

Statements and System Variables

This section contains special considerations concerning Natural DML statements, Natural SQL statements, and Natural system variables when used with DB2.

It mainly consists of information also contained in the Natural documentation set where each Natural statement and variable is described in detail.

This section covers the following topics:

- Natural DML Statements
 - Natural SQL Statements
 - Natural System Variables
 - Error Handling
 - Stored Procedures in Natural
-

Natural DML Statements

This section summarizes particular points you have to consider when using Natural DML statements with DB2. Any Natural statement not mentioned in this section can be used with DB2 without restriction.

- BACKOUT TRANSACTION
- DELETE
- END TRANSACTION
- FIND
- GET
- HISTOGRAM
- READ
- STORE
- UPDATE

BACKOUT TRANSACTION

This statement undoes all database modifications made since the beginning of the last logical transaction. Logical transactions can start either after the beginning of a session or after the last SYNCPOINT, END TRANSACTION, or BACKOUT TRANSACTION statement.

How the statement is translated and which command is actually issued depends on the TP-monitor environment:

- If this command is executed from a stored procedure in a user written Natural program, Natural for DB2 executes the underlying rollback operation. This sets the caller into a must-rollback state. If this command is executed from a stored procedure on behalf of the Natural error processing (implicit ROLLBACK), Natural for DB2 does not execute the underlying rollback operation, thus allowing the caller to receive the original Natural error.
- Under CICS, the BACKOUT TRANSACTION statement is translated into an EXEC CICS ROLLBACK command. However, in pseudo-conversational mode, only changes made to the database since the last terminal I/O are undone. This is due to CICS-specific transaction processing.

Note:

Be aware that with terminal input in database loops, Natural switches to conversational mode if no file server is used.

- In batch mode and under TSO, the BACKOUT TRANSACTION statement is translated into an SQL ROLLBACK command.

Note:

If running in a DSNMTV01 environment the BACKOUT TRANSACTION statement is ignored if the used PSB has been generated without the CMPAT=YES option.

- Under IMS/TM, the BACKOUT TRANSACTION statement is translated into an IMS Rollback (ROLB) command. However, only changes made to the database since the last terminal I/O are undone. This is due to IMS/TM-specific transaction processing.

As all cursors are closed when a logical unit of work ends, a BACKOUT TRANSACTION statement must not be placed within a database loop; instead, it has to be placed outside such a loop or after the outermost loop of nested loops.

If an external program written in another standard programming language is called from a Natural program, this external program should not contain its own ROLLBACK command if the Natural program issues database calls, too. The calling Natural program should issue the BACKOUT TRANSACTION statement on behalf of the external program.

If a program tries to backout updates which have already been committed, for example by a terminal I/O, a corresponding Natural error message (NAT3711) is returned.

DELETE

The DELETE statement is used to delete a row from a DB2 table which has been read with a preceding FIND, READ, or SELECT statement. It corresponds to the SQL statement DELETE WHERE CURRENT OF *cursor-name*, which means that only the row which was read last can be deleted.

Example:

```
FIND EMPLOYEES WITH NAME = 'SMITH'
      AND FIRST_NAME = 'ROGER'
DELETE
```

Natural would translate the above Natural statements into SQL and assign a cursor name (for example, CURSOR1) as follows:

```
DECLARE CURSOR1 CURSOR FOR
SELECT FROM EMPLOYEES
      WHERE NAME = 'SMITH' AND FIRST_NAME = 'ROGER'
DELETE FROM EMPLOYEES
      WHERE CURRENT OF CURSOR1
```

Both the SELECT and the DELETE statement refer to the same cursor.

Natural translates a DML DELETE statement into an SQL DELETE statement in the same way it translates a FIND statement into an SQL SELECT statement.

A row read with a FIND SORTED BY cannot be deleted due to DB2 restrictions explained with the FIND statement. A row read with a READ LOGICAL cannot be deleted either.

DELETE when using the File Server

If a row rolled out to the file server is to be deleted, Natural re-reads automatically the original row from the database to compare it with its image stored in the file server. If the original row has not been modified in the meantime, the DELETE operation is performed. With the next terminal I/O, the transaction is terminated, and the row is deleted from the actual database. If any modification is detected, however, the row is not deleted, and an error message is returned.

Since a DELETE statement requires that Natural re-reads a single row, a unique index must be available for the respective table. All columns which comprise the unique index must be part of the corresponding Natural view.

END TRANSACTION

This statement indicates the end of a logical transaction and releases all DB2 data locked during the transaction. All data modifications are committed and made permanent.

How the statement is translated and which command is actually issued depends on the TP-monitor environment:

- If this command is executed from a stored procedure, Natural for DB2 does not execute the underlying commit operation. This allows the stored procedure to commit updates against non DB2 databases.
- Under CICS, the END TRANSACTION statement is translated into an EXEC CICS SYNCPOINT command. If the file server is used, an implicit end-of-transaction is issued after each terminal I/O. This is due to CICS-specific transaction processing in pseudo-conversational mode.
- In batch mode and under TSO, the END TRANSACTION statement is translated into an SQL COMMIT WORK command.

Note:

If running in a DSNMTV01 environment the END TRANSACTION statement is ignored if the used PSB has been generated without the CMPAT=YES option.

- Under IMS/TM, the END TRANSACTION statement is not translated into an IMS CHKP call, but is ignored. Due to IMS/TM-specific transaction processing, an implicit end-of-transaction is issued after each terminal I/O.

Except when used in combination with the SQL WITH HOLD clause, an END TRANSACTION statement must not be placed within a database loop, since all cursors are closed when a logical unit of work ends. Instead, it has to be placed outside such a loop or after the outermost loop of nested loops.

If an external program written in another standard programming language is called from a Natural program, this external program should not contain its own COMMIT command if the Natural program issues database calls, too. The calling Natural program should issue the END TRANSACTION statement on behalf of the external program.

Note:

With DB2, the END TRANSACTION statement cannot be used to store transaction data.

FIND

The FIND statement corresponds to the SQL SELECT statement.

| |
|---|
| <p>Example:</p> <p>Natural statements:</p> <pre>FIND EMPLOYEES WITH NAME = 'BLACKMORE' AND AGE EQ 20 THRU 40 OBTAIN PERSONNEL_ID NAME AGE</pre> <p>Equivalent SQL statement:</p> <pre>SELECT PERSONNEL_ID, NAME, AGE FROM EMPLOYEES WHERE NAME = 'BLACKMORE' AND AGE BETWEEN 20 AND 40</pre> |
|---|

Natural internally translates a FIND statement into an SQL SELECT statement. The SELECT statement is executed by an OPEN CURSOR command followed by a FETCH command. The FETCH command is executed repeatedly until either all records have been read or the program flow exits the FIND processing loop. A CLOSE CURSOR

command ends the SELECT processing.

The WITH clause of a FIND statement is converted to the WHERE clause of the SELECT statement. The basic search criterion for a DB2 table can be specified in the same way as for an Adabas file. This implies that only database fields which are defined as descriptors can be used to construct basic search criteria and that descriptors cannot be compared with other fields of the Natural view (that is, database fields) but only with program variables or constants.

Note:

As each database field (column) of a DB2 table can be used for searching, any database field can be defined as a descriptor in a Natural DDM.

The WHERE clause of the FIND statement is evaluated by Natural **after** the rows have been selected via the WITH clause. Within the WHERE clause, non-descriptors can be used and database fields can be compared with other database fields.

Note:

DB2 does not have sub-, super-, or phonetic descriptors.

A FIND NUMBER statement is translated into a SELECT statement containing a COUNT(*) clause. The number of rows found is returned in the Natural system variable *NUMBER.

The FIND UNIQUE statement can be used to ensure that only one record is selected for processing. If the FIND UNIQUE statement is referenced by an UPDATE statement, a non-cursor ("searched") UPDATE operation is generated instead of a cursor-oriented (positioned) UPDATE operation. Therefore, it can be used if you want to update a DB2 primary key. It is, however, recommended to use Natural SQL ("searched" UPDATE statement) to update a primary key.

In static mode, the FIND NUMBER and FIND UNIQUE statements are translated into a SELECT SINGLE statement.

The FIND FIRST statement cannot be used. The PASSWORD, CIPHER, COUPLED and RETAIN clauses cannot be used either.

The SORTED BY clause of a FIND statement is translated into the SQL SELECT ... ORDER BY clause, which follows the search criterion. Because this produces a read-only result table, a row read with a FIND statement that contains a SORTED BY clause cannot be updated or deleted.

A limit on the depth of nested database loops can be specified at installation time. If this limit is exceeded, a Natural error message is returned.

FIND when using the File Server

As far as the file server is concerned, there are no programming restrictions with selection statements. It is, however, recommended to make yourself familiar with its functionality considering performance and file server space requirements.

GET

This statement is ISN-based and therefore cannot be used with DB2 tables.

HISTOGRAM

The HISTOGRAM statement returns the number of rows in a table which have the same value in a specific column. The number of rows is returned in the Natural system variable *NUMBER.

Example:

Natural statements:

```
HISTOGRAM EMPLOYEES FOR AGE
OBTAIN AGE
```

Equivalent SQL statement:

```
SELECT COUNT(*), AGE FROM EMPLOYEES
WHERE AGE > -999
GROUP BY AGE
ORDER BY AGE
```

Natural translates the HISTOGRAM statement into an SQL SELECT statement, which means that the control flow is similar to the flow explained for the FIND statement.

READ

The READ statement can also be used to access DB2 tables. Natural translates a READ statement into an SQL SELECT statement.

READ PHYSICAL and READ LOGICAL can be used; READ BY ISN, however, cannot be used, as there is no DB2 equivalent to Adabas ISNs. The PASSWORD and CIPHER clauses cannot be used either.

Since a READ LOGICAL statement is translated into a SELECT ... ORDER BY statement - which produces a read-only table -, a row read with a READ LOGICAL statement cannot be updated or deleted (see Example 1). The start value can only be a constant or program variable; any other field of the Natural view (that is, any database field) cannot be used.

A READ PHYSICAL statement is translated into a SELECT statement without an ORDER BY clause and can therefore be updated or deleted (see Example 2).

Example 1:

Natural statements:

```
READ PERSONNEL BY NAME
OBTAIN NAME FIRSTNAME DATEOFBIRTH
```

Equivalent SQL statement:

```
SELECT NAME, FIRSTNAME, DATEOFBIRTH FROM PERSONNEL
WHERE NAME >= ' '
ORDER BY NAME
```

Example 2:

Natural statements:

```
READ PERSONNEL PHYSICAL
OBTAIN NAME
```

Equivalent SQL statement:

```
SELECT NAME FROM PERSONNEL
```

If the READ statement contains a WHERE clause, this clause is evaluated by the Natural processor **after** the rows have been selected according to the descriptor value(s) specified in the search criterion.

READ when using the File Server

As far as the file server is concerned there are no programming restrictions with selection statements. It is, however, recommended to make yourself familiar with its functionality considering performance and file server space requirements.

STORE

The STORE statement is used to add a row to a DB2 table. The STORE statement corresponds to the SQL statement INSERT.

Example:

Natural statement:

```
STORE RECORD IN EMPLOYEES
  WITH PERSONNEL_ID = '2112'
      NAME           = 'LIFESON'
      FIRST_NAME     = 'ALEX'
```

Equivalent SQL statement:

```
INSERT INTO EMPLOYEES (PERSONNEL_ID, NAME, FIRST_NAME)
VALUES ('2112', 'LIFESON', 'ALEX')
```

The PASSWORD, CIPHER and USING/GIVING NUMBER clauses cannot be used.

UPDATE

The Natural DML UPDATE statement updates a row in a DB2 table which has been read with a preceding FIND, READ, or SELECT statement. It corresponds to the SQL statement UPDATE WHERE CURRENT OF *cursor-name* (positioned UPDATE), which means that only the row which was read last can be updated.

UPDATE when using the File Server

If a row rolled out to the file server is to be updated, Natural automatically re-reads the original row from the database to compare it with its image stored in the file server. If the original row has not been modified in the meantime, the UPDATE operation is performed. With the next terminal I/O, the transaction is terminated and the row is definitely updated on the database. If any modification is detected, however, the row is not updated and an error message is returned.

Since an UPDATE statement requires re-reading a single row by Natural, a unique index must be available for this table. All columns which comprise the unique index must be part of the corresponding Natural view.

UPDATE with FIND/READ

As explained with the FIND statement, Natural translates a FIND statement into an SQL SELECT statement. When a Natural program contains a DML UPDATE statement, this statement is translated into an SQL UPDATE statement and a FOR UPDATE OF clause is added to the SELECT statement.

Example:

```
FIND EMPLOYEES WITH SALARY < 5000
  ASSIGN SALARY = 6000
  UPDATE
```

Natural would translate the above Natural statements into SQL and assign a cursor name (for example, CURSOR1) as follows:

```
DECLARE CURSOR1 CURSOR FOR
SELECT SALARY FROM EMPLOYEES WHERE SALARY < 5000
  FOR UPDATE OF SALARY
UPDATE EMPLOYEES SET SALARY = 6000
  WHERE CURRENT OF CURSOR1
```

Both the SELECT and the UPDATE statement refer to the same cursor.

Due to DB2 logic, a column (field) can only be updated if it is contained in the FOR UPDATE OF clause; otherwise updating this column (field) is rejected. Natural includes automatically all columns (fields) into the FOR UPDATE OF clause which have been modified anywhere in the Natural program or which are input fields as part of a Natural map.

However, a DB2 column is not updated if the column (field) is marked as "not updateable" in the Natural DDM. Such columns (fields) are removed from the FOR UPDATE OF list without any warning or error message. The columns (fields) contained in the FOR UPDATE OF list can be checked with the LISTSQL command.

The Adabas short name in the Natural DDM determines whether a column (field) can be updated.

The following table shows the ranges that apply:

| Short-Name Range | Type of Field |
|------------------|---|
| AA - N9 | non-key field that can be updated |
| Aa - Nz | non-key field that can be updated |
| OA - O9 | primary key field |
| PA - P9 | ascending key field that can be updated |
| QA - Q9 | descending key field that can be updated |
| RA - X9 | non-key field that cannot be updated |
| Ra - Xz | non-key field that cannot be updated |
| YA - Y9 | ascending key field that cannot be updated |
| ZA - Z9 | descending key field that cannot be updated |
| 1A - 9Z | non-key field that cannot be updated |
| 1a - 9z | non-key field that cannot be updated |

Be aware that a primary key field is never part of a FOR UPDATE OF list. A primary key field can only be updated by using a non-cursor UPDATE operation (see also the section UPDATE).

A row read with a FIND statement that contains a SORTED BY clause cannot be updated (due to DB2 limitations as explained with the FIND statement). A row read with a READ LOGICAL cannot be updated either (as explained with the READ statement).

If a column is to be updated which is redefined as an array, it is strongly recommended to update the whole column and not individual occurrences; otherwise, results are not predictable. To do so, in reporting mode you can use the OBTAIN statement (as described in the Natural Statements documentation), which must be applied to all field occurrences in the column to be updated. In structured mode, however, all these occurrences must be defined in the corresponding Natural view.

The data locked by an UPDATE statement are released when an END TRANSACTION (COMMIT WORK) or BACKOUT TRANSACTION (ROLLBACK WORK) statement is executed by the program.

Note:

If a length indicator field or null indicator field is updated in a Natural program without updating the field (column) it refers to, the update of the column is not generated for DB2 and thus no updating takes place.

UPDATE with SELECT

In general, the DML UPDATE statement can be used in both structured and reporting mode. However, after a SELECT statement, only the syntax defined for Natural structured mode is allowed:

UPDATE [RECORD] [IN] [STATEMENT] [(r)]

This is due to the fact that in combination with the SELECT statement, the DML UPDATE statement is only allowed in the special case of:

```

...
SELECT ...
  INTO VIEW view-name
...

```

Thus, only a whole Natural view can be updated; individual columns (fields) cannot.

Example:

```

DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 NAME
  02 AGE
END-DEFINE

SELECT *
  INTO VIEW PERS
  FROM SQL-PERSONNEL
  WHERE NAME LIKE 'S%'

  IF NAME = 'SMITH'
    ADD 1 TO AGE
  UPDATE
  END-IF

END-SELECT
...

```

In combination with the DML UPDATE statement, any other form of the SELECT statement is rejected and an error message is returned.

In all other respects, the DML UPDATE statement can be used with the SELECT statement in the same way as with the Natural FIND statement described earlier in this section and in the Natural Statements documentation.

Natural SQL Statements

This section covers points you have to consider when using Natural SQL statements with DB2. These DB2-specific points partly consist in syntax enhancements which belong to the Extended Set of Natural SQL syntax. The Extended Set is provided in addition to the Common Set to support database-specific features.

This section covers the following topics:

- Common Syntactical Items
- CALLDBPROC
- COMMIT
- DELETE
- INSERT
- PROCESS SQL
- ROLLBACK
- SELECT
- UPDATE

Common Syntactical Items

The following syntactical items are either DB2-specific and do not conform to the standard SQL syntax definitions (that is, to the Common Set of Natural SQL syntax) or impose restrictions when used with DB2 (see also SQL Statements in the Natural Statements documentation).

atom

An atom can be either a parameter (that is, a Natural program variable or host variable) or a constant. When running dynamically, however, the use of host variables is restricted by DB2. For further details, refer to the relevant literature on DB2.

comparison

The following three comparison operators are specific to DB2 and belong to the Natural Extended Set.

↯ =
↯ >
↯ <

factor

The following three factors are specific to DB2 and belong to the Natural Extended Set:

special-register
scalar-function (scalar-expression, ...)
scalar-expression unit
case-expression

scalar-function

A scalar-function is a built-in function that can be used in the construction of scalar computational expressions. Scalar functions are specific to DB2 and belong to the Natural Extended Set.

The following scalar functions are supported:

CHAR
COALECE
DATE
DAY
DAYS
DECIMAL
DIGITS
FLOAT
HEX
HOUR
INTEGER
LENGTH
MICROSECOND
MINUTE
MONTH
NULLIF
SECOND
STRIP
SUBSTR
TIME
TIMESTAMP
VALUE
VARGRAPHIC
YEAR

Each scalar function is followed by one or more scalar expressions in parentheses. The number of scalar expressions depends upon the scalar function. Multiple scalar expressions must be separated from one another by commas.

Example:

```
SELECT NAME  
  INTO NAME  
  FROM SQL-PERSONNEL  
  WHERE SUBSTR ( NAME, 1, 3 ) = 'Fri'  
  . . .
```

scalar-operator

The concatenation operator (CONCAT or "//") does not conform to standard SQL. It is specific to DB2 and belongs to the Natural Extended Set.

special-register

The following special registers do not conform to standard SQL. They are specific to DB2 and belong to the Natural Extended Set:

CURRENT TIMEZONE
CURRENT DATE
CURRENT TIME
CURRENT_TIMEZONE
CURRENT_DATE
CURRENT_TIME
CURRENT TIMESTAMP
CURRENT SQLID
CURRENT PACKAGESET
CURRENT SERVER
CURRENT DEGREE
CURRENT RULES

A reference to a special register returns a scalar value.

Using the command SET CURRENT SQLID, the creator name of a table can be substituted by the current SQLID. This enables you to access identical tables with the same table name but with different creator names.

units

Units, also called "durations", are specific to DB2 and belong to the Natural Extended Set.

The following units are supported:

YEAR
YEARS
MONTH
MONTHS
DAY
DAYS
HOUR
HOURS
MINUTE
MINUTES
SECOND
SECONDS
MICROSECOND
MICROSECONDS

case-expression

| |
|--|
| $\text{CASE } \left\{ \begin{array}{l} \textit{searched-when-clause} \dots \\ \textit{simple-when-clause} \end{array} \right\} \left[\text{ELSE } \left\{ \begin{array}{l} \text{NULL} \\ \textit{scalar expression} \end{array} \right\} \right] \text{END}$ |
|--|

Case-expressions do not conform to standard SQL and are therefore supported by the Natural SQL Extended Set only.

Example:

```

DEFINE DATA LOCAL
01 #EMP
02 #EMPNO (A10)
02 #FIRSTNME (A15)
02 #MIDINIT (A5)

```

```

02 #LASTNAME (A15)
02 #EDLEVEL (A13)
02 #INCOME (P7)
END-DEFINE
SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME,
      (CASE WHEN EDLEVEL < 15 THEN 'SECONDARY'
            WHEN EDLEVEL < 19 THEN 'COLLEGE'
            ELSE 'POST GRADUATE'
            END ) AS EDUCATION, SALARY + COMM AS INCOME
INTO
#EMPNO, #FIRSTNME, #MIDINIT, #LASTNAME,
#EDLEVEL, #INCOME
FROM DSN8510-EMP
WHERE (CASE WHEN SALARY = 0 THEN NULL
        ELSE SALARY / COMM
        END ) > 0.25

DISPLAY #EMP
END-SELECT
END

```

CALLDBPROC

The CALLDBPROC statement allows you to call DB2 stored procedures. It supports the result set mechanism of DB2 Version 5 and it enables you to call DB2 stored procedures written in Natural.

For a detailed description of the syntax of the statement, see the Natural Statements documentation.

If the CALLDBPROC statement is executed dynamically, all parameters and constants are mapped to the variables of the following DB2 SQL statement:

```
CALL :hv USING DESCRIPTOR :sqlda statement
```

:hv denotes a host variable containing the name of the procedure to be called and *:sqlda* is a dynamically generated sqlda describing the parameters to be passed to the stored procedure.

If the CALLDBPROC statement is executed statically, the constants of the CALLDBPROC statement are also generated as constants in the generated assembler SQL source for the DB2 precompiler.

If the SQLCODE created by the CALL statement indicates that there are result sets (SQLCODE +466 and +464), Natural for DB2 runtime executes a

```
DESCRIBE PROCEDURE :hv INTO :sqlda
```

statement in order to retrieve the result set locator values of the result sets created by the invoked stored procedure. These values are put into the RESULT SETS variables specified in the CALLDBPROC statement. Each RESULT SETS variable specified in a CALLDBPROC for which no result set locator value is present is reset to zero. The result set locator values can be used to read the result sets by means of the READ RESULT SET statement as long as the database transaction which created the result set has not yet issued a COMMIT or ROLLBACK.

If the result set was created by a cursor WITH HOLD, the result set locator value remains valid after a COMMIT operation.

Unlike other Natural SQL statements, CALLDBPROC enables you (optionally!) to specify a SQLCODE variable following the GIVING keyword which will contain the SQLCODE of the underlying CALL statement. If GIVING is specified, it is up to the Natural program to react on the SQLCODE (error message NAT3700 is not issued by the runtime).

Parameter data types supported by the CALLDBPROC statement:

| Natural Format/Length | DB2 Data Type |
|---------------------------------------|--------------------------|
| <i>An</i> | CHAR(<i>n</i>) |
| B2 | SMALLINT |
| B4 | INT |
| <i>Bn</i> ; <i>n</i> not equal 2 or 4 | CHAR(<i>n</i>) |
| F4 | REAL |
| F8 | DOUBLE PRECISION |
| I2 | SMALLINT |
| I4 | INT |
| <i>Nnn.m</i> | NUMERIC(<i>nn+m,m</i>) |
| <i>Pnn.m</i> | NUMERIC(<i>nn+m,n</i>) |
| <i>Gn</i> | GRAPHIC(<i>n</i>) |
| <i>An/1:m</i> | VARCHAR(<i>n*m</i>) |

CALLMODE=NATURAL

This parameter allows DB2 stored procedures written in Natural to be invoked. Stored procedures written in Natural are Natural subprograms which execute in the stored procedure address space.

If the CALLMODE=NATURAL parameter is specified, an additional parameter describing the parameters passed to the Natural stored procedure is passed from the client, i.e. caller, to the server, i.e. stored procedure. The parameter is of format VARCHAR from the viewpoint of DB2. Therefore, every stored procedure written in Natural has to be defined in the SYSIBM.SYSPROCEDURES table with this VARCHAR parameter as the first in its PARMLIST row.

From the viewpoint of the caller, i.e. the Natural program, and from the viewpoint of the stored procedure, i.e. Natural subprogram, this additional parameter is invisible. It is passed as first parameter by the Natural for DB2 runtime and it is used as on the server side to build the copy of the passed data in the Natural thread and the corresponding CALLNAT statement. Additionally, this parameter serves as a container for error information created during execution of the Natural stored procedure by the Natural runtime. It also contains information on the library where you are logged on and the Natural subprogram to be invoked.

The following table describes the first parameter passed between the caller and the stored procedure if CALLMODE = NATURAL is specified.

| NAME | FORMAT | PROCESSING MODE SERVER |
|------------------------------|------------|---------------------------------------|
| STCBL | I2 | Input (size of following information) |
| Procedure Information | | |
| STCBLENG | A4 | Input (printable STCBL) |
| STCBID | A4 | Input ('STCB') |
| STCBVERS | A4 | Input (version of STCB '310') |
| STCBUSER | A8 | Input (user ID) |
| STCBLIB | A8 | Input (library) |
| STCBPROG | A8 | Input (calling program) |
| STCBPSW | A8 | Unused (password) |
| STCBSTNR | A4 | Input (CALLDBPROC statement number) |
| STCBTCP | A8 | Input (procedure called) |
| STCBPANR | A4 | Input (number of parameters) |
| Error Information | | |
| STCBERNR | A5 | Output (Natural error number) |
| STCBSTAT | A1 | Unused (Natural error status) |
| STCBLIB | A8 | Unused (Natural error library) |
| STCBPRG | A8 | Unused (Natural error program) |
| STCBLVL | A1 | Unused (Natural error level) |
| STCBOTP | A1 | Unused (error object type) |
| STCBEDYL | A2 | Output (error text length) |
| STCBEDYT | A88 | Output (error text) |
| | A100 | Reserved for future use |
| Parameter Information | | |
| FORMAT_DESCRIPTION | A variable | Input |

The FORMAT_DESCRIPTION contains a description for each parameter passed to the stored procedure consisting of parameter type, format specification and length. Parameter type is the AD attribute of the CALLNAT statement as described in the Natural Statements documentation.

Each parameter has the following format description element in the FORMAT_DESC string

atl,p[,dl]...

where

- *a* is an attribute mark which specifies the parameter type:

| Mark | Type | Equivalent AD Attribute | Equivalent DB2 Clause |
|------|----------------|-------------------------|-----------------------|
| M | modifiable | AD=M | INOUT |
| O | non-modifiable | AD=O | IN |
| A | input only | AD=A | OUT |

- *t* is one of the following Natural format tokens:

| <i>t</i> | Description | <i>l</i> | <i>p</i> | <i>dl</i> | Example |
|----------|-----------------------|----------|----------|--------------------|-------------------------|
| A | Alphanumeric | 1-253 | 0 | 1-32767 or - | A30,0 or A30,0,10 |
| N | Numeric unpacked | 1-29 | 0-7 | - | N10,3 |
| P | Packed numeric | 1-29 | 0-7 | - | P13,4 |
| I | Integer | 2 or 4 | 0 | - | I2,0 |
| F | Floating point | | 0 | - | I4,0 |
| B | Binary | | 0 | - | B23,0 |
| D | Date (unsupported) | | | | |
| T | Time (unsupported) | | | | |
| L | Logical (unsupported) | | | | |

- *l* is an integer denoting the length/scale of the field. For numeric and packed numeric fields, *l* denotes the total number of digits of the field that is, the sum of the digits left and right of the decimal point. The Natural format N7.3 is, for example, represented by N10.3. See also the table above.
- *p* is an integer denoting the precision of the field. It is usually 0, except for numeric and packed fields where it denotes the number of digits right of the decimal point. See also the table above.
- *dl* is also an integer denoting the occurrences of the alphanumeric array (alphanumeric only). See also the table above.

This descriptive/control parameter is invisible to the calling Natural program and to the called Natural stored procedure, but it has to be defined in the parameter definition of the stored procedure row in the SYSIBM.SYSPROCEDURES table for the stored procedure.

The following table shows the number of parameters which have to be defined in the SYSIBM.SYSPROCEDURES table depending on the number of user parameters and whether the client (i.e. the caller of a stored procedure for DB2 for OS/390) and the server (i.e. the stored procedure for DB2 for OS/390) is written in Natural or in a different host language. *n* denotes the number of 'user' parameters.

| Client\Server | Natural | not Natural |
|---------------|--------------|--|
| Natural | <i>n</i> + 1 | <i>n</i> (CALLMODE=NONE) <i>n</i> +1 (CALLMODE=NATURAL) |
| not Natural | <i>n</i> + 1 | N |

Example issuing CALLDBPROC and READ RESULT SET statements:

```

DEFINE DATA LOCAL
1 ALPHA          (A8)
1 NUMERIC        (N7.3)
1 PACKED         (P9.4)
1 VCHAR          (A20/1:5) INIT      <'DB25SGCP'>
1 INTEGER2       (I2)
1 INTEGER4       (I4)
1 BINARY2        (B2)
1 BINARY4        (B4)
1 BINARY12       (B12)
1 FLOAT4         (F4)
1 FLOAT8         (F8)
1 INDEX-ARRAY   (I2/1:11)
1 INDEX-ARRAY1  (I2)
1 INDEX-ARRAY2  (I2)
1 INDEX-ARRAY3  (I2)
1 INDEX-ARRAY4  (I2)
1 INDEX-ARRAY5  (I2)
1 INDEX-ARRAY6  (I2)
1 INDEX-ARRAY7  (I2)
1 INDEX-ARRAY8  (I2)
1 INDEX-ARRAY9  (I2)
1 INDEX-ARRAY10(I2)
1 INDEX-ARRAY11(I2)
1 #RESP         (I4)
1 #RS1          (I4) INIT <99>
1 #RS2          (I4) INIT <99>
LOCAL
1 V1 VIEW OF SYSIBM-SYSTABLES
2 NAME
1 V2 VIEW OF SYSIBM-SYSPROCEDURES
2 PROCEDURE
2 RESULT_SETS
1 V (I2) INIT <99>
END-DEFINE
CALLDBPROC 'DAEFDB25.SYSPROC.SNGSTPC' DSN8510-EMP
ALPHA INDICATOR :INDEX-ARRAY1
NUMERIC INDICATOR :INDEX-ARRAY2
PACKED INDICATOR :INDEX-ARRAY3
VCHAR(*) INDICATOR :INDEX-ARRAY4
INTEGER2 INDICATOR :INDEX-ARRAY5
INTEGER4 INDICATOR :INDEX-ARRAY6
BINARY2 INDICATOR :INDEX-ARRAY7
BINARY4 INDICATOR :INDEX-ARRAY8
BINARY12 INDICATOR :INDEX-ARRAY9
FLOAT4 INDICATOR :INDEX-ARRAY10
FLOAT8 INDICATOR :INDEX-ARRAY11
RESULT SETS #RS1 #RS2
CALLMODE=NATURAL
READ (10) RESULT SET #RS2 INTO VIEW V2 FROM SYSIBM-SYSTABLES
WRITE 'PROC F RS :' PROCEDURE 50T RESULT_SETS
END-RESULT
END

```

COMMIT

The SQL COMMIT statement indicates the end of a logical transaction and releases all DB2 data locked during the transaction. All data modifications are made permanent.

COMMIT is a synonym for the Natural END TRANSACTION statement.

No transaction data can be provided with the COMMIT statement.

If this command is executed from a stored procedure, Natural for DB2 does not execute the underlying commit operation. This allows the stored procedure to commit updates against non DB2 databases.

Under CICS, the COMMIT statement is translated into an EXEC CICS SYNCPOINT command. If the file server is used, an implicit end-of-transaction is issued after each terminal I/O. This is due to CICS-specific transaction processing in pseudo-conversational mode.

Under IMS/TM, the COMMIT statement is not translated into an IMS Checkpoint command, but is ignored. An implicit end-of-transaction is issued after each terminal I/O. This is due to IMS/TM-specific transaction processing.

Unless when used in combination with the WITH HOLD clause, a COMMIT statement must not be placed within a database loop, since all cursors are closed when a logical unit of work ends. Instead, it has to be placed outside such a loop or after the outermost loop of nested loops.

If an external program written in another standard programming language is called from a Natural program, this external program should not contain its own COMMIT command if the Natural program issues database calls, too. The calling Natural program should issue the COMMIT statement on behalf of the external program.

DELETE

Both the "cursor-oriented" or "positioned" and the "non-cursor" or ""searched"" SQL DELETE statement are supported as part of Natural SQL; the functionality of the "positioned" DELETE statement corresponds to that of the Natural DML DELETE statement.

With DB2, a table name in the FROM clause of a "searched" DELETE statement can be assigned a *correlation-name*. This does not correspond to the standard SQL syntax definition and therefore belongs to the Natural Extended Set.

The "searched" DELETE statement must be used, for example, to delete a row from a self-referencing table, since with self-referencing tables a positioned DELETE is not allowed by DB2.

INSERT

The INSERT statement is used to add one or more new rows to a table.

Since the INSERT statement can contain a select expression, all the DB2-specific syntactical items described above apply.

PROCESS SQL

The PROCESS SQL statement is used to issue SQL statements to the underlying database. The statements are specified in a *statement-string*, which can also include constants and parameters.

The set of statements which can be issued is also referred to as Flexible SQL and comprises those statements which can be issued with the SQL statement "EXECUTE".

In addition, Flexible SQL includes the DB2-specific statements CONNECT, SET CURRENT PACKAGESET, SET CURRENT DEGREE, SET CURRENT RULES, SET CURRENT SQLID, SET *host-variable*= *special-register* RELEASE, SET CONNECTION, and CALL.

Note:

To avoid transaction synchronization problems between the Natural environment and DB2, the COMMIT and ROLLBACK statements must not be used within PROCESS SQL.

CALL

Natural for DB2 now supports the DB2 Version 4 CALL statement by means of the PROCESS SQL statement. However, the syntax of the CALL statement is restricted as shown below.

| |
|---|
| $\text{CALL } \left\{ \begin{array}{l} \textit{procedure-name} \\ \textit{host-variable} \end{array} \right\} \left[\left(\left\{ \begin{array}{l} \text{[:U:] } \textit{host-variable} \\ \textit{constant} \\ \text{NULL} \end{array} \right\} \dots \right) \right]$ |
|---|

The using descriptor parameter list format of the CALL statement is not supported.

Every host variable specified in the CALL parameter list should be prefixed with :U or the prefix should be omitted, regardless how the parameters are defined in the SYSIBM.SYSPROCEDURES tables. To use :G as *host-variable* prefix is strictly forbidden.

Example:

```
PROCESS SQL DB2-DDM

  <<CALL DB2PROC
      (:U:#USER,
       :U:#DATE,
       'ALPHA',
       NULL
      )
  >>
```

DB2PROC is a procedure name to be defined in SYSIBM.PROCEDURES.

#USER, #DATE are Natural variables.

'ALPHA' is a literal.

NULL is a keyword representing the NULL value.

Whether data are returned by the called procedure is only determined by the definition of call parameter list in the SYSIBM.SYSPROCEDURE table for the called procedure.

ROLLBACK

The SQL ROLLBACK statement undoes all database modifications made since the beginning of the last logical transaction. Logical transactions can start either after the beginning of a session or after the last COMMIT/END TRANSACTION or ROLLBACK/BACKOUT TRANSACTION statement. All records held during the transaction are released.

ROLLBACK is a synonym for the Natural statement BACKOUT TRANSACTION.

If this command is executed from a stored procedure in a user written Natural program, Natural for DB2 executes the underlying rollback operation. This sets the caller into a must-rollback state. If this command is executed from a stored procedure on behalf of the Natural error processing (implicit ROLLBACK), Natural for DB2 does not execute the underlying rollback operation, thus allowing the caller to receive the original Natural error.

Under CICS, the ROLLBACK statement is translated into an EXEC CICS ROLLBACK command. However, if the file server is used, only changes made to the database since the last terminal I/O are undone. This is due to CICS-specific transaction processing in pseudo-conversational mode.

Under IMS/TM, the ROLLBACK statement is translated into an IMS Rollback (ROLB) command. However, only changes made to the database since the last terminal I/O are undone. This is due to IMS/TM-specific transaction processing.

As all cursors are closed when a logical unit of work ends, a ROLLBACK statement must not be placed within a database loop; instead, it has to be placed outside such a loop or after the outermost loop of nested loops.

If an external program written in another standard programming language is called from a Natural program, this external program should not contain its own ROLLBACK command if the Natural program issues database calls, too. The calling Natural program should issue the ROLLBACK statement on behalf of the external program.

SELECT

Cursor-Oriented Selection

Like the Natural FIND statement, the cursor-oriented SELECT statement is used to select a set of rows (records) from one or more DB2 tables, based on a search criterion. Since a database loop is initiated, the loop must be closed by a LOOP (reporting mode) or END-SELECT statement. With this construction, Natural uses the same loop processing as with the FIND statement.

In addition, no cursor management is required from the application program; it is automatically handled by Natural.

OPTIMIZE FOR integer ROWS Clause

[OPTIMIZE FOR *integer* ROWS]

The OPTIMIZE FOR *integer* ROWS clause is used to inform DB2 in advance of the number (*integer*) of rows to be retrieved from the result table. Without this clause, DB2 assumes that all rows of the result table are to be retrieved and optimizes accordingly.

This optional clause is useful if you know how many rows are likely to be selected, because optimizing for *integer* rows can improve performance if the number of actually selected rows does not exceed the *integer* value (which can be in the range from 0 to 2147483647).

Example:

```
SELECT name INTO #name FROM table
WHERE AGE = 2 OPTIMIZE FOR 100 ROWS
```

WITH Clauses

Isolation Level

[WITH { CS
RR
UR }]

This option allows you to specify an explicit isolation level with which the statement is to be executed.

- CS means cursor stability.
- RR means repeatable read.
- UR means uncommitted read.

WITH UR can only be specified within a SELECT statement and when the table is read-only. The default isolation level is determined by the isolation of the package or plan into which the statement is bound. The default isolation level also depends on whether the result table is read-only or not. To find out the default isolation level, refer to the IBM documentation.

Note:

This option also works for non-cursor selection.

WITH HOLD Clause

[WITH HOLD]

The WITH HOLD clause is used to prevent cursors from being closed by a commit operation within database loops. If WITH HOLD is specified, a commit operation commits all the modifications of the current logical unit of work, but releases only locks that are not required to maintain the cursor. This optional clause is mainly useful in batch mode; it is ignored in CICS pseudo-conversational mode and in IMS message-driven programs.

Example:

```
SELECT name INTO #name FROM table
WHERE AGE = 2 WITH HOLD
```

WITH RETURN

[WITH RETURN]

The WITH RETURN clause is used to create result sets. Therefore it should only be used in programs which operate as stored procedure. If the WITH RETURN clause is specified in a SELECT statement, the underlying cursor remains open when the associated processing loop is left, except when the processing loop had read all rows of the result set itself. During first execution of the processing loop, only the cursor is opened. The first row is not yet fetched. This allows the Natural program to return a full result set to the caller of the stored procedure. It is up to the Natural program to decide how many rows are processed by the program itself and how many unprocessed rows are returned to the caller of the stored procedure. If it wants to process rows of the select operation itself, it should code

```
IF *counter =1 ESCAPE TOP END-IF
```

in order to avoid processing of the first "empty row" in the processing loop. If it decides by some criteria of its own to terminate its own processing of rows, it should code

```
If condition ESCAPE BOTTOM END-IF
```

If the program reads all rows of the result set, the cursor is closed and no result set is returned for this SELECT WITH RETURN to the caller of the stored procedure.

The following programs are examples for retrieving full result sets (Example 1) and partial result sets (Example 2).

Example 1:

```

DEFINE DATA LOCAL
. . .
END DEFINE
*
*   Return all rows of the result set
*
SELECT * INTO VIEW V2
                FROM SYSIBM-SYSPROCEDURES
                WHERE RESULT_SETS > 0
                WITH RETURN

ESCAPE BOTTOM
END-SELECT
END

```

Example 2:

```

DEFINE DATA LOCAL
. . .
END DEFINE
*
*   Read the first two rows and return the rest as result set
*
SELECT * INTO VIEW V2
                FROM SYSIBM-SYSPROCEDURES
                WHERE RESULT_SETS > 0
                WITH RETURN

WRITE PROCEDURE *COUNTER
IF *COUNTER = 1 ESCAPE TOP END-IF
IF *COUNTER =3 ESCAPE BOTTOM END-IF
END-SELECT
END

```

Non-Cursor Selection - SELECT SINGLE

The Natural statement **SELECT SINGLE** provides the functionality of a non-cursor selection (singleton **SELECT**); that is, a select expression that retrieves at most one row without using a cursor.

Since DB2 supports the singleton **SELECT** command in static SQL only, in dynamic mode, the Natural **SELECT SINGLE** statement is executed like a set-level **SELECT** statement, which results in a cursor operation. However, Natural checks the number of rows returned by DB2. If more than one row is selected, a corresponding error message is returned.

UPDATE

Both the "cursor-oriented" or "positioned" and the "non-cursor" or "'searched'" SQL **UPDATE** statement are supported as part of Natural SQL. Both of them reference either a table or a Natural view.

With DB2, the name of a table or Natural view to be referenced by a "searched" **UPDATE** can be assigned a *correlation-name*. This does not correspond to the standard SQL syntax definition and therefore belongs to the Natural Extended Set.

The "searched" **UPDATE** statement must be used, for example, to update a primary key field, since DB2 does not allow updating of columns of a primary key by using a positioned **UPDATE** statement.

Note:

If you use the SET * notation, all fields of the referenced Natural view are added to the FOR UPDATE OF and SET lists. Therefore, ensure that your view contains only fields which can be updated; otherwise, a negative SQLCODE is returned by DB2.

Natural System Variables

When used with DB2, there are restrictions for the following Natural system variables:

- *ISN
- *NUMBER

*ISN

As there is no DB2 equivalent of Adabas ISNs, the system variable *ISN is not applicable to DB2 tables.

*NUMBER

When used with a FIND NUMBER or HISTOGRAM statement, *NUMBER contains the number of rows actually found.

When applied to data from a DB2 table in any other case, the system variable *NUMBER only indicates whether any rows have been found. If no rows have been found, *NUMBER is "0". Any value other than "0" indicates that at least one row has been found; however, the value contained in *NUMBER has no relation to the number of rows actually found.

The reason is that if *NUMBER were to produce a valid number, Natural would have to translate the corresponding FIND statement into an SQL SELECT statement including the special function COUNT(*); however, a SELECT containing a COUNT function would produce a read-only result table, which would not be available for updating. In other words, the option to update selected data was given priority in Natural over obtaining the number of rows that meet the search criteria.

To obtain the number of rows affected by the Natural SQL statements ""searched" UPDATE", ""searched" DELETE" and INSERT, the Natural subprogram NDBNROW is provided. Alternatively, you can use the Natural system variable *ROWCOUNT as described in the Natural Programming Reference documentation.

Error Handling

In contrast to the normal Natural error handling, where either an ON ERROR statement is used to intercept execution time errors or standard error message processing is performed and program execution is terminated, the enhanced error handling of Natural for DB2 provides an application controlled reaction to the encountered SQL error.

Two Natural subprograms, NDBERR and NDBNOERR, are provided to disable the usual Natural error handling and to check the encountered SQL error for the returned SQL code. This functionality replaces the "E" function of the DB2SERV interface, which is still provided but no longer documented.

For further information on Natural subprograms provided for DB2, see the section Natural Subprograms.

Example:

```
DEFINE DATA LOCAL
01 #SQLCODE           ( I4 )
01 #SQLSTATE          ( A5 )
01 #SQLCA             ( A136 )
01 #DBMS              ( B1 )
END-DEFINE
```

```

*
*           Ignore error from next statement
*
CALLNAT 'NDBNOERR'
*
*           This SQL statement produces an SQL error
*
INSERT INTO SYSIBH-SYSTABLES (CREATOR, NAME, COLCOUNT)
VALUES ('SAG', 'MYTABLE', '3')
*
*           Investigate error
*
CALLNAT 'NDBERR' #SQLCODE #SQLSTATE #SQLCA #DBMS
*
IF #DBMS NE 2                               /* not DB2
    MOVE 3700 TO *ERROR-NR
END-IF
*
DECIDE ON FIRST VALUE OF #SQLCODE
    VALUE 0, 100                             /* successful execution
        IGNORE
    VALUE -803                               /* duplicate row
        /* UPDATE existing record
        /*
        IGNORE
    NONE VALUE
        MOVE 3700 TO *ERROR-NR
END-DECIDE
*
END

```

Stored Procedures in Natural

- Writing Stored Procedures
- Security
- Example Stored Procedure

Writing Stored Procedures

To write stored procedures in Natural

1. Determine the format and attributes of the parameters which are passed between the caller and the stored procedure.
Consider creating a Natural parameter data area.
stored procedures do not support data groups and redefinitions within their parameters.
2. Decide the linkage of the stored procedure (simple or simple with NULL).
If you have decided to use simple with NULL linkage, you have to append the parameter data area within the Natural stored procedure by the NULL indicator area, which contains a NULL indicator (I2) for each other parameter. See also the DB2 literature by IBM.
3. Decide which and how many result sets the stored procedure will return to the caller.
4. Code your stored procedure as a Natural subprogram.
 - **Returning result sets**
To return result sets code a SELECT statement with the WITH RETURN option.
To return the whole result set code an ESCAPE BOTTOM immediately after the SELECT.
To return part of the result set code an IF *COUNTER = 1 ESCAPE TOP END-IF immediately following the SELECT statement. This ensures that you do not process the first empty row that is returned by the SELECT WITH RETURN statement. To stop row processing execute an ESCAPE BOTTOM statement.
If you do not leave the processing loop initiated by the SELECT WITH RETURN via ESCAPE

BOTTOM, the result set created is closed and nothing is returned to the caller.

- **Attention when accessing other databases**

You can access other databases (for instance Adabas) within a DB2 stored procedure written in Natural. But you should keep in mind that your access to other databases is neither synchronized with the updates done by the caller of the stored procedure, nor with the updates done against DB2 within the stored procedure.

- **NDB handling of COMMIT and ROLLBACK statements**

DB2 does not allow a stored procedure to issue COMMIT or ROLLBACK statements (the execution of those statements puts the caller into a must-rollback state). Therefore, the NDB runtime handles those statements as follows when they are issued from a stored procedure:

COMMIT against DB2 will be skipped.

This allows the stored procedure to commit Adabas updates without getting a must-rollback state from DB2.

ROLLBACK against DB2 will be skipped if it is created by Natural itself.

ROLLBACK against DB2 will be executed if it is user-programmed.

Thus, after a Natural error, the caller receives the Natural error information and not the unqualified must-rollback state. Additionally, this function ensures that, if the user program backs out the transaction, every database transaction of the stored procedure is backed out.

5. Enter a row describing your stored procedure into the SYSIBM.SYSPROCEDURES table.

This can be done by SYSDB2 Procedure Maintenance. In order to define a stored procedure written in Natural, the following data has to be entered:

| Column | Data |
|-------------|---|
| PROCEDURE | The name of your Natural subprogram representing your stored procedure. |
| LOADMOD | The name of your Natural for DB2 server stub module. |
| LINKAGE | Either blank or N as you decided at 2. |
| IBMREQD | N |
| PGM_TYPE | Depending on the specification of the MAIN parameter of the Natural for DB2 server stub module. |
| RESULT_SETS | As you decided at 3. |
| LANGUAGE | ASSEMBLER |
| PARMLIST | <p>The description of the parameters you determined at 1. The first field of the PARMLIST must be defined as Natural internal parameter description STCB. It must be specified as a VARCHAR field with DATA INOUT and the following size: 274 + 13*N</p> <p>where N is the number of parameters passed to the stored procedured (STCB not counted).</p> <p>If you have created a Natural parameter data area for the stored procedure, you can easily derive the PARMLIST from your PDA by pressing PF5 on your PARMLIST screen of SYSDB2 and entering the library and the name of the PDA. In this case, the VARCHAR field for the parameter description is generated automatically.</p> |

6. Code your Natural program invoking the stored procedure via the CALLDBPROC statement.

Code the parameters in the CALLDBPROC statement in the same sequence as they are specified in the stored procedure. Define the parameters in the calling program in a format that is compatible with the format defined in the stored procedure.

If you use result sets, specify a RESULT SETS clause in the CALLDBPROC statement followed by a number of result set locator variables of FORMAT (I4). The number of result set locator variables should be the same as the number or result sets created by the stored procedure. If you specify less than are created, some result sets are lost. If you specify more than are created, the remaining result set locator variables are lost. The sequence of

locator variables corresponds to the sequence in which the result sets are created by the stored procedure. Keep in mind that the fields into which the result set rows are read have to correspond to the fields used in the Select WITH RETURN statement, which created the result set.

Security

A Natural stored procedure is executed with the authorization of the user ID provided by DB2, i.e., the user authorization of the address space the stored procedure is running in, or with the authorization of the caller of the procedure if External Security is set to Yes.

Example Stored Procedure NDBPURGN

This section describes the example stored procedure NDBPURGN, a Natural subprogram which purges Natural objects from the buffer pool used by the Natural stored procedures server.

Below is information on:

- Members of NDBPURGN
- Defining the Stored Procedure
- Stored Procedure Control Block (STCB)

Members of NDBPURGN

The example stored procedure NDBPURGN comprises the following text members which are stored in the library SYSDB2:

| Member | Explanation |
|----------|--|
| CR5PURGN | Input member for SYSDB2 ISQL. Contains SQL statements used to declare NDBPURGN in DB2 Version 5. |
| CR6PURGN | Input member for SYSDB2 ISQL. Contains SQL statements used to declare NDBPURGN in DB2 Version 6 or above. |
| NDBPURGP | The client (Natural) program which <ul style="list-style-type: none"> ● Requests the name of the program to be purged and the library where it resides, ● Invokes the stored procedure NDBPURGN and ● Reports the outcome of the request. |
| NDBPURGN | The stored procedure which purges objects from the buffer pool. NDBPURGN invokes the user exit USR0340N supplied in the library SYSEXT. Therefore, USR0340N must be available in the library defined as the steplib for the execution environment. |

Defining the Stored Procedure

 To define the example stored procedure NDBPURGN

- Define the stored procedure in the DB2 catalog by using the SQL statements provided as text members CR5PURGN (for DB2 Version 5) and CR6PURGN (for DB2 Version 6).
- Specify the name of the Natural stored procedure stub (here: NDB31SRV) as LOADMOD (V5) or EXTERNAL NAME (V6). The Natural stored procedure stub is generated during the installation by assembling the NDBSTUB macro.

- As the first parameter, pass the internal Natural parameter STCB to the stored procedure. The STCB parameter is a VARCHAR field which contains information required to invoke the stored procedure in Natural:
 - The program name of the stored procedure and the library where it resides,
 - The description of the parameters passed to the stored procedure and
 - The error message created by Natural if the stored procedure fails during the execution.

The STCB parameter is generated automatically by the CALLMODE=NATURAL clause of the CALLDBPROC statement and is removed from the parameters passed to the Natural stored procedure by the server stub. Thus, STCB is invisible to the caller and the stored procedure. However, if a non-Natural client intends to call a Natural stored procedure, he has to pass the STCB parameter explicitly. See also Stored Procedure Control Block below.

Stored Procedure Control Block (STCB)

Below is the Stored Procedure Control Block (STBC) generated by the CALLMODE=NATURAL clause as generated by the stored procedure NDBPURGN before and after execution. Changed values are emphasized in boldface:

STCB before Execution:

| | | | | | | |
|--------|------------------|----------|----------|------------------|-----------------------------|----------|
| 004C82 | 0132F0F3 | F0F6E2E3 | C3C2F3F1 | F040C8C7 | *..0306STCB310 HG* | 11097D42 |
| 004C92 | D2404040 | 4040C8C7 | D2404040 | 4040D5C4 | *K HGK ND* | 11097D52 |
| 004CA2 | C2D7E4D9 | C7D74040 | 40404040 | 4040F0F5 | *BPURGP 05* | 11097D62 |
| 004CB2 | F7F0D5C4 | C2D7E4D9 | C7D5F0F0 | F0F6 F0F9 | *70NDBPURGN000 609 * | 11097D72 |
| 004CC2 | F9F9F9 40 | 40404040 | 40404040 | 40404040 | * 999 * | 11097D82 |
| 004CD2 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097D92 |
| 004CE2 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097DA2 |
| 004CF2 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097DB2 |
| 004D02 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097DC2 |
| 004D12 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097DD2 |
| 004D22 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097DE2 |
| 004D32 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097DF2 |
| 004D42 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097E02 |
| 004D52 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097E12 |
| 004D62 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097E22 |
| 004D72 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097E32 |
| 004D82 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097E42 |
| 004D92 | 40404040 | D4C1F86B | F0D4C1F4 | F06BF0D4 | * MA8,0MA40,0M* | 11097E52 |
| 004DA2 | C2F26BF0 | D4C2F26B | F0D4C9F2 | 6BF0D4C9 | *I2,0MI2,0MI2,0MI* | 11097E62 |
| 004DB2 | F26BF04B | | | | *2,0. * | 11097E72 |

STCB after Execution:

| | | | | | | |
|--------|----------|----------|----------|----------|---------------------|----------|
| 004C82 | 0132F0F3 | F0F6E2E3 | C3C2F3F1 | F040C8C7 | *. .0306STCB310 HG* | 11097D42 |
| 004C92 | D2404040 | 4040C8C7 | D2404040 | 4040D5C4 | *K HGK ND* | 11097D52 |
| 004CA2 | C2D7E4D9 | C7D74040 | 40404040 | 4040F0F5 | *BPURGP 05* | 11097D62 |
| 004CB2 | F7F0D5C4 | C2D7E4D9 | C7D5F0F0 | F0F6F0F0 | *70NDBPURGN000600* | 11097D72 |
| 004CC2 | F0F0F040 | 40404040 | 40404040 | 40404040 | *000 * | 11097D82 |
| 004CD2 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097D92 |
| 004CE2 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097DA2 |
| 004CF2 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097DB2 |
| 004D02 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097DC2 |
| 004D12 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097DD2 |
| 004D22 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097DE2 |
| 004D32 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097DF2 |
| 004D42 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097E02 |
| 004D52 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097E12 |
| 004D62 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097E22 |
| 004D72 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097E32 |
| 004D82 | 40404040 | 40404040 | 40404040 | 40404040 | * * | 11097E42 |
| 004D92 | 40404040 | D4C1F86B | F0D4C1F4 | F06BF0D4 | * MA8,0MA40,0M* | 11097E52 |
| 004DA2 | C2F26BF0 | D4C2F26B | F0D4C9F2 | 6BF0D4C9 | *I2,0MI2,0MI2,0MI* | 11097E62 |
| 004DB2 | F26BF04B | | | | *2,0. * | 11097E72 |