February 2001

In this issue

3 DB2 REXX Language Support – part 2
12 Have you been cubed? – further thoughts
18 E-business enhancements with DB2 Universal Database Version 7.1
28 DB2 fast close and start
48 DB2 news

© Xephon plc 2001
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 ($33.50) each including postage.

DB2 Update on-line
Code from DB2 Update can be downloaded from our Web site at http://www.xephon.com/db2update.html; you will need the user-id shown on your address label.

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 contact us at any of the addresses above and we will send you a copy of our Notes for Contributors, or you can download a copy from www.xephon.com/contnote.html.

© Xephon plc 2001. All rights reserved. None of the text in this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior permission of the copyright owner. Subscribers are free to copy any code reproduced in this publication for use in their own installations, but may not sell such code or incorporate it in any commercial product. No part of this publication may be used for any form of advertising, sales promotion, or publicity without the written permission of the publisher. Copying permits are available from Xephon in the form of pressure-sensitive labels, for application to individual copies. A pack of 240 labels costs $36 (£24), giving a cost per copy of 15 cents (10 pence). To order, contact Xephon at any of the addresses above.

Printed in England.
DB2 REXX Language Support – part 2

This month we conclude the article describing DB2 for OS/390 REXX Language Support.

Handling errors and warnings
DB2 does not support the SQL WHENEVER statement in a REXX procedure. To handle SQL errors and warnings, use the following methods:

• To test for SQL errors or warnings, test the SQLCODE or SQLSTATE value and the SQLWARN values after each EXECSQL call. This method does not detect errors in the REXX interface to DB2.

• To test for SQL errors or warnings, or errors or warnings from the REXX interface to DB2, test the REXX RC variable after each EXECSQL call – see Figure 1.

<table>
<thead>
<tr>
<th>Return code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No SQL warning or error occurred.</td>
</tr>
<tr>
<td>+1</td>
<td>An SQL warning occurred.</td>
</tr>
<tr>
<td>-1</td>
<td>An SQL error occurred.</td>
</tr>
</tbody>
</table>

Figure 1: REXX return codes after SQL statements

You can also use the REXX SIGNAL ON ERROR and SIGNAL ON FAILURE keyword instructions to detect negative values of the RC variable and transfer control to an error routine.

Continuation for SQL statements
SQL statements that span lines follow REXX rules for statement continuation. You can break the statement into several strings, each of which fits on a line, and separate the strings with commas or with concatenation operators followed by commas.
For example, either of the following statements is valid:

```sql
EXECSQL,
    "UPDATE DSN8510.DEPT",
    "SET MGRNO = '000010'",
    "WHERE DEPTNO = 'D11'"
"EXECSQL " ||,
    " UPDATE DSN8510.DEPT "$ ||,
    " SET MGRNO = '000010' "$ ||,
    " WHERE DEPTNO = 'D11' "$
```

**Comments**

You cannot include REXX comments (/* ... */) or SQL comments (-- ) within SQL statements. However, you can include REXX comments anywhere else in the procedure.

**Data types**

All REXX data is string data.

Therefore, when a REXX procedure assigns input data to a table column, DB2 converts the data from a string type to the table column type.

When a REXX procedure assigns column data to an output variable, DB2 converts the data from the column type to a string type.

**Setting the isolation level of SQL statements in a REXX procedure**

When you install DB2 REXX Language Support, you bind four packages for accessing DB2, each with a different isolation level:

- DSNREXRR – Repeatable Read (RR)
- DSNREXRS – Read Stability (RS)
- DSNREXCS – Cursor Stability (CS)
- DSNREXUR – Uncommitted Read (UR).

To change the isolation level for SQL statements in a REXX procedure, execute the **SET CURRENT PACKAGESET** statement to select the package with the isolation level you need. For example, to change the isolation level to cursor stability, execute this SQL statement:

```sql
ADDRESS DSNREXX "EXECSQL SET CURRENT PACKAGESET='DSNREXCS'"
```
A SAMPLE DB2 REXX APPLICATION
This section contains an example of a complete DB2 REXX application named REXX02SO.

This very simple application is only intended to demonstrate DB2 REXX concepts which were explained in the previous section.

Creating the execution environment of the sample application
Before running the sample application, you should create its execution environment:

```/*
//STEPØØ1 EXEC PGM=IKJEFTØ1
//SYSTSPRT DD SYSOUT=*  
//SYSPRINT DD SYSOUT=*  
//SYSTSIN DD *
DSN SYSTEM(DB2B)
RUN PROGRAM(DSNTEP2) PLAN(DSNTEP2)  
END
//*/
//SYSIN DD *
   DROP DATABASE DBIVPØ1 ;  
   COMMIT;
   DROP STOGROUP SGIVPØ1 ;  
/*
//STEPØØ2 EXEC PGM=IKJEFTØ1
//SYSTSPRT DD SYSOUT=*  
//SYSPRINT DD SYSOUT=*  
//SYSTSIN DD *
DSN SYSTEM(DB2B)
RUN PROGRAM(DSNTEP2) PLAN(DSNTEP2)  
END
//*/
//SYSIN DD *
   CREATE STOGROUP SGIVPØ1
      VOLUMES('*')
      VCAT TDB2
   ;  
   COMMIT;
   CREATE DATABASE DBIVPØ1  
      STOGROUP SGIVPØ1
      BUFFERPOOL BPØ
   ;
/*
//STEPØØ3 EXEC PGM=IKJEFTØ1
```
/*
CREATE TABLESPACE TS1IVPØ1
   IN DBIVPØ1
   LOCKSIZE ANY
   USING STOGROUP SGIVPØ1
      PRIQTY 1
      SECQTY 1
;
COMMIT;
CREATE TABLE CRIVPØ1.TABLE1
   (COL1 CHAR(4),
    COL2 INTEGER,
    COL3 DECIMAL(5,2),
    COL4 VARCHAR(10))
   IN DBIVPØ1.TS1IVPØ1;
COMMIT;
*/

/*
STEPØØ4 EXEC PGM=IKJEFTØ1
//SYSTSPRT DD SYSOUT=* 
//SYSPRINT DD SYSOUT=* 
//SYSTSIN DD *  
DSN SYSTEM(DB2B) 
RUN PROGRAM(DSNTEP2) PLAN(DSNTEP2) 
END 
/** 
*/ 

/*
STEPØØ6 EXEC PGM=IKJEFTØ1 
//SYSTSPRT DD SYSOUT=*  
//SYSPRINT DD SYSOUT=* 
//SYSTSIN DD * 
DSN SYSTEM(DB2B) 
-TERM UTIL(DSNTEX) 
END 
/** 
*/ 

REXX source code for the sample application

The following source code shows how to use the DB2 REXX language support to issue SQL statements from a REXX procedure:

/* REXX */
SUBSYS = DB2B
SQLSTMT1 = "SELECT COL1, COL2, COL3, COL4 FROM CRIVP01.TABLE1"
SQLSTMT2 = "INSERT INTO CRIVP01.TABLE1",
            "VALUES('DDDD',4,444.44,'DDDDDDDD')"
SQLSTMT3 = "DELETE FROM CRIVP01.TABLE1",
            "WHERE COL1 = 'DDDD'"
SQLSTMT4 = "UPDATE CRIVP01.TABLE1",
            "SET COL4 = 'ZZZZZZZ'",
            "WHERE COL1 = 'AAAA'"
SQLSTMT5 = "UPDATE CRIVP01.TABLE1",
            "SET COL4 = ------",
            "WHERE COL1 = ------"
SQLSTMT6 = "SELECT COL1, COL2, COL3, COL4 FROM CRIVP01.TABLE1",
            "WHERE COL1 = ?"

ADDRESS TSO "SUBCOM DSNREXX" /* HOST CMD ENV AVAILABLE ? */
IF RC THEN S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX')
ADDRESS DSNREXX "CONNECT" SUBSYS
IF SQLCODE ≠ Ø THEN CALL SQLCA
CALL SELECT_ALL
CALL SELECT_ONE
CALL INSERT
CALL SELECT_ALL
CALL DELETE
CALL SELECT_ALL
CALL UPDATE
CALL SELECT_ALL
ADDRESS DSNREXX "DISCONNECT"
EXIT
SELECT_ALL:
    LINENUM = Ø
    SAY ""
SAY "SQLSTMT1: " SQLSTMT1
ADDRESS DSNREXX "EXEC SQL DECLARE C1 CURSOR FOR S1"
IF SQLCODE ≠ Ø THEN CALL SQLCA
ADDRESS DSNREXX "EXEC SQL PREPARE S1 FROM :SQLSTMT1"
IF SQLCODE ≠ Ø THEN CALL SQLCA
ADDRESS DSNREXX "EXEC SQL DESCRIBE S1 INTO :OUTSQLDA"
IF SQLCODE ≠ Ø THEN CALL SQLCA
ADDRESS DSNREXX "EXEC SQL OPEN C1"
IF SQLCODE ≠ Ø THEN CALL SQLCA
SAY "NUMBER OF COLUMNS: " OUTSQLDA.SQLD
DO UNTIL(SQLCODE ≠ Ø)
    ADDRESS DSNREXX "EXEC SQL FETCH C1 USING DESCRIPTOR :OUTSQLDA"
    IF SQLCODE = Ø THEN
        DO
            LINENUM = LINENUM + 1
            SAY "* ROW NUMBER: " LINENUM
            DO I = 1 TO OUTSQLDA.SQLD
                SAY " > COLUMN NUMBER: " I
                SAY " COLUMN NAME: " OUTSQLDA.I.SQLNAME
                SAY " COLUMN TYPE: " OUTSQLDA.I.SQLTYPE
                SAY " COLUMN VALUE: " OUTSQLDA.I.SQLDATA
            END
        END
    END
END
RETURN

SELECT_ONE:
    VAL_COL1 = 'AAAA'
    LINENUM = Ø
    SAY ""
    SAY "SQLSTMT6: " SQLSTMT6
ADDRESS DSNREXX "EXEC SQL DECLARE C6 CURSOR FOR S6"
IF SQLCODE ≠ Ø THEN CALL SQLCA
ADDRESS DSNREXX "EXEC SQL PREPARE S6 FROM :SQLSTMT6"
IF SQLCODE ≠ Ø THEN CALL SQLCA
ADDRESS DSNREXX "EXEC SQL DESCRIBE S6 INTO :OUTSQLDA"
IF SQLCODE ≠ Ø THEN CALL SQLCA
ADDRESS DSNREXX "EXEC SQL OPEN C6 USING :VAL_COL1"
IF SQLCODE ≠ Ø THEN CALL SQLCA
SAY "NUMBER OF COLUMNS: " OUTSQLDA.SQLD
DO UNTIL(SQLCODE ≠ Ø)
    ADDRESS DSNREXX "EXEC SQL FETCH C6 USING DESCRIPTOR :OUTSQLDA"
    IF SQLCODE = Ø THEN
        DO
            LINENUM = LINENUM + 1
            SAY "* ROW NUMBER: " LINENUM
            DO I = 1 TO OUTSQLDA.SQLD
                SAY " > COLUMN NUMBER: " I
                SAY " COLUMN NAME: " OUTSQLDA.I.SQLNAME
                SAY " COLUMN TYPE: " OUTSQLDA.I.SQLTYPE
            END
        END
    END
END
RETURN
SAY "    COLUMN VALUE: " OUTSQLDA.I.SQLDATA
END
END
RETURN
INSERT:
SAY ""
SAY "SQLSTMT2: " SQLSTMT2
ADDRESS DSNREXX "EXECSQL " SQLSTMT2
IF SQLCODE ¬= Ø THEN CALL SQLCA
ADDRESS DSNREXX "EXECSQL COMMIT"
IF SQLCODE ¬= Ø THEN CALL SQLCA
RETURN
DELETE:
SAY ""
SAY "SQLSTMT3: " SQLSTMT3
ADDRESS DSNREXX "EXECSQL " SQLSTMT3
IF SQLCODE ¬= Ø THEN CALL SQLCA
ADDRESS DSNREXX "EXECSQL COMMIT"
IF SQLCODE ¬= Ø THEN CALL SQLCA
RETURN
UPDATE:
SAY ""
SAY "SQLSTMT4: " SQLSTMT4
VAL_COL1 = 'AAAA'
VAL_COL4 = 'HHHHHHHH'
ADDRESS DSNREXX "EXECSQL " SQLSTMT4
IF SQLCODE ¬= Ø THEN CALL SQLCA
SAY ""
VAL_COL1 = '')BBB"
VAL_COL4 = 'YYYYYYYY'
SQLSTMT5  = "UPDATE CRIVPØ1.TABLE1",
           "SET COL4 = " || VAL_COL4 || "",
           "WHERE COL1 = " || VAL_COL1
SAY "SQLSTMT5: " SQLSTMT5
ADDRESS DSNREXX "EXECSQL " SQLSTMT5
IF SQLCODE ¬= Ø THEN CALL SQLCA
ADDRESS DSNREXX "EXECSQL COMMIT"
IF SQLCODE ¬= Ø THEN CALL SQLCA
RETURN
SQLCA:
   SAY "SQLCODE = " SQLCODE
   SAY "SQLSTATE = " SQLSTATE
EXIT

Running the DB2 REXX sample application
You run DB2 REXX procedures under TSO.
In a batch environment, you might use statements like these to invoke procedure REXX02SO:

```
//STEP001 EXEC PGM=IKJEFT01
//STEPLIB DD DISP=SHR,DSN=TLINK.DB2.SDSNLOAD
//SYSEXEC DD DISP=SHR,DSN=SMAINT.I990557.DB2.JCL
//SYSTSPRT DD SYSOUT=*  
//SYSPRINT DD SYSOUT=* 
//SYSTSIN DD * 
REXX02SO 
/*
```

The SYSEXEC dataset contains your REXX application, and the SYSTSIN dataset contains the command that you use to invoke the application.

**Result of the sample application**

When you run the sample application, you get the following result:

```
READY
REXX02SO
SQLSTMT1: SELECT COL1, COL2, COL3, COL4 FROM CRIVP01.TABLE1
NUMBER OF COLUMNS:  4
  * ROW NUMBER:  1
    > COLUMN NUMBER: 1
      COLUMN NAME:    COL1
      COLUMN TYPE:    453
      COLUMN VALUE:   AAAA
    > COLUMN NUMBER: 2
      COLUMN NAME:    COL2
      COLUMN TYPE:    497
      COLUMN VALUE:   1
    > COLUMN NUMBER: 3
      COLUMN NAME:    COL3
      COLUMN TYPE:    485
      COLUMN VALUE:   111.11
    > COLUMN NUMBER: 4
      COLUMN NAME:    COL4
      COLUMN TYPE:    449
      COLUMN VALUE:   AAAAAAAA
  * ROW NUMBER:  2
    > COLUMN NUMBER: 1
      COLUMN NAME:    COL1
      COLUMN TYPE:    453
      COLUMN VALUE:   BBBB
    > COLUMN NUMBER: 2
      COLUMN NAME:    COL2
      COLUMN TYPE:    497
```
<table>
<thead>
<tr>
<th>Column Number</th>
<th>Column Name</th>
<th>Column Type</th>
<th>Column Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>COL1</td>
<td>453</td>
<td>AAAAA</td>
</tr>
<tr>
<td>2</td>
<td>COL2</td>
<td>497</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>COL3</td>
<td>485</td>
<td>111.11</td>
</tr>
<tr>
<td>4</td>
<td>COL4</td>
<td>449</td>
<td>AAAAAAAA</td>
</tr>
</tbody>
</table>

SQLSTMT6: SELECT COL1, COL2, COL3, COL4 FROM CRIVPØ1.TABLE1 WHERE COL1 = ?

* Row Number: 1

Editor’s note: the example output continues in a similar vein.

Patrick Renard
CTRNE (France) © Xephon 2001
Have you been cubed? – further thoughts

The original article *Have you been cubed?* was printed in *DB2 Update*, Issue 98, December 2000. The DB2 expanded GROUP BY syntax is reprinted for your convenience:

```plaintext
>> GROUP BY grouping-expression |
    grouping-sets
    super-groups

>> GROUPING SETS grouping-expression |
    super-groups
    grouping-expression
    super-groups
```

The grouping-expression is the original DB2 that identifies the grouping columns.

Restrictions on its use are:

- Column names must be unambiguous.
- Total length must be <= 254 bytes.
- It cannot include scalar-fullselect, variant function, or external action.

GROUPING SETS allow multiple grouping clauses in a single statement. It is logically equivalent to a UNION of two or more row groups in a single result table. GROUPING SETS can be a simple element, parenthesis delimited element list, or super-group.

Super-group syntax looks like:

```plaintext
>> ROLLUP (grouping-expression-list)  
CUBE (grouping-expression-list)   
   grand-total (-)
```

ROLLUP calculates the *super-aggregate* sub-total rows applying the same column functions that are used to obtain regular rows.

CUBE returns a result table containing all the ROLLUP rows plus...
additional *cross-tabulation* rows where the GROUP BY syntax is *identical*.

The grouping-expression-list defines elements.

The grand-total is self-explanatory. ROLLUP and CUBE automatically compute a grand total so grand-total specification is unnecessary.

```
>>>GROUPING (expression) <<
```

is used to indicate whether a row specified by *expression* is a regular row or a super-aggregate row. It returns 0 or 1 where 0 means a regular row whereas 1 indicates a super-aggregate NULL row usable for sub-totals.

<table>
<thead>
<tr>
<th>alumni_id</th>
<th>surname</th>
<th>school_id</th>
<th>date</th>
<th>contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>124</td>
<td>paul</td>
<td>bus</td>
<td>Ø32ØØØ</td>
<td>95ØØ</td>
</tr>
<tr>
<td>124</td>
<td>paul</td>
<td>bus</td>
<td>Ø62ØØØ</td>
<td>9ØØØ</td>
</tr>
<tr>
<td>124</td>
<td>paul</td>
<td>bus</td>
<td>Ø92ØØØ</td>
<td>85ØØ</td>
</tr>
<tr>
<td>632</td>
<td>vesely</td>
<td>law</td>
<td>Ø32ØØØ</td>
<td>85ØØ</td>
</tr>
<tr>
<td>632</td>
<td>vesely</td>
<td>law</td>
<td>Ø62ØØØ</td>
<td>8ØØØ</td>
</tr>
<tr>
<td>632</td>
<td>vesely</td>
<td>law</td>
<td>Ø92ØØØ</td>
<td>75ØØ</td>
</tr>
</tbody>
</table>

*Figure 1: Alumni table values*

The alumni table in Figure 1 is used for all the examples.

Let’s assume a user wants to know the individual contributions in order by alumni, school, month, and year with subtotals and grand totals for each predicate:

```
SELECT alumni_id, surname, school_id, YEAR(date) AS year, MONTH(date) AS month, contribution
FROM alumni
GROUP BY GROUPING SETS
(ROLLUP(YEAR(date), MONTH(date), alumni_id),
 ROLLUP(YEAR(date), MONTH(date), school_id))
ORDER BY alumni_id, year, MONTH(date), contribution DESC
--year included in order by to process multiyear results
```

Figure 2 is displayable as is but is user-unfriendly! Users would probably be confused by what the values signify in ‘contribution’, why there are duplicate contribution values, and why there are empty columns (- denotes NULL). There are many techniques for making
Figure 2 user-friendly including using SQL GROUPING combined with COALESCE to designate which ROLLUP and whether rows are regular or super-aggregate. A simpler method is to use an embedded static SQL application program. You can use almost any programming language for embedded SQL ranging from COBOL to REXX. Any acceptable programming language has the capability of analysing Figure 2 to make it user-friendly. Below is the pseudocode:

* assumes program has stored result table in
  temporary table named Table-2
EXEC SQL
ALTER Table-2 ADD explanation CHAR(30)
* insert explanation column as last column
END-EXEC.

EXEC SQL
DECLARE alumni-cursor CURSOR FOR
  SELECT alumni_id, surname, school_id, year, month,
  contribution
  INTO Table-2
END-EXEC.

Figure 2: SELECT result table
contribution, explanation
FROM Table-2
FOR UPDATE OF explanation
END-EXEC.

EXEC SQL
OPEN alumni-cursor
END-EXEC.

EXEC SQL
WHENEVER NOT FOUND GO TO close
END-EXEC.

fetch
EXEC SQL
FETCH alumni-cursor
INTO :alumni-id, :surname, :school-id, :year, :month,
:contribution, :explanation
END-EXEC.

IF :month NULL THEN NEXT SENTENCE ELSE GO TO fetch.

EXEC SQL
UPDATE Table-2
SET :explanation = 'alumni total contribution'
WHERE CURRENT OF alumni-cursor
END-EXEC.

GO TO fetch.

IF :alumni-id NULL AND :surname NULL AND :school-id NULL THEN NEXT SENTENCE ELSE GO TO fetch.

EXEC SQL
UPDATE Table-2
SET :explanation = 'month contribution'
WHERE CURRENT OF alumni-cursor
END-EXEC.

GO TO fetch.

IF :alumni-id NULL AND :surname NULL AND :school-id NULL AND :month NULL THEN NEXT SENTENCE
ELSE    GO TO    fetch.

EXEC SQL
    UPDATE    Table-2
    SET    :explanation = 'annual contribution'
    WHERE CURRENT OF    alumni-cursor
END-EXEC.

GO TO    fetch.

IF    :alumni-id NULL
AND    :surname NULL
AND    :school-id NULL
AND    :month NULL
AND    :year NULL
THEN NEXT SENTENCE
ELSE    GO TO    fetch.

EXEC SQL
    UPDATE    Table-2
    SET    :explanation = 'grand total'
    WHERE CURRENT OF    alumni-cursor
END-EXEC.

GO TO    fetch.

IF    :alumni-id NULL
AND    :surname NULL
THEN NEXT SENTENCE
ELSE    GO TO    fetch.

EXEC SQL
    UPDATE    Table-2
    SET    :explanation = 'month total by school'
    WHERE CURRENT OF    alumni-cursor
END-EXEC.

GO TO    fetch.

IF    :alumni-id NULL
AND    :surname NULL
AND    :month NULL
THEN NEXT SENTENCE
ELSE    GO TO    fetch.

EXEC SQL
    UPDATE    Table-2
    SET    :explanation = 'grand total by school'
    WHERE CURRENT OF    alumni-cursor
END-EXEC.
GO TO fetch.

IF :alumni-id NULL AND :surname NULL AND :school-id NULL AND :month NULL THEN NEXT SENTENCE ELSE GO TO fetch.

EXEC SQL
  UPDATE Table-2
  SET :explanation = 'annual contribution'
  WHERE CURRENT OF alumni-cursor
END-EXEC.

GO TO fetch.


EXEC SQL
  UPDATE Table-2
  SET :explanation = 'grand total'
  WHERE CURRENT OF alumni-cursor
END-EXEC.

GO TO fetch.

close
EXEC SQL
  CLOSE alumni-cursor
END-EXEC.

terminate program

If your users would be unhappy about showing annual and grand totals twice then:

EXEC SQL
  DELETE FROM Table-2
  WHERE CURRENT OF alumni-cursor
END-EXEC.

The pseudo program above needs to know about the result table output. Therefore, you should print it before writing the code.
SUMMARY

1 Code and debug expanded GROUP BY SQL for user requests.
2 Print the result table.
3 Code the embedded static SQL application program using knowledge of the result table output to simplify IF statements for determining subtotal and grand total rows. You can probably build a library of programs to shorten the overall process.
4 Provide your users with a friendly report.

Combining embedded SQL programs with expanded GROUP BY permits sophisticated user-friendly reports.

Eric Garrigue Vesely
Principal Analyst
Workbench Consulting (Malaysia) © Xophon 2001

E-business enhancements with DB2 Universal Database Version 7.1

The term ‘e-Business’ has managed to stay ‘hot’, and with good reason. Businesses continue to move their applications and processes to run in the ‘e-space’. But what does this e-space entail? e-business isn’t only about selling books or auctioning off merchandise on the Web, e-business covers all facets of your business. When you look up directions to your favourite shop on the Internet, that’s e-business. When your company communicates corporate directives and programs to you over an intranet, that’s e-business. The IBM e-business model is illustrated in Figure 1. The road to e-business starts out with a transformation. In this phase, the company is dedicated to moving both internal and external lines of business functions to electronic format. In order to facilitate this, the company builds applications. In the next phase, the company architects an environment to run the new applications. Finally, the company leverages the knowledge of the information that it has collected. DB2 Universal Database (DB2) is an
integral component of the *run* and *leverage* quadrants. DB2 can provide your business with a scalable and robust environment in which to conduct business, as well as offering the analysis and metric tools to understand your business better.

CONSTRUCTS TO BUILDING E-BUSINESS APPLICATIONS

DB2 Version 7.1 offers seven major e-business constructs to help businesses not only build e-business applications, but also port applications from other relational database management systems. The seven major e-business constructs that have been added to this release of DB2 are declared temporary tables, application savepoints, identity columns, stored procedure language, stored procedure builder, nested stored procedures, and materialized generated columns.

Declared temporary tables

DB2 applications can now create and use temporary tables in applications. Previously, application developers had to create regular tables to function as temporary tables. These ‘fake’ tables would be logged by the database, their names had to be unique, and ultimately these tables had to be dropped; all of this was the responsibility of the application. Now, the application does not have to worry about managing temporary tables with this new feature – DB2 can do it for
you! There are no performance issues associated with these tables from locking, or catalog contention. A temporary table is available for the duration of a database connection and dropped when the application disconnects from the database.

If your current application creates tables to process large amounts of data and drops those tables once the application has finished manipulating that data, consider using declared temporary tables instead of regular tables. An example of an SQL statement that would create a temporary table is shown in Example 1:

```
CONNECT TO MYDB;
DECLARE GLOBAL TEMPORARY TABLE T1 (...)
    ON COMMIT PRESERVE ROWS NOT LOGGED;
```

**Application savepoints (aka compound SQL savepoints)**

You can now programmatically roll back to a specified savepoint rather than rolling back an entire unit of work. Savepoints are very similar to compound SQL statements, but have the added advantage of a great deal of control over rollback processing. An application savepoint enables the developer to group several SQL statements into an executable block. If any statement in this defined block fails, only that statement is rolled back. In the event of a failed transaction, the application developer can choose to roll back to the savepoint instead of rolling back the entire unit of work. Keep in mind, however, that when the block of SQL statements has finished executing, it still needs to be committed to the database. So, if two SQL statements were processed and the third was rolled back, that database would still not have a committed transaction written to the logs. Example 2 shows an application savepoint:

```
INSERT INTO t1
START SAVEPOINT
OPEN CURSOR c1 USING s1
ROLLBACK TO SAVEPOINT
```

**Identity columns**

Identity columns assign sequential numbers to rows in a table. Think of an identity column as a counter. In Example 3, an SQL statement
that creates a table is shown:

```
CREATE TABLE CUSTOMER (ACCOUNT SMALLINT NOT NULL
GENERATED ALWAYS AS IDENTITY (INCREMENT BY 10)
```

You can see that the *Account* column is an identity column, with a
defined increment of 10. The first row that gets inserted into this row
will be account 10. Each subsequent row inserted into this table will
have a sequential increment of 10 (second row 20, third row 30, … nth
row n*10). When you do not use identity columns to automatically
generate unique primary keys, a common design is to store a counter
in a table with a single row. Each transaction locks this table,
increments the number, and then commits the transaction to unlock
the counter. Unfortunately, this design allows only a single transaction
to increment the counter at a time. Driving this function into the
database engine resolves these problems and guarantees the unique
value that the application wants.

Application developers should be aware that, if a transaction happened
to roll back, a gap would be created in the sequence, which may or may
not be acceptable, depending on your business rules. For example,
assume that you have entered three new employees into the
EMPLOYEE table. You may set the EMPLOYEE_ID column as an
identity column (with an increment of 10) and defined it as the primary
key. If you inserted three employees into the table, but the second
insertion was rolled back, the sequence EMPLOYEE_ID column
would have two rows in it – 10 and 30. The next insert into this table
would have an EMPLOYEE_ID of 40. This is an inherent complexity
with identity columns.

**Stored procedure language**

DB2 Version 7.1 has implemented a non-proprietary and open stored
procedure language called SQL/PL. SQL/PL supports building DB2
stored procedures that employ SQL in a manner consistent with the
Persistent Stored Module definition of the ANSI SQL99 standard.
This means that all of your stored procedures are portable. PL/SQL is
enabled for DB2 for Unix, Windows, OS/2, AS/400, and OS/390.

In addition to the SQL/PL support, DB2 Version 7.1 includes support
for the creation of user-defined functions, table functions, and row
functions through the use of SQL statements. This reduces the need to write these functions in a high-level language. SQL-bodied functions are written completely in SQL so no external language is used. This means that SQL parameters do not need to be transformed into parameters of a host-language. It also means that, after the function completes the result, these SQL parameters do not need to be mapped back to SQL. Structured data types make use of SQL-bodied functions for transform functions and methods.

**Stored Procedure Builder**

The Stored Procedure Builder (SPB) in DB2 Version 7.1 is a Java-based graphical tool that supports the rapid development of DB2 stored procedures. Using the SPB, you can perform the following tasks:

- Create new stored procedures.
- Build stored procedures on local and remote DB2 servers.
- Modify and rebuild existing stored procedures.
- Run stored procedures to test and debug the execution of installed stored procedures.

The SPB in DB2 is integrated into the tools that applications developers commonly use to develop applications. For example, you can launch SPB from any of the following development or management applications:

- Microsoft Visual C++
- Microsoft Visual Basic
- IBM Visual Age for Java
- DB2 Control Center.

An example of the SPB is shown in Figure 2.

Furthermore, to help you with migrations from other vendors’ proprietary stored procedure languages (such as PL/SQL by Oracle and Transact-SQL by Microsoft) to the open standard supported by
DB2, IBM provides tools that you can download free of charge at www.ibm.com/software/data/db2/migration/.

**Nested stored procedures**

Your DB2 stored procedures can contain CALL statements to call other stored procedures. This feature, called nested stored procedures, enables you to reuse existing stored procedures and design more complex applications. These CALL statements can nest stored procedure calls up to 16 levels. Each host language can call only its own language. For example, a C-based stored procedure must call another C-based stored procedure.

**Materialized generated columns**

The materialized generated columns feature lets you create columns within tables that are based on other columns within the table. This feature could be simulated using an update and insert trigger, but implementing this feature directly into the table definition gives the user additional flexibility and performance.
Example 4 shows how a new column could be generated as the uppercase of the lastname column:

```sql
CREATE TABLE t1 (  
  name VARCHAR(20),  
  uname GENERATED ALWAYS AS (UCASE(name)));
CREATE INDEX i1 ON t1(uname);
SELECT name FROM t1 WHERE UCASE(name) = 'SMITH'
```

This column actually exists within the table, but it does not need to be included as part of an insert statement. The user can now create an index on this column value and do searches against it. This allows for a form of case-insensitive searching.

The support of materialized generated columns provides potential performance improvements in three areas:

- **Pre-computed expressions**
  
  Using generated columns, the user defines an additional column to assign the result of an expression. When issuing a query that includes the same expression, the generated column can be used directly, or query rewrite replaces the expression with the generated column.

- **Index on expressions**
  
  It is possible to create a non-unique index on a generated column. Using query rewrite as above allows for an inexpensive implementation of index on expression.

- **Increased JOIN choices for the optimizer**
  
  A query of the form `SELECT c1, c2, c3 FROM T1, T2 WHERE T1.c1 = T2.c2 + T2.c3` usually ends up with a nested loop join because the optimizer does not handle other forms of joins on expressions. If the user adds a column `c4 = c2 + c3` to T2, query rewrite can substitute the expression by `c4` and the optimizer has a choice of possibly better join strategies.
numeric and character data to images, video, voice, complex
documents, spatial objects, and more. You can bring any or all of these
data types together in one SQL query and manipulate them with
powerful built-in functions. DB2 Version 7.1 introduces the XML and
Net.Search Extenders – each will help customers transact over the
Web.

**XML Extender**

The XML Extender provides new data types that let you store XML
documents in DB2 databases and new functions that help you work
with these structured documents. Entire XML documents can be
stored in DB2 databases as character data or stored as external files
that are still managed by DB2. Retrieval functions allow you to
retrieve either the entire XML document or individual elements or
attributes.

Furthermore, the XML Extender gives you powerful control over the
way XML documents are stored in the database. For example, you can
store your XML document either in a *column* or as a *collection*. In the
first case, the whole XML document is stored in a column; in the
second, the document is stored in rows and columns as relational data.
The method of storage has no effect on the composition and
decomposition of XML documents by the database engine.

You should also be aware that Net.Data, which connects Web
applications to DB2, now has built-in XML exploitation. This allows
you to generate XML tags as output from your Net.Data macro,
instead of having to manually enter the tags.

**Net.Search Extender**

The DB2 Net.Search Extender contains a DB2 stored procedure that
adds the power of fast full-text retrieval to Net.Data Java or DB2 CLI
applications. It offers application programmers a variety of search
functions, such as fuzzy searching, stemming, Boolean operators, and
section searching.

Searching using Net.Search Extender can be particularly advantageous
with applications designed to run over the Internet, when search
performance on large indexes and scalability according to concurrent queries are important factors. The Net.Search Extender is considered to be an ‘in-memory’ database search. The Net.Search Extender avoids expensive table look-ups at search time, which you would normally use when displaying the results of a search. It does this by letting you store table columns in main memory at indexing time. At search time, you can extend the results list using the in-memory information without having to access the table.

DB2 Net.Search Extender is similar to the DB2 Text Extender, but provides a higher search speed. There are some trade-offs, however. Some of the powerful features and granular searches that are available in the Text Extender are not available in the Net.Search Extender. The Net.Search Extender is packaged with features that make it useful in a Web environment. Figure 3 shows the extent of the scalability of the Net.Search Extender. This example was based on an on-line book e-tailer.

As you can see, an almost hundredfold increase in users (registered as hits/day) had minimal affect on the response time of the database.

OTHER ENHANCEMENTS
There are some other miscellaneous enhancements to DB2 Version 7.1. This section briefly covers just two of them.
Active log size limit increased to 32GB

The active log size limit was increased from 4GB to 32GB in DB2 Version 7.1. This gives you the ability to handle longer-running transactions and not be concerned with running out of log space (which would ultimately shut down your database). While it is not recommended that your application’s transactions contain 32GB of information without a commit to the database, the increased limit can give you more peace-of-mind.

Ability to execute ODBC/CLI (dynamic) applications as static applications

DB2 Version 7.1 allows you to convert your ODBC queries into static SQL queries. This provides a performance gain and increased security for applications that use canned queries. A canned query is a query that is infrequently changed. This helps transactions over the Web since the query does not have to be prepared and compiled as a dynamic query does.

With DB2 Version 7.1, not only does IBM continue its role as the database innovator in e-business, but IBM also makes it easy – with ease of use and lower total cost of ownership – for customers to transform their operations to take advantage of these new technologies. If you conduct business over the Web, don’t miss the opportunities offered by the e-business enhancements in DB2.

Paul Zikopoulos  
Database Specialist with the DB2 Sales Support team  
IBM (Canada)
DB2 fast close and start

INTRODUCTION
This article describes some batch REXX procedures that provide an automated and fast way to stop DB2.

STOP AND START USER DATABASE
In order to release memory from the DBM1 address space this function perform a stop and a start of all user databases.

DB2 FAST STOP SUBSYSTEM
One of the most frequent problems in a DB2 environment where a lot of spacenames are opened for work is the long time needed for DB2 shutdown. When the system operators do the -STOP DB2 command, each spacename opened (TABLESPACE/INDEXSPACE) must be kept by the DSNxDBM1 address space in sequence mode. To reduce this time I have created a batch REXX procedure that performs a parallel STOP command for every database in the DB2 subsystem. This STOP command will produce a faster way to keep all DB2 VSAM spaces.

START DB2 SUBSYSTEM
This function performs the DB2 start subsystem in ACCESS(*) or ACCESS(MAINT) according to your own needs.

PARAMETER DESCRIPTION
The following parameters describe how you can customize the REXX EXEC batch procedure:

- SUBSYS – DB2 subsystem name.
- MAXJOB – number of stop commands executed at the same time (1 - 8).
• STOSYS – variable to stop DB2 subsystem (STOP=YES / STOP=NO)

• STASYS – variable to start DB2 subsystem (START=YES / START=MAINT / START=NO).

The procedure may be customized for several functions as follows:

• STOP/START USER database:
  'STOP=no,START=no'

• STOP DB2:
  'STOP=yes,START=no'

• STOP DB2 + START DB2:
  'STOP=yes,START=yes'

CHECKLIST FOR INSTALLATION

Follow these steps to install the components of the REXX procedure:

• Allocate a USER.LIBRARY.

• Copy all REXX, macro, PROC, and PARMs into the USER.LIBRARY:
  – REXX: §DB2STP0, §DB2STP1, §DB2STP2, §DB2PAR0, §DB2SDSF, §DB2ALL0
  – Macro: §MDB2039, §MDB2001, §MDB2037, §MDB2036
  – PROC: DB2REXX1
  – Sample job: STOPDSNZ.

• Customize §DB2PAR0 REXX for the global environment setting highlighted variables.

• Download program DELAY00 from the Xephon Web site and put it into a library available during batch execution.

  The Xephon Web address is www.xephon.com/cgi-bin/xephon2/tdisplay?/m115a03.txt.

• Customize PROC DB2REXX1 according your environment.
• Customize job STOPDSNZ according to your environment.

The sample jobs and parameter are customized for a DB2 subsystem whose name is DSNZ and the DB2 subsystem command prefix is ‘+’.

The test environment is DB2 Version 5 in an OS/390 Version 6 environment.

@DB2STP0 REXX EXEC
/* REXX */
trace ?o

/*--------*   DB2 Fast Close & Start   *----------*/
arg parmin ; parm = translate(parmin,' ',',')
nparm   = words(parm); subsys = word(parm,1); maxjob = word(parm,2)
stosys  = word(parm,3); stasys  = word(parm,4)

elapsed = time('R')

/*---  Test input parameter    ---*/

/elom00*/

if nparm < 4 then do ; say '' ; say '' ; say '>>>>>>>>'
say '>>>>>>>>  Parameter list is incomplete !!!!! ' parmin
say '>>>>>>>>' ;say '' ; say '' ; exit ; end

if maxjob > 8 then do ; say '' ; say '' ; say '>>>>>>>>>

say '>>>>>>>> The variable MAXJOB is wrong !!!!!'
say '>>>>>>>> Specify a number between 1 - 8'
say '>>>>>>>>' ;say '' ; say '' ; exit ; end

if stosys ¬= 'STOP=NO' & stosys ¬= 'STOP=YES' then do
say '' ; say '' ; say '>>>>>>>>'
say '>>>>>>>>  The variable STOSYS is wrong !!!!!'
say '>>>>>>>> Specify : STOP=YES/STOP=NO'
say '>>>>>>>>' ;say '' ; say '' ; exit ; end

/endom1*

if stasys ¬= 'START=NO' & stasys ¬= 'START=YES' &

stasys ¬= 'START=MAINT' then do ; say '' ; say '' ; say '>>>>>>>>>
say '>>>>>>>> The variable START is wrong !!!!!
say '>>>>>>>> Specify : START=YES/START=NO/START=MAINT'
say '>>>>>>>>' ;say '' ; say '' ; exit ; end

/endom2*/

if stosys = 'STOP=NO' & (stasys = 'START=YES' | stasys = START='MAINT')

then do
say '' ; say '' ; say '>>>>>>>>>' ; say '>>>>>>>>>
say '>>>>>>>> Unable to Start DB2 without Stop !!!!!
say '>>>>>>>>' ;say '' ; say '' ; exit ; end

/endom3*/

/*--  Parameters assignment   ---*/
Call @db2par subsys ; if word(result,1) = 99 then exit

$1par =word(result,1); $accn =word(result,2); $class
/*---   Work areas initialization   ---*/
blk     =  ; ct1 = Ø ; #a = Ø ; #b = Ø ; today = date(w) ; ora = time()
wr1     = substr(subsys,4,1) ; outisf1 = $hiwork'.'subsys'.@DB2SDSF.ISFOUT'
outisf2 = $hiwork'.'subsys'.@DB2SDSF.ISFIN'
/*implØ4*/
/*---  Reset parameter into job   ---*/
inpds1 = $jcllib'(STOP'subsys')'
xx=OUTTRAP(trpØ9.) ; "ispexec edit dataset('"inpds1"') macro(@mdb2Ø37)"
xx=OUTTRAP(OFF)
if rc > Ø then do ; do #a = 1 to trpØ9.Ø ; say trpØ9.#a ; end ; exit ; end
"ispexec lminit dataid("inp1") dataset('"$jcllib"') enq(shr)"
"ispexec lmmstats dataid("inp1") member('"STOP'subsys")
user("DEFAULT")"
/*---  Sysin allocation file   ---*/
say '" ; outdsØ = $hiwork'.subsys'.@DB2STPØ.SYSIN'
prmalloc = subsys' 'outdsØ' Ø 5,1 f,b 8Ø 2792Ø sysin yes'
call @db2allØ prmalloc ; if word(result,1) = 99 then exit ; jobw = sysin
sk.1='   SELECT NAME FROM SYSIBM.SYSDATABASE         '  
sk.2='    WHERE NAME NOT LIKE ''DSN%''   ;           '  
sk.Ø=2; Call writerec
/*---  Systsinp allocation file   ---*/
outds1= $hiwork'.subsys'.@DB2STPØ.SYSTSINP'
prmalloc = subsys' 'outds1' Ø 1,1 f,b 8Ø 2792Ø systsinsp yes'
call @db2allØ prmalloc ; if word(result,1) = 99 then exit
sk.1='DSN SYSTEM('"subsys")

sk.2='RUN PROGRAM('step2pgm') PLAN('step2pln') LIB('''$runlib''')    
sk.3='END     

/*---  Sysprint allocation file   ---*/
outds2= $hiwork'.subsys'.@DB2STPØ.SYSPRINT'
prmalloc = subsys' 'outds2' Ø 15,15 f,b,a 121 121Ø sysprint yes'
call @db2allØ prmalloc : if word(result,1) = 99 then exit

/*---  Stop/Start allocation job file   ---*/
outdssts= $hiwork'.subsys'.@DB2STPØ.JOBSTST'
prmalloc = subsys' 'outdssts' Ø 15,15 f,b ØØ 2792Ø fists yes'
call @db2allØ prmalloc ; if word(result,1) = 99 then exit

/*---  DB2 query to extract data   ---*/
xx=outtrap(trpØ4.) ; address tso "ex "'outds1"'" ; xx=outtrap(off)
if rc > Ø then do ; analisi = substr(trpØ4.1,1,8)
    if analisi = 'DSNE1ØØI' then do ; subs = center(subsys,1Ø)
        say '' ; say '' ; say '>>>>>>' ; say '>>>>>>'
        say '>>>>>> DB2 subsystem 'subs'<--- is not active'
        say '>>>>>>' ; say '' ; say '' ; end
    else do ; do #a = 1 to trpØ4.Ø ; say trpØ4.#a ; end ; end ; exit ; end

/*---  Macro to process sysprint output   ---*/
xx=OUTTRAP(trpØ5.) ; "ispexec edit dataset(''outds2'') macro(@mdb2ØØ1)"
xx=OUTTRAP(OFF)
if rc > Ø then do ; do #a = 1 to trpØ5.Ø ; say trpØ5.#a ; end ; exit ; end

/*---  Start process output   ---*/
xx=outtrap(trpØ6.)
"execio * diskr sysprint (stem sysprint. finis" ; "free
fi(sysprint)"
xx=outtrap(off)
if rc > Ø then do
    do #a = 1 to trpØ6.Ø ; say trpØ6.#a ; end
    say '' ; say '' ; say '>>>>>>>>' ;
    say '>>>>>>>> Error reading file ''outds2''' ;
    say '>>>>>>>> RC='rc'. Verify.' ;
    say '>>>>>>>>' ; say '' ; say '' ; exit ; end
if sysprint.Ø = Ø then do ; say '' ; say '' ; say '' ; say '>>>>>>>>>>'
    say '>>>>>>>> The file ''outds2''' is empty.' ;
    say '>>>>>>>> Probably, there are problems during DB2 access !!!!'
    say '>>>>>>>>' ; say '' ; say '' ; exit ; end

/*---  Write Stop/Start jobs   ---*/
Do d = 1 to sysprint.Ø
    dbsel = word(sysprint.d,2) ; jobw = FISTS
    "alloc da(''outdssts'' ) f(''jobw'' ) mod reuse"
    if ctl > maxjob then ctl = Ø
    ctl = ctl + 1 ; jna = STOPDB || substr(subsys,4,1) || ctl
    nr = center(d,5) ; db = center(dbsel,8)
sk.1='//'jna' JOB ('$accn'),,'DB2 Fast Close',CLASS=''$class',

sk.2='/' MSGCLASS=Z,USER=''$user',REGION=''$region',

sk.3='/' MSGLEVEL=(''$msglvl'')
sk.4='*JOBPARM BYTES=999999,LINES=9999
sk.5='*-------------------------------------------------- *
sk.6='* ------ Stop/Start 'db' ------ *
sk.7='*-------------------------------------------------- *
sk.8='JOBLIB DD DISP=SHR,DSN='$dsnload
sk.9='*STEPØØ EXEC PGM=IKJEFTØ1,DYNAMNBR=2Ø
sk.10='*SYSTSPRT DD SYSOUT=Z
sk.11='*SYSTSIN DD *
sk.12=' DSN SYSTEM('subsys')
sk.13=' -STOP DB('dbsel') SPACENAM(*)
sk.14=' -START DB('dbsel') SPACENAM(*)
sk.15='*-------------------------------------------------- *

    sk.Ø=15 ; Call writerec ; end
xx=outtrap(trpØ7.) ; address tso "submit "outdssts"");
xx=outtrap(off)
if rc > Ø then do ; do #a = 1 to trpØ7.Ø ; say trpØ7.#a ; end ; exit ; end
say ''; say ''; say time()
say time() '---->> 'sysprint.Ø' Stop/Start jobs has been submitted ....
say time(); say ''; say ''
if stosys = 'STOP=YES' then do
    /*--- Stop/Start user DB monitor ---*/
say ' ' ; "******************************************************************' ;
say ' ' ; 'time()' ' '*
say ' ' ; Stop/Start user Database is running '*
say ' ' ; '*
say ' ' ; Please Stand By........ '*
say ' ' ; '*
say ' ' ; "******************************************************************' ;
say ' ' ; "free fi(sysin)" ; "free fi(systsinp)" ; "free fi(fists)"
    Call @db2stpl subsys ; if word(result,1) = 99 then exit
/**implØ5*/
/**implØ6*/
/*--- Start DB2 after STOP ---*/
if Stasys = 'START=YES' | Stasys = 'START=MAINT' then do
    address tso "delayØØ 3Ø" ; parmsdsf = subsys ||
    '.subsys'CHLO,15,12Ø'
    Call @db2sdsf parmsdsf ; if word(result,1) = 99 then do
        $exitc = 99 : return $exitc ; exit ; end
/**implØ7*/
if word(result,1) = ØØ then do ; say '' ; say '>>>>>>>>>>'
    if Stasys = 'START=YES' then
        say '>>>>>>>>>> Starting in progress for DB2 subsystem 'subsys
else
    say '>>>>>>>>>> Starting in progress for DB2 subsystem 'subsys' in ACCESS MAINT' ; say '>>>>>>>>>>'; say '' ; say '' ; end
/*--- Start DB2 ---*/
jobw = FIOUT ; "alloc da("outdsØ") f("jobw") mod reuse"
com1 = "$TA" || wr1 || 001 ; com2 = CA || wr1 || 001
if Stasys = 'START=YES' then com5 = 'STA DB2'
else com5 = 'STA DB2,ACCESS(MAINT)'
sk.1 = "/*com1","l=10,"'com1","$VS,"""$ctsubs||com5""":"com2""""
sk.0=1 ; Call writerec
xx=outtrap(trp10.) ; address tso "submit ""outds0"""" ;
xx=outtrap(off)
if rc > 0 then do ; do #a = 1 to trp10.0 ; say trp10.#a ; end ;
exit ; end
/*impl08*/
end
elapsed = time('R') ; elapsed = elapsed % 60 ; say '' ; say '>>>>>>>>>
say '>>>>>>>>' ; say '' ; say '' ; end ; Call Free ; exit
/*--- Write routine output records ---*/
Writerec :
"EXECIO * DISKW "jobw" (STEM sk. FINIS"
Clean:
DO f = 1 to sk.0 ; sk.f = blk ; end ; return
/*--- Free dataset ---*/
Free :
xx=outtrap(trpdummy.)
address tso "delete ""outds0"""" ; address tso "delete ""outds1""""
address tso "delete ""outds2"""" ; address tso "delete ""outdssts"""
address tso "delete ""outisf1"""" ; address tso "delete ""outisf2"""
xx=outtrap(off) ; return
@DB2STOP1 REXX EXEC
/*/ REXX */
trace ?o
******/--- DB2 STOP MONITOR ---*/
arg parmin ; parm = translate(parmin,' ',',')
nparm = words(parm) ; Subsys = word(parm,1)
/*--- Test input parameter list ---*/
if nparm < 1 then do ; say '' ; say '' ; say '>>>>>>>>>
say '>>>>>>>>' ; say '' ; say '' ; $exitc = 99 ; return $exitc ;
exit ; end
/*--- Parameters assignment ---*/
Call @db2parØ subsys ; if word(result,1) = 99 then do
$exitc = 99 ; return $exitc ; exit ; end
$1par =word(result,1); $accn =word(result,2); $class
=word(result,3)
$msgcla =word(result,4); $region =word(result,5); $msglvl
=word(result,6)
$notif =word(result,7); $user =word(result,8); $unitda
=word(result,9)
$unitta =word(result,10); $esunit =word(result,11); $prt
=word(result,12)
$hiwork =word(result,13); $db2ver =word(result,14); $ctsubs
=word(result,15)
/*---   Work areas initialization   ---*/
blk    =   ; loop00 = Ø
/*---   Verify Stop/Start command   ---*/
parmsdsf = subsys || ',STOPDB' substr(subsys,4,1)'*,9Ø,8Ø'
Call @db2sdsf parmsdsf
if word(result,1) = 99 then do ; $exitc = 99 ; return $exitc ; exit ; end
if word(result,1) = 98 then do ; ctr = right(word(result,2),3,'Ø')
say '' ; say '>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>'
say '>>>>>>>         !!!!!!   W A R N I N G    !!!!!!       >>>>>>>'
say '>>>>>>>                                                >>>>>>>'
say '>>>>>>>2 hours have passed since the beginning of STOP >>>>>>>'
say '>>>>>>> of user database. There are still 'ctr' jobs   >>>>>>>'
say '>>>>>>> running. Verify the cause of the problem.     >>>>>>>'
say '>>>>>>> Exit from DB2 fast Stop close procedure        >>>>>>>'
say '>>>>>>>                                                >>>>>>>'
say '>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>'
say '' ; say '>>>>>>>>'
say '>>>>>>>> DB2 Stop/Start user DataBase ended. Start DB2 Stop
precedure'
say '>>>>>>>>>' ; say '' ; end
/*-- Verify and cancel active thread   --*/
say '>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>'
say '>>>>>>> Verify & CANCEL THREAD still active           >>>>>>>'
say '>>>>>>>                                                >>>>>>>'
say '' ; say '' ; say ''
Call @db2stp2 subsys ; if word(result,1) = 99 then do
  $exitc = 99 ; return $exitc ; exit ; end
/*--- Stop DB2 subsystem   ---*/
outdsØ= $hiwork'.'subsys'.@DB2STPØ.SYSIN' ; jobw = FIOUt
"alloc da("outdsØ") f("jobw") shr reuse"
wr1 = substr(subsys,4,1)
jname = STOP || subsys ; com1 = "$TA" || wr1 || 000
com2 = CA || wr1 || ØØ : com3 = "$TA" || wr1 || ØØ
com4 = CA || wr1 || ØØ1 ; ct = $ctsubs
sk.1= "/*/com1",l=1Ø,"com1","$VS,","ce"STOP DDF
MODE(FORCE)"":"com2""
sk.2= "/*/com3",l=6Ø,"com3","$VS,","ce"STOP DB2"":"com4""
sk.Ø=2 ; Call writerec
xx=outtrap(trpØ3.) ; address tso "submit "outdsØ"" ; xx=outtrap(off)
if rc > Ø then do ; do #a = 1 to trpØ3.Ø
say trpØ3.#a ; end ; $exitc = 99  ; return $exitc ; exit ; end
/*--- Verify DB2 address space ---*/
say '' ; say ' ******************************************************************
say ' * DB2 Address Space close monitor *
say ' * subsys'DIST/'subsys'MSTR/'subsys'DBM1/'subsys'IRLM *
say ' * Please Stand By....... *
say ' * *
say ' ******************************************************************
; say ''
do while ctr2 ¬= Ø
   address tso "delayØØ 15"
   loopØØ = loopØØ + 1 ; ctrl1 = Ø ; ctrl2 = Ø ; ctrl3 = Ø
   cvtmser=d2x(c2d(storage(lØ,4))+6Ø) ;
   base=d2x(c2d(storage(cvtmser,4)))
   cscb=d2x(c2d(storage(base,4)))
do while cscb ¬= 'ØØØØØØØØ'
   nameaddr=d2x(x2d(cscb)+8) ; jobname=storage(nameaddr,8)
   jobstopØ = DSN || substr(subsys,4,1)
   jobstop1 = DSN || substr(subsys,4,1) || DBM1
   jobstop2 = DSN || substr(subsys,4,1) || IRLM
   /*--- Attivo DSNx* ---*/
   if left(jobname,4) = jobstopØ then do ; ctr2 = ctrl2 + 1 ; end
   /*--- DSN?DBM1 active ---*/
   if left(jobname,1) ¬=' ' & left(jobname,8) = jobstop1 then do
      ctr1 = ctrl1 + 1 ; end
   /*--- Only DSN?IRLM already active ---*/
   if left(jobname,1) ¬=' ' & left(jobname,8) = jobstop2 then do
      ctr3 = ctrl3 + 1 ; end
   cscb=d2x(c2d(storage(cscb,4))) ; end
if ctrl1 = Ø & ctrl2 = Ø & ctr3 = Ø then do ; say '' ; say '>>>>>>>>>' ;
   say '>>>>>>>>> All DB2 address spaces are ended. DB2 Stop OK'
   say '>>>>>>>>>' ; say '' ; leave ; end
   /*--- Only DSN?DBM1 already active ---*/
if loopØØ < 8 & ctrl2 = 1 & ctrl1 = 1 & ctrl3 = Ø then do
   say '' ; say '>>>>>>>>>' ;
   say '>>>>>>>>> I am waiting for DSN'wr1'DBM1 ending '
   say '>>>>>>>>>' ; say '' ; end
/* After two minutes from db2 stop command, DSNxDBM1 */
/* is already active. Cancel address space */
if loopØØ = 8 & ctrl2 = 1 & ctrl1 = 1 & ctrl3 = Ø then do

jobw = FIOOUT ; "alloc da('"outdsØ"') f("jobw") shr reuse"

sk.1= "/**VS,'C DSN'' ||substr(subsys,4,1)|| 'DBM1''

sk.2= "/**VS,'FORCE DSN'' ||substr(subsys,4,1)|| 'DBM1''

sk.Ø=2 ; Call writerec

xx=outtrap(trpØ3.); address tso "submit ''outdsØ''" ;

if rc > Ø then do ; do #a = 1 to trpØ3.Ø

  say trpØ3.#a ; end  ; $exitc = 99 ; return $exitc ; exit ;
end

say ' ' ; say ' ' ; say ' ' ; time()

say ' ' ; say ' ' ; say ' ' ; say 'The DSN'wr1'DBM1 don''t close.

say ' ' ; say 'The address space will be cancelled ||||

say ' ' ; say ' ' ;

end

/*--- exit for long wait ---*/

if loopØØ > 12Ø & ctr2 > 1 then do ; say ' ; say ' ' ; say ' ' ; say '

  say ' ' ;

end

/*--- exit for problem during ending phase ---*/

if ctrl1 = Ø & ctr2 > 1 &ctr3 = Ø then do ; say ' ' ; say ' ' ; say ' ' ; say 'Problem during DB2 stop processing. There are

  say ' ' ;

say ' ' ;

  say 'DSN'wr1'MSTR/DSN'wr1'DIST already active but DSN'wr1'DBM1

is not present'

  say ' ' ;

say ' ' ; say ' ' ; $exitc = 99 ; return $exitc ; exit ;
end

/*--- exit for IRLM problem in ending phase ---*/

if loopØØ > 6Ø & ctrl1 = Ø & ctr2 = 1 & ctr3 = 1 then do ; say ' ' ; say ' ' ; say ' ' ; say 'Prob

say ' ' ;

say ' ' ;

say ' ' ;

  say ' ' ;

end

"free fi(fiout)" ; exit

"EXECIO * DISKW "jobw" (STEM sk. FINIS"

DO f = 1 to sk.Ø ; sk.f = blk ; end ; return

@DB2STP2 REXX EXEC

/* REXX */

trace ?o
/*-----------*    DISPLAY & CANCEL THREAD    *----------*/

arg parmin ; parm = translate(parmin,' ',',')
nparm  = words(parm) ; subsys = word(parm,1)

/*---  Test input parameter list   ---*/
if nparm < 1 then do ; say '' ; say '' ; say '>>>>>>>>'
say '>>>>>>>> Parameter list is incomplete ||||| ' parmin
say '>>>>>>>>' ; say '' ; say '' ; $exitc = 99 ; return $exitc;
exit ; end

/*---  Parameters assignment   ---*/
call @db2parØ subsys
if word(result,1) = 99 then do ; $exitc = 99 ; return $exitc ; exit ; end
$lpar   =word(result,1);  $accn    =word(result,2);  $class
=word(result,3)
$msgcla =word(result,4);  $region  =word(result,5);  $msglvl
=word(result,6)
$notif  =word(result,7);  $user    =word(result,8);  $unitda
=word(result,9)
$unitta =word(result,10); $esunit  =word(result,11); $prt
=word(result,12)
$hiwork =word(result,13); $db2ver  =word(result,14); $ctsubs
=word(result,15)
$librexx =word(result,16); $parmlib =word(result,17); $proclib
=word(result,18)
$jcllib  =word(result,19); $report  =word(result,20); $isptenu
=word(result,21)
$isppenu =word(result,22); $ispmenu =word(result,23); $ispslib
=word(result,24)
$plilink =word(result,25); $sibmlnk =word(result,26); $sortlib
=word(result,27)
$runlib  =word(result,28); $dsnload =word(result,29); $tep2pgm
=word(result,30)
$tep2pln =word(result,31); $unlopgm =word(result,32); $unlopln
=word(result,33)
$dsnproc =word(result,34)

/*---   Work areas initialization   ---*/
blk = #a = Ø ; #c = 1 ; #d = Ø ; #v = 1 ; fineelab = no ;
clearrec = no
timeexit = Ø ; tcanjob = STOP || subsys
address ispexec 'vput (tcanjob) profile'
address ispexec 'vput (tomegdb) profile'

/*---  Sysin allocation file   ---*/
"alloc dummy f(sysin)"

/*---  Systsinp allocation file   ---*/
outds1= $hiwork'.subsys'.@DB2STP2.DTHREAD.SYSTSINP'
prmalloc = subsys' 'outds1' Ø 5,1 f,b 8Ø 2792Ø systsinp yes'
call @db2allØ prmalloc
if word(result,1) = 99 then do ; $exitc = 99 ; return $exitc ; exit ; end

/*---  Systsprt allocation file   ---*/
outds2='$hiwork'.subsys,@DB2STP2.DTHREAD.SYSTSPRT'
prmalloc = subsys'outds2' Ø 45,15 f b Ø 27920 systspri yes'
call @db2allØ prmalloc
if word(result,1) = 99 then do ; $exitc = 99 ; return $exitc ; exit ;
end

/*-- Display Thread ---*/
call Dthread ; analisi = substr(trpcmdØØ.1,1,8)
select

/*--- DB2 subsystem not active ---*/
when analisi = 'DSNE1ØØI' then do
  subs = center(subsys,1Ø) ; say '>>>>>>' ; say '>>>>>>'
say '>>>>>> DB2 subsystem --->'subs'<--- is not active'
say '>>>>>>' ; say '>>>>>>' ; say '' ; say ''
call free ; $exitc = 99 ; return $exitc ; exit ; end

/*--- User not authorized ---*/
when analisi = 'DSN9Ø16I' then do
  say '>>>>>>' ; say '>>>>>>' ; say '>>>>>> User not authorized !!!!!'
say '>>>>>>' ; say '>>>>>>' ; say '' ; say ''
call free ; $exitc = 99 ; return $exitc ; exit ; end

/*--- Output display O.K. ---*/
when analisi = 'DSNV4Ø1I' then do
  /*--- No DB2 connection ---*/
  if systspri.Ø = Ø then do ; fineelab = YES ; end
  /*--- Prepare CANCEL command ---*/
  else do
    jobw = systsinp ; "alloc da('outds1') f("jobw") shr reuse"
sk.1='DSN SYSTEM('subsys')'
    DO #b = 1 to systspri.Ø
      dthr_nword = words(systspri.#b)
      dthr_last = word(systspri.#b,dthr_nword)
      if dthr_last > Ø then do ; #c = #c + 1
        sk.#c='CANCEL THREAD('word(systspri.#b,dthr_nword)')' ; end
      dthr_conn = word(systspri.#b,1) ; end
  /*--- Test active THREAD ---*/
  if #c = 1 then fineelab = YES
  else do
    sk.Ø=#c ; call Writerec ; tipcom = 'CANCEL Thread'
    Call Rundb2 ;end ; end ; end
  /*--- Output non previsto ---*/
  otherwise
    say '' ; say '>>>>>>>>>>' ; say '>>>>>>>>>> Unpredictable Error.'
    Exit.
    say '>>>>>>>>>>' ; say '>>>>>> $exitc = 99 ; return $exitc ; exit ; end
  Do while fineelab ¬= YES
    address tso "delayØØ Ø15"
call Dthread ; #c = 1
    DO #b = 1 to systspri.Ø
      dthr_nword = words(systspri.#b);dthr_last =
      word(systspri.#b,dthr_nword)
if dthr_last > Ø then do
   #c = #c + 1 ; dthr_cicsallØ = word(systspri.#b,5)
   dthr_cicsall1 = word(systspri.#b,6)
   if dthr_cicsallØ = 'CICSALL' | dthr_cicsall1 = 'CICSALL' then
      do
         #d = #d + 1
         jk.#d= "/*$VS, 'F CICS' ||word(systspri.#b,1)' ,DSNC STOP FORCE'''
         end ; end ; end
      if #c = 1 then fineelab = YES
      /*--- No DB2 connection ---*/
      if systspri.Ø = Ø then do ; fineelab = YES ; iterate ; end
      /*--- DSNC stop for CICS connection already active ---*/
      if timeexit = 40 & #d > Ø then do ; say '' ; say ' >>>>>>>>
      say ' >>>>>>>> 10 minutes have passed since last CANCEL THREAD command.'
      say ' >>>>>>>> There are CICS connections already active. Start DSNC STOP'
      say ' >>>>>>>> FORCE command for those connections.'
      say ' >>>>>>>>'' ; say '' ; say ''
      /*--- Write & submit DSNC command ---*/
      outdsØ= $hiwork'.subsys'.@DB2STPØ.SYSIN'
      jobw = FIOUT ; "alloc da("outdsØ") f("jobw") shr reuse"
      sk.1= jk.1
      if #d > 1 then do ; do #z = 2 to #d
         #y = #z - 1
         if jk.#z ¬= jk.#y then do
            #v = #v + 1 ; sk.#v = jk.#z ; end ; end ; end
      sk.Ø= #v ; Call writerec
      xx=outtrap(trpØ6.) ; address tso "submit "outdsØ"'' ;
      xx=outtrap(off)
      if rc > Ø then do ; do #a = 1 to trpØ6.Ø
         say trpØ6.#a ; end ;
      $exitc = 99  ; return $exitc ; exit ;
      end ;
      / *--- Exit for non-cancellable thread ---*/
      if timeexit = 60 then do ; say '' ; say ' >>>>>>>>
      say ' >>>>>>>> 15 minutes have passed since last CANCEL THREAD command.'
      say ' >>>>>>>> There are connections still active. Exit from DB2
      say ' >>>>>>>> CANCEL THREAD procedure. Verify active thread with
      say ' >>>>>>>> command '$ctsubs'DISPLAY THREAD(*)
      say ' >>>>>>>>'' ; say '' ; say ''
      $exitc = 99 ; return $exitc ; exit ; end
      timeexit = timeexit + 1  ; end
      say '' ;say ' >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><
      say '' ;say ' >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
      say ' >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
      say '' ;say '' ; call free ; exit
      / *--- Display Thread routine ---*/
Dthread :
   jobw = systsinp ; "alloc da("outds1") f("jobw") shr reuse"
   sk.1='DSN SYSTEM('subsys')
   sk.2='DIS THREAD(*)

© 2001. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.
sk.3='END
sk.Ø=3 ; call Writerec
tipcom = 'DISPLAY Thread' ; Call Rundb2 ; jobw = systspri
"alloc da('"outds2'"') f("jobw") shr reuse"
"execio * diskw systspri (stem trpcmd00. finis"
#trpcmd00 = trpcmd00.Ø ; clearrec = YES
/*--- Macro output display ---*/
xx=OUTTRAP(trp04.) ; "ispexec edit dataset('"outds2'"
macro(@mdb2036)"
xx=OUTTRAP(OFF)
if rc > Ø then do ; do #a = 1 to trp04.Ø
   say trp04.#a ; end ; $exitc = 99 ; return $exitc ; exit ; end
/*--- Test output macro ---*/
xx=outtrap(trp05.)
"execio * diskr systspri (stem systspri. finis" ; "free
fi(systspri)"
xx=outtrap(off)
if rc > Ø then do
   do #a = 1 to trp05.Ø ; say trp05.#a ; end
   say ''; say ''; say '>>>>>>>>>>'
   say '>>>>>>>>>> Error reading file '"outds2'"
   say '>>>>>>>>>> RC='rc'. Verify. ' ;
   say '>>>>>>>>>>'; say ''; say ''
   $exitc = 99 ; return $exitc ; exit ; end ; return
/*--- DB2 command routine ---*/
Rundb2 :
wrk = center(tipcom,14)
say ''; say ' **********************************************************' ;
say '*' ; say ' **' ; say 'time()' '*'
say '*' ; say ' wrk ' command is running '*'
say '*' ; say '*'
say '*' ; say ' Please Stand By.......
say '*' ;
say ' **********************************************************' ;
say '' ;
/*--- Clean trace record ---*/
if clearrec = YES then do
   DO #g = 1 to #trpcmd00 ; trpcmd00.#g = blk ; end ;
   xx=outtrap(trpcmd00.) ; address tso "ex 'outds1'"
;xx=outtrap(off); #e = Ø
   do #e = 1 to trpcmd00.Ø ; say trpcmd00.#e ; end ; say ''; return
/*--- Write routine output records ---*/
Writerec :
"EXECIO * DISKW "jobw" (STEM sk. FINIS"
Clean:
   DO #f = 1 to sk.Ø ; sk.#f = blk ; end ; return
/*--- Free dataset ---*/
Free:
   xx=outtrap(trpdummy.)
   "free fi(fiout)" ; "free fi(sysin)"
address tso "delete "outds1"" ; address tso "delete "outds2""
xx=outtrap(off) ; return

@DB2ALL0 REXX EXEC
/* REXX */
trace ?o
/*-----------*    FILE ALLOCATION ROUTINE    *----------*/
arg parmin
nparmin =words(parmin) ; subsys =word(parmin,1); vdsfile =word(parmin,2)
vdir =word(parmin,3); vspace =word(parmin,4); vrecfm =word(parmin,5)
vlrecl =word(parmin,6); vblksize =word(parmin,7); vddfile =word(parmin,8)
vokdisp1 =word(parmin,9)
/*---  Test input parameter ---*/
if nparmin < 9 then do ; say ''; say ''; say '<<<<<<<<<<< Parameter list is incomplete !!!!! ' parmin
say '<<<<<<<<<<<'; say ''; say ''; exit ; end
/***  Parameters assignment ---*/
Call @db2parØ subsys ; if word(result,1) = 99 then exit
$1par =word(result,1); $accn =word(result,2); $class =word(result,3)
$msgcla =word(result,4); $region =word(result,5); $msglvl =word(result,6)
$notif =word(result,7); $user =word(result,8); $unitda =word(result,9)
$unitta =word(result,10); $esunit =word(result,11); $prt =word(result,12)
$hiwork =word(result,13); $db2ver =word(result,14); $ctsubs =word(result,15)
$librexx =word(result,16); $parmlib =word(result,17); $proclib =word(result,18)
$jcllib =word(result,19); $report =word(result,20); $isptenu =word(result,21)
$isppenu =word(result,22); $ispmenu =word(result,23); $ispslib =word(result,24)
$plilink =word(result,25); $sibmlink =word(result,26); $sortlib =word(result,27)
$runlib =word(result,28); $dsnload =word(result,29); $tep2pgm =word(result,30)
$tep2pln =word(result,31); $unlopgm =word(result,32); $unlopln =word(result,33)
$dsnproc =word(result,34)
/*---  File allocation ---*/
xx=outtrap(trpallocØ.) ; address tso "delete "vdsfile"
"alloc da("vdsfile") dir("vdir") space("vspace") dsorg(ps)","recfm("vrecfm") lrecl("vlrecl") blksize("vblksize") tracks ",

42 © 2001. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.
"unit("$esunit") new catalog f("vddfile")" ; xx=outtrap(off)
if rc > Ø then do
    do #a = 1 to trpallocØ.Ø ; say trpallocØ.#a ; end
    say '' ; say '' ; say '';
    say '>>>>>>>>' ; say '' ; say '' ;
    $exitc = 99 ; return $exitc ; exit ; end
else do
    if vokdispl = 'YES' then do
        vdsfilec = left(vdsfile,46) ; say '>>>>>>>> 'vdsfilec'
        Allocation OK
    end ; end ; return

@DB2SDSF REXX EXEC
/* REXX */
trace ?o

/*-----------*      DISPLAY JOB STATUS       *----------*/
arg parmin ; parm = translate(parmin,' ',',')
nparm = words(parm) ; subsys = word(parm,1) ; jobcheck =
    word(parm,2)
delaytim = word(parm,3); timeexit = word(parm,4)

/*---  Test input parameter   ---*/
if nparm < 1 then do ; say '' ; say '' ; say '';
    say '>>>>>>>> Parameter list is incomplete !!!!! ' parmin
    say '>>>>>>>>' ; say '' ; say '' ; $exitc = 99 ; return $exitc ;
    exit ; end

/*---  Parameters assignment   ---*/
call @db2parØ subsys
if word(result,1) = 99 then do ; $exitc = 99 ; return $exitc ; exit ;
end
$lpar    =word(result,1);  $accn    =word(result,2);  $class =
    word(result,3)
$msgccia =word(result,4); $region =word(result,5); $msglvl =
    word(result,6)
$notif   =word(result,7); $user   =word(result,8); $unitda =
    word(result,9)
$unitta  =word(result,10); $esunit =word(result,11); $prt =
    word(result,12)
$hiwork  =word(result,13); $db2ver =word(result,14); $ctsubs =
    word(result,15)
$librexx =word(result,16); $parmlib =word(result,17); $proclib =
    word(result,18)
$jcclib  =word(result,19); $report =word(result,20); $isptenu =
    word(result,21)
$isppenu =word(result,22); $ispmenu =word(result,23); $ispslib =
    word(result,24)
$plilink =word(result,25); $sibmlnk =word(result,26); $sortlib =
    word(result,27)
$runlib = word(result, 28); $dsnload = word(result, 29); $step2pgm = word(result, 30); $step2pln = word(result, 31); $unlopgm = word(result, 32); $unlopln = word(result, 33); $dsnproc = word(result, 34)

/ *--- Work areas initialization ---*/
blk = ; wtimeexit = 0
/ *--- isfout file allocation ---*/
outds1 = $hiwork'.subsys'.@DB2SDSF.ISFOUT'
dsn = sysdsn('''outds1''')
if dsn = OK then do
  prmalloc = subsys' 'outds1' 0 60,30 f,b,a 133 0 isfout yes'
call @db2allØ prmalloc
  if word(result, 1) = 99 then do;$exitc = 99;return $exitc ; exit ; end ; end
else do
  jobw = isfout ; "alloc da(''outds1'') f(''jobw'') shr reuse'' ; end
/ *--- isfin file allocation ---*/
outds2 = $hiwork'.subsys'.@DB2SDSF.ISFIN'
dsn = sysdsn('''outds2''')
if dsn = OK then do
  prmalloc = subsys' 'outds2' 0 1,1 f,b 80 27920 isfin yes'
call @db2allØ prmalloc ; if word(result, 1) = 99 then do
    $exitc = 99 ; return $exitc ; exit ; end ; end
else do
  jobw = isfin ; "alloc da(''outds2'') f(''jobw'') shr reuse'' ; end
  jobw = isfin
sk.1 = 'ST 'jobcheck
sk.Ø = 1 ; Call writerec ; Call versdsf
do while isfout.Ø = 0
  wtimeexit = wtimeexit + 1
  say ''
  say '------> 'time()' Nr. 'right(isfout.Ø,3,'Ø') job 'jobcheck' are running'
  address tso "delayØØ "delaytim"" ; call versdsf
if wtimeexit = timeexit then do
  $jobnum = isfout.Ø ; $exitc=98 ; return $exitc $jobnum ; exit ; end ; end
call Free ; $exitc = 00 ; return $exitc ; exit
/ *--- Write routine output records ---*/
Writerec :
"EXECIO * DISKW "jobw" (STEM sk. FINIS"
Clean:
  DO #f = 1 to sk.Ø ; sk.#f = blk ; end ; return
/ *--- Routin SDSF ---*/
Versdsf :
xx=outtrap(trpØØ.) ; "ispexec select pgm(SDSF) "
if rc > 0 then do
  say '' ; say ''>>>' ; say ''>>> SDSF Call Failed RC = 'rc
  say '>>>>>> Verify output. Exit from elaboration||||||'
say '>>>>>>'; say ''; address tso "printoff ('"outds1"') class(X)"
"free fi(isfin)"; "free fi(isfout)"
$exitc = 99; return $exitc; exit; end; xx=outtrap(off)
/**--- Macro to process SDSF output ---*/
xx=OUTTRAP(trp01.); "ispexec edit dataset ('"outds1"') macro(@MDB2039)"
xx=OUTTRAP(OF)
if rc > 0 then do
  do #a = 1 to trp01.Ø
    say trp01.#a; end
$exitc = 99; return $exitc; exit; end
xx=OUTTRAP(OFF)
if rc > 0 then do
  do #a = 1 to trp02.Ø
    say trp02.#a; end
  say ''; say ''; say '>>>>>>>>>> 'outds1' Allocation KO '
  say '>>>>>>>>>> RC='rc'. Verify. '
  say ''; say ''; $exitc = 99; return $exitc; exit; end; return
/**--- Free dataset ---*/
Free:
xx=outtrap(trpdummy.)
"free fi(isfin)"; "free fi(isfout)"
xx=outtrap(off); return

@DB2PAR0 REXX EXEC
/* REXX */
trace ?o
/*----------- DB2 INITIALIZATION VARIABLES *-----------*/
/*/ DB2 version & subsystem id assignment ---*/
arg subsys
select
  when subsys = DSNS then do ; db2ver = 51 ; $ctsubs = "-" ; end
  when subsys = DSNZ then do ; db2ver = 51 ; $ctsubs = "+" ; end
  otherwise
say '>>>>>>>>>
say '>>>>>>>>> ---------- ATTENTION PLEASE <<<<----- '
say '>>>>>>>>>
say '>>>>>>>>> Wrong DB2 subsystem '
say '>>>>>>>>> >>>>>> 'subsys' <<<<<
say ''; say ''; $exitc = 99; return $exitc; exit; end
/*/ job class assignment ---*/
call valsysid
select
  when result = SYSS then $class = P ; when result = SYSZ then $class = S
 otherwise
say '>>>>>>>>>
say '>>>>>>>>> ---------- ATTENTION PLEASE <<<<----- '
say '>>>>>>>>>
say '>>>>>>>>> Unknown MVS subsystem '
say '>>>>>>>>> >>>>>> 'result' <<<<<
say ''; say ''; $exitc = 99; return $exitc; exit; end
address ispexec 'vget (zacctnum) shared'

/*--- General variables ---*/
$lpar = result /* LPAR */
$accn = zacctnum /* Account name */
$class = $class /* Class */
$msgcla = 'X' /* Message class */
$region = '4M' /* Region */
$msglvl = '1,1' /* Message level */
$notif = userid() /* Notify */
$user = userid() /* User */
$unitda = '3390' /* Type of unit dasd */
$unitta = 'ROBOT' /* Type of unit tape */
$esunit = 'WORKA' /* Esoteric work name */
$prt = 'NØZ5' /* Printer name */
$hiwork = 'DB2WORK' /* Hi-level work areas */
$db2ver = db2ver /* Versione DB2 */
$ctsubs = $ctsubs /* Carattere subsystem */

/*--- DB2 V5.1 variables "OS/390" LE ---*/
select
  when $db2ver = '51' then do
    $librexx = 'user.library' /* Rexx library */
    $parmlib = 'user.library' /* Parmlib */
    $proclib = 'DSN510.PROCLIB' /* Proclib */
    $jcllib = 'user.library' /* Jcl */
    $report = 'DSN510.REPORT.LOGDB2' /* Report */
    $isptenu = 'ISP.SISPTENU' /* ISPF library */
    $isppenu = 'ISP.SISPPENU' /* ISPF library */
    $ispmenu = 'ISP.SISPMENU' /* ISPF library */
    $ispslib = 'ISP.SISPSLIB' /* ISPF library */
    $plilink = 'CEE.SCEELKED' /* PLI */
    $sibmlnk = 'CEE.SCEERUN' /* PLI */
    $sortlib = 'SYS1.SYNCSORT.LVL36F.LINKLIB' /* Sort */
    $runlib = 'DSN510.RUNLIB.LOAD' /* Runlib */
    $dsnload = 'SYS1.DSN510.SDSNLOAD' /* DB2 system library */
    $step2pgm = 'DSNTEP2' /* DSNTEP2 (program) */
    $step2pln = 'DSNTEP51' /* DSNTEP51 (plan) */
    $unlopgm = 'DSNTIAØ1' /* DSNTEP51 (plan) */
    $unlopln = 'DSNTIBØ1' /* DSNTEP51 (plan) */
    $dsnproc = 'DSNUPROD' /* Proclib */
  end
otherwise
  say ''; say ''; say '>>>>>>>>';
  say '>>>>>>>> Wrong DB2 version. Stop elaboration !!!!!';
  say '>>>>>>>>'; say ''; say '
  $exitc = 99; return $exitc; exit; end

return $lpar $accn $class $msgcla $region $msglvl $notif ,
$user $unitda $unitta $esunit $prt $hiwork ,
$db2ver $ctsubs $librexx $parmlib $proclib $jcllib $report.
$isptenu $isppenu $ispmenu $ispslib.
$pilink $sipmlnk $sortlib $runlib $dsnload.
$tep2pgm $tep2pln $unlopgm $unlopln $dsnproc

valsysid:
numeric digits 1Ø

   cvt = addc(16,'ØØ') ; system = addc(cvt,Ø154)
   return storage(d2x(ad1+x2d(ad2)),4)
   addc:arg ad1,ad2
   return c2d(storage(d2x(ad1+x2d(ad2)),4))

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

Jaiwant K Jonathan
DB2 DBA
QSS (USA) © Xephon 2001

Need help with a DB2 problem or project?

Maybe we can help:

• If it’s on a topic of interest to other subscribers, we’ll commission an article on the subject, which we’ll publish in DB2 Update, and which we’ll pay for – it won’t cost you anything.

• If it’s a more specialized, or more complex, problem, you can advertise your requirements (including one-off projects, freelance contracts, permanent jobs, etc) to the hundreds of DB2 professionals who visit DB2 Update’s home page every week. This service is also free of charge.

Visit the DB2 Update Web site, http://www.xephon.com/db2update.html, and follow the link to Suggest a topic or Opportunities for DB2 specialists.
Metagon Technologies has introduced its DQbroker Extended Enterprise, described as a secure real-time inter-corporate data access solution, providing a scalable peer-to-peer distributed processing model, which enables the company’s DQbroker to deliver secure data integration to corporate trading partners via the Web.

Specifically it accesses data across IBM, Unix, NT, and other servers and workstations and can access data sources including DB2, FoxPro, Informix, Neon, ODBC, Oracle, Progress, Sequential, SQL Server, Sybase, and VSAM.

It can also provide real-time access to PC-based data, such as Microsoft Excel spreadsheets. It supports relational, non-relational data, and XML data sources.

For further information contact:
Metagon Technologies, PO Box 2810, Matthews, NC 28106-2810, USA.
Tel: (704) 847 2390.

---

IBM has announced the beta of Version 7.1 of its Essbase-based DB2 OLAP Server for OS/390, promising improved functions, performance, and scalability.

There’s improved performance of load/calc, when used with the multidimensional storage option now based on VSAM, allowing larger cubes to be built within the same batch window.

There’s a choice of storage, including multi-dimensional file or relational database, based on individual applications’ needs.

The DB2 OLAP Integration Server now runs on System/390 natively. It helps map DB2 data into OLAP structures more easily and supports drill-through into DB2.

Other features include support of attributes such as colours and sizes, improved scalability via large outlines and data exports, optimized I/O operations, claimed better calculation functions, and new features in spreadsheet add-ins, such as a new Query Designer and support of attributes to improve end-user productivity.

For further information contact your local IBM representative.

---

Neon Systems has announced certification of its iWave J2EE-compliant JDBC Driver for use with BEA’s WebLogic Server 6.0.

iWave is an end-to-end integration product providing JDBC access to a growing list of information systems running on OS/390.

Besides support for DB2 for OS/390 data sources, it covers non-relational data residing in IMS/DB, VSAM, and ADABAS data sources and applications within CICS/TS and IMS/TM transactions.

Support for the J2EE platform with JDBC includes JNDI, connection pooling, and distributed transactions via two-phase commit.

For further information contact:
Neon Systems, 14100 Southwest Fwy, Suite 500, Sugar Land, TX 77478, USA.
Tel: (281) 491 4200.