_sysid=zsysid
sid = SSID
ssid_err = Ø
/*** SITESPEC  ***/
 if p_sysid  = 'PROD' then
    select
      when sid = 'PROD'
         then do
            nop
         end
      when sid = 'PRD2'
         then do
            nop
         end
      otherwise
         ssid_err =1
    end
 if p_sysid  = 'TEST'' then
   select
      when sid = 'DBT1'
         then do
            nop
         end
      when sid = 'DBT2'
         then do
            nop
         end
      when sid = 'DBT3'
         then do
            nop
         end
      when sid = 'DBP1'
         then do
            nop
         end
      when sid = 'DBP2'
         then do
            nop
         end
      when sid = 'DBP3'
         then do
            nop
         end
      otherwise
```

```
         ssid_err =1
   end
if ssid_err = 1 then
do
   ERRMSG = 'Invalid SSID for '||MC
   ERRCOL = SSID
   "ISPEXEC ADDPOP ROW(4) COLUMN(6)"
   "ISPEXEC DISPLAY PANEL(RCVR51)"
   "ISPEXEC REMPOP ALL "
    signal BEGIN
end
CHKEDT:
if EDFLG = 'Y' then
do
   ADDRESS ISPEXEC "EDIT  DATASET('"DBLST"')"
   EDFLG = 'N'
   signal BEGIN
end
cmd = 'address tso "DCOPYCA2" SSID DBLST ICDTE1 ICDTE2 ICT1 ICT2 '
cmd = cmd||' ACCOUNT OUTUNL TSID VCATNAM JCLCRD2 JCLCRD3'
cmd = cmd||' TGTDSN SRCDSN TCRETOR'
interpret cmd
say 'Press Enter to quit...'; pull temp
exit(Ø)
```

DCOPYCA2

```
/* rexx */
/*********************************************************************/
/*  DCOPYCAL  - by Jaiwant Jonathan                                  */
/*                                                                   */
/* Invocation: Invoked from DCOPY for generating DSN1COPY jobs       */
/*                                                                   */
/*  Panel driven                                                     */
/*                                                                   */
/*  This REXX is to generate the DSN1COPY jobs for:                  */
/*  copying image copies onto a DB2 dataset for either               */
/*  a list containing database names or database and                 */
/*  tablespace names. This has to be given as a sequential input     */
/*  dataset. Other inputs to be specifed are the DB2 subsystem       */
/*  ID, the ICDATE, the ICTIMEs between which the image copy         */
/*  or quiesce was done, and an output dataset.                      */
/*  The inputs are captured and validated by DCOPY and passed        */
/*  to this program as arguments                                     */
/*                                                                   */
/*  Special Requirements:                                            */
/*  ---------------------                                            */
/*  This REXX uses the DB2-REXX Interface facility provided by       */
/*  IBM.                                                             */
```

```
/*****************************************************************/
trace o
clear
PARSE UPPER ARG P_ssid P_dblst P_icdte1 P_icdte2 P_ict1 P_ict2,
     P_act P_out P_tsid P_vcat P_jc2 P_jc3 P_tdsn P_sdsn P_tcretor
if left(strip(P_act),1) ¬= '('  then
   P_act = '('||strip(P_act)
if right(strip(P_act),1) ¬= ')'  then
   P_act = ')'||strip(P_act)
address tso "ispexec  vget (zsysid)"
p_sysid=zsysid
/*  pass parameters to local values    */
sid = strip(P_ssid)
tsid = strip(P_tsid)
/*  say  P_ssid P_dblst P_icdte1 P_icdte2 P_ict1 P_ict2 P_out P_typ   */
I_lstdsn = strip(P_dblst)
I_icdte1 = strip(P_icdte1)
I_icdte2 = strip(P_icdte2)
I_btime = strip(P_ict1)
I_etime = strip(P_ict2)
ods_name = strip(P_out)
P_typ    = strip(P_typ)
P_jc2    = strip(P_jc2,'T',' ')
P_jc3    = strip(P_jc3,'T',' ')
plus = 2
cd = date(U)
us_date = substr(cd,7,2)||substr(cd,1,2)||substr(cd,4,2)
/*  build rest of sys. array   */
/*  1 thru 5 array is built in WRITEMEM as it needs member name */
RBLDSYS:
sys.7 = "//SYSPRINT DD SYSOUT=* "
sys.8 = "//SYSUDUMP DD SYSOUT=* "
sys.9 = "//UTPRINT  DD SYSOUT=* "
sys.10= "//SYSOUT   DD SYSOUT=* "
sys.11= "//SYSUT1   DD SPACE=(CYL,(500,500)),UNIT=(SYSDA,5)"
sys.12= "//SORTWK01 DD SPACE=(CYL,(500,500),RLSE),UNIT=(SYSDA,5)"
sys.13= "//SORTWK02 DD SPACE=(CYL,(500,500),RLSE),UNIT=(SYSDA,5)"
sys.14= "//SORTWK03 DD SPACE=(CYL,(500,500),RLSE),UNIT=(SYSDA,5)"
sys.15= "//SORTWK04 DD SPACE=(CYL,(500,500),RLSE),UNIT=(SYSDA,5)"
sys.16= "//SORTWK05 DD SPACE=(CYL,(500,500),RLSE),UNIT=(SYSDA,5)"
sys.17= "//SORTWK06 DD SPACE=(CYL,(500,500),RLSE),UNIT=(SYSDA,5)"
sys.18= "//SORTWK07 DD SPACE=(CYL,(500,500),RLSE),UNIT=(SYSDA,5)"
sys.19= "//SORTWK08 DD SPACE=(CYL,(500,500),RLSE),UNIT=(SYSDA,5)"
sys.20= "//*  "
sys.21= "//SYSIN  DD  * "
STRTUT:
stt.1 = "//STRTACUT  EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LE)"
stt.2 = "//STEPLIB  DD DISP=SHR,DSN=SGDP."||tsid||".DSNEXIT"
stt.3 = "//         DD DISP=SHR,DSN=SGDP."||tsid||".DSNLOAD"
stt.4 = "//SYSTSPRT DD SYSOUT=*,DCB=BLKSIZE=121  "
```

```
stt.5 = "//SYSPRINT DD SYSOUT=*,DCB=BLKSIZE=121 "
stt.6 = "//SYSOUT   DD  SYSOUT=*  "
stt.7 = "//SYSUDUMP DD SYSOUT=D,HOLD=YES "
stt.8 = "//SYSTSIN  DD *  "
stt.9 = " DSN SYSTEM ("||tsid||")"
Call P000_MAIN
exit(0)
/* Main routine */
P000_MAIN:
/* Allocate and read the input list of DBNAMEs and/or TSNAMEs */
Call GETDBLST
Call GETTGTLST
Call GETSRCLST
pref = strip(sysvar(syspref))
/* Delete output dataset name if it exists   */
xx = outtrap("zap.","*")
address tso "delete '"ods_name"'"
xx = outtrap("OFF")
/*  allocate the output dataset anew  */
address tso "alloc f(rpds) new unit(hsm) space(5,10)",
          "cyl dir(50) reuse dsname('"ods_name"')",
          "dsorg(po) blksize(3120) lrecl(80) recfm(f b)"
if  rc>0 then
do
   say  'Failed during allocation of 'ods_name
   address tso "free f(lstdd)"
   address tso "free f(tgtdd)"
   address tso "free f(srcdd)"
   exit(8)
end
say 'Results will be generated into...' ods_name
s = 0
/*  Connect to DB2  */
Call PERFCONN
j = 0
tot = dbl.0
prdb = ''
I_tsname = ''
do i = 1 to tot
   Parse Var dbl.i dbname  tsname  rest
   C_dbname=strip(dbname);
   C_tsname=strip(tsname);
   I_tsname=strip(tsname);
   if i = tot then
   do
      I_dbname = C_dbname
      I_tsname = C_tsname
      Call PROCROW
      leave
   end
```

```
   do j = i+1 to tot
      Parse Var dbl.j dbname  tsname  rest
      N_dbname=strip(dbname)
      N_tsname=strip(tsname)
      if C_dbname = N_dbname &  N_tsname <> '' then
      do
         I_tsname = I_tsname||"', '"||N_tsname
         i = j
         I_dbname = C_dbname
         iterate
      end
      else
      do
         I_dbname = C_dbname
      end
   end
   Call PROCROW
   if i = tot then
   do
      leave
   end
   I_tsname = ''
end
trace o
Call CLEANUP
Call PERFDISC
say 'Doing cleanup...'
xx = outtrap("zap.","*")
address tso "free f(lstdd)"
address tso "FREE ddname(rpds)"
xx = outtrap("OFF")
say; say 'Completed processing..'
say 'Refer member REPORT in 'ods_name
return
/*  Allocate and read input dataset for DBNAMEs and/or TSNAMEs  */
GETDBLST:
"ALLOCATE DD(lstdd) DSN('"I_lstdsn"') REUSE SHR"
if  rc>Ø then
do
   say  'Failed during allocation of 'I_lstdsn
   pull temp
   exit(8)
end
"execio * DISKR lstdd (FINIS STEM dbl."
return
GETTGTLST:
"ALLOCATE DD(tgtdd) DSN('"P_tdsn"') REUSE SHR"
if  rc>Ø then
do
   say  'Failed during allocation of 'P_tdsn
```

```
   pull temp
   exit(8)
end
return
GETSRCLST:
"ALLOCATE DD(srcdd) DSN('"P_sdsn"') REUSE SHR"
if  rc>Ø then
do
   say  'Failed during allocation of 'P_sdsn
   pull temp
   exit(8)
end
return
/* Form query using input row from flat file */
PROCROW:
   STMT =" SELECT  A.DBNAME, A.TSNAME, DSNUM, FILESEQNO, "
   STMT = STMT||" DSNAME, S.NAME, "
   STMT=STMT||"ICDATE, ICTIME, T.PARTITIONS ",
         " FROM   SYSIBM.SYSCOPY A, SYSIBM.SYSTABLESPACE  T, ",
         " SYSIBM.SYSTABLES S ",
         "   WHERE A.DBNAME IN  ('"I_dbname"')"
   if I_tsname ¬= '' then
      STMT = STMT||" AND  A.TSNAME IN ('"I_tsname"')"
   if I_icdte1 = I_icdte2 then
   do
      STMT = STMT||" AND ICDATE = '"I_icdte1"'",
            " AND ICTIME BETWEEN  '"I_btime"' AND '"I_etime"'"
   end
   else
   do
      STMT = STMT||" AND (( ICDATE = '"I_icdte1"'",
         " AND ICTIME >= '"I_btime"' ) OR  ",
         " ( ICDATE = '"I_icdte2"'",
         " AND ICTIME <= '"I_etime"' ))"
   end
   STMT = STMT||" AND ICTYPE = 'F' ",
               " AND ICUNIT = 'T'  "
   if P_recsite = 'Y' then
      STMT = STMT||" AND ICBACKUP = 'RP' "
   else
      STMT = STMT||" AND ICBACKUP = '  ' "
   STMT = STMT||" AND TIMESTAMP = ( SELECT MAX(TIMESTAMP) FROM  ",
         " SYSIBM.SYSCOPY B ",
         " WHERE  A.DBNAME = B.DBNAME ",
         "   AND  A.TSNAME = B.TSNAME ",
         "   AND  A.DSNUM  = B.DSNUM  "
   if I_icdte1 = I_icdte2 then
   do
      STMT = STMT||" AND B.ICDATE = '"I_icdte1"'",
            " AND B.ICTIME BETWEEN  '"I_btime"' AND '"I_etime"'"
```

```
   end
   else
   do
      STMT = STMT||" AND (( B.ICDATE = '"I_icdte1"'",
         " AND B.ICTIME >= '"I_btime"' ) OR  ",
         " ( B.ICDATE = '"I_icdte2"'",
         " AND B.ICTIME <= '"I_etime"' ))"
   end
   STMT = STMT ||"   AND  A.ICTYPE = B.ICTYPE ) ",
         "   AND  A.DBNAME = T.DBNAME ",
         "   AND  A.TSNAME = T.NAME ",
         "   AND  A.DBNAME = S.DBNAME ",
         "   AND  A.TSNAME = S.TSNAME ",
         " ORDER BY 1,2,3,4  ",
         " WITH UR  "
  /*     " ORDER BY 1,2,3,4,6,7,8 "  commented for performance  */
ERRFLG = 0
Call PROCQRY
/*  if first fetch is successful, then process all rows       */
/*  otherwise retrieve next row from flat file for processing */
if ERRFLG = 0 then
   Call PROCRSLT
else
do
   say 'Error in processing ... refer member REPORT in 'ods_name
   Call CLEANUP
   address tso "free f(lstdd)"
   exit(20)
end
return
/*  Prepare Dynamic SQL using STMT and perform first fetch  */
/*  Call PROCRSLT if successful to process rows otherwise   */
/*  write error message and continue to next input row      */
/*  If severe error, then exit program                      */
PROCQRY:
ADDRESS DSNREXX "EXECSQL DECLARE C1 CURSOR FOR S1"
/* Say 'Declare SQLCODE = ' SQLCODE  */
if SQLCODE ¬= 0 then
do
   Call SQLCA
   exit
end
ADDRESS DSNREXX "EXECSQL PREPARE S1 FROM :STMT"
/*  Say 'Prepare SQLCODE = ' SQLCODE    */
if SQLCODE ¬= 0 then
do
   Call SQLCA
   exit
end
/* Say 'opening C1 '             */
```

```
ADDRESS DSNREXX "EXECSQL OPEN C1"
/*Say 'Open SQLCODE = ' SQLCODE  */
if SQLCODE ¬= Ø then
   Call SQLCA
ADDRESS DSNREXX "EXECSQL FETCH C1 INTO :A1, :A2, :A3, :A4, :A5,",
                ":A6, :A7, :A8, :A9 "
/*
IF  SQLCODE = Ø  Then  Do
       Line = ' '
       Line = Line||A1||A2||A3||A4||A5||A6||A7||A8||A9
       say Line
end
 */
if SQLCODE ¬= Ø then
do
   Call ERRPT
   ERRFLG = 1
   return
end
/*********************************************************/
/******  DEBUG  TEXT - UNCOMMENT IF REQUIRED  ***********/
/*
IF  SQLCODE = Ø  Then  Do
       Line = ' '
       Line = Line||A1||A2||A3||A4||A5||A6||A7||A8||A9
       say Line
end
*/
/*********************************************************/
return
/*  Process results of query and form output members  */
PROCRSLT:
memcnt=Ø; row_cnt = Ø
prevts = ''
prevdb = ''
first = 1;
mno = Ø;
j = Ø ; l = Ø; u = Ø
I_dbname = strip(I_dbname)
I_tsname = strip(I_tsname)
If strip(I_tsname) = '' | length(strip(I_tsname)) > 8 then
   Memname = substr(I_dbname,1,8)
else
   Memname = I_tsname
Do while length(Memname) < 8
   Memname  = ' '||Memname
end
len= length(Memname);
Do until (SQLCODE ¬= Ø )
   db = strip(A1)       /* database name          */
```

```
   ts = strip(A2)       /* tablespace name        */
   prt = strip(A3)      /* partition num or DSNUM */
   fseq = strip(A4)     /* fileseq number         */
   dsn = strip(A5)      /* dataset name of imagecpy */
   tbn = strip(A6)      /* tbl name               */
   icd = strip(A7)      /* icdate of syscopy rec  */
   ict = strip(A8)      /* ictime of syscopy rec  */
   numprts = strip(A9)      /* number of partitions   */
   if db ¬= prevdb then
   do
      say 'Processing subset 'db' ...'
      prevdb = db
   end
   row_cnt = row_cnt + 1
   if fseq = 1 & first = Ø then
   do
      mno = mno+1
      memname = overlay(mno,memname,6)
      Call  FILLJCL
      Call  WRITEMEM
      first = 1
      l = Ø ; j = Ø ; u=Ø
      row_cnt = 1
   end
   if  first = 1  then
   do
      prevts = ts
      Call  BILDSTEP
      Call  BILDINDX
      first = Ø
   end
   else /* first ¬= 1  */
   do
      if  prevts = ts  then
      do
         Call BILDSTEP
      end
      else /*  prevts ¬=  ts   */
      do
         Call BILDSTEP
         Call  BILDINDX
         prevts = ts
      end
   end
ADDRESS DSNREXX "EXECSQL FETCH C1 INTO :A1, :A2, :A3, :A4, :A5,",
                ":A6, :A7, :A8, :A9"
/*******************************************************/
/***  DEBUGGING  TEXT   - UNCOMMENT IF REQUIRED  ********/
/*
   IF  SQLCODE = Ø  Then  Do
```

```
          Line = ' '
          Line = Line||A1||A2||A3||A4||A5||A6||A7||A8||A9
          say Line
   end
*/
/*********************************************************/
end
ADDRESS DSNREXX "EXECSQL CLOSE C1"
if mno > Ø then
do
   mno=mno+1
   memname = overlay(mno,memname,6)
end
Call FILLJCL
Call  WRITEMEM
return
CLEANUP:
ods_mem = ods_name||"(REPORT)"
address tso "alloc f(outdd) mod dsname('"ods_mem"')"
hdr.1 = 'ICDATEs used ...'I_icdte1  I_icdte2
hdr.2 = 'ICTIMES between 'I_btime' and 'I_etime
hdr.3 = 'Input dataset 'I_lstdsn
hdr.4 = 'NO  DBNAME   TSNAME   MEMBER    STATUS'
hdr.5 = '-------------------------------------'
address tso "execio *  DISKW outdd (stem hdr.  "
address tso "execio *  DISKW outdd (stem rpt. FINIS "
address tso "FREE ddname(outdd)"
return
PERFCONN:
/* Set up STEPLIBs                                    */
/*                                                    */
ADDRESS TSO
lib.Ø = 2
lib.1 = "SGDP."||sid||".DSNLOAD"
lib.2 = "SGDP."||sid||".DSNEXIT"
L_Dsname = ""
  do i = 1 to lib.Ø
    L_Dsname = L_Dsname||"'"||_ib.i||"' "
  end
/*  address tso "ALLOC F(STEPLIB) DA("L_Dsname") SHR REU "   */
  "STEPLIB DA("L_Dsname") SHR "
'SUBCOM  DSNREXX'            /* Is host command env avlbl ?  */
IF  RC then                  /*  If not, then add it         */
    S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX')
ADDRESS  DSNREXX             /* exec all further cmds in DSNREXX */
ADDRESS DSNREXX "CONNECT" sid
Say 'Connect RC ...' RC
return
/* Disconnect from DB2  */
PERFDISC:
```

```
ADDRESS  DSNREXX " DISCONNECT"
Say 'Disconnect RC ...' RC
if SQLCODE ¬= Ø then
do
   Call SQLCA
   exit
end
S_RC = RXSUBCOM('DELETE','DSNREXX','DSNREXX')
return
/*  Report error in query processing  */
ERRPT:
   do while length(I_dbname) < 8
      I_dbname = ' '||I_dbname
   end
   do while length(I_tsname) < 8
      I_tsname = ' '||I_tsname
   end
   sno=i
   do while length(sno) < 3
      sno = ' '||sno
   end
   rpt.i = sno||' '||I_dbname||' '||I_tsname||' '||Memname
   rpt.i = rpt.i||' - Query did not retrieve any rows'
return
/*  Useful in analyzing SQL codes    */
SQLCA:
TRACE O
SAY 'SQLCODE ='SQLCODE
SAY 'SQLERRMC ='SQLERRMC
SAY 'SQLERRP ='SQLERRP
SAY 'SQLERRD ='SQLERRD.1',',
|| SQLERRD.2',',
|| SQLERRD.3',',
|| SQLERRD.4',',
|| SQLERRD.5',',
|| SQLERRD.6
SAY 'SQLWARN ='SQLWARN.Ø',',
|| SQLWARN.1',',
|| SQLWARN.2',',
|| SQLWARN.3',',
|| SQLWARN.4',',
|| SQLWARN.5',',
|| SQLWARN.6',',
|| SQLWARN.7',',
|| SQLWARN.8',',
|| SQLWARN.9',',
|| SQLWARN.1Ø
SAY 'SQLSTATE='SQLSTATE
SAY 'SQLCODE ='SQLCODE
say  'SQLERRMC ='SQLERRMC';' ,
```

```
|| 'SQLERRP ='SQLERRP';' ,
|| 'SQLERRD ='SQLERRD.1',',
            || SQLERRD.2',',
            || SQLERRD.3',',
            || SQLERRD.4',',
            || SQLERRD.5',',
            || SQLERRD.6';' ,
|| 'SQLWARN ='SQLWARN.Ø',',
            || SQLWARN.1',',
            || SQLWARN.2',',
            || SQLWARN.3',',
            || SQLWARN.4',',
            || SQLWARN.5',',
            || SQLWARN.6',',
            || SQLWARN.7',',
            || SQLWARN.8',',
            || SQLWARN.9',',
            || SQLWARN.1Ø';' ,
            || 'SQLSTATE='SQLSTATE';'
return
/*  The JCL JOB card, EXEC card and work file definitions */
FILLJCL:
jcl.1 = "//"||substr(Memname,1,8)||" JOB "||P_act||","
jcl.1 = jcl.1||"'"||Memname||" FASTRCVR',"
jcl.2 = "//  "||P_jc2
jcl.3 = "//  "||P_jc3||"        "
jcl.4 = "//*                                                  "
jcl.5 = "/*JOBPARM SYSAFF=819Ø "
jcl.6 = "//* ------------------------------------------------    "
jcl.7 = "//*       FASTRCVR JOB                                 "
jcl.8 = "//*                                                   "
jcl.9 = "//*       GENERATED USING ...... ON "||us_date||"        "
jcl.1Ø= "//* USING "||I_lstdsn
jcl.11= "//* ------------------------------------------------    "
jcl.12= "//*                                                   "
jcl.13= "//JOBLIB   DD DISP=SHR,DSN=SGDP."||tsid||".DSNEXIT"
jcl.14= "//         DD DISP=SHR,DSN=SGDP."||tsid||".DSNLOAD"
jcl.15= "//*  "
return
/**    end  of  FILLJCL     **/
/*  Build the COPY DD cards and SYSIN cards for RECOVERY */
BILDSTEP:
j = j+1
csno = strip(row_cnt)
do while length(csno) < 4
   csno = 'Ø'||csno
end
parmstr = "FULLCOPY,OBIDXLAT,RESET,NUMPARTS("||>umprts||")"
cpy.j = "//DSN1"||csno||" EXEC PGM=DSN1COPY,"
j = j+1
```

```
cpy.j = "//        PARM='"||parmstr||"',COND=(4,LE)"
j = j+1
cpy.j = "//DSNTRACE DD DUMMY    "
j = j+1
cpy.j = "//SYSUDUMP DD SYSOUT=* "
j = j+1
cpy.j = "//SYSPRINT DD SYSOUT=* "
j = j+1
cpy.j  = "//SYSUT1   DD DSN="||dsn||","
j = j+1
cpy.j  = "//         DISP=(OLD)"
if  (csno = 1) | (fseq = 1) then
do
   cpy.j = cpy.j||",UNIT=349Ø,VOL=(,RETAIN)"
end
else
do
   cpy.j = cpy.j||",UNIT=349Ø,"
   j=j+1
   if plus = 1 then
      cpy.j =  "//         VOL=(,RETAIN)"
   else
/*    cpy.j =  "//         VOL=(,RETAIN,REF=*."||prevdd||".SYSUT1)" */
      cpy.j =  "//         VOL=(,RETAIN)"
end
prevdd = "DSN1"||csno
Call MAPOBID
odsn = P_vcat||".DSNDBD."||>db||"."||ts||".I0Ø01.AØØ1"
j = j+1
cpy.j  = "//SYSUT2   DD DSN="||?dsn||","
j = j+1
cpy.j  = "//         DISP=OLD"
if smapped = Ø then
do
   sdbid = 'XXXX'
   spsid = 'XXXX'
   sobid = 'XXXX'
   say 'unable to find source mapping for ' db ts tbn
end
if tmapped = Ø then
do
   tdbid = 'XXXX'
   tpsid = 'XXXX'
   tobid = 'XXXX'
   ndb   = 'XXXXXXXX'
   say 'unable to find target mapping for ' db ts tbn
end
j = j+1
cpy.j  = "//SYSXLAT  DD * "
j = j+1
```

```
cpy.j  = sdbid","tdbid||" "
j = j+1
cpy.j  = spsid","tpsid||" "
j = j+1
cpy.j  = sobid","tobid||" "
j = j+1
cpy.j  = "/*"
return
/*  Build the SYSIN cards for rebuilding INDEXES         */
/*  and for the start in util mode before index rebuild  */
BILDINDX:
/*******************************
   ****** this is for starting indiv ts in ut mode  ******
   ****** to use this mode you must stop indiv ts only ****
*/
u = u+1
str.u = " -START DATABASE("||>db||") SPACE("||ts||") ACCESS(UT) "
stp.u = " -STOP DATABASE("||>db||") SPACE("||ts||") "
/*************/
l = l+1
rbd.l = "  REBUILD INDEX (ALL) TABLESPACE "||>db||"."||ts
return
/*  Allocate a member and write the JCL  */
WRITEMEM:
   ods_mem = ods_name||"("||Memname||")"
   address tso "alloc f(outdd) mod dsname('"ods_mem"')"
sys.1= "//* "
sys.2= "//RBLDINDX   EXEC PGM=DSNUTILB,PARM='"||tsid||","||Memname||"'"
sys.2= sys.2||",COND=(4,LE)"
sys.3 = "//*RBLDINDX  EXEC                                        "
sys.4 = "//*
PGM=DSNUTILB,PARM='"||tsid||","||Memname||",RESTART(PHASE)'"
sys.4= sys.4||",COND=(4,LE)"
sys.5 = "//STEPLIB  DD DISP=SHR,DSN=SGDP."||tsid||".DSNEXIT"
sys.6 = "//         DD DISP=SHR,DSN=SGDP."||tsid||".DSNLOAD"
/*  this is for starting all TS in a DB in UT mode */
/***
str.1 = " -START DATABASE("||db||") SPACE(*) ACCESS(UT) "
***/
   address tso "execio *  DISKW outdd (stem jcl.  "
stt.1 = "//STRTUTS1  EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LE)"
   address tso "execio *  DISKW outdd (stem stt.  "
   address tso "execio *  DISKW outdd (stem str.  "
stt.1 = "//STOPTSUT  EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LE)"
   address tso "execio *  DISKW outdd (stem stt.  "
   address tso "execio *  DISKW outdd (stem stp.  "
   address tso "execio *  DISKW outdd (stem cpy.  "
stt.1 = "//STRTUTS2  EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LE)"
   address tso "execio *  DISKW outdd (stem stt.  "
   address tso "execio *  DISKW outdd (stem str.  "
```

```
   address tso "execio *  DISKW outdd (stem sys.  "
   address tso "execio *  DISKW outdd (stem rbd. FINIS "
   drop jcl.
   drop cpy.
   drop str.
   drop stp.
   drop rbd.
   xx = outtrap("zap.","*")
   address tso "FREE ddname(outdd)"
   xx = outtrap("OFF")
   s = s+1
   sno=s
   do while length(sno) < 3
      sno = ' '||sno
   end
   if I_tsname = '' then
      I_tsname = '        '
   do
      rpt.s = sno||' '||I_dbname||' '||I_tsname||' '||Memname
      rpt.s = rpt.s||'  Successfully written'
   end
return
MAPOBID:
address tso
smapped = Ø
tmapped = Ø
do forever
   "execio 1 DISKR srcdd "
   if RC=2 then leave
   pull inrec1
   parse var inrec1 w1 w2 w3 w4 w5 w6 w7
   if strip(w1) = db & strip(w2) = ts then
   do
      sdbid = strip(w4)
      spsid = strip(w5)
      sobid = strip(w6)
      smapped = 1
      leave
   end
end
"execio Ø DISKR srcdd (FINIS "
do forever
   "execio 1 DISKR tgtdd "
   if rc=2 then leave
   pull inrec1
   parse var inrec1 w1 w2 w3 w4 w5 w6 w7
   if strip(w2) = ts & strip(w3) = tbn & strip(w7) = P_tcretor then
   do
      tdbid = strip(w4)
      tpsid = strip(w5)
```

```
      tobid = strip(w6)
      ndb = strip(w1)
      tmapped = 1
      leave
   end
end
"execio Ø DISKR tgtdd (FINIS "
return
```

DSCPY

DSCPY - Panel definition

```
)ATTR
+ TYPE(TEXT) COLOR(WHITE)
~ TYPE(TEXT) COLOR(RED) HILITE(BLINK)
{ TYPE(OUTPUT) COLOR(RED)
$ TYPE(OUTPUT) COLOR(GREEN)
¬ TYPE(TEXT) COLOR(GREEN) INTENS(LOW) HILITE(REVERSE)
_ TYPE(INPUT) COLOR(YELLOW) INTENS(HIGH)
! TYPE(OUTPUT) COLOR(RED) HILITE(BLINK) CAPS(OFF)  PAD(' ')
@ TYPE(INPUT) COLOR(RED) INTENS(LOW) HILITE(REVERSE)
)BODY
+ System: {MC  +        ~ DSN1CPY
+                        ~ *******
+ Source SSID :_SSID+     Target SSID :_TSID+
+ DBNAME/TSNAME List PDS :_DBLST                                    +
+ Edit this dataset (Y/N):_Z+
+
+ ICDATE and ICTIME values of SYSCOPY records
+  After  which to recover :_ICDTE1+(yymmdd)   :_ICT1  +(hhmmss)
+  Before which to recover :_ICDTE2+(yymmdd)   :_ICT2  +(hhmmss)
+
+ //XXXXXXXX JOB@ACCOUNT          +,'DSN1CPY',
+ //_JCLCRD2                                                  +
+ //_JCLCRD3                                                  +
+                                                             +
+ Dataset for DSN1CPY JCLs:_OUTUNL                             +
+ Dataset for Source OBIDs:_SRCDSN                             +
+ Dataset for Target OBIDs:_TGTDSN                             +
+
+ Give the Target DATABASE VCATNAME :_VCATNAM +and CREATOR :_TCRETOR +
+
      !ERRMSG                    !ERRCOL
+
¬F3 - End +     $CONTMSG                                      +
+
)INIT
  .ZVARS= '(EDFLG)'
```

```
)PROC
   if (.PFKEY = 'PFØ3') &PF3 = EXIT
   VER(&SSID,NB,LIST,DBT3,DBP2,DBT1,DBT2,DBP6,DBP1,PROD)
   VER(&TSID,NB,LIST,DBT3,DBP2,DBT1,DBT2,DBP6,DBP1)
   VER(&DBLST,NB)
   VER(&EDFLG,NB,LIST,Y,N)
   VER(&ICDTE1,NB)
   VER(&ICDTE2,NB)
   VER(&ICT1,NB)
   VER(&ICT2,NB)
   VER(&OUTUNL,NB)
   VER(&VCATNAM,NB)
   VER(&TCRETOR,NB)
   VER(&SRCDSN,NB)
   VER(&TGTDSN,NB)
   VER(&ACCOUNT,NB)
   VER(&JCLCRD2,NB)
   VER(&JCLCRD3,NB)
)END
```

RCVR51

```
)ATTR
+ TYPE(TEXT) COLOR(WHITE)
)BODY WINDOW (6Ø,5)
+
+ &ERRMSG                                          :+ &ERRCOL
+
+  Enter to continue ...
+
)INIT
)PROC
)END
```

Jaiwant K Jonathan
IBM Certified Solutions Expert
QSS (USA)

Code from individual articles of *DB2 Update*, and complete issues in Acrobat PDF format, can be accessed on our Web site at *www.xephon.com/db2*. You will be asked to enter a word from the printed issue.

Checking data to be loaded

One of our *DB2 Update* subscribers, Denis Augé, has sent in the following question.

We work on an IBM OS/390 (MVS). We want to control our files before loading them into DB2 tables with the DB2 utility LOAD. We have to perform a lot of loads and the result must be OK. If it's not, we must restore the previous DB2 tables. So, to minimize the risk of doing a restore, we want to examine the files first. We want to control whether the data in the file is OK: if a column has a numeric definition, the tool had to identify whether the data (in the file) is numeric and can be loaded by the DB2 LOAD utility. We want to know whether there is a utility or a tool to give us this sort of control?

Pav Kumar-Chatterjee answered:

If I understand your situation correctly, you want to verify that the data you are loading into DB2 is 'clean' (ie you are loading characters into character fields, numerics into numeric fields, etc). This is an ETL (extract, transformation, load) process, which should be done outside of DB2. DB2 does not have an ETL function. Two products are available on OS/390 to manage this ETL process. If you just want to extract and load, then I recommend using Warehouse Manager, and if you want to do some data transformation as well, you can use Datastage.

Pav Kumar-Chatterjee
Data and Business Intelligence Group
IBM Software Business

Have you come across any undocumented features in DB2 Version 7? Why not share your discovery with others? Send your findings to the editor, Trevor Eddolls, at any of the addresses shown on page 2 or e-mail trevore@xephon.com.

Computer Associates has announced Release 6.2 of its BrightStor CA-Vantage Storage Resource Manager (SRM) for z/OS and OS/390.

It has more support for DB2 Universal Database Server for z/OS and OS/390 and enables sites to monitor and control DB2 space usage without, it's claimed, special DB2 expertise. It also includes an open interface that enables storage management information to be integrated into popular desktop business applications, such as Microsoft Outlook or Lotus Notes.

Support for DB2 UDB for mainframes includes database, table, and tablespace views and raises the number of storage objects that the tool can manage to more than 300.

It also has better support for IBM and StorageTek virtual tape systems and features a new graphical configuration client that's said to simplify installation and maintenance.

For further information contact:
Computer Associates International, One Computer Associates Plaza, Islandia, NY 11749, USA.
Tel: (631) 342 6000.
URL: http://ca.com/products/vantage_os390.htm.

* * *

Magic Software has announced the release of its Magic eDeveloper Version 9.01 SP3, which gains native gateways for both Informix and DB2 UDB.

Magic eDeveloper gives customers in heterogeneous environments the ability to manage and integrate data between Informix and DB2 UDB databases across eServer platforms.

Magic eDeveloper is said to provide a single tool for creating solutions using data in Informix, DB2, or both, native access to both Informix and DB2, integrated business processes to access different databases all within one program module, and physical table migration between databases.

For further information contact:
Magic Software Enterprises, 5 Haplada Street, Or Yehuda 60218, Israel.
Tel: (972-3) 538 9292.
URL: http://www.magicsoftware.com.

* * *

ArtinSoft has enhanced its Informix 4GL automatic migration engine to support upgrades to Java on Oracle and DB2. It will also support J2EE and CORBA platforms and will enable companies, says the firm, to migrate to these new environments faster and more cheaply than doing it manually.

For further information contact:
ArtinSoft, Ofiplaza del Este, Edificio B, Piso 1, 100m oeste Rotonda de la Bandera, San Pedro, Costa Rica.
Tel: (506) 283 3144
URL: http://www.artinsoft.com/en/informix/informix_page.asp.

* * *