SWG BetaWorks

DB2 9 for z/OS Technical Education Series

"DB2 V8 Multi-Row – How to put it into Dynamic SQL in COBOL"

Paul Fletcher (Fletchpl@uk.ibm.com)

Important Disclaimer

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.

WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.

IN ADDITION, THIS INFORMATION IS BASED ON IBM’S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE.

IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.

NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:

- CREATING ANY WARRANTY OR REPRESENTATION FROM IBM (OR ITS AFFILIATES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS); OR
- ALTERING THE TERMS AND CONDITIONS OF THE APPLICABLE LICENSE AGREEMENT GOVERNING THE USE OF IBM SOFTWARE.

Agenda

- **Take sample Dynamic SQL programs DSN8BCU1 & 2**
- **How the code works**
- **What Multi-Row Fetch is**
- **How to change the code to implement Multi-Row Fetch**

Single Row Fetch in Dynamic SQL – DSN8BCU1

- **Sample program Documented in Appendix D of Application Programming And SQL Guide**
- **Input is table name**
- **Program adds SELECT * before table name and Executes the Statement**
- **SQLDA used to map storage to put column data into instead of FETCH INTO host variables**
- **Output is similar to output from DSNTIAUL**

SQLDA Contents

- **Column Name – provided by DB2**
- **Column Length – provided by DB2**
- **Column Type – provided by DB2**
- **Pointer to storage for DB2 to put the data into – provided by the program**
- **Pointer to storage for DB2 to put the indicator variable into – provided by the program**

Using Pointers in COBOL

- **Pointers and based variables in the sample COBOL program**
  - **COBOL has a POINTER type and a SET statement that provide pointers and based variables. The SET statement sets a pointer from the address of an area in the linkage section or another pointer; the statement can also set the address of an area in the linkage section.**

Storage allocation for the sample COBOL program

- **COBOL does not provide a means to allocate main storage within a program and know the address of that storage. You can achieve the same end by having an initial program which allocates the storage, and then calls a second program that manipulates the pointer. (COBOL does not permit you to directly manipulate the pointer because errors and abends are likely to occur.)**
- **The initial program is extremely simple. It includes a working storage section that allocates the maximum amount of storage needed. This program then calls the second program, passing the area or areas on the CALL statement.**
- **The second program defines the area in the linkage section and can then use pointers within the area. If you need to allocate parts of storage, the best method is to use indexes or subscripts. You can use subscripts for arithmetic and comparison operations.**

Program DSN8BCU1

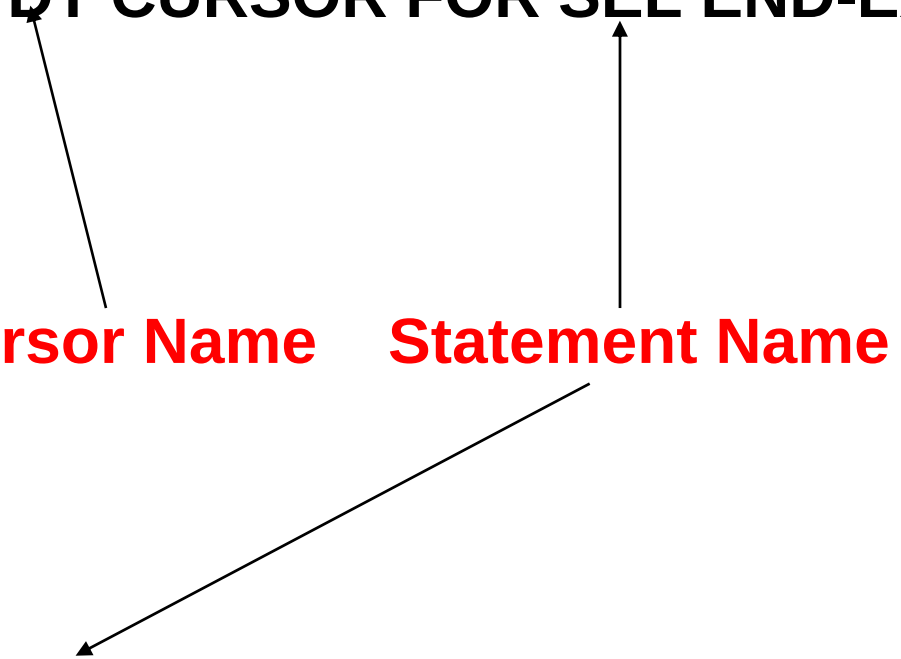
```
/
 IDENTIFICATION DIVISION.
*-----------------------
 PROGRAM-ID.    DSN8BCU1
*
 ENVIRONMENT DIVISION.
*
 CONFIGURATION SECTION.
 DATA DIVISION.
*
 WORKING-STORAGE SECTION.
*
 01  WORKAREA-IND.
         02  WORKIND PIC S9(4) COMP OCCURS 750 TIMES.
 01  RECWORK.
         02  RECWORK-LEN PIC S9(8) COMP VALUE 32700.
         02  RECWORK-CHAR PIC X(1) OCCURS 32700 TIMES.
*
 PROCEDURE DIVISION.
*
           CALL 'DSN8BCU2' USING WORKAREA-IND RECWORK.
           GOBACK.
```

DSN8BCU2 – Program Logic

```
*PSEUDOCODE*
  PROCEDURE
  EXEC SQL DECLARE DT CURSOR FOR SEL END-EXEC.
  EXEC SQL DECLARE SEL STATEMENT END-EXEC.
  INITIALIZE THE DATA, OPEN FILES.
  OBTAIN STORAGE FOR THE SQLDA AND THE DATA RECORDS.
  READ A TABLE NAME.
  OPEN SYSREC01.
  BUILD THE SQL STATEMENT TO BE EXECUTED
  EXEC SQL PREPARE SQL STATEMENT INTO SQLDA END-EXEC.
  SET UP ADDRESSES IN THE SQLDA FOR DATA.
  INITIALIZE DATA RECORD COUNTER TO 0.
  EXEC SQL OPEN DT END-EXEC.
  DO WHILE SQLCODE IS 0.
  EXEC SQL FETCH DT USING DESCRIPTOR SQLDA END-EXEC.
  ADD IN MARKERS TO DENOTE NULLS.
  WRITE THE DATA TO SYSREC01.
  INCREMENT DATA RECORD COUNTER.
  END.
  EXEC SQL CLOSE DT END-EXEC.
  INDICATE THE RESULTS OF THE UNLOAD OPERATION.
  CLOSE THE SYSIN, SYSPRINT, AND SYSREC01 FILES.
```

DECLARING THE CURSOR AND STATEMENT

EXEC SQL DECLARE DT CURSOR FOR SEL END-EXEC.



EXEC SQL DECLARE SEL STATEMENT END-EXEC.

Move the Table Name

MOVE TNAME TO STMTTAB.

add table name after the SELECT *

MOVE STMTBLD TO STMTCHAR.

move the full SELECT statement

MOVE 750 TO SQLN.

set the occurrences in SQLDA to the maximum allowed

Preparing the Statement

```
EXEC SQL
PREPARE SEL INTO :SQLDA
FROM :STMTBUF
END-EXEC.
```

DB2 will take the statement held in STMTBUF and work out the best access path (like a BIND), it will also populate the SQLDA with the name, type and length of each column that will be returned

SQLDA

```
01 SQLDA.
   05 SQLDAID PIC X(8).                    Eyecatcher normally contains 'SQLDA'
   05 SQLDABC PIC S9(9) BINARY.            Total Length Of SQLDA
   05 SQLN    PIC S9(4) BINARY.            Number of occurrences – normally set to maximum 750
   05 SQLD    PIC S9(4) BINARY.            Number of columns – populated by DB2 after PREPARE
   05 SQLVAR  OCCURS 0 TO 750 TIMES DEPENDING ON SQLN.
    10 SQLVAR1.
      15 SQLTYPE PIC S9(4) BINARY.         Data Type of column
      15 SQLLEN  PIC S9(4) BINARY.         Column Length
      15 FILLER  REDEFINES SQLLEN.
        20 SQLPRECISION   PIC X.           Used for Decimal columns only
        20 SQLSCALE       PIC X.
      15 SQLDATA POINTER.                  Storage address where DB2 is to put the data
      15 SQLIND  POINTER.                  Storage address where DB2 is to put Indicator variable
      15 SQLNAME.
        49 SQLNAMEL PIC S9(4) BINARY.      Length of column name
        49 SQLNAMEC PIC X(30).             Column name
```

Data Types Catered for in DSN8BCU2

```
* DATA TYPES FOUND IN SQLTYPE, AFTER REMOVING THE NULL BIT
*
 77  VARCTYPE                PIC S9(4) COMP VALUE +448.
 77  CHARTYPE                PIC S9(4) COMP VALUE +452.
 77  VARLTYPE                PIC S9(4) COMP VALUE +456.
 77  VARGTYPE                PIC S9(4) COMP VALUE +464.
 77  GTYPE                   PIC S9(4) COMP VALUE +468.
 77  LVARGTYP                PIC S9(4) COMP VALUE +472.
 77  FLOATYPE                PIC S9(4) COMP VALUE +480.
 77  DECTYPE                 PIC S9(4) COMP VALUE +484.
 77  INTTYPE                 PIC S9(4) COMP VALUE +496.
 77  HWTYPE                  PIC S9(4) COMP VALUE +500.
 77  DATETYP                 PIC S9(4) COMP VALUE +384.
 77  TIMETYP                 PIC S9(4) COMP VALUE +388.
 77  TIMESTMP                PIC S9(4) COMP VALUE +392.
```

Note an even number denotes NOT NULL and an odd number denotes NULL

So 452 would be CHAR NOT NULL and 453 is CHAR NULL

The complete list can be found in Table 138 in Appendix F of SQL Reference

Determining the length of each column

- **DB2 populates the SQLDA with the lengths but**
  - 2 must be added for the length field of variable columns
  - Decimal columns have
    - 1 byte for precision
    - 1 byte for scale
    - CALC-DECIMAL-LEN.
    - DIVIDE COLUMN-LEN BY 256 GIVING COLUMN-PREC
    - REMAINDER COLUMN-SCALE.
    - MOVE COLUMN-PREC TO COLUMN-LEN.
    - ADD ONE TO COLUMN-LEN.
    - DIVIDE COLUMN-LEN BY TWO GIVING COLUMN-LEN.

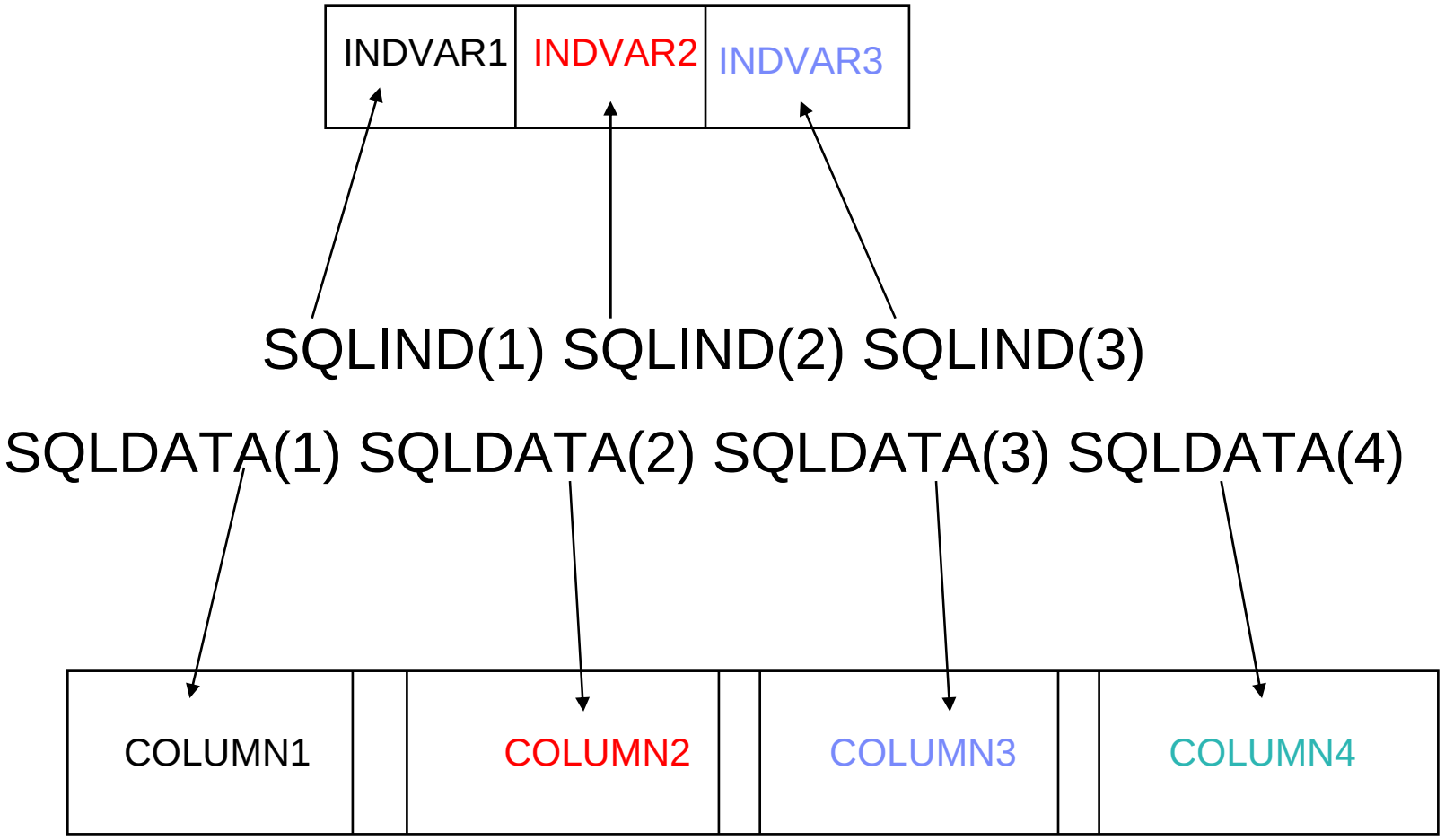
Fetching the Data

```
EXEC SQL
FETCH DT                          Fetch from the cursor called DT
USING DESCRIPTOR :SQLDA           Put the data into the storage
                                  Pointed to by the SQLDA
END-EXEC
```

The program then checks for Nulls, if it finds any it puts a ? After the data field.

The record is then written out exactly the way it was put into storage by the FETCH – no manipulation is needed.

Pointing the SQLDA to the storage



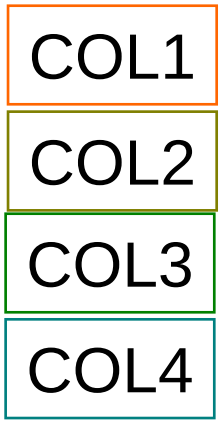
Multi-row FETCH

- Returns multiple rows on one API crossing
- "wide" cursor with locks on multiple rows
- Supports scrollable and non-scrollable, static and dynamic SQL
- Significant performance boost
- DSNTEP4 = DSNTEP2 + MRF

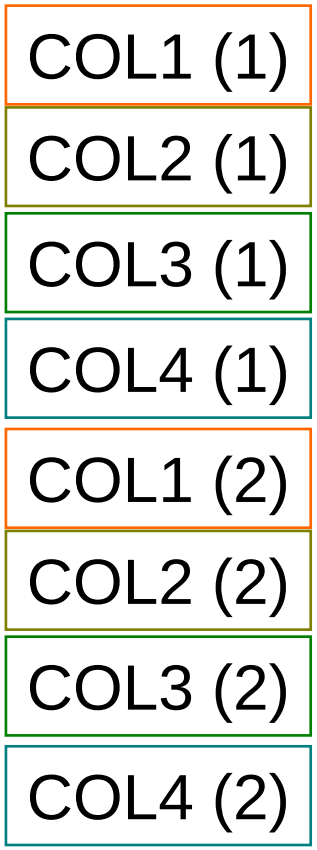
```
DECLARE C1 CURSOR
    WITH ROWSET POSITIONING
    FOR SELECT COL1, COL2 FROM T1;
OPEN C1;
FETCH FROM C1
    FOR :hv ROWS  INTO  :ARRAY1, :ARRAY2;
```

What The Arrays Do NOT contain

Single Row



Multi Row (2 rows)



What The Arrays Do contain

Single Row

COL1
COL2
COL3
COL4

Multi Row (2 rows)

COL1 (1)	COL2 (1)	COL3 (1)	COL4 (1)
COL1 (2)	COL2 (2)	COL3 (2)	COL4 (2)

There is an array for each column

NOT for each Row

What needs to be changed to cater for Multi-Row

- **2 new programs created DSN8BCM1 & 2**
- **Working storage in first program increased 100 times to cater for 100 rows being returned**
- **Linkage Section in second program changed in same way**
- **SQL needs changing**
- **SQLDA needs changing to tell DB2 we want Multi-Rows**
- **Code needs to be changed for multiple columns being returned**

SQL Changes

EXEC SQL

DECLARE DT CURSOR

WITH ROWSET POSITIONING New parameter for Multi-Row

FOR SEL

END-EXEC.

Prepare remains unchanged and will populate SQLDA with Column names and lengths as before

SQLDA Changes

- **Before the Fetch can be executed the SQLDA must be changed to tell DB2 that you want multiple columns returning**
- **To do this the SQLNAME field must contain**
  - Length must be set to 8
  - First 2 bytes of text set to hex 0000
  - Bytes 5 and 6 hex 0001 – indicating that arrays are to be used for this column
  - Bytes 7 and 8 contains number of occurrences

Code for changing SQLDA

Working storage – new fields

```
03  W-REDEF-SQLNAME.
      05 FILLER              PIC S9(8) COMP VALUE 0.
      05 W-MULTI-ROW-IND     PIC S9(4) COMP VALUE +1.
      05 W-MULTI-ROW-NUM     PIC S9(4) COMP VALUE +100.
```

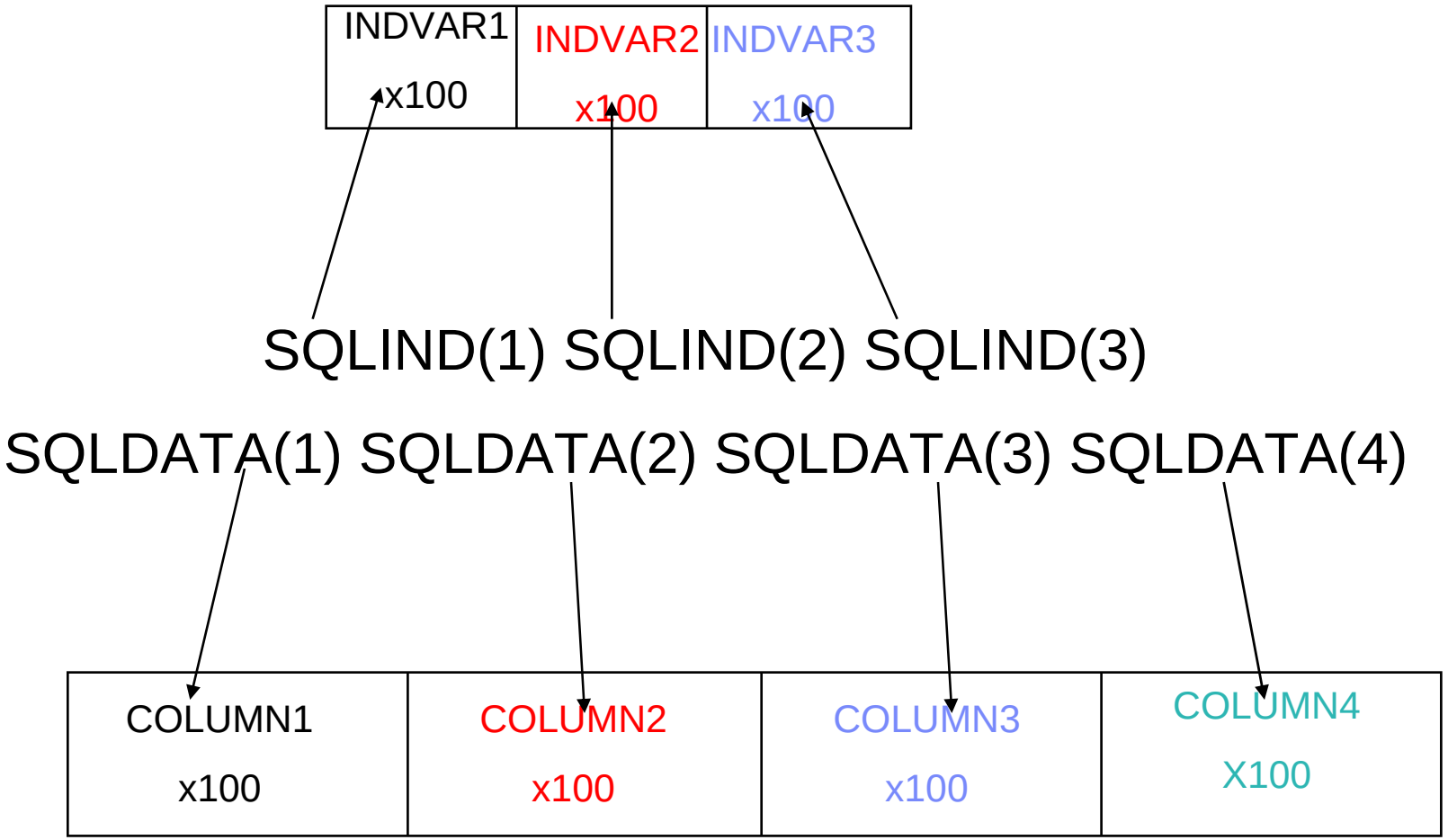
New Code before opening the cursor

```
PERFORM VARYING W-SUB FROM 1 BY 1
        UNTIL W-SUB > SQLD
          MOVE 8 TO SQLNAMEL (W-SUB)
          MOVE W-REDEF-SQLNAME TO SQLNAMEC (W-SUB)
END-PERFORM.
```

SQLDA Changes

- **If your code uses the SQLNAME field to put the column names into a report after each FETCH you must remember to move each SQLNAME field to another area of working storage after the PREPARE.**
- **Pointers to indicator variables and host variables need to cater for the number of occurrences you want to fetch - to do this the code is changed to multiply the length field for each column by 100**

Setting Up The SQLDA



Fetching the Data with new parameters

```
EXEC SQL
        FETCH NEXT ROWSET DT
        FOR 100 ROWS
        USING DESCRIPTOR :SQLDA
END-EXEC.
EXEC SQL
        GET DIAGNOSTICS
        :DIAG-ROW-COUNT = ROW_COUNT
END-EXEC.
```

Code Changes

- **Because Muti-Row Fetch brings back mutiple columns NOT mutiple rows then the code is now far more complex**
- **Old code Fetched into an area containing one row and was written from the same area**
- **New code has 100 occurrences of each column so each occurrence of each column must be moved to another area of storage to reflect the same layout as before**

Code Changes – Old Code (in DSN8BCU2)

```
MOVE ONE TO INDCOUNT.
PERFORM NULLCHK UNTIL INDCOUNT = SQLD.
MOVE REC1-LEN TO REC01-LEN.
WRITE REC01 FROM LINKAREA-REC.
ADD ONE TO ROWCOUNT.
NULLCHK.
    IF IND(INDCOUNT) < 0 THEN
    SET ADDRESS OF LINKAREA-QMARK TO
  WORKINDPTR(INDCOUNT)
    MOVE QMARK TO INDREC.
```

Code Changes – New Code (in DSN8BCM2)

```
PERFORM VARYING ROW-SUB FROM 1 BY 1
  UNTIL ROW-SUB > DIAG-ROW-COUNT
    MOVE REC-LEN TO REC01-LEN
    MOVE 1 TO WRITE-START
    PERFORM VARYING I FROM 1 BY 1
    UNTIL I > SQLD
      MOVE COL-START (I) TO INP-POS
      MULTIPLY COL-LEN (I) BY ROW-SUB GIVING CALC-RESULT
      SUBTRACT COL-LEN (I) FROM CALC-RESULT
      ADD CALC-RESULT TO INP-POS
      MOVE COL-LEN (I) TO SINGLE-COL-LEN
      MOVE REC1-CHARS (INP-POS : SINGLE-COL-LEN)
      TO REC01-CHARS (WRITE-START : SINGLE-COL-LEN)
      ADD SINGLE-COL-LEN TO WRITE-START
      IF COL-NULL(I) = 'Y'
        SUBTRACT 1 FROM I GIVING J
        MULTIPLY J BY 100
        GIVING NULL-POS
        ADD ROW-SUB TO NULL-POS
        IF IND(NULL-POS) < 0 THEN
          MOVE '?' TO REC01-CHARS (WRITE-START : 1)
        ELSE
          MOVE SPACE TO REC01-CHARS (WRITE-START : 1)
        END-IF
        ADD 1 TO WRITE-START
      END-IF
    END-PERFORM
    WRITE REC01
    ADD ONE TO ROWCOUNT
  END-PERFORM.
```

First loop to process each row

Second loop to process each column within the row

Copy the column into the record

Checks for nullable columns and inserts ? In the record If Null found

Using DSNTIAR to Obtain the Message

Working Storage

```
01  ERROR-MESSAGE.
      02 ERROR-LEN PIC S9(4) COMP VALUE +960.
      02 ERROR-TEXT PIC X(120) OCCURS 8 TIMES
         INDEXED BY ERROR-INDEX.
77  ERROR-TEXT-LEN        PIC S9(8) COMP VALUE +120.
```

Total Length of following 4 fields

Length of each message line

Procedure Division

```
CALL 'DSNTIAR' USING SQLCA ERROR-MESSAGE ERROR-TEXT-LEN
PERFORM ERROR-PRINT VARYING ERROR-INDEX
         FROM 1 BY 1 UNTIL ERROR-INDEX GREATER THAN 8
ERROR-PRINT.
       WRITE MSGREC FROM ERROR-TEXT (ERROR-INDEX)
       AFTER ADVANCING 1 LINE.
```

Coding GET DIAGNOSTICS

- **Working Storage**

```
03  W-DIAG-SUB           PIC S9(9) COMP.
03  W-DB2-RETURNED-SQLCODE PIC S9(9) COMP  VALUE 0.
03  W-DB2-RETURNED-SQLSTATE PIC X(5).
03  W-DB2-ROW-NUMBER       PIC S9(31) COMP-3.
03  W-DIAG-ERRORS         PIC S9(9) COMP.
01  W600-DIAG-AREA.
  10  W600-DIAGNOSTICS.
    49  W600-DIAGLEN       PIC S9(4) COMP VALUE 0.
    49  W600-DIAG          PIC X(32672).
```

These fields must be defined Correctly or GET DIAGNOSTICS Will not work

- **Procedure Division**

```
EXEC SQL
  GET DIAGNOSTICS :W-DIAG-ERRORS = NUMBER
END-EXEC
IF W-DIAG-ERRORS > 0
  PERFORM VARYING W-DIAG-SUB FROM 1 BY 1
  UNTIL W-DIAG-SUB > W-DIAG-ERRORS
    EXEC SQL
      GET DIAGNOSTICS CONDITION :W-DIAG-SUB
      :W-DB2-RETURNED-SQLCODE = DB2_RETURNED_SQLCODE ,
      :W-DB2-RETURNED-SQLSTATE = RETURNED_SQLSTATE ,
      :W600-DIAGNOSTICS  = MESSAGE_TEXT ,
      :W-DB2-ROW-NUMBER     = DB2_ROW_NUMBER
    END-EXEC
```

This tells us how many errors have been found

Loop round so we get all errors

Get next error

Get SQLCODE

Get SQLSTATE

Get Message

Row number which caused this error

Any Questions ?

If you have any questions please contact

Ian Cook - ian_cook@uk.ibm.com

Or

Paul Fletcher – fletchpl@uk.ibm.com

