To multi fetch or to code for performance in general

Kurt Struyf
Competence Partners

Why Multi Row fetch ?

- Technique to save up to 60% of DB2 cpu
- Easy to use
- Changes application logic
- Use arrays in application programs
  - FETCH into the array
  - INSERT, DELETE and UPDATE from the array
- Program can control size of rowset
- Rowset positioning used to position in fetch

In detail

Multi-Row in detail

- Introduction
  - What is a Rowset?
  - What are Host Variable Arrays
- Statements
  - Multi-Row FETCH statement
  - Positioned UPDATE & DELETE statement
  - Multi-Row INSERT statement
- Logic in application program
  - Differences in error handling
  - Treatment of End of Multi-Row FETCH

What is a rowset ?

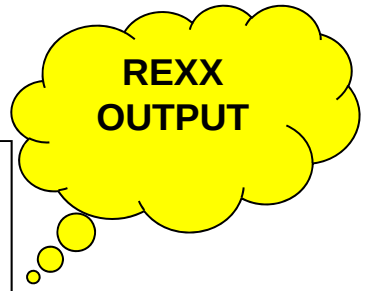
- A rowset is a group of rows for the result set of a query that are returned by a single FETCH

- You’re program can control the size of the rowset
- Maximum rowset size is 32.767 rows
- Adjust “commit counter” to rowset size
- Company default of more then 20 rows
- DCLGEN not supported, own “rexx” to adapt
  - maximum used in DCLGEN generated copies e.g.: 100 rows
    beware of to compiler limits e.g. COBOL
    - elementary item : max. 16Mb
    - complete working storage : max 128

Host Variable Arrays

- Foresee arrays in your program to work with multi-row SQL operations
  - A host variable array is an array in which each element of the array contains a value for the same column

- Work with dynamic arrays, for example:




```
01  NB-rowsetsize              PIC S9(09) COMP-5.
01  NS-rowsetindex             PIC S9(09) COMP-5.
01  WS-A-ROWSET.
    03  NM-NAME
        OCCURS 100 TIMES.
        49  NM-NAME-LEN        PIC S9(04) COMP.
        49  NM-NAME-TEXT       PIC X(50).
    03  NB-NUMBER              PIC S9(09) COMP-4
        OCCURS 100 TIMES.
```

instead of

```
01  WS-A-ROW.
    03  NM-NAME.
        49  NM-NAME-LEN        PIC S9(04) COMP.
        49  NM-NAME-TEXT       PIC X(50).
    03  NB-NUMBER              PIC S9(09) COMP-4.
```

Multi-Row FETCH (1)

- This is a single FETCH statement that can retrieve multiple rows of data as a rowset

- Cursor declaration needs to change:

```
EXEC SQL
   DECLARE cursor-name CURSOR FOR
   SELECT column1
         ,column2 FROM table-name;
END-EXEC
```

BECOMES

```
EXEC SQL
   DECLARE cursor-name CURSOR WITH ROWSET POSITIONING FOR
   SELECT column1
         ,column2 FROM table-name;
END-EXEC
```

Then you can FETCH multiple rows at-a-time from the cursor

Multi-Row FETCH (2)

On the FETCH statement

- the amount of rows requested can be specified
- the FETCH direction can be specified (first, NEXT, prior, last)

```
EXEC SQL
   FETCH cursor-name
   INTO …
END-EXEC
```

becomes

```
EXEC SQL
   FETCH NEXT ROWSET FROM cursor-name
   FOR :rowset-size ROWS
   INTO …
END-EXEC
```

- The rowset size can be defined as a constant or a variable

```
01  rowset-size PIC S9(09) COMP-5.
```

Positioned update & delete

The new syntax allows you to specify the row to be updated or deleted in the rowset.

For example:

```
EXEC SQL
  UPDATE table-name
     SET MY_NAME = :xxx
     WHERE CURRENT OF cursor-name
     FOR ROW :row-number OF ROWSET
END-EXEC
```

```
EXEC SQL
  DELETE FROM table-name
  WHERE CURRENT OF cursor-name
  FOR ROW :row-number OF ROWSET
END-EXEC
```

BE CAREFUL with

```
EXEC SQL
  UPDATE table-name
     SET MY_NAME = :xxx
     WHERE CURRENT OF cursor-name
END-EXEC
```

```
EXEC SQL
  DELETE FROM table-name
  WHERE CURRENT OF cursor-name
END-EXEC
```

Multi-Row INSERT (1)

A single INSERT statement can add multiple rows of data from an array.

for example:

```
EXEC SQL
  INSERT INTO table-name
         ( column
          ,column )
  VALUES ( :hostvar-array
          ,:hostvar-array )
  FOR :rowset-size ROWS
 [ATOMIC option]
END-EXEC
```

instead of

```
EXEC SQL
  INSERT INTO table-name
         ( column
          ,column )
  VALUES ( :hostvar
          ,:hostvar )
END-EXEC
```

Multi-Row INSERT (2)

[ATOMIC Option]

- **ATOMIC**
  If a single row insert fails, all fail.

- **NOT ATOMIC CONTINUE ON SQLEXCEPTION**
  If errors occur during execution of INSERT, processing continues.
  Diagnostics are available for each failed row.

Error Handling – GET DIAGNOSTICS (1)

GET DIAGNOSTICS statement

- Returns SQL error and diagnostic information and can be performed for
  - The entire statement
  - Each condition (when multiple conditions occur)

- Enable more diagnostic information to be returned than can be contained in SQLCA (no CALL DSNTIAR needed)
  - SQL error message tokens larger that 70 bytes are supported whereas the SQLCA cannot

Error Handling – GET DIAGNOSTICS (2)

Simply returning the SQLCODE may no longer be enough.

After detecting an error you might need to retrieve error information for multiple errors that may have occurred.

```
IF SQLCODE NOT = 0
   EXEC SQL
     GET DIAGNOSTICS
       :amount-errors   = NUMBER
       :row-count       = ROW_COUNT
   END-EXEC
END-IF
```

- **ROW_COUNT** : after a DELETE, INSERT, UPDATE, FETCH this item contains the number of rows deleted, inserted, updated or fetched.

- **NUMBER** : contains the amount of conditions (errors) of the last SQL statement. (Only for INSERT statement > 1)

Error Handling – GET DIAGNOSTICS (3)

Looping to get (and treat) all the error information:

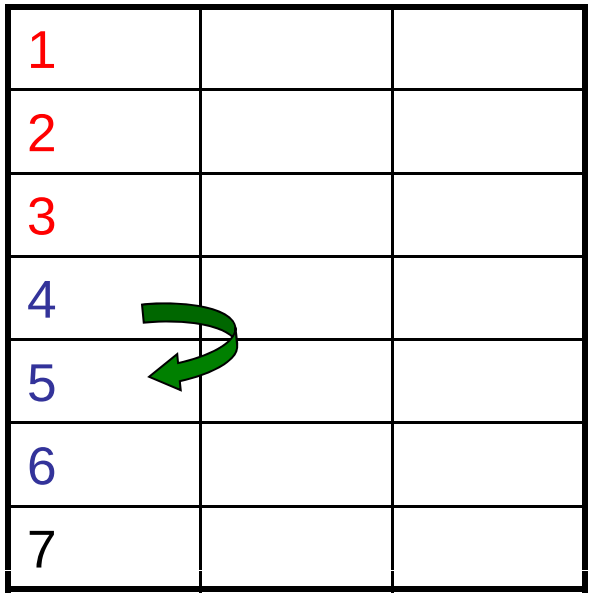
```
MOVE 0 TO index
PERFORM amount-errors TIMES
  ADD 1 TO index
  EXEC SQL
    GET DIAGNOSTICS CONDITION :index
        :xx = MESSAGE_TEXT
        :xx = DB2_MESSAGE_ID
        :xx = DB2_REASON_CODE
        :xx = DB2_RETURNED_SQLCODE
        :xx = DB2_ROW_NUMBER
  END-EXEC
  ... own error handling (e.g. reporting)
END-PERFORM
```

- `MESSAGE_TEXT` : message associated with the error
- `DB2_MESSAGE_ID` : id associated with error message
- `DB2_REASON_CODE` : reason code for error (extended info)
- `DB2_RETURNED_SQLCODE` : SQLCODE of the condition
- `DB2_ROW_NUMBER` : the row number on which DB2 detected the error

Related error codes

-246	STATEMENT USING CURSOR name SPECIFIED NUMBER OF ROWS num-rows WHICH IS NOT VALID WITH dimension
-248	A POSITIONED DELETE OR UPDATE FOR CURSOR name SPECIFIED ROW n OF ROWSET, BUT THE ROW IS NOT CONTAINED WITHIN THE CURRENT ROWSET
-253	A NON-ATOMIC statement STATEMENT SUCCESSFULLY COMPLETED FOR SOME OF THE REQUESTED ROWS, POSSIBLY WITH WARNINGS, AND ONE OR MORE ERRORS
-254	A NON-ATOMIC statement STATEMENT ATTEMPTED TO PROCESS MULTIPLE ROWS OF DATA, BUT ERRORS OCCURRED
-589	A POSITIONED DELETE OR UPDATE STATEMENT FOR CURSOR name SPECIFIED A ROW OF A ROWSET, BUT THE CURSOR IS NOT POSITIONED ON A ROWSET
-20185	CURSOR name IS NOT DEFINED TO ACCESS ROWSETS, BUT A CLAUSE WAS SPECIFIED THAT IS VALID ONLY WITH ROWSET ACCESS

Positioning and last fetch



FETCH **NEXT ROWSET FROM** cursor-name
FOR :rowset-size ROWS ➔ **rowset-size = 3**

FIRST FETCH NEXT ROWSET FROM cursor-name FOR :rowset-size ROWS

SECOND FETCH NEXT ROWSET FROM cursor-name
FOR :rowset-size ROWS

Be careful
FETCH FROM cursor-name **into…**

THIRD FETCH NEXT ROWSET FROM cursor-name
FOR :rowset-size ROWS
➔**Incomplete a.k.a SQLCODE +100**

```
IF SQLCODE = 100 THEN
   EXEC SQL
            GET DIAGNOSTICS
            :row-count = ROW_COUNT
   END-EXEC
            MOVE row-count TO rowset-size
END-IF
```

Multi row Insert – Update - Delete

INSERT :

First fill the rowset (via loop) and then INSERT complete rowset.

At the end of the program: do last INSERT (incomplete rowset).

UPDATE and DELETE :

Treat every row of the fetched rowset one by one and decide which one to update or delete (via loop).

Multi Row Fetch our measurements

- Performance results may differ:
  - < 5 rows : poor performance (worse than before)
  - 10 – 100 rows : best performance
  - > 100 rows : no improvement anymore
- Following data is based upon treatment of 1 million rows (in seconds CPU).

	Via row	Via rowset	Gain on DB2 in CPU seconds
FETCH	16	6	10 (-60%)
FETCH + UPDATE via row	76	66	10 (-15%)
FETCH + UPDATE via rowset	76	60	16 (-35%)

Provide skeletons- select

```
...
 PFETCH-KP01NSC1.
*---------------*
     EXEC SQL FETCH NEXT ROWSET FROM KP01NSC1
         FOR :NB-KP01NSTR-MAX ROWS
         INTO  :DCLKP01NSTR.NM-ID-NR
              ,:DCLKP01NSTR.NS-LAST-NR
     END-EXEC.

 ROWSET-KP01NSC1.
*---------------*
     PERFORM PFETCH-KP01NSC1
     EVALUATE SQLCODE
       WHEN 0
            CONTINUE
       WHEN 100
            SET EOF-KP01NSC1 TO TRUE
            PERFORM SQLDIAGNOSTICS
*           set size incomplete rowset
            MOVE NB-DIAG-ROWS TO NB-KP01NSTR-MAX
       WHEN OTHER
            PERFORM SQLERRORDISP
     END-EVALUATE
*    treat now all rows of the rowset
     MOVE 0 TO NS-KP01NSTR
     PERFORM NB-KP01NSTR-MAX TIMES
       ADD 1 TO NS-KP01NSTR
       ////-->
```

Provide skeletons-update

```
EXEC SQL  UPDATE KP01NSTV
    SET NM_ID_NR            = :DCLKP01NSTV.NM-ID-NR
  WHERE CURRENT OF KP01NSC1
    FOR ROW :NS-KP01NSTR OF ROWSET
END-EXEC

EVALUATE  SQLCODE
 WHEN     0       CONTINUE
 WHEN   100   ???????????
 WHEN OTHER       PERFORM SQLERRORDISP
END-EVALUATE
```

Provide skeletons-delete

```
EXEC SQL  DELETE FROM KP01NSTV
  WHERE CURRENT OF KP01NSC1
    FOR ROW :NS-KP01NSTR OF ROWSET
END-EXEC

EVALUATE  SQLCODE
 WHEN     0       CONTINUE
 WHEN   100       CONTINUE
 WHEN OTHER       PERFORM SQLERRORDISP
END-EVALUATE
```

Provide skeletons-insert

```
   …
      ADD 1 TO NS-KP01NSTR
     MOVE     NM-ID-NR
           TO NM-ID-NR            OF DCLKP01NSTR (NS-KP01NSTR)
     MOVE     NS-LAST-NR
           TO NS-LAST-NR          OF DCLKP01NSTR (NS-KP01NSTR)

     IF NS-KP01NSTR = NB-KP01NSTR-MAX
*       rowset full, do now insert
        PERFORM INSERT-ROWSET-KP01NSTR
     END-IF
     .

 INSERT-ROWSET-KP01NSTR.
     MOVE 0 TO NS-KP01NSTR
     EXEC SQL
      INSERT  INTO KP01NSTV
              (NM_ID_NR
              ,NS_LAST_NR)
      VALUES (:DCLKP01NSTR.NM-ID-NR
             ,:DCLKP01NSTR.NS-LAST-NR )
      FOR :NB-KP01NSTR-MAX ROWS
      NOT ATOMIC CONTINUE ON SQLEXCEPTION
    END-EXEC
```

```
EVALUATE  SQLCODE
      WHEN    0
       CONTINUE
      WHEN -253
*      some inserts have failed
       PERFORM SQLDIAGNOSTICS
       MOVE 0 TO NB-DIAG-COND
       PERFORM DIAG-AMOUNT-ERRORS TIMES
         ADD 1 TO NB-DIAG-COND
         EXEC SQL
          GET DIAGNOSTICS CONDITION NB-
DIAG-COND
            :TE-DIAG-MESSAGE =
MESSAGE_TEXT
           ,:CO-DIAG-MESSAGE  =
DB2_MESSAGE_ID
        …
```

Provide skeletons-error handling

```
Get diagnostic information of last executed SQL statement
 SQLDIAGNOSTICS.
     EXEC SQL
       GET DIAGNOSTICS
         :NB-DIAG-ERRORS   = NUMBER
        ,:NB-DIAG-ROWS     = ROW_COUNT
     END-EXEC
     .
PERFORM SQLDIAGNOSTICS
     MOVE 0                TO NB-DIAG-COND
     PERFORM NB-DIAG-ERRORS TIMES
       ADD 1 TO NB-DIAG-COND
       EXEC SQL
         GET DIAGNOSTICS CONDITION :NB-DIAG-COND
             :TE-DIAG-MESSAGE       = MESSAGE_TEXT
            ,:CO-DIAG-MESSAGE       = DB2_MESSAGE_ID
            ,:CO-DIAG-REASON        = DB2_REASON_CODE
            ,:CO-DIAG-SQLCODE       = DB2_RETURNED_SQLCODE
           …
```

Provide skeletons-error handling

```
*
* WORK FIELDS FOR SQL DIAGNOSTICS
*
 01  WS-DIAG-SQL.
     03  NB-DIAG-ERRORS           PIC S9(09) COMP.
     03  NB-DIAG-ROWS             PIC S9(18) COMP-3.
 01  WS-DIAG-COND.
     03  NB-DIAG-COND             PIC S9(09) COMP.
     03  TE-DIAG-MESSAGE.
         49  TE-DIAG-MESSAGE-LEN  PIC S9(04) COMP.
         49  TE-DIAG-MESSAGE-TEXT PIC X(150).
     03  CO-DIAG-MESSAGE          PIC X(10).
     03  CO-DIAG-REASON           PIC S9(09) COMP.
     03  CO-DIAG-SQLCODE          PIC S9(09) COMP.
     03  NS-DIAG-ROW              PIC S9(18) COMP-3.
 01  WS-DIAG-DISP-COND.
     03  CO-DIAG-DISP-REASON      PIC +9(09).
     03  CO-DIAG-DISP-SQLCODE     PIC +9(05).
     03  NS-DIAG-DISP-ROW         PIC 9(18).
```

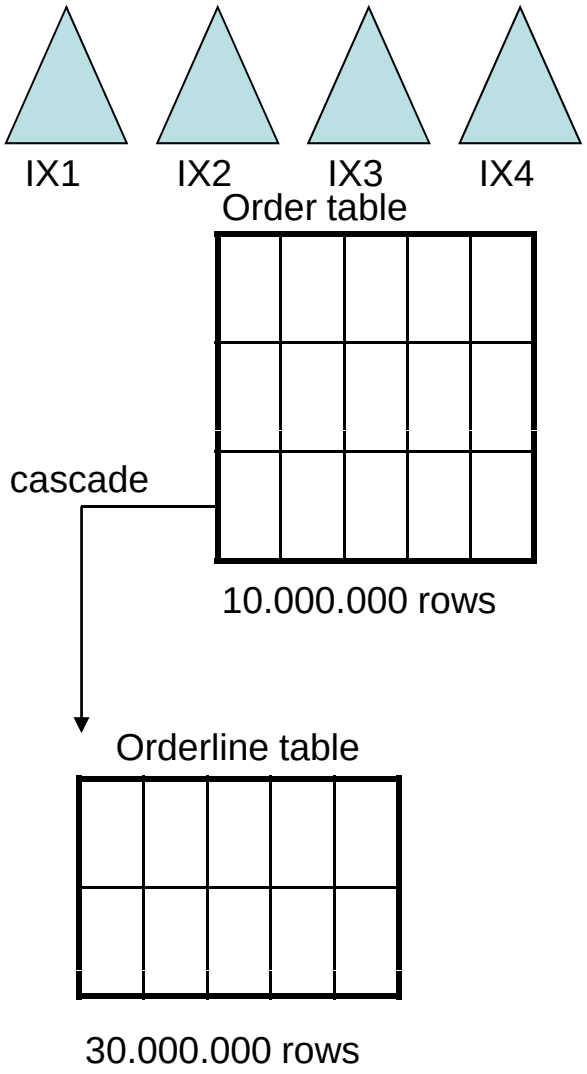
Multi Row fetch reminder

- Do not use single and multiple row fetch for the same cursor in one program
- Be aware of compiler limits
  - elementary item : max. 16Mb
  - complete working storage : max 128 Mb
- Last FETCH on a rowset can be ‘incomplete’
- May not be fastest solution e.g mass delete

Mass Delete

```
DELETE
    FROM ORDER
    WHERE ORDERDATE  <  :HV
```

100,000 ORDER rows are deleted



Mass Delete Solution 1

Cursor

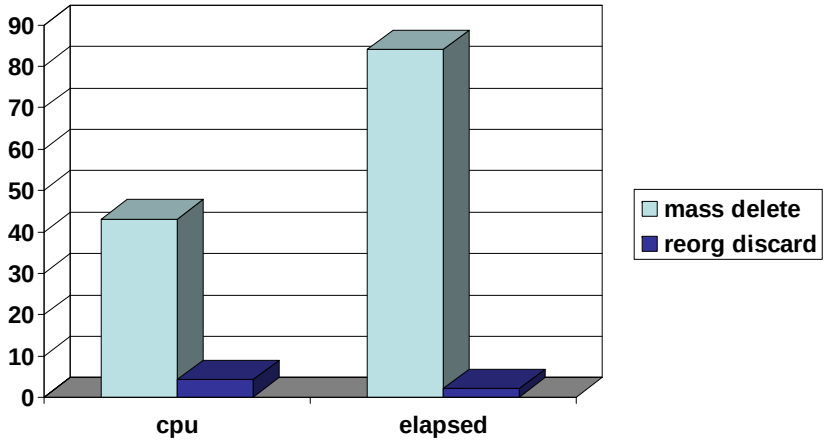
```
DECLARE ORDER CURSOR WITH  HOLD FOR
   SELECT . . .
      FROM ORDER
      WHERE ORDERDATE < :HV
      FOR UPDATE

OPEN ORDER

do until eof
   FETCH ORDER
   DELETE
      FROM ORDER
      WHERE CURRENT OF ORDER
   COMMIT
end

CLOSE ORDER
```

- Be ware of commit overhead
- Consider multi row delete (V8 and up)
- Consider reorg discard (V8 and up)

Mass Delete Solution 2

Reorg discard

```
REORG TABLESPACE db.ORDER_TS
LOG NO COPYDDN(SYSCOPY)
SHRLEVEL CHANGE
DISCARD FROM TABLE ORDER_TB
WHEN ORDERDATE < CURRENT DATE- 5 YEAR
```



- Extremely fast
  - CPU only 10%
  - Elapsed only 5%
- More indexes, more interesting it becomes

Questions ?

Kurt.struyf@cp.be