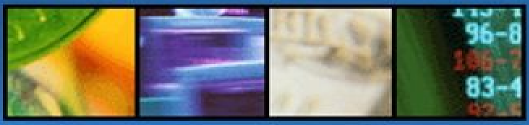

1 DSNTEP2 and DSNTEP4 – in-depth analysis!

This description of IBM's DB2 sample programs DSNTEP2 and DSNTEP4 bases on DB2 for z/OS Version 10. These programs, DB2 and z/OS are property of the IBM Corporation. Information can be found in the appropriate program product documentation.
May 2011

Table of Contents:

1.1 INVOCATION 1
1.2 PARAMETERS 2
1.3 VALID INPUT 3
1.4 RESTRICTIONS 4
1.5 SUPPORTED SQL DATA TYPE 5
1.6 COMMIT AND ROLLBACK PROCESSING 5
1.7 RETURN CODE AND ERROR HANDLING 5
1.8 USE OF DSNTEP2 AND DNSTEP4 8
1.9 PROGRAM INTERNAL FORMATTING OPTIONS - LIMITS 8
1.10 OUTPUT 8
1.11 IMPROVING PERFORMANCE AND FUNCTIONALITY 9
1.12 VARIATIONS OF IKJEFT01 11

(C) www.ruban.de, Boeblingen, Germany

DSNTEP2 is a sample dynamic SQL program that is written in the PL/I language. With this program, you can execute any SQL statement that can be executed dynamically. You can use the source version of DSNTEP2 and modify it to meet your needs, or, if you do not have a PL/I compiler at your installation, you can use the object code version of DSNTEP2.

DSNTEP4, which appeared first with DB2 Version 8, is a sample dynamic SQL program that is written in the PL/I language. This program is identical to DSNTEP2 except DSNTEP4 uses multi-row fetch for increased performance. If you use DSNTEP2, then changing to DSNTEP4 can save a lot of cpu time if you fetch large numbers of rows. You can use the source version of DSNTEP4 and modify it to meet your needs, or, if you do not have a PL/I compiler at your installation, you can use the object code version of DSNTEP4.

For program preparation see installation jobs DSNTEJ1P (source) and DSNTEJ1L (object). You can find the PL/I source code in the appropriate *dsnlvl*.SDSNSAMP library members DSNTEP2 and DSNTEP4. The DBRM's will be created by the precompiler, but have also been provided as members DSN§EP2L and DSN§EP4L of the SDSNSAMP library.

The DSNTEP2 and DSNTEP4 do not work in compatibility mode (CM) due to new functions (NEWFUN) used inside the code. You cannot precompile and cannot bind the modules in CM mode. If you need to deploy these programms in CM mode you have to work with code of the previous version instead.

1.1 Invocation

```
//...      JOB ...
//DSN      EXEC PGM=IKJEFT01[1]
//STEPLIB  DD   DSN=dsnlvl.RUNLIB.LOAD,DISP=SHR
//SYSTSPRT DD   SYSOUT=*
//SYSTSIN  DD   *
 DSN SYSTEM(ssid)
  RUN PROGRAM(program) -
       [PLAN(planname)] -
       [PARM('/parms')] -
      [LIB('dsnlvl.RUNLIB.LOAD']
 END
//SYSIN    DD   *
-- your SQL Statements;
/*
//SYSPRINT DD {SYSOUT=*|DSN=...,DCB=(LRECL=133),...}
```

SSID:	DB2 subsystem identifier
Program:	DSNTEP2 and DSNTEP4
Planname:	no specification, or DSNTEP2 and DSNTEP4
Parms:	up to 3 optional parameters, see next section
Lib:	Specify `LIB` parameter or //`STEPLIB` JCL statement. Only necessary, if not specified within z/OS LINKLST or JOBLIB JCL statement. Programs may reside in *dsnlvl*.RUNLIB.LOAD, or *dsnlvl*.SDSNLOAD, or even other libraries. This depends on individual installation.

[1] IKJEFT*xx*: Possible variations, see section „Variations of IKJEFT01“on page 11.

1.2 Parameters

Parameters are optional. You may specify up to 3 parameters. The parameter string cannot have more than 60 bytes.

Parameter	**Meaning**
ALIGN(MID)	Set alignment to center
ALIGN(LHS)	Set alignment to left
NOMIXED	Assume all SBCS characters
MIXED	Recognize mixed characters
SQLFORMAT(SQL)	Specifies how DSNTEP2 or DSNTEP4 pre-processes SQL statements: This is the preferred mode for SQL statements other than SQL procedural language. When you use this option, which is the default, DSNTEP2 or DSNTEP4 collapses each line of an SQL statement into a single line before passing the statement to DB2. DSNTEP2 or DSNTEP4 also discards all SQL comments.
SQLFORMAT(SQLCOMNT)	Specifies how DSNTEP2 or DSNTEP4 pre-processes SQL statements: This mode is suitable for all SQL, but it is intended primarily for SQL procedural language processing. When this option is in effect, behavior is similar to SQL mode, except that DSNTEP2 or DSNTEP4 does not discard SQL comments. Instead, it automatically terminates each SQL comment with a line feed character (hex 25), unless the comment is already terminated by one or more line formatting characters. Use this option to process SQL procedural language with minimal modification by DSNTEP2 or DSNTEP4.
SQLFORMAT(SQLPL)	Specifies how DSNTEP2 or DSNTEP4 pre-processes SQL statements: This mode is suitable for all SQL, but it is intended primarily for SQL procedural language processing. When this option is in effect, DSNTEP2 or DSNTEP4 retains SQL comments and terminates each line of an SQL statement with a line feed character (hex 25) before passing the statement to DB2. Lines that end with a split token are not terminated with a line feed character. Use this mode to obtain improved diagnostics and debugging of SQL procedural language.
SQLTERM(*termintor-char*)	Specify this parameter to indicate the character that you use to end each SQL statement. You can use any special character except one of those listed in the following table. SQLTERM(;) is the default. blank X'40' comma , X'6B' double quotation mark " X'7F' left parenthesis (X'4D' right parenthesis) X'5D' single quotation mark ' X'7D'

(C) www.ruban.de, Boeblingen, Germany

Parameter	Meaning
	underscore _ X'6D'
TOLWARN(YES)	Allow SQL warnings. If a warning occurs when DSNTEP2 or DSNTEP4 executes an OPEN or FETCH for a SELECT statement, it continues to process the SELECT statement.
TOLWARN(NO)	Do not allow SQL warnings (default). If a warning occurs when DSNTEP2 or DSNTEP4 executes an OPEN or FETCH for a SELECT statement, DSNTEP2 or DSNTEP4 stops processing the SELECT statement. If SQLCODE +445 or SQLCODE +595 occurs when DSNTEP2 or DSNTEP4 executes a FETCH for a SELECT statement, DSNTEP2 or DSNTEP4 continues to process the SELECT statement. If SQLCODE +802 occurs when DSNTEP2 or DSNTEP4 executes a FETCH for a SELECT statement, DSNTEP2 or DSNTEP4 continues to process the SELECT statement if the TOLARTHWRN control statement is set to YES.

1.3 Valid Input

An input line consists of characters from columns 1-72. If an input statement spans over multiple lines, the lines are concatenated and blanks are removed such that only one blank occurs between words.

Input SQL statements will be transferred to the statement buffer with one blank between words. Blanks in delimited strings will be transferred into the statement buffer exactly as they appear in the input statement.

Longest SQL statement may have 2 meagabytes. Multiple blanks will be removed before further internal procesing. You may raise this limit changing an internal constant. (See section „Program Internal Formatting options - Limits“ below.)

<u>Following input will be accepted by DSNTEP2:</u>

1. All valid dynamic SQL commands
2. These static SQL commands:
   CONNECT
   SET CONNECTION
   SET QUERYNO
   RELEASE
3. These commands, which are used to terminate processing:
   END
   EXIT
   QUIT
4. These commands, which provide brief HELP text:
   HELP
   H
   ?
5. EXEC SQL, which is permitted as a prefix to test statements that will be put in programs.
6. Standard comments, which are of these forms:
   - All text in any line that starts with an asterisk (*)
     - Not valid within an SQL statement
   - All text that follows a double hyphen (--), to the end of the line
     - Can start in any column
     - Is valid within an SQL statement
7. Functional comments, which are of these forms:

```
--#SET ROWS_FETCH n
```

(C) www.ruban.de, Boeblingen, Germany

where 'n' is a non-negative integer that indicates the maximum number of rows to be FETCHed for each subsequent SELECT statement. Use -1 to indicate that all rows should be fetched.

```
--#SET ROWS_OUT n
```

where 'n' is a non-negative integer that indicates the maximum number of rows to be outputted for each subsequent SELECT statement. Use -1 to indicate that all rows should be outputted.

```
--#SET TOLWARN    x
```

where 'x' is a YES/NO, allow SQL warnings or not

```
--#SET TERMINATOR x
```

where 'x' is a one-byte character to be used to terminate the next SQL statement. Any character is valid except a blank, comma, single or double quote, underscore, or parenthesis.

```
--#SET TOLARTHWRN x
```

where 'x' is a YES/NO, allow Arithmetic warning

```
--#SET PREPWARN x
```

Indicates how DSNTEP2 and DSNTEP4 is to handle a PREPARE SQLWARNING message, where 'x' is a YES/NO.

```
--#SET SQLFORMAT x
```

Specifies how DSNTEP2 or DSNTEP4 pre-processes SQL statements before passing them to DB2. Select one of the following options:, where 'x' is a SQL/SQLCOMNT/SQLPL.

```
--#SET MAXERRORS n
```

where n is the number of errors allowed before the program should continue. Use -1 to indicate that the program should tolerate an unlimited number of errors. Severe SQL errors will cause cause program termination whenever encountered.

```
--#SET MULT_FETCH n
```

(DSNTEP4 only!) where 'n' is a non-negative integer between 1 to 32767 that indicates the amount of rows to be fetched for each subsequent FETCH statement. Default is 1.

(C) www.ruban.de, Boeblingen, Germany

1.4 Restrictions

- This module uses the semicolon as the default terminator for SQL statements. The user may specify an alternate termination character by either of these means:
  - Use the SQLTERM parameter when invoking DSNTEP2 SQLTERM parameter.
  - Use the following "functional comment" in the SQL statement stream. Note that this allows you to change the SQL terminator "on the fly":

    ```
    --#SET TERMINATOR x
    ```

    where 'x' is the terminator character.

    For either SQLTERM or TERMINATOR, any character is valid except a blank, comma, single or double quote, underscore, or parenthesis.
- Comments prefixed by two hyphens (--) are allowed within SQL statements.
- The maximum length of SQL statements that this program can handle is STMTMAX, due to its allocation of space (STMTBUF). The first INPUTL characters of each input record are inserted into this buffer.
- For statements other than SELECT, only error information or 'SUCCESSFUL" is put out, with the number of rows updated, inserted, or deleted. For a SELECT statement, the output is more detailed.

- DSNTEP2 and DSNTEP4 write results to the data set that is defined by the SYSPRINT DD statement. SYSPRINT data must have a logical record length of 133 bytes (LRECL=133). Otherwise, the program issues return code 12 with abend U4038 and reason code 1.

1.5 Supported SQL Data Type

CHAR	VARCHAR	LONG VARCHAR
DECIMAL	SMALLINT	INTEGER
DATE	TIME	TIMESTAMP
FLOAT	FLOAT(n)	DOUBLE PRECISION
REAL	BLOB	CLOB
DBCLOB	ROWID	

1.6 Commit and Rollback Processing

Applications that use the TSO attachment facility can explicitly define units of work by using the SQL COMMIT and ROLLBACK statements. DSNTEP2 and DSNTEP4 itself do not issue COMMIT or ROLLBACK SQL statements if they are not provided as input coded in the SYSIN data set.

In TSO applications, a unit of work starts when the first updates of a DB2 object occur. A unit of work ends when one of the following conditions occurs:

- The program issues a subsequent COMMIT statement. At this point in the processing, your program has determined that the data is consistent; all data changes that were made since the previous commit point were made correctly.
- The program issues a subsequent ROLLBACK statement. At this point in the processing, your program has determined that the data changes were not made correctly and, therefore, should not be permanent. A ROLLBACK statement causes any data changes that were made since the last commit point to be backed out.
- The program terminates and returns to the DSN command processor, which returns to the TSO Terminal Monitor Program (TMP).

The first and third conditions in the preceding list are called a commit point. A commit point occurs when you issue a COMMIT statement or your program terminates normally. If the application program is unable to reach a point of consistency, or if the application terminates after changing database information but before declaring a point of consistency, DB2 aborts all of the uncommitted changes.

For error situations DSNTEP2 and DSNTEP4 can handle, a PL/I built-in function `PLIRETC(`*`return code`*`)` is called which sets the return code for JES JCL, but do not abnormally abend the program. Then the program normally ends and passes control to the DSN commands processor.

1.7 Return Code and Error Handling

In general both of the sample programs will end with following return codes:

1.7.1.1 Re-turn cod e	Meanin
0	Successful completion
4	An SQL statement received a warning code.
8	An SQL statement received an error code.
12	The length of an SQL statement was more than n bytes (see limits), an SQL statement returned a severe error code (-8nn or -9nn), or an error occurred in the SQL message formatting routine

(C) www.ruban.de, Boeblingen, Germany

- **Normal Exit (RC=0):**
  No errors were found in the source and no errors occurred during processing.
  If the only non-zero SQL codes generated are +100s, the return code from DSNTEP2 or DSNTEP4 will be 0.

  1. The following message will be generated when a HELP request is made:
     ```
     ALL DYNAMIC SQL COMMANDS ARE SUPPORTED.
     THE FOLLOWING STATIC SQL COMMANDS ARE ALSO
     SUPPORTED:
     CONNECT  SET CONNECTION  SET QUERYNO  RELEASE
     ```
  2. The following message will be generated for all input statements:
     ```
     ***INPUT STATEMENT:   SQL input statement
     ```
  3. The following message will be generated when a non select SQL statement is entered:
     ```
     RESULT OF SQL STATEMENT:
     ```
     all SQL messages printed ---
  4. The following messages will be generated when a zero SQLCODE is returned:
     ```
     SUCCESSFUL command OF nnn OBJECT(S)
     ```
     The command is ALTER, CREATE, or DROP.
     ```
     command SUCCESSFUL
     ```
     The command is COMMIT,GRANT,LOCK,REVOKE, ROLLBACK, or SET
     ```
     SUCCESSFUL command OF nnn ROW(S)
     ```
     The command is SELECT, DELETE, INSERT, or UPDATE
  5. The following message will be generated when a 'not found' condition was encountered as a result of an open cursor:
     ```
     "NOT FOUND" CONDITION ENCOUNTERED DURING OPEN
     ```
  6. The following message will be generated when an unrecognizable SQLTYPE is encountered:
     ```
     INVALID SQLTYPE mmm ENCOUNTERED FOR FIELD # nnn
     ```
     mmm is the SQLTYPE code
     nnn is the field number on the SELECT that was unrecognized.

- **Error Exit (RC=4, RC=8 or RC=12):**
  Errors were found in the source, or occurred during processing. DSNTEP2 or DSNTEP4 does not produce an ABEND code.

  If one of these errors are encountered, DSNTEP2 or DSNTEP4 stops processing immediatelly with error exit (RC=8):

  - -804: BAD PARMLIST OR SQLDA
  - -805: NO PLAN FOR APPLICATION PGM
  - -902: SYSTEM ERROR
  - -904: UNAVAILABLE RESOURCE
  - -906: PRIOR SQL ERROR DISABLED FURTHER EXECUTION
  - -911: ROLLBACK DUE TO DEADLOCK
  - -913: FAILURE DUE TO DEADLOCK
  - -922: CONNECTION NOT AUTHORIZED
  - -923: CONNECTION NOT ESTABLISHED
  - -924: DB2 CONNECTION INTERNAL ERR
  - -927: LANGUAGE INTERFACE WAS CALLED WHEN CONNECTING ENVT WAS NOT ESTABLISHED

(C) www.ruban.de, Boeblingen, Germany

(C) www.ruban.de, Boeblingen, Germany

Other negative SQL Codes will be ignored up the maximum number of errors (MAXERRORS, see section „Program Internal Formatting options - Limits“ below), which is 10 by default. One more statement with a negative SQL code will be treated as error (RC=8) and will lead to an error exit.

Warnings, possibly returned by FETCH operations (see SQLCA.SQLWARN0) are gathered. The maximum warning code (at least RC=4) will be returned at the end of DSNTEP2, if no error occurs.

If the only non-zero SQL codes generated are +100‘s, the return code from DSNTEP2 or DSNTEP4 will be 0.

- RETURN CODE = 4 - warning-level errors detected.
  SQLWARNING found during execution.
  REASON CODE = none
- RETURN CODE = 8 - errors detected.
  SQLERROR   found during execution.
  REASON CODE = none
- RETURN CODE = 12 - severe errors detected.
  Unable to open files.
  Internal error, error message routine return code.
  Statement is too long.
  SQL buffer overflow.
  REASON CODE = none
  Invalid functional comment (--#SET) encountered

1. The following message will be generated when a SQL error is found:
   ```
   SQLERROR ON command  COMMAND SQL_function FUNCTION
   ```
2. The following message will be generated when a SQL warning is found:
   ```
   SQLWARNING ON command COMMAND SQL_function FUNCTION
   ```
3. The following message will be generated when an input SQL statement is greater than MAXARRAY size:
   ```
   **ERROR:  SQL STATEMENT GREATER THAN nnn CHARACTERS.
   STMT:
   SQL_statement.
   ```
   nnn is MAXARRAY which is the maximum array defined in the program.
   SQL_statement is the current SQL statement being processed.
4. The following message will be generated when the size of the SQL statement (SQLDA + FIELD BUFFERS) is greater than MAXARRAY:
   ```
   **ERROR:  SQL BUFFER OVERFLOW.  MAXIMUM SIZE IS nn
   ```
   nn is MAXARRAY. A return code of 12 is set.
5. The following message will be generated when a non select SQL statement is entered:
   ```
   RESULT OF SQL STATEMENT:
   RETURN CODE nnn FROM MESSAGE ROUTINE DSNTIAR.
   any SQL messages printed ---
   ```
6. The following message will be generated when an invalid functional comment (--#SET) is entered
   ```
   DSNTEP2 halted due to a functional comment (--#SET) error:
   Invalid value, <token> specified for <command>
   ```

1.8 Use of DSNTEP2 and DNSTEP4

DSNTEPx is a very useful sample IBM DB2 program when used to execute simple SQL statements. For complex requirements, which are intrinsically interdependent, these programs are not suitable for the following reasons:

- DSNTEPx never initiates a COMMIT or ROLLBACK by itself – those have to be supplied as SYSIN input. From perspective of the TSO Call Attachment Facility DSNTEPx always successfully ends and therfore issues an implicit COMMIT, regardless of the return code.
- Certain negative SQLCODE's lead to immediate end of execution. But on the other hand DSNTEPx will tolerate a certain number of minor errors which depends on settings.

Conclusion: DSNTEPx should be used very carefully and with knowledge of its peculiarities. In situation with complex SQL logic a program (Cobol, PL / I) or a script (REXX, SQL stored procedure) should be used instead. Incorrect use can lead to inconsistent data!

1.9 Program Internal Formatting options - Limits

Variable Name	Value	Meaning
PAGEWIDTH	133	Maximum width of a page in characters (including the control character in column one)
MAXROW#LN	6	Maximum number of digits for the row numbers in the output.
MAXPAGWD	125	Print line width controller = maximum width - 1 (for control character) - MAXROW#LN (length of the column display) - 1 (a '-' between the column number display the SQL output display).
MAXCOLWD	120	Maximum number of characters in a character data type column. Truncation occurs when this number is exceeded.
MAXPAGLN	60	Maximum number of lines on the print output pages 2 thrn N. Page 1 will have MAXPAGLN + 1 lines.
PAGESIZE	4096	Size of a page. All storage allocation of the SQL buffer area will be a multiple of this value.
MAXNCOLS	100	Initial maximum number of columns in an answer, times 2 in case a double SQLDA is required for LOBs and/or UDTs. An initial setting of 100 will handle an SQL statement of at least 50 columns. If an SQL statement described into a single SQLDA has more than 100 columns -or- an SQLDA described into a double SQLDA has more than 50 columns, a larger SQLDA area will be allocated.
MAXBUFFER STMTMAX	2097152	Maximum length of any large array size (OUTBUF, BUFFSQL, COLSTART, COLLN and STMTBUF) (increased in V8 from 32760 to 2M).
RECLEN	72	Length of the input record
MAXERRORS	10	Number of errors allowed before the program is terminated. Severe SQL errors will cause program termination whenever encountered.

1.10 Output

DSNTEP2 and DSNTEP4 write informational and error messages to the data set allocated to SYSPRINT. An output record has (as supplied default) not more than 133 bytes. When you allocate a new data set with the SYSPRINT DD statement, either specify a DCB with LRECL=133, or do not specify the DCB parameter.

(C) www.ruban.de, Boeblingen, Germany

The answer for a SELECT may be visualized by a table and my may split into several pages if necessary. Those will show the answer, giving the pages from left to right, then top to bottom, as shown in an example below. The numbers and internal boxes represent pages inside the answer.

1	2	3
4	5	6
7	8	...

1.11 Improving performance and functionality

The original DSNTEP2 and DSNTEP4 BIND statements are supplied by IBM in installation job DSNTEJ1L on library SDSNSAMP:

```
BIND PACKAGE (DSNTEP2) MEMBER(DSN§EP2L) +
     CURRENTDATA(NO) ACT(REP) ISO(CS) ENCODING(EBCDIC)
BIND PLAN(DSNTEPÜÜ) PKLIST(DSNTEP2.*) +
     CURRENTDATA(NO) ACT(REP) ISO(CS) ENCODING(EBCDIC) SQLRULES(DB2)
```

and

```
BIND PACKAGE (DSNTEP4) MEMBER(DSN§EP4L) +
     CURRENTDATA(NO) ACT(REP) ISO(CS) ENCODING(EBCDIC)
BIND PLAN(DSNTP4ÜÜ) PKLIST(DSNTEP4.*) +
     CURRENTDATA(NO) ACT(REP) ISO(CS) ENCODING(EBCDIC) SQLRULES(DB2)
```

... and these statements – as an example for DSNTEP2 – yield in ...

```
BIND PACKAGE(DSNTEP2)       MEMBER(DSN§EP2L)+
     LIBRARY('lib-name')+
     OWNER(id)              QUALIFIER(id)      SQLERROR(NOPACKAGE)+
     VALIDATE(RUN)          ISOLATION(CS)        CURRENTDATA(NO)+
     DEGREE(1)              EXPLAIN(NO)          REOPT(NONE)+
     KEEPDYNAMIC(NO)        DBPROTOCOL(DRDA)     IMMEDWRITE(NO)+
     ENCODING(273)          ROUNDING(HALFEVEN)+
     ENABLE(*)+
     ACTION(ADD|REPLACE)   [REPLVER(vvrr.PMnnnnn)]
END
BIND PLAN(DSNTEP2)          OWNER(id)            QUALIFIER(id)+
     NODEFER(PREPARE)       VALIDATE(RUN)        ISOLATION(CS)+
     CACHESIZE(3072)        CURRENTDATA(NO)      DEGREE(1)+
     SQLRULES(DB2)          ACQUIRE(USE)         RELEASE(COMMIT)+
     EXPLAIN(NO)            REOPT(NONE)          KEEPDYNAMIC(NO)+
     IMMEDWRITE(NO)         DBPROTOCOL(DRDA)     ENCODING(273)+
     ROUNDING(HALFEVEN)     DISCONNECT(EXPLICIT)+
     PKLIST(DSNTEP2.*)+
     ENABLE(*)+
     ACTION(ADD|REPLACE)   [RETAIN]
END
```

To improve access path for dynamic SQL statements you may want to apply the **REOPT** BIND option. Read how it works:

> The REOPT(ONCE) (V8) bind option tries to combine the benefits of REOPT(ALWAYS) and dynamic statement caching. The idea of REOPT(ONCE) is to re-optimize the access path only once (using the first set of input variable values) no matter how many times the same statement is executed.

(C) www.ruban.de, Boeblingen, Germany

If you specify the new REOPT(AUTO) (V9) bind option, DB2 automatically determines whether a new access path is required to further optimize the performance of a statement for each execution. REOPT(AUTO) or REOPT(ONCE) only applies to dynamic statements that can be cached. If dynamic statement caching is turned off and DB2 executes a statement that is bound with REOPT(AUTO) or REOPT(ONCE), no reoptimization occurs.

REOPT specifies whether DB2 determines access paths at bind time and again at execution time for statements that contain:

- Input host variables
- Parameter markers
- Special registers

Your input to DSNTEP2, DSNTEP4 or DSNTIAUL usually does not contain input host variable or parameter markers, but less or more often special registers like CURRENT DATE, CURREN TIME and others. And these predicates and statements may tremendously profit from re-optimized acces path!

Some minor drawbacks using this technique:

- DB2‘s DSC (dynamic statement cache) must be active
- a large number of small and short-running SQL may raise re-optimization overhead
- DSNTEP2 ends with return code 4 if UPDATE or DELETE statemen occurs without WHERE condition. The PREPWARN NO option has no effect on that .
- EXPLAIN/PLAN_TABLE possibly shows different access path as chosen after re-optimization a run-time.

(C) www.ruban.de, Boeblingen, Germany

DEFER(PREPARE) and distributed processing:

Specify the bind option DEFER(PREPARE) to improve performance, instead of NODEFER (PREPARE), when binding dynamic or static SQL for DB2 private protocol access and when binding dynamic SQL for DRDA access. DB2 does not prepare the dynamic SQL statement until that statement executes. (The exception to this situation is dynamic SELECT, which combines PREPARE and DESCRIBE, regardless of whether the DEFER(PREPARE) option is in effect.) When a dynamic SQL statement accesses remote data, the PREPARE and EXECUTE statements can be transmitted over the network together and processed at the remote location. Responses to both statements can be sent back to the local subsystem together. This reduces network traffic, which improves the performance of the dynamic SQL statement.

To acccess data on remote servers you first have to BIND the DSNTEP2, DSNTEP4 or DSNTIAUL package at the remote server. Then you must add the package and location to the package list of the plan. But you can also BIND the plan „at all locations“: Just use „*“ to tell DB2 to use the required package at every location connected to.

The BIND of package and plan – as an example for DSNTEP2 – finally looks like ...

```
BIND  PACKAGE(collection)     MEMBER(DSN§EP2L)+
      LIBRARY('lib-name')+
      OWNER(id)               QUALIFIER(id)         SQLERROR(NOPACKAGE)+
      DEFER(PREPARE)          VALIDATE(RUN)         ISOLATION(CS)+
      CURRENTDATA(NO)         DEGREE(1)             EXPLAIN(NO)+
      REOPT(ONCE)             KEEPDYNAMIC(NO)       DBPROTOCOL(DRDA)+
      IMMEDWRITE(NO)          ENCODING(273)         ROUNDING(HALFEVEN)+
      ENABLE(*)+
      ACTION(ADD)
END
BIND  PLAN(MYTEP2)            OWNER(id)             QUALIFIER(id)+
      DEFER(PREPARE)          VALIDATE(RUN)          ISOLATION(CS)+
      CACHESIZE(3072)         CURRENTDATA(NO)        DEGREE(1)+
      SQLRULES(DB2)           ACQUIRE(USE)           RELEASE(COMMIT)+
      EXPLAIN(NO)             REOPT(ONCE)            KEEPDYNAMIC(NO)+
      IMMEDWRITE(NO)          DBPROTOCOL(DRDA)       ENCODING(273)+
      ROUNDING(HALFEVEN)      DISCONNECT(EXPLICIT)+
      PKLIST(*.collection.*)+
      ENABLE(*)+
      ACTION(ADD)
END
```

1.12 Variations of IKJEFT01

The program needed to execute TSO/E commands from the background is a terminal monitor program (TMP), which may be one of the following:

PGM=IKJEFT01
When a command completes with a non-zero return code, the program goes to the next command. When a command abends, the step ends with a condition code of 12 (X'C').

PGM=IKJEFT1A
If a command or program being processed by IKJEFT1A ends with a system abend, IKJEFT1A causes the job step to terminate with a X'04C' system completion code. IKJEFT1A also returns to the caller the completion code from the command or program in register 15.
If a command or program being processed by IKJEFT1A ends with a user abend, IKJEFT1A saves the completion code in register 15 and then terminates.
If a command, program or REXX exec being processed by IKJEFT1A returns a non-zero return code to IKJEFT1A, IKJEFT1A saves this return code in register 15 and then terminates. Non-zero return codes to IKJEFT1A from CLISTs will not affect the contents of register 15 and the TMP will continue processing.
For a non-zero return code or an abend from a command or program that was not given control directly by IKJEFT1A, no return code is saved in register 15, and IKJEFT1A does not terminate.

PGM=IKJEFT1B
If a command or program being processed by IKJEFT1B ends with a system or user abend, IKJEFT1B causes the job step to terminate with a X'04C' system completion code. IKJEFT1B also returns to the caller the completion code from the command or program in register 15.
If a command, program, CLIST, or REXX exec being processed by IKJEFT1B returns a non-zero return code to IKJEFT1B, IKJEFT1B saves this return code in register 15 and then terminates.
For a non-zero return code or abend completion code from a program or command that was not given control by IKJEFT1B, no return code is saved in register 15, and IKJEFT1B does not terminate.

See z/OS TSO/E Customization for information on abend and non-zero return code processing by these alternate programs, as well as conditional disposition processing for data sets.

(C) www.ruban.de, Boeblingen, Germany