# Converting To Multi-Row Fetch With Ease

Anthony Tichonoff
Florida Hospital MIS
January 2007

FLORIDA HOSPITAL

# Multi-Row Fetch

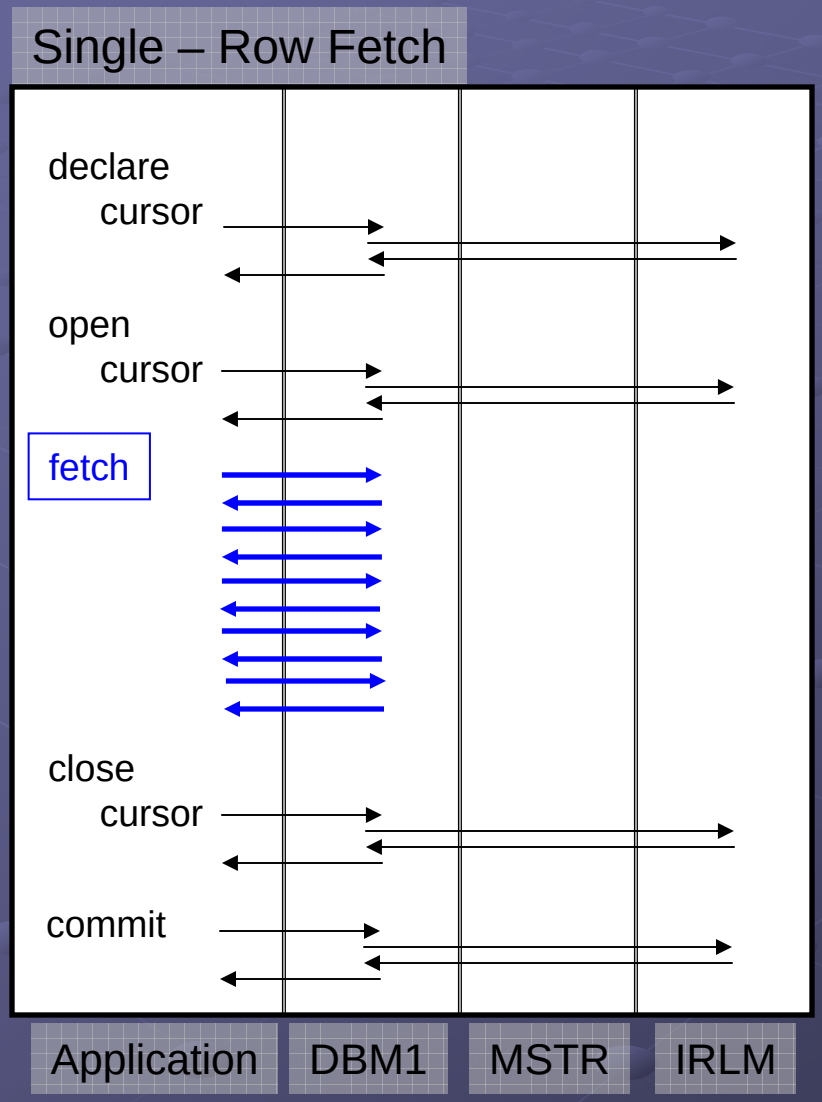
- Agenda
  - Why use multi-row fetching
  - What I need to know using multi-row fetching
  - Extras about multi-row fetching
  - How to bridge into multi-row fetching with ease

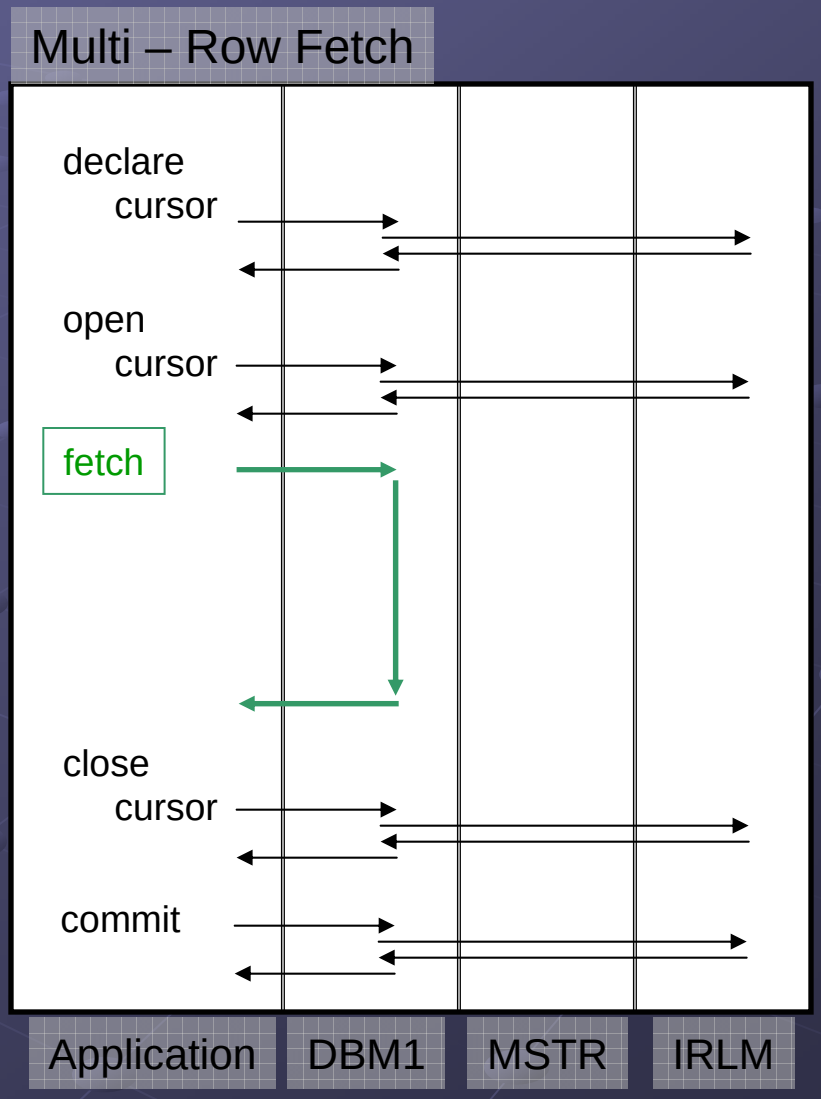
# Why use multi-row fetching?

# Why use multi-row fetch

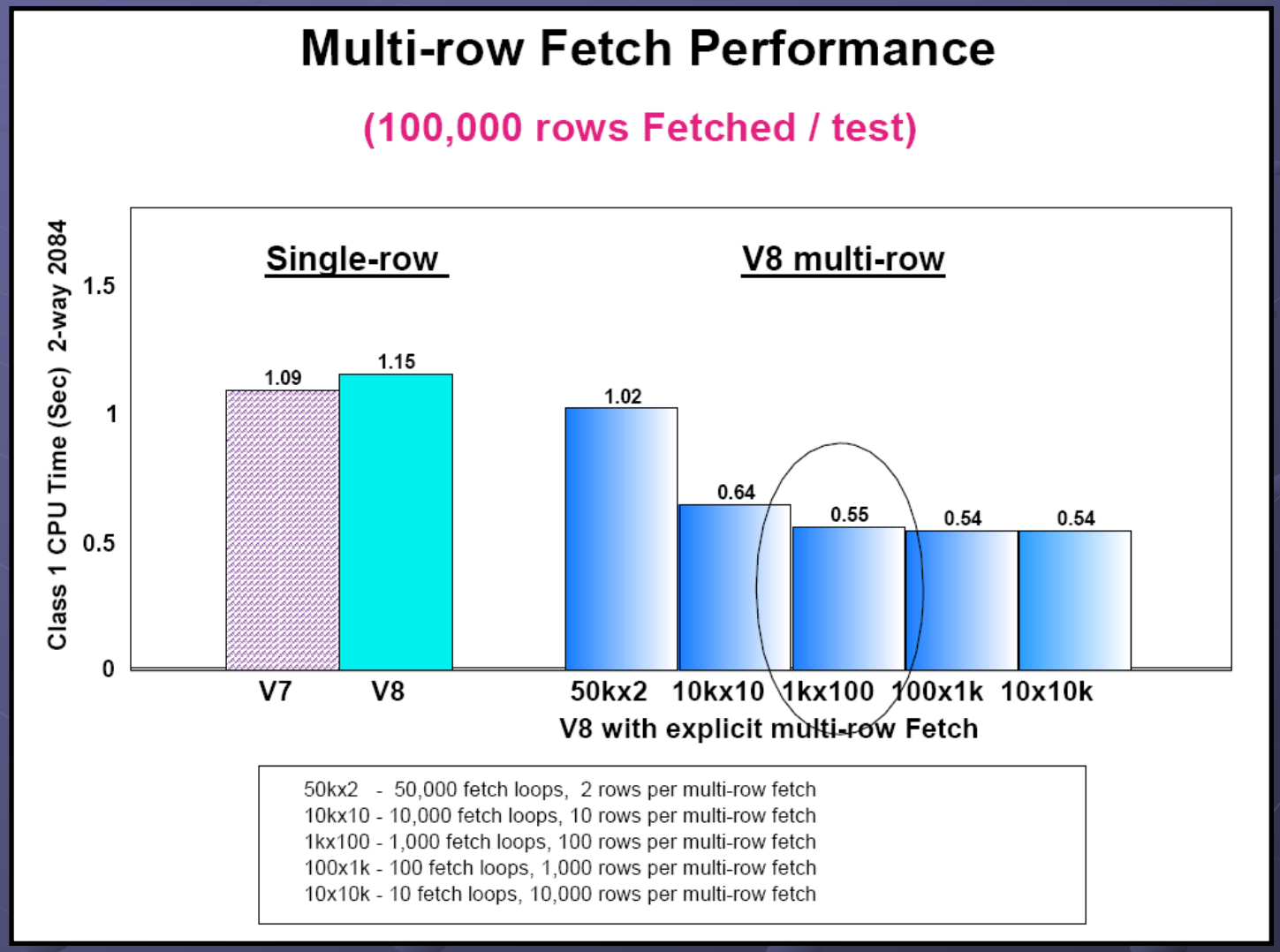
- Performance Advantages
  - Improves Throughput
    - Fewer database access calls
    - Lower network operations
  - Lowers CPU consumption
  - V8 ROU (Return On Upgrade)

# Advantages: Performance



# CPU Consumption



# Return on Upgrade

- What does V8 cost you?
  - Performance objective is less than 10% average regression.
  - Typical customer workload regression is expected to be 5 to 10% higher on average, differing by work load:

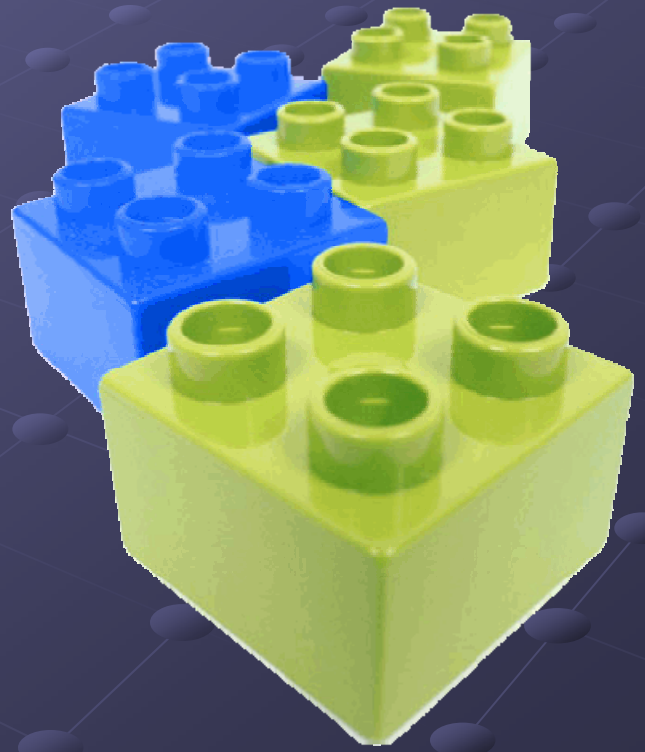
| | | |
|---|---|---|
| online transaction | 0% | +15% |
| transaction in data sharing | -5% | +10% |
| batch | -5% | +20% |
| insert | -5% | +5% |
| fetch, select, update | +5% | +20% |
| batch data sharing | -10% | +15% |
| batch DRDA | -20% | +15% |
| utility | -5% | +10% |
| query | -20% | +15% |

- Return on Upgrade
  - Options with significant potential to offset an expected increase include multi-row fetch, multi-row insert, long term page fix and rebind.

# What you need to know.

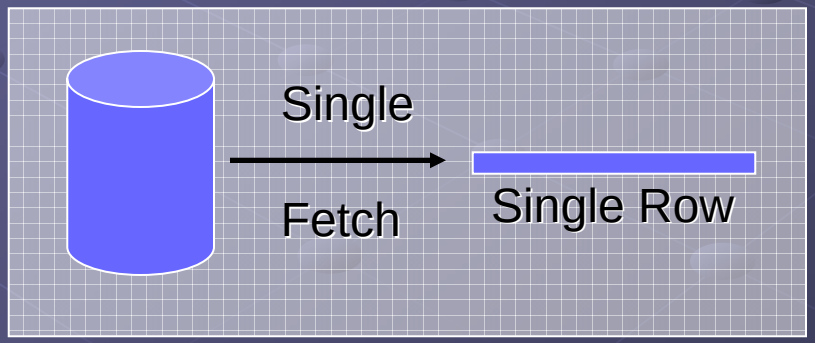
# Requirements for Multi-row

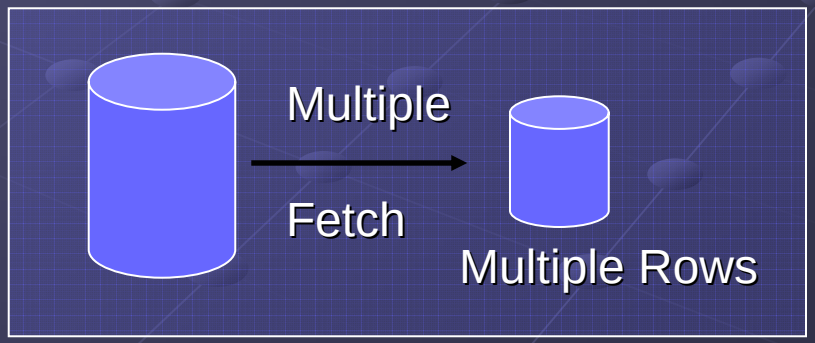
- Understanding the foundation
  - What is a rowset
  - New SQL syntax
    - Declare
    - Fetch
  - Host variable arrays
  - Error handling



# What is a rowset?

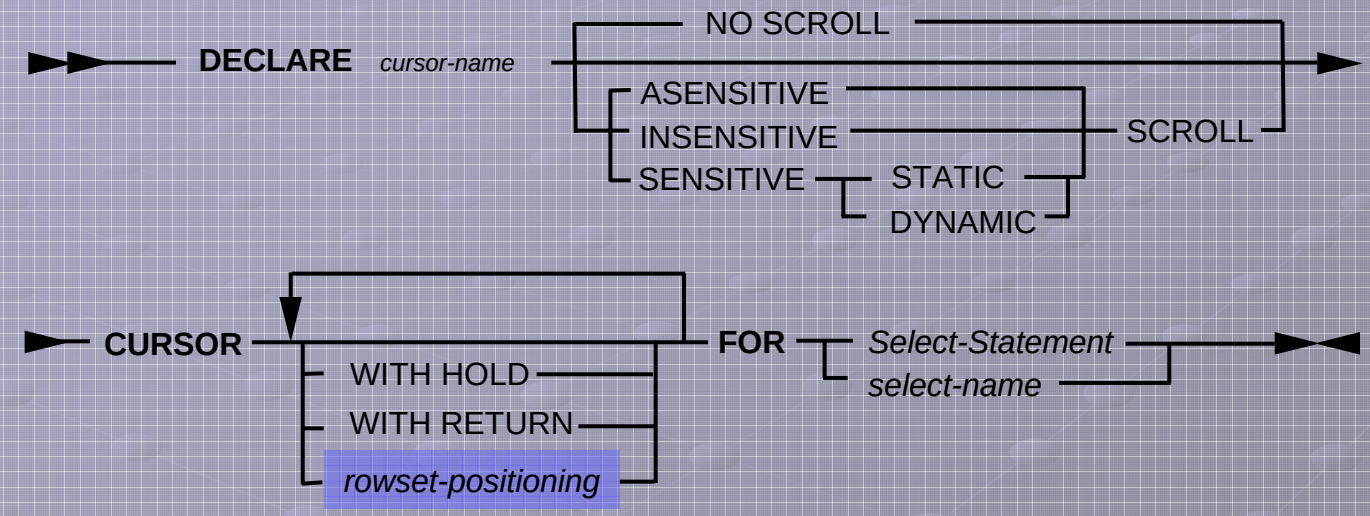
- A group of rows for the result table of a query that are returned by a single fetch statement.
- The maximum size of the rowset is 32767
- The program fetch call controls how many rows are returned in a rowset.
- It is possible to intertwine single-row and multi-row fetches within the same program.



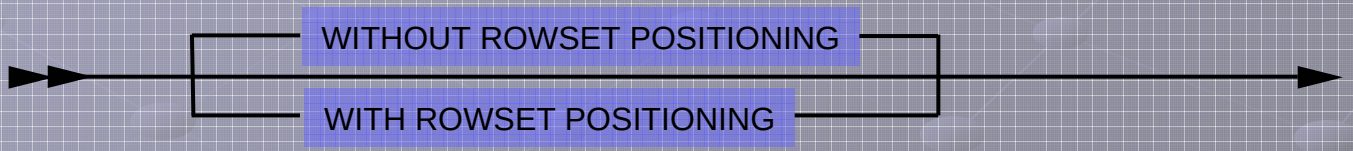



# Declare Syntax Changes

## DECLARE CURSOR Syntax




***rowset-positioning:***




Default is ***WITHOUT ROWSET POSITIONING***

# Declare Syntax Example

- SQL Declare with rowset positioning

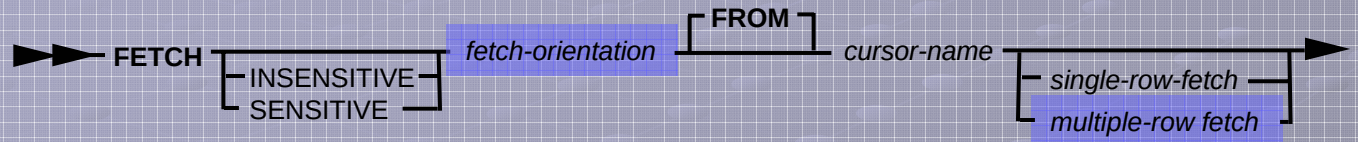
Declare C1 as the cursor of a query to retrieve a rowset from table PayRoll

```
Exec SQL
      Declare C1 Cursor
      With Rowset Positioning For
      Select *
      From PayRoll
      Where ….
End Exec
```

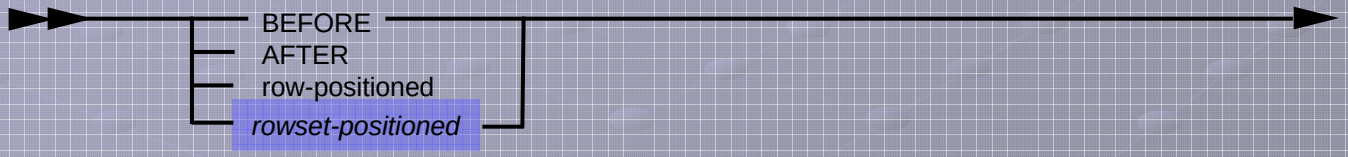
With Rowset Positioning specifies whether multiple rows of data can be accessed as a rowset on a single Fetch statement

# Fetch Syntax Changes

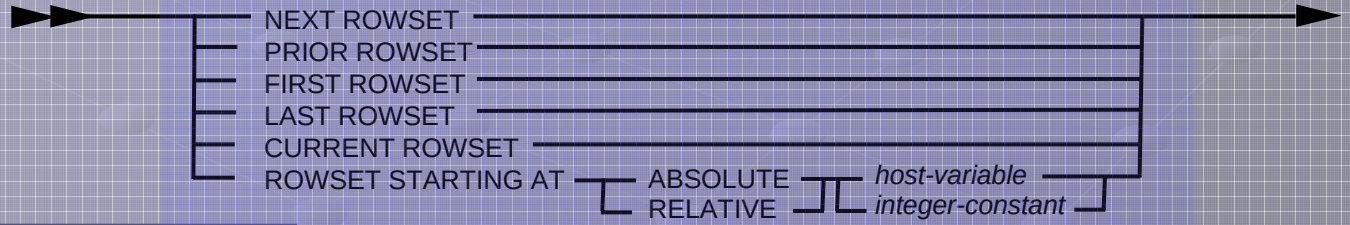
## FETCH Syntax



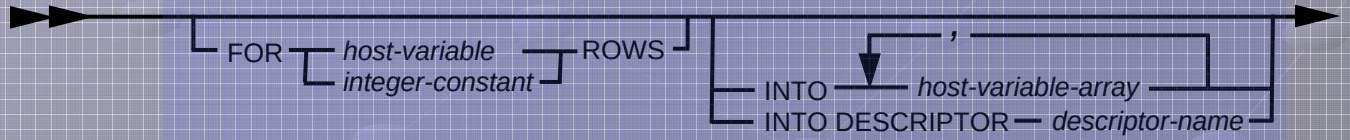

***fetch-orientation:***




***rowset-positioned:***




***multiple-row-fetch:***



# Fetch Syntax Example

- SQL Fetch with rowset positioning

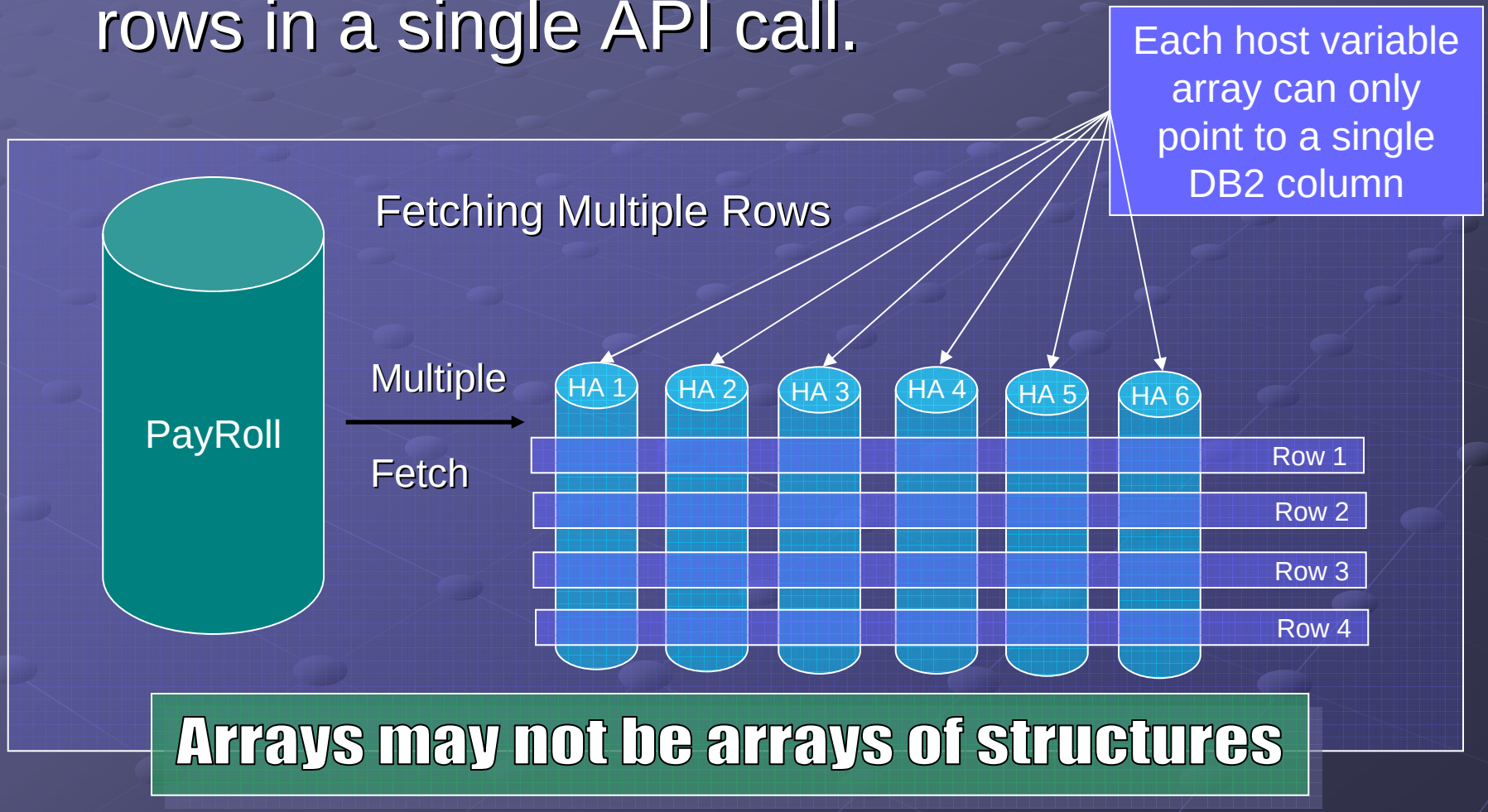
Fetch the next rowset

```
Exec SQL
      Fetch Next Rowset From C1
      For :Rowset-Limit Rows
      Into
            :HA_C1
           ,:HA_C2
           ,:HA_C3
End Exec
```

**For :HV Rows** is not required. The fetch statement will default to the row limit from the previous fetch statement.

# Host variable arrays

- Fetching into arrays will return multiple rows in a single API call.



# Host variable arrays

- Host variable array declaration

```
EXEC SQL DECLARE TEST.PAYROLL TABLE
( Column_01                    CHAR(1)    NOT NULL,
  Column_02                    TIMESTAMP  NOT NULL,
  Column_03                    CHAR(8)    NOT NULL,
  Column_04                    CHAR(1)    NOT NULL,
  Column_05                    CHAR(8)    NOT NULL,
) END-EXEC.

 01  DCLHVA.
     05  HVA-Column-01          PIC X(1)  OCCURS 100 TIMES.
     05  HVA-Column-02          PIC X(26) OCCURS 100 TIMES.
     05  HVA-Column-03          PIC X(8)  OCCURS 100 TIMES.
     05  HVA-Column-04          PIC X(1)  OCCURS 100 TIMES.
     05  HVA-Column-05          PIC X(8)  OCCURS 100 TIMES.
```

**Arrays may not be arrays of structures**

# Error handling multi-row fetch

- Handling multi-row SQL RC
  - SQL RC is the result for the ***rowset*** not for the ***row***.
  - Use the Get Diagnostics statement to return each row's SQL RC

```
EXEC SQL
    GET DIAGNOSTICS
       :rows-returned = Row_Count
      ,:err-count = Number
END-EXEC
```

returns statement information

```
EXEC SQL
  GET DIAGNOSTICS CONDITION :cond
     :sqlcode = DB2_Returned_SQLCode
    ,:sqlstate = Returned_SQLState
END-EXEC
```

returns condition information

# Error handling multi-row fetch

- Handling multi-row SQL RC (continue)
  - If SQL RC = 0 then ***all rows*** within the rowset have a SQL RC = 0. (Get Diagnostics is not necessary)
  - If SQL RC = +100 then the EOT has been found, but rows could possibly exists within the host variable array to process. (Get Diagnostics is not necessary)
  - Any other RC not equal to 0 or +100 on a rowset we must treat as you have in the past or use Get Diagnostics.

# Information from the SQLCA

- Useful data within the SQLCA after the rowset Fetch.
  - SQLCODE
  - SQLSTATE
  - SQLERRD(3) contains the actual number of row returned.
  - SQLWARN flags are set to represent all warnings accumulated while processing the Fetch.

# Extras about multi-row fetching.

# Helpful Extras Tips

- Keeping track of your multi-row cursors information
  - SQLCODE - per cursor
  - Rows Returned - per cursor
  - Current array position - per cursor
  - Max array limits - per table


- Create ways to prevent exceeding your host variable array boundaries

# Helpful Extras Tips

- How big do we make the arrays
  - Appropriate array size your
    - online apps
    - batch apps
- Creating standard host variable array copybooks.
  - Add max limits for the array

# Helpful Extras Tips

- Simple error checking for multi-row fetching

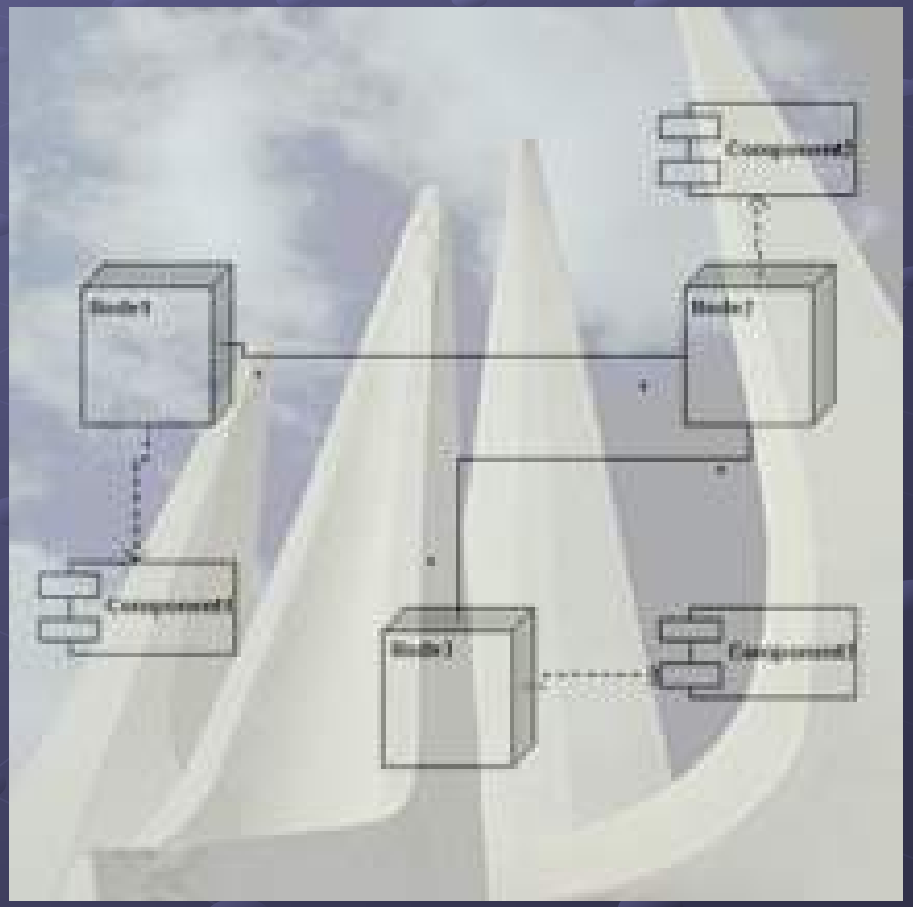
```
Fetch Next Rowset
From CS1
For :row-limt Rows
Into
   :HVA-1
  ,:HVA-2
  ,:HVA-3
```

```
If (SQLCode = 0  Or  SQLCode = 100)  And
   SQLERRD(3) > 0

            - Process SQLERRD(3) rows

Else If SQLCode = 100 and  SQLERRD(3) = 0

            -  Close Cursor

Else Handle SQL Error
```

# How to bridge into multi-row fetching with ease.

# Bridging to Multi-row fetching

- Steps to discuss
  - Our Plan
  - Our Design
  - Coding
  - Automation
  - Execution



# Bridging to Multi-row fetching

- Our Plan
  - Limit the amount of coding necessary to convert to multi-row fetch.
  - Limit the changes to our error handling.
  - Make the changes fit our current program’s architecture framework.
  - Keep the changes few and simple.

# Bridging to Multi-row fetching

- Our Design (Limit the changes)

Our Goals:
- limit change
- keep it simple

- What we have

Declare
Open
Fetch
Loop Until Done
Process Work
Fetch Next
Close

- What we want

Declare **(syntax changes)**
Open
Fetch **(syntax changes and wrapper)**
Loop Until Done
Process Work
Fetch Next
Close

# Bridging to Multi-row fetching

- Our Design
  - Modifying the Declare with a simple syntax change
  - Modifying the Fetch…

**Add Pre-Fetch Code**
(Handles decision where to get next row)

**Fetch SQL Statement**
(Changes made to acquire multi-row fetch)

**Add Post-Fetch Code**
(Handles RCs and array positioning)

**Add Variable Assignments**
(Move :HVA elements into :HV)

# Coding the Working Storage

## Table Working Storage

```
 01   CP160A-ROWSET-MAX                  PIC S9(04) COMP
%                                          VALUE {_ROWSET_SIZE_MAX}.
 01   CP160A-ROWSET-LIMIT                PIC S9(04) COMP
%                                          VALUE {ROWSET_SIZE}.
```

## Cursor Working Storage

```
%01   {CURSOR}-ROWSET-STORAGE.
%     05   {CURSOR}-SQLCODE                    PIC S9(09) COMP-4
                                                 VALUE ZERO.
%     05   {CURSOR}-ROWSET-RETURN              PIC S9(04) COMP
                                                 VALUE ZERO.
%     05   {CURSOR}-SUB1                       PIC S9(04) COMP
                                                 VALUE ZERO.
```

**Don't forget to code your Host Variable Arrays**

# Coding the Working Storage

Table Host Variable Array

```
 01  DCLVCP97160A.
     05  CP160A-F-STATS-RECR          PIC X(1)
%                                        OCCURS {ROWSET_SIZE} TIMES.
     05  CP160A-T-MODF                PIC X(26)
%                                        OCCURS {ROWSET_SIZE} TIMES.
     05  CP160A-I-OPRT-MODF           PIC X(8)
%                                        OCCURS {ROWSET_SIZE} TIMES.
     05  CP160A-I-ACTN-MODF           PIC X(1)
%                                        OCCURS {ROWSET_SIZE} TIMES.
     05  CP160A-I-TERM-MODF           PIC X(8)
%                                        OCCURS {ROWSET_SIZE} TIMES.
     05  CP160A-I-CLIE                PIC X(2)
%                                        OCCURS {ROWSET_SIZE} TIMES.
     05  CP160A-I-N-STAN              PIC X(18)
%                                        OCCURS {ROWSET_SIZE} TIMES.
     05  CP160A-I-CODE                PIC X(10)
%                                        OCCURS {ROWSET_SIZE} TIMES.
     05  CP160A-C-SHOR-DESC           PIC X(15)
```

# Coding the Working Storage

ISCL code for the host variable array size

```
%%IF {%DEF ROWSET_SIZE}
%  %UNDEF ROWSET_SIZE
%%ENDIF

%%IF {%DEF ROWSET_SIZE_LOCAL}
%  %DEFINE ROWSET_SIZE {ROWSET_SIZE_LOCAL}
% %ELSE
%  %DEFINE ROWSET_SIZE {_ROWSET_SIZE_SYS}
%%ENDIF

%%IF {ROWSET_SIZE} > {_ROWSET_SIZE_MAX}
%  %MESSAGE 'THE ROWSET SIZE MUST BE'
%  %MESSAGE '<= '{_ROWSET_SIZE_MAX}
% %FAIL
%%ENDIF
```

# Coding the Prefetch

```
%%IF {_MULTIROW_FETCH_MODE} = 'PRE_FETCH'
*----------------------
*-- PRE-FETCH CHECK
*----------------------
*--
*-- MAKE SURE ROWSET-LIMIT WAS NOT SET ABOVE THE ROWSET SIZE
*-- LIMIT OF THE ARRAY.
%    IF {ROWSET_PREFIX}-ROWSET-LIMIT > {ROWSET_SIZE}
%       MOVE {ROWSET_SIZE}      TO {ROWSET_PREFIX}-ROWSET-LIMIT
     END-IF
*--
*-- GET NEXT ROW; PASS BACK ALL ROWS WITHIN THE ROWSET
*-- BEFORE FETCHING THE NEXT ROWSET
%    IF {CURSOR}-ROWSET-RETURN > {CURSOR}-SUB1
%       ADD +1                        TO {CURSOR}-SUB1
     ELSE
*--
*-- RESET SQLCODE FROM LAST ROWSET FETCH AND AFTER ALL ROWS
*-- HAVE BEEN PROCESS FROM THE LAST ROWSET
%       IF {CURSOR}-SUB1 > 0                    AND
%          {CURSOR}-SUB1 = {CURSOR}-ROWSET-RETURN AND
%          {ROWSET_PREFIX}-ROWSET-LIMIT > {CURSOR}-ROWSET-RETURN

%          MOVE {CURSOR}-SQLCODE     TO WS960-SQLCODE
                                        SQLCODE
        ELSE
*--
*-- PEFORM THE NEXT ROWSET FETCH.
*--
```

Check array boundaries

Get Next Array Row

At the end reset SQL return code

Fetch next rowset

# Coding the Fetch

Before

```
EXEC SQL
    FETCH CUR001
    NTO
        :TAB01-I-CLIE
       ,:TAB01-I-PROG
       ,:WS400-CP18-COUNT
END-EXEC
```

After

```
EXEC SQL
    FETCH NEXT ROWSET FROM CUR001
    FOR :TAB01A-ROWSET-LIMIT ROWS
    INTO
        :TAB01A-I-CLIE
       ,:TAB01A-I-PROG
       ,:WS400A-CP18-COUNT
END-EXEC
```

# Coding the Post-Fetch

```
%%IF {_MULTIROW_FETCH_MODE} = 'POST_FETCH'
*----------------------
*-- POST-FETCH CHECK
*----------------------
*--
*-- POST FETCH WORK
*--   1. MOVE ROWS FETCHED VALUE INTO ROWSET RETURN VARIABLE
*--   2. MUST KEEP TRACK OF THE SQLCODE FROM THE FETCH TO
*--      PASS BACK TO THE CALLER WHEN THE LAST ROW FROM THE
*--      ROWSET IS RETURNED.
%          MOVE SQLERRD(3)          TO {CURSOR}-ROWSET-RETURN
           MOVE SQLCODE             TO WS960-SQLCODE
%                                      {CURSOR}-SQLCODE

*--   3. WHEN NORMAL FETCH OR LAST ROWSET RETURNED IS A NOT
*--      FOUND THEN RESET ROWSET POINTER AND FORCE SQLCODE
*--      TO NORMAL
           IF WS960-R-NORMAL
              OR
%             (WS960-R-NOTFND AND {CURSOR}-ROWSET-RETURN > 0)
%             MOVE 1                TO {CURSOR}-SUB1

              SET WS960-R-NORMAL     TO TRUE
              MOVE WS960-SQLCODE     TO SQLCODE
           END-IF
*--
        END-IF
     END-IF
```

Just executed the fetch

Store rows returned & SQL RC per cursor

If RC = 0 or (+100 & no array rows) reset array pointer & SQL code

# Coding the Data Moves

```
*-- CUR001 HOST ARRAY VARIABLE MOVES TO HOST VARIABLES
     IF WS960-R-NORMAL
        MOVE TAB01A-I-CLIE (CUR001-SUB1)      TO TAB01-I-CLIE

        MOVE TAB01A-I-PROG (CUR001-SUB1)      TO TAB01-I-PROG

        MOVE TAB01A-CP18-COUNT (CUR001-SUB1) TO WS400-CP18-COUNT

     END-IF
```

Let eliminate code changes by using our existing host variables.

```
%%IF {_MULTIROW_FETCH_MODE} = 'PRE_FETCH'
*----------------------
*-- PRE-FETCH CHECK
*----------------------
*--
*-- MAKE SURE ROWSET-LIMIT WAS NOT SET ABOVE THE ROWSET SIZE
*-- LIMIT OF THE ARRAY.
%    IF {ROWSET_PREFIX}-ROWSET-LIMIT > {ROWSET_SIZE}
%       MOVE {ROWSET_SIZE}      TO {ROWSET_PREFIX}-ROWSET-LIMIT
     END-IF
*--
*-- GET NEXT ROW; PASS BACK ALL ROWS WITHIN THE ROWSET
*-- BEFORE FETCHING THE NEXT ROWSET
%    IF {CURSOR}-ROWSET-RETURN > {CURSOR}-SUB1
%       ADD +1                        TO {CURSOR}-SUB1
     ELSE
*--
*-- RESET SQLCODE FROM LAST ROWSET FETCH AND AFTER ALL ROWS
*-- HAVE BEEN PROCESS FROM THE LAST ROWSET
%       IF {CURSOR}-SUB1 > 0                    AND
%          {CURSOR}-SUB1 = {CURSOR}-ROWSET-RETURN AND
%          {ROWSET_PREFIX}-ROWSET-LIMIT > {CURSOR}-ROWSET-RETURN

%          MOVE {CURSOR}-SQLCODE     TO WS960-SQLCODE
                                        SQLCODE
        ELSE
*--
*-- PEFORM THE NEXT ROWSET FETCH.
*--
```

```
%             (WS960-R-NOTFND AND {CURSOR}-ROWSET-RETURN > 0)
%             MOVE 1                  TO {CURSOR}-SUB1

              SET WS960-R-NORMAL      TO TRUE
              MOVE WS960-SQLCODE      TO SQLCODE
            END-IF
*--
         END-IF
      END-IF
```

Fetch

Pre-Fetch

Next Subscript | Reset Code | Call Fetch

Post-Fetch

Move :HVA

row sqlcode

+100

rows returned

2

subscript

2

Host Variables

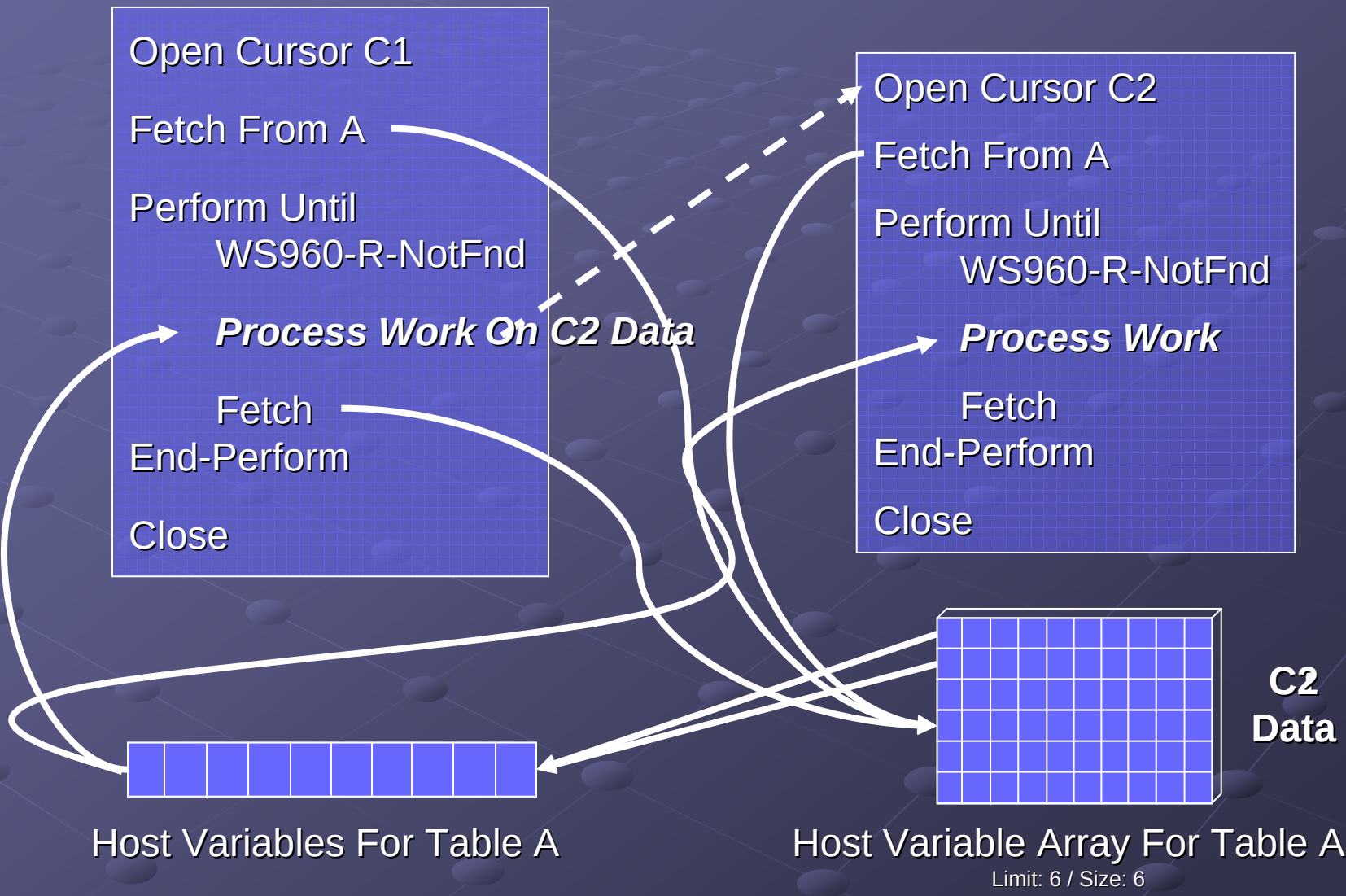
RC: +100

Host Variable Array

Limit: 6 / Size: 6

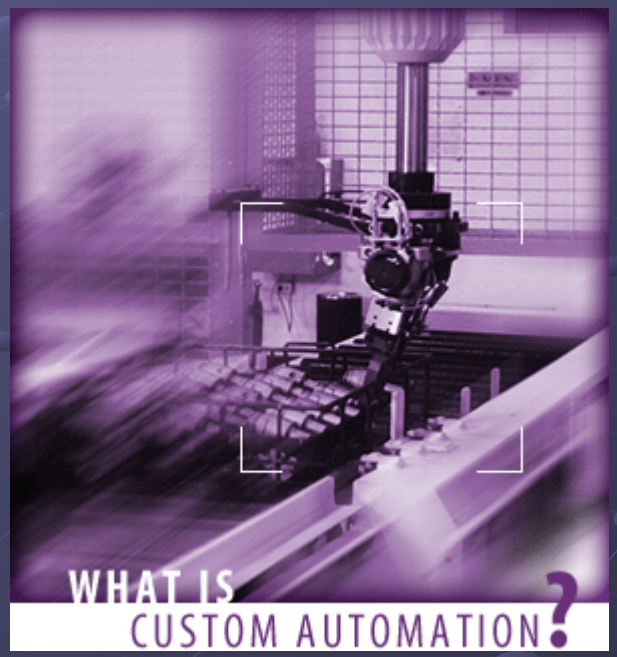
DB2 Data

# Beware!


Open Cursor C1
Fetch From A
Perform Until
WS960-R-NotFnd
Process Work On C2 Data
Fetch
End-Perform
Close
Open Cursor C2
Fetch From A
Perform Until
WS960-R-NotFnd
Process Work
Fetch
End-Perform
Close
C2
Data
Host Variables For Table A
Host Variable Array For Table A
Limit: 6 / Size: 6

# Bridging to Multi-row fetching

- Automation
  - The technique, method, or system of operating or controlling a process by highly automatic means, as by electronic devices, reducing human intervention to a minimum.




- My definition
  - Lets minimize human error
  - Built a Rexx program with ISPF macros to alter existing Cobol code.

# Multi-Row Fetch

- Points to remember
  - Use multi-row fetching for a Return on Upgrade
  - Know the basics for utilizing multi-row fetch
  - Extras about multi-row fetching
  - Find ways to quickly convert with little code changes to gain back resources.

# Converting To Multi-Row Fetch With Ease

## Questions?

Thanks for attending.

Anthony Tichonoff
anthony.tichonoff@flhosp.org
Florida Hospital MIS