Multiple Row Processing

This document covers the multiple row functionality for DB2 databases.

You have to operate against DB2 for z/OS Version 8 or higher to use these features.

Natural for DB2 provides two kinds of multiple row processing features:

- Standard multiple row processing
- This feature does not influence the program logic. Although the Natural native DML and Natural SQL DML provide clauses for specification of the multi-fetch-factor, the Natural program operates with one database row and from the program point of view only one row is received from or is send to the database.
- Advanced multiple row processing
- This feature has a lot of impact on the program logic, as it allows the retrieval of multiple rows from the database into the program storage by a single Natural SQL SELECT statement into a set of arrays. Additionally it is possible to insert multiple rows into the database from a set of arrays by the Natural SQL INSERT statement.

The following topics are covered:

Purpose of Multi-Fetch Feature

In standard mode, Natural does not read multiple records with a single database call; it always operates in a one-record-per-fetch mode. This kind of operation is solid and stable, but can take some time if a large number of database records are being processed.

To improve the performance of those programs, you can use the Multi-Fetch Clause in the FIND, READ or HISTOGRAM statements. This allows you to specify the number of records read per database access.

```
{ FIND      } [ MULTI-FETCH { ON                    } ]
{ READ      } [             { OFF                   } ]
{ HISTOGRAM } [             { OF multi-fetch-factor } ]
```

Where the *multi-fetch-factor* is either a constant or a variable with a format integer (I4).

To improve the performance of the Natural SQL SELECT statements, you can use the With_Rowset_Positioning Clause to specify a multi-fetch-factor.

$$\left[\ \texttt{WITH ROWSET POSITIONING FOR} \left\{\begin{array}{l}[:]\ \mathit{row\_hv}\\ \mathit{integer}\end{array}\right\} \texttt{ROWS}\ \right]$$

At statement execution time, the runtime checks if a *multi-fetch-factor* greater than 1 is supplied for the database statement.

If the *multi-fetch-factor* is

less than or equal to 1	the database call is continued in the usual one-record-per-access mode.
greater than 1	the database call is prepared dynamically to read multiple records (e.g. 10) with a single database access into an auxiliary buffer (multi-fetch buffer). If successful, the first record is transferred into the underlying data view. Upon the execution of the next loop, the data view is filled directly from the multi-fetch buffer, without database access. After all records are fetched from the multi-fetch buffer, the next loop results in the next record set being read from the database. If the database loop is terminated (either by end-of-records, ESCAPE, STOP, etc.), the content of the multi-fetch buffer is released.

Considerations for Multi-Fetch Usage

- The program does not receive "fresh" records from the database for every loop, but operates with images retrieved at the most recent multi-fetch access.
- If a dynamic direction change (IN DYNAMIC...SEQUENCE) is coded for a READ / HISTOGRAM statement, the multi-fetch feature is not possible and leads to a corresponding syntax error at compilation.
- The size occupied by a database loop in the multi-fetch buffer is determined according to the rule:

header + sqldaheader + columns*(sqlvar+lise) + mf*(udind + sum(collen) + sum(LF(columns) + sum(nullind))

=

32 + 16 + columns*(44+12) + mf*(1 + sum(collen) + sum(LF(column)) + sum(2))

where

- header denotes the length of the header of a entry in the DB2 multifetch buffer, i.e. 32
- sqldaheader denotes the length of the header of a sqlda, i.e. 16
- columns denotes the number of receiving fields of a SQL request

- sqlvar denotes the length of a sqlvar, i.e. 44
- lise denotes the length of a NDB specific sqlvar extension
- mf denotes the multifetch factor, i.e. the number of rows fetched by one database call
- collen denotes the length of the receiving field
- LF(column) denotes the size of the length field of the receiving field, i.e. 0 for fixed length fields, 2 for variable length fields, and 4 for large object columns (LOBs)
- nullind denotes the length of a null indicator, i.e. 2

Size of the Multi-Fetch Buffer

The multifetch buffer is released at terminal i/o in pseudo conversional mode. Therefore there is no size limitation for the DB2 multifetch buffer (DB2SIZE6). The buffer will be automatical enlarged if necessary.

Support of TEST DBLOG Q

When multi-fetch is used, real database calls are only submitted to get a new set of records.

The TEST DBLOG Q facility is also called from the NDB multi fetch handler for every rowset fetch from DB2 and for every record moved from the multi fetch buffer to the program storage. The events are distinguished by the literal "MULTI FETCH .." and "<BUFF FETCH ..."

Example: TEST DBLOG List Break-Out

```
10:51:57             ***** NATURAL Test Utilities *****            2006-01-27
User HGK                        - DBLOG Trace -                 Library NDB42
M No   R SQL Statement (truncated)     CU SN SREF M Typ SQLC/W Program  Line LV
_    1   SELECT EMPNO,FIRSTNME,LASTNAM 01 01 0260 D DB2        MF000001 0260 01
_    2     MULTI FETCH  NEX            01 01 0260 D DB2        MF000001 0260 01
_    3     <BUFF FETCH  NEX            00 00 0260 D DB2        MF000001 0260 01
_    4     <BUFF FETCH  NEX            00 00 0260 D DB2        MF000001 0260 01
_    5     <BUFF FETCH  NEX            00 00 0260 D DB2        MF000001 0260 01
_    6     <BUFF FETCH  NEX            00 00 0260 D DB2        MF000001 0260 01
_    7     <BUFF FETCH  NEX            00 00 0260 D DB2        MF000001 0260 01
_    8     <BUFF FETCH  NEX            00 00 0260 D DB2        MF000001 0260 01
_    9     <BUFF FETCH  NEX            00 00 0260 D DB2        MF000001 0260 01
_   10     <BUFF FETCH  NEX            00 00 0260 D DB2        MF000001 0260 01
_   11     <BUFF FETCH  NEX            00 00 0260 D DB2        MF000001 0260 01
_   12     <BUFF FETCH  NEX            00 00 0260 D DB2        MF000001 0260 01
_   13     <BUFF FETCH  NEX            00 00 0260 D DB2        MF000001 0260 01
_   14     <BUFF FETCH  NEX            00 00 0260 D DB2        MF000001 0260 01
_   15     <BUFF FETCH  NEX            00 00 0260 D DB2        MF000001 0260 01
_   16     <BUFF FETCH  NEX            00 00 0260 D DB2        MF000001 0260 01
_   17     <BUFF FETCH  NEX            00 00 0260 D DB2        MF000001 0260 01
Command ===>
```

where column No represents the following:

1	is a open cursor DB2 call.
2	is a "real" database call that reads a set of records via multi-fetch (see " `MULTI FETCH NEX` " in column SQL Statement).
3-17	are "no real" database calls, but only entries that document that the program has received these records from the multi-fetch buffer (see " `<BUFF FETCH NEX` " in column SQL Statement).

Multiple rows to program (Advanced)

The feature allows programs to retrieve multiple rows from DB2 into arrays.

This feature is only available with the SELECT statement.

Prerequisites

In order to use this feature you have to

- set the compiler option DB2ARRY=ON (either via OPTIONS statement or `COMPOPT` command or CMPO profile parameter).
- specify a list of receiving arrays in the INTO Clause of the SELECTstatement.
- specify the number of rows to be retrieved from the database by a single FETCH operation via the WITH ROWSET POSITIONING Clause.
- Specify a variable receiving the number of rows retrieved from the database via the ROWS_RETURNED Clause.

DB2ARRY=ON

DB2ARRY=ON is necessary to allow the specification of arrays in the INTO Clause. DB2ARRY=ON also prevents the usage of arrays as sending or receiving fields for DB2 CHAR/VARCHAR /GRAPHIC/VARGRAPHIC columns. Instead Natural scalar fields with the appropriate length have to be used.

INTO Clause

Each array specified in the INTO Clause has to be contiguous (one occurrence following immediately by another, this is expected by DB2) and has to be one-dimensional. The arrays are filled from the first occurrence (low) to last occurrence (high). The first array occurrences compose the first row of the received rowset, the second array occurrences compose the second row of the received rowset. The array occurrences of the nth index compose the nth row returned from DB2. If a LINDICATOR or INDICATOR Clauses are used in the INTO Clause for arrays, the specified length indicators or null indicators have also to be arrays. The number of occurrences of LINDICATOR and INDICATOR arrays have to equal or greater than the number of occurrences of the master array.

WITH ROWSET POSITIONING Clause

The WITH_ROWSET_POSITIONING Clause is used to specify the number of rows to be retrieved from the database by one processing cycle. The specified number has to be equal or smaller than the minimum of occurrences of all specified arrays. If a variable, not a constant, is specified the actual content of the variable will be used during each processing cycle. The specified number has to be greater 0 and smaller than 32768.

ROWS_RETURNED Clause

The ROWS_RETURNED Clause is used to specify a variable, which will contain the number of rows read from the database during the actual fetch operation. The format of the variable has to be I4.

Restrictions and Constraints

Natural Views

It is not possible to use Natural arrays of views in the INTO clause, i.e. the use of keyword VIEW is not possible.

File Server usage and positioned UPDATE and DELETE

The purpose of this feature is to reduce the number of database and database interface calls for bulk batch processing. Therefore it is not recommended to use this kind of programming in online CICS or IMS environments, when terminal I/Os occur within open cursor loops, i.e. the file server is used. A fortiori it is not possible to perform a positioned UPDATE or DELETE statement after terminal I/O.

Example:

```
DEFINE DATA LOCAL
01 NAME            (A20/1:10)
01 ADDRESS         (A100/1:10)
01 DATEOFBIRTH     (A10/1:10)
01 SALARY          (P4.2/1:10)
01 L$ADDRESS       (I2/1:10)
01 ROWS            (I4)
01 NUMBER          (I4)
01 INDEX           (I4)
END-DEFINE
OPTIONS DB2ARRY=ON
ASSIGN NUMBER := 10
SEL.
SELECT NAME, ADDRESS , DATEOFBIRTH, SALARY
       INTO  :NAME(*),                             /* <-- ARRAY
             :ADDRESS(*) LINDICATOR :L$ADDRESS(*), /* <-- ARRAY
             :DATEOFBIRTH(1:10),                   /* <-- ARRAY
             :SALARY(01:10)                        /* <-- ARRAY
      FROM NAT-DEMO
      WHERE NAME > ' '
      WITH ROWSET POSITIONING FOR :NUMBER ROWS     /* <-- ROWS REQ
      ROWS_RETURNED :ROWS                          /* <-- ROWS RET
  IF ROWS > 0
    FOR INDEX = 1 TO  ROWS STEP 1
      DISPLAY
              INDEX (EM=99) *COUNTER (SEL.) (EM=99) ROWS (EM=99)
              NAME(INDEX)
              ADDRESS(INDEX) (AL=20)
              DATEOFBIRTH(INDEX)
              SALARY(INDEX)
    END-FOR
  END-IF
END-SELECT
END
```

Multiple rows from program (Advanced)

The feature allows programs to insert multiple rows into a DB2 table from arrays.

This feature is only available with the INSERT statement.

Prerequisites

In order to use this feature you have to

- set the compiler option DB2ARRY=ON (either via OPTIONS statement or `COMPOPT` command or CMPO profile parameter).
- specify a list of sending arrays in the VALUES Clause of the INSERT statement.
- specify the number of rows to be inserted into the database by a single INSERT statement via the FOR n ROWS Clause.

DB2ARRY=ON

DB2ARRY=ON is necessary to allow the specification of arrays in the VALUES Clause. DB2ARRY=ON also prevents the usage of arrays as sending or receiving fields for DB2 CHAR/VARCHAR /GRAPHIC/VARGRAPHIC columns. Instead Natural scalar fields with the appropriate length have to be used.

VALUES Clause

Each array specified in the VALUES Clause has to be contiguous (one occurrence following immediately by another, this is expected by DB2) and has to be one-dimensional. The arrays are read from the first occurrence (low) to last occurrence (high). The first array occurrences compose the first row inserted into the database, the second array occurrences compose the second row inserted into the database. The array occurrences of the nth index compose the nth row inserted into the database. If a LINDICATOR or INDICATOR Clauses are used in the VALUES Clause for arrays, the specified length indicators or null indicators have also to be arrays. The number of LINDICATOR and INDICATOR array occurrences has to be equal or greater than the number of occurrences of the master array.

FOR n ROWS Clause

The FOR n ROWS Clause is used to specify how many rows are to be inserted into the database table by one INSERT statement. The specified number has to be equal or smaller than the minimum of occurrences of all specified arrays in the VALUES clause. The specified number has to be greater 0 and smaller than 32768.

Restrictions and Constraints

Natural Views

It is not possible to use Natural arrays of views in the VALUES clause, i.e. the use of keyword VIEW is not possible.

Static execution

Due to DB2 restrictions it is not possible to execute multiple row inserts in static mode. Therefore multiple row inserts are not generated static and are always dynamically prepared and executed by Natural for DB2.

It is not possible to use Natural arrays of views in the INTO clause, i.e. the use of keyword VIEW is not possible.

Example:

```
DEFINE DATA LOCAL
01 NAME        (A20/1:10)  INIT <'ZILLER1','ZILLER2','ZILLER3','ZILLER4'
                                ,'ZILLER5','ZILLER6','ZILLER7','ZILLER8'
                                ,'ZILLER9','ZILLERA'>
01 ADDRESS     (A100/1:10) INIT <'ANGEL STREET 1','ANGEL STREET 2'
                                ,'ANGEL STREET 3','ANGEL STREET 4'
                                ,'ANGEL STREET 5','ANGEL STREET 6'
                                ,'ANGEL STREET 7','ANGEL STREET 8'
                                ,'ANGEL STREET 9','ANGEL STREET 10'>
01 DATENATD (D/1:10)  INIT <D'1954-03-27',D'1954-03-27',D'1954-03-27'
                            ,D'1954-03-27',D'1954-03-27',D'1954-03-27'
                            ,D'1954-03-27',D'1954-03-27',D'1954-03-27'
                            ,D'1954-03-27'>
01 SALARY      (P4.2/1:10) INIT <1000,2000,3000,4000,5000
                                ,6000,7000,8000,9000,9999>
01 SALARY_N    (N4.2/1:10) INIT <1000,2000,3000,4000,5000
                                ,6000,7000,8000,9000,9999>
01 L§ADDRESS   (I2/1:10) INIT <14,14,14,14,14,14,14,14,14,15>
01 N§ADDRESS   (I2/1:10) INIT <00,00,00,00,00,00,00,00,00,00>
01 ROWS        (I4)
01 INDEX       (I4)
01 V1 VIEW OF NAT-DEMO_ID
02 NAME
02 ADDRESS     (EM=X(20))
02 DATEOFBIRTH
02 SALARY
01 ROWCOUNT  (I4)
END-DEFINE
OPTIONS DB2ARRY=ON                  /* <-- ENABLE DB2 ARRAY
ROWCOUNT := 10
INSERT INTO NAT-DEMO_ID
       (NAME,ADDRESS,DATEOFBIRTH,SALARY)
       VALUES
       (:NAME(*),                   /* <-- ARRAY
        :ADDRESS(*)                 /* <-- ARRAY
        INDICATOR :N§ADDRESS(*)     /* <-- ARRAY
        LINDICATOR :L§ADDRESS(*),   /* <-- ARRAY DB2 VCHAR
        :DATENATD(1:10),            /* <-- ARRAY NATURAL DATES
        :SALARY_N(01:10)            /* <-- ARRAY NATURAL NUMERIC
       )
       FOR :ROWCOUNT ROWS
SELECT * INTO VIEW V1 FROM NAT-DEMO_ID WHERE NAME > 'Z'
DISPLAY V1                          /* <-- VERIFY INSERT
END-SELECT
BACKOUT
END
```