IDUG

2026

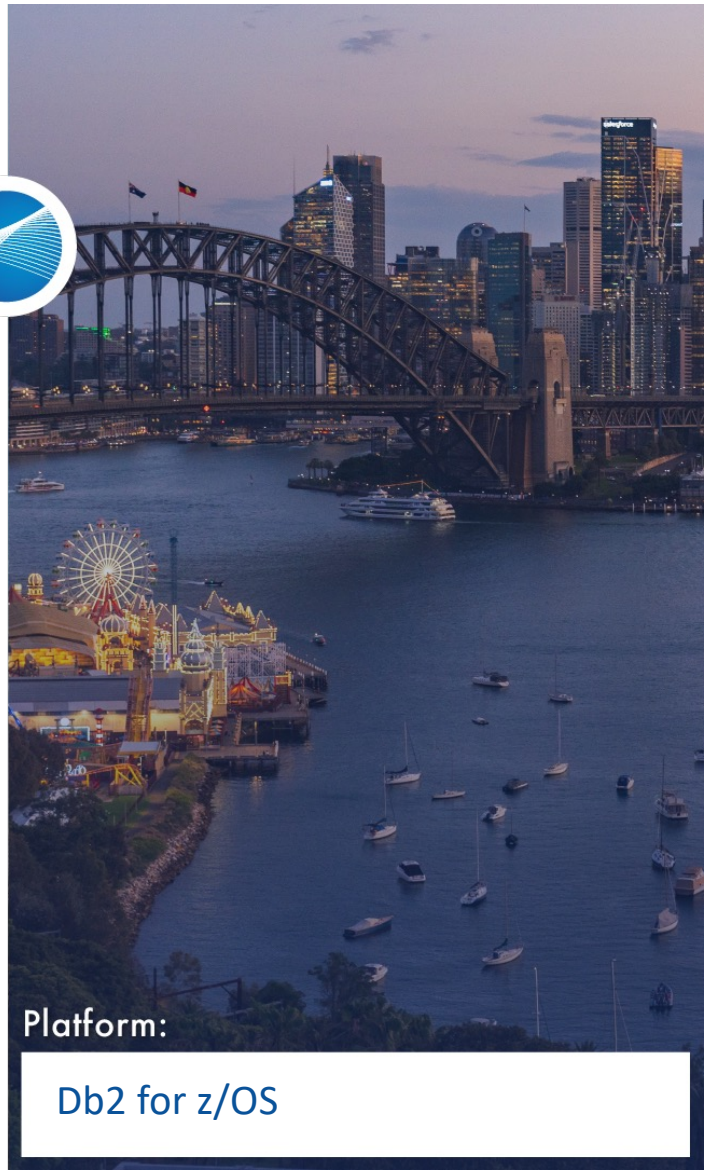
Sydney | March 16 - 18

AU Db2 TECH CONFERENCE

Enhance Performance with Db2 Multi-Row Processing

Chris Crone, Broadcom

Session Code: App Dev 5



Platform:

Db2 for z/OS

How I Tested – Disclaimer

- Z16 60X/z15 60x – Dedicated Performance Sysplex/Datasharing GP
  - Only used z16
    - 2 GCPs
    - 3 zIIPs
    - 1 TB Memory (Generous Bufferpools)
  - Db2 13 FL V13R1M506
  - Dedicated DS8K DASD (only cabled to Performance LPARs)
- Db2 Implicit Tables – Non-Partitioned
  - Indexes added/deleted to emphasize overhead or for needed function
  - Data Generated from HIS is repetitive
- Testcases with “no business logic”, “no think time” will likely overestimate real world benefits
- Your Results **WILL VARY**
  - TEST, TEST, TEST

Presentation Topics

- Overview
- Multi-Row FETCH
- Multi-Row INSERT
- SELECT FROM …
- MERGE (aka UPSERT)
- Conclusion

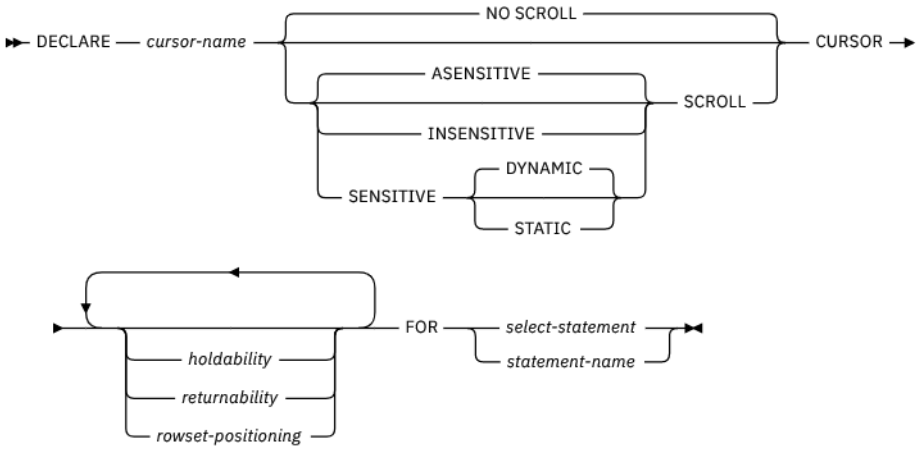

Overview
IDUG
2026 Australia Db2 Tech Conference

Overview

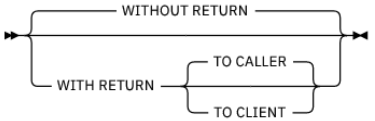
- Standard INSERT and FETCH process one row of data at a time
- DELETE, UPDATE, and INSERT with SUBSELECT can process one or more rows of data (Not really thesubject of this presentation)
- Db2 also supports a plethora of other Multi-Row processing
  - Multi-Row FETCH
  - Multi-Row INSERT
  - SELECT FROM (INSERT, UPDATE, DELETE, MERGE)
  - MERGE
- Multi-Row Processing can save significant CPU and Elapsed Time
- Let’s Get Started

Fetching Multiple Rows

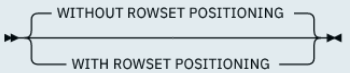
DECLARE CURSOR



returnability:



rowset-positioning:



cursor-name – names the cursor

NO SCROLL/SCROLL – Controls cursor scrollability

Avoid ASENSITIVE (default) due to ambiguity

holdability – WITH/WITHOUT HOLD – controls cursor behavior on COMMIT

returnability – WITH/WITHOUT RETURN –controls if the cursor is to be returned from a procedure as a result set

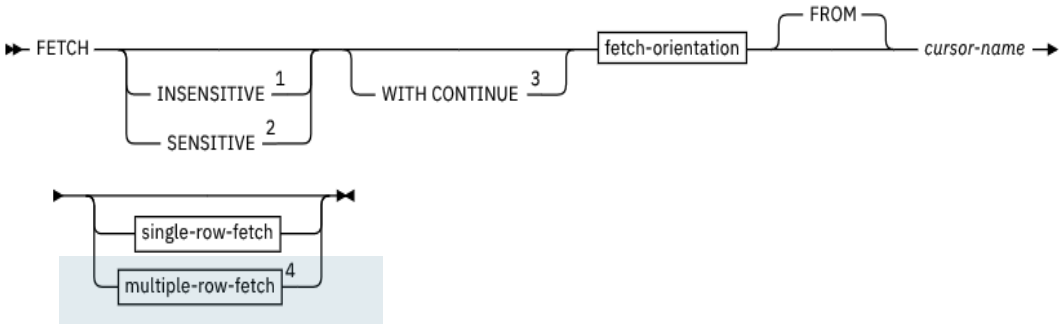
rowset-positioning – WITH/WITHOUT ROWSET POSITIONING – Specifies if multiple rows can be accessed as a rowset

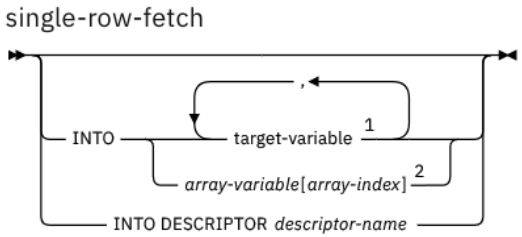
FOR

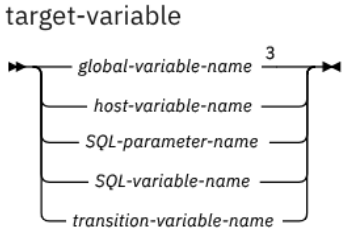
select-statement – Specifies the result table via a select-statement

statement-name – Identifies the prepared select-statement

FETCH







SENSITIVE/INSENSITIVE – Controls sensitivity to changes made outside this cursor. Complex option that is lightly supported.

WITH CONTINIUE – Allows a subsequent FECTH CURRENT CONTINUE to access truncated LOB or XML columns following an initial FETCH

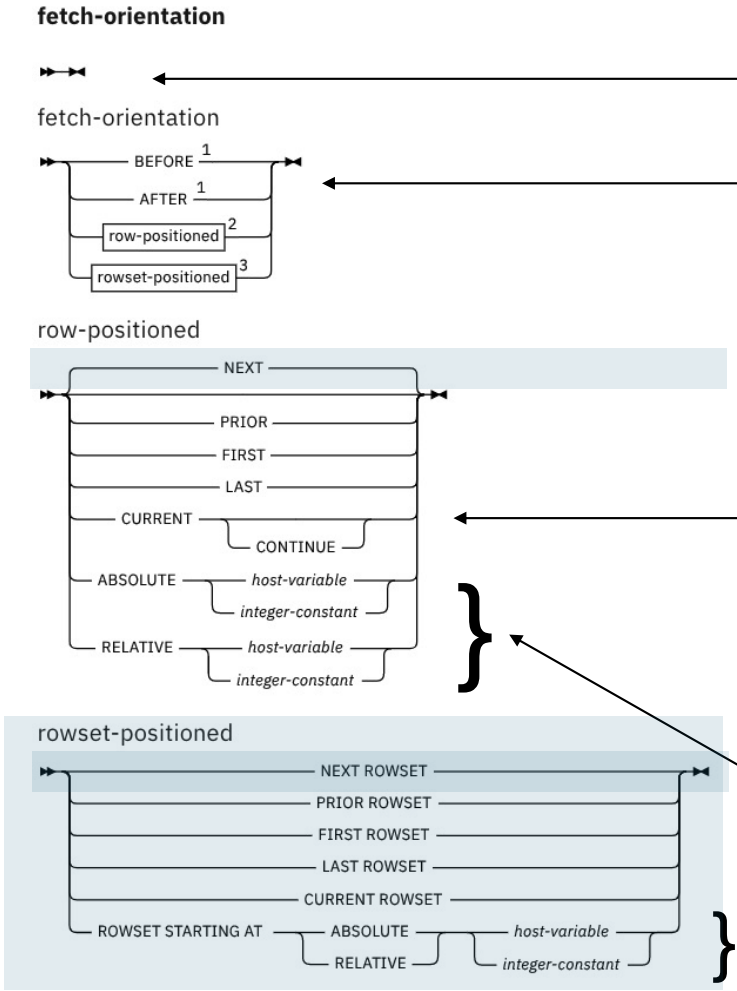
fetch-orientation

NEXT – Default behavior (Other options will be discussed later)

INTO target-variable or DESCRIPTOR – Specifies target of FETCH

FETCH without an INTO clause simply advances the cursor.

FETCH (cont)



The clause is optional – i.e. Everything below is optional

BEFORE and AFTER are positioning FETCH statements and do not return data

CURRENT may not return the Row you "think" you are on

CONTINUE is used to return additional LOB or XML data (if previous FETCH was truncated

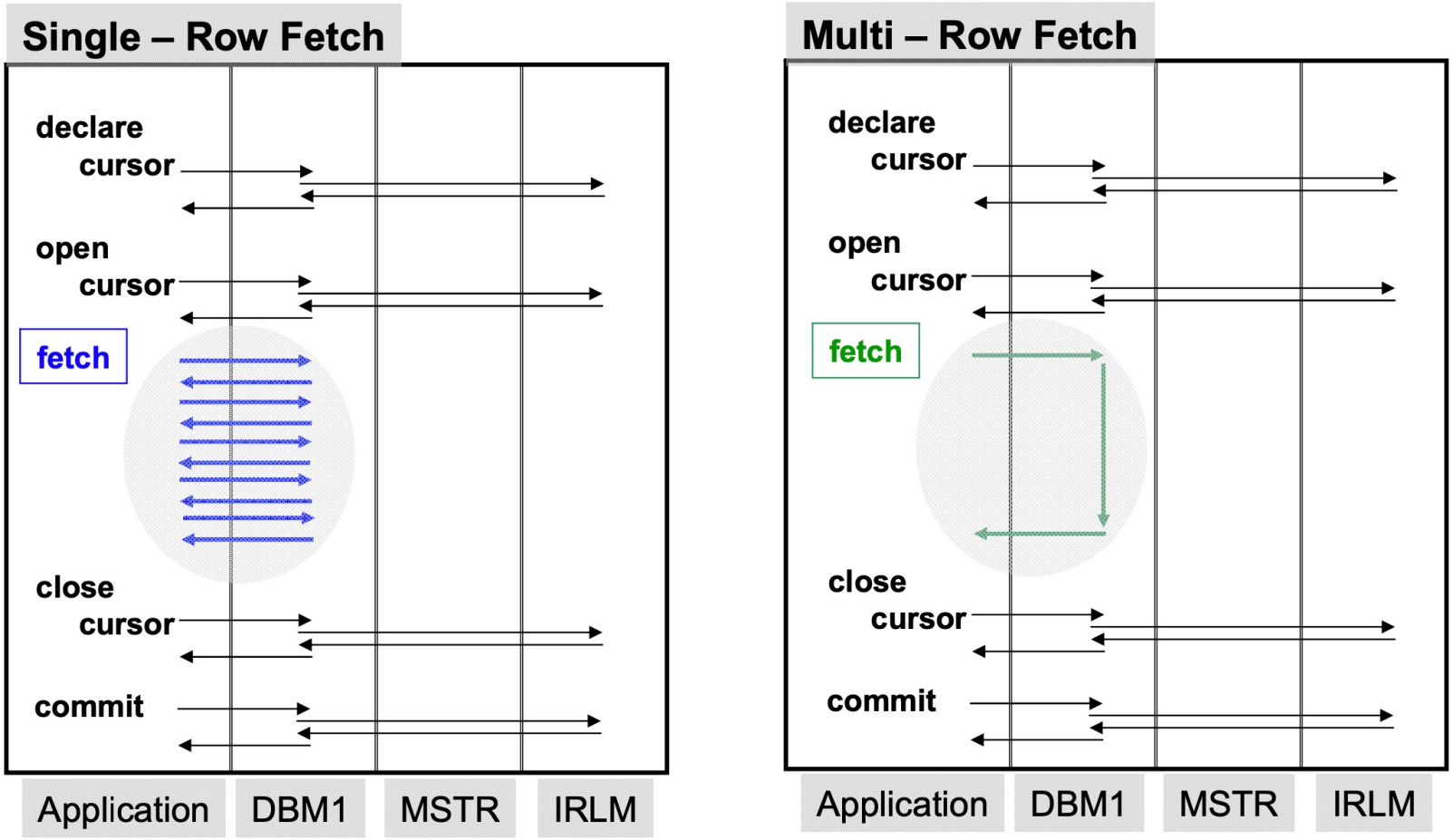
ABSOLUTE and RELATIVE provide the ability

To move to a position quickly

Simple Fetch Example


```
DECLARE CURSOR CS1 FOR SELECT NAME FROM EMP WHERE NAME =:HV1;

OPEN CURSOR CS1 USING :HV1;

Loop until SQLCODE = 100;
  FETCH CS1 INTO :OHV1;
End Loop;

CLOSE CURSOR CS1;

COMMIT;
```

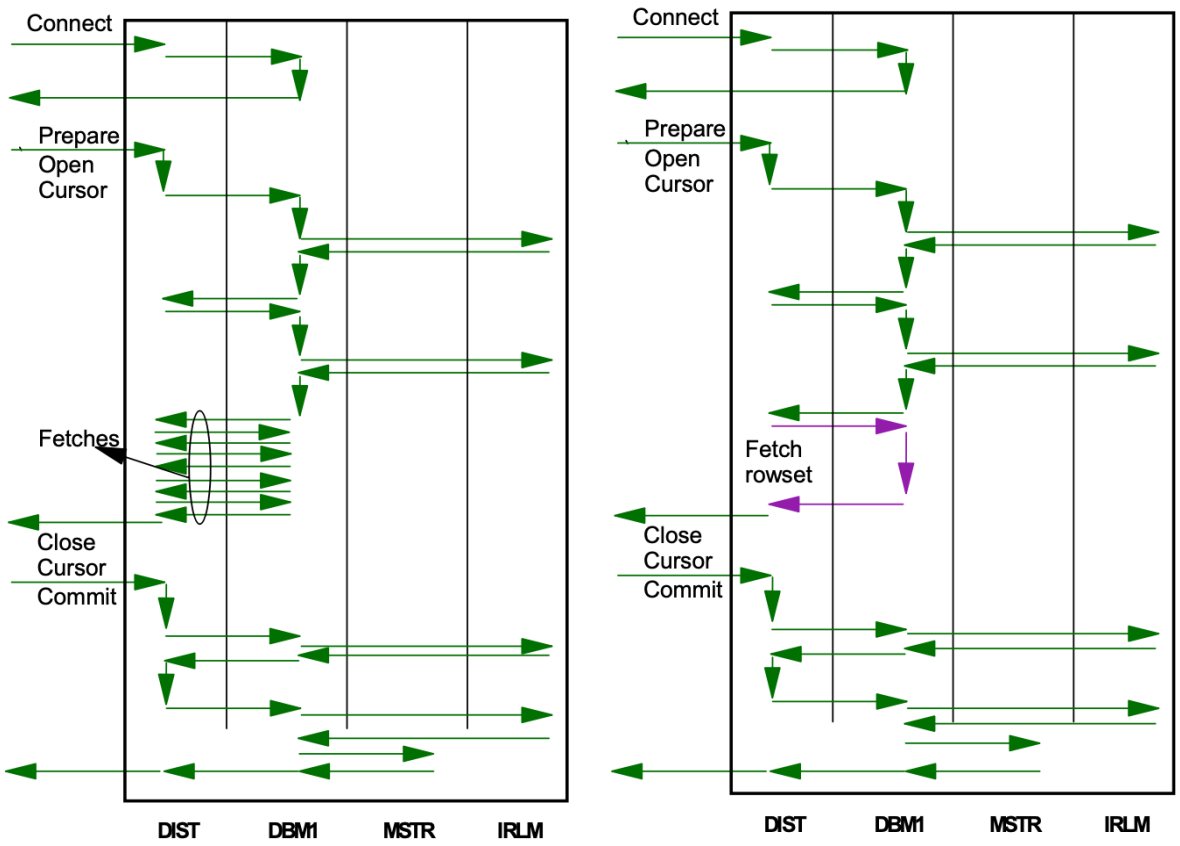
Why Fetch Multiple Rows?



Source – Db2 V8 Performance Redbook – Figure 3.1

Why MRF – For Distributed Applications

Multi-row fetch distributed before V8 vs. V8



Source – Db2 V8 Performance Redbook – Figure 7.1

Multi-Row Fetch

Multiple Row Fetch allows fetching of multiple rows into a host variable array

COBOL Example: Declare a CURSOR C1 and fetch 10 rows using a multi-row FETCH

```
 01 OUTPUT-VARS.
      05 LNAME OCCURS 10 TIMES.
        49 LNAME-LEN PIC S9(4)COMP-4 SY C.
        49 LNAME-DATA PIC X(15).
      05 EMP_NO PIX X(6) OCCURS 10 TIMES.

EXEC SQL
 DECLARE CS1 CURSOR WITH ROWSET POSITIONING FOR
  SELECT LASTNAME, EMPNO FROM EMP END-EXEC.

EXEC SQL
 OPEN CS1 END-EXEC.

EXEC SQL
 FETCH NEXT ROWSET FROM C1 FOR 10 ROWS
  INTO :LNAME, EMP_NO  END-EXEC.
```

Note: No arrays of structures – except for VARCHAR

Need – "WITH ROWSET POSITIONING" clause

Need – "NEXT ROWSET" and "FOR n ROWS" clauses

Multi-Row Fetch SQL Changes

```
DECLARE CURSOR CS1 FOR SELECT LNAME WITH ROWSET POSITIONING
          FROM EMP WHERE LNAME =:HV1;
OPEN CURSOR CS1 USING :HV1;

DO WHILE ( ton of conditions )        /* DSNTEP4 has example – search FETCH_NUM */
  FETCH NEXT ROWSET FOR 10 ROWS FROM CS1 INTO :LNAME;
  Rows_FETCHED = SQLERRD(3);
  FOR i = 1 to Rows_FETCHED
    /* Do something with LNAME(I);  */
  END;  /* FOR */
End;   /* DO While */

CLOSE CURSOR CS1;

COMMIT;
```

DSNTEP2 (Abridged FETCH LOOP)

```
/*****************************************************************/
/* FETCH ALL ROWS THAT FULFILL THE SELECT EXPRESSION. THE END OF  */
/* DATA AND SQL ERRORS ARE DENOTED BY A NON ZERO S QLCODE.        */
/*****************************************************************/
 DO WHILE( ( SQLCODE = ZERO             /* FETCH while no errors    */
           | SQLCODE = TRUNCWRN         /* or truncation only       */
           | SQLCODE = 595
           | (SQLCODE = ARITHWRN & TOLARTHWRN = YES)
           | (SQLCODE > 0 & TOLWARN ^= NO))                 /*@44*/
         & ( ROWS_FETCH = -1            /* and unresticted limit    */
           | ROWS_FETCH > RECNT ) );    /* or still under limit     */
   EXEC SQL FETCH C1 USING DESCRIPTOR :SQLDA; /* GET DATA          */
   RECNT = RECNT+ONE;                 /* COUNT THE # OF RECORDS FETCHED */
   IF TOLWARN ^= QUIET                /* If not suppress SQLWARNs   *@44*/
    & ( SQLWARN0 ^= ' '               /* AND WE GET A WARNING             */
      | (                             /* ... OR WE DIDN'T GET A WARNING */
         SQLCODE > 0                  /* ... BUT SQLCODE IS POSITIVE    */
         & SQLCODE ^= 100             /* ... AND NOT AT END OF DATA     */
         )
      ) THEN                          /* SAY SO                   @44*/
      DO;                                                /* KEF0059 */
        CALL PRINTCA;                 /* PRINT OUT THE SQLCA              */
      END;                            /*                    END KEF0059 */
    IF( ROWS_OUT = -1                 /* If output count is unrestricted*/
     | ROWS_OUT >= RECNT ) THEN       /* or <= actual output              */
       CALL CONROW;                   /* CONVERT FETCHED RECORD INTO @27*/
/*****************************************************************/
/* IF A PAGE FULL OF INFORMATION HAS BEEN COLLECTED THEN PRINT OUT */
/* THE INFORMATION. RESET THE NECESSARY VARIABLES.                */
/*****************************************************************/
    IF LASTROW+LCT >= MAXPAGLN THEN                     /* @PL56283 */
     DO;                              /* LASTROW + LCT = MAXPAGLN     */
       /* Process End of Page */
     END;                             /* END LASTROW + LCT = MAXPAGLN */
 END; /* END DO WHILE */
```

DSNTEP4 (Abridged FETCH Loop)

```
/*****************************************************************/
/* FETCH ALL ROWS THAT FULFILL THE SELECT EXPRESSION. THE END OF  */
/* DATA AND SQL ERRORS ARE DENOTED BY A NON ZERO SQLCODE.        */
/*****************************************************************/
 DO WHILE( ( SQLCODE = ZERO                    /* FETCH while no errors*/
           | SQLCODE = TRUNCWRN                /* or truncation only   */
           | SQLCODE = MULTWRN                 /* or mult warnings  @00*/
           | (SQLCODE = ARITHWRN & TOLARTHWRN = YES)
           | (SQLCODE > 0 & TOLWARN ^= NO))                  /*@44*/
         & ( ROWS_FETCH = -1                   /* and unresticted limit*/
           | ROWS_FETCH > RECNT )              /* or still under limit */
         & HALTPROC = NO );                    /* NO HALT PROCESS      */
  IF(ROWS_FETCH ^= -1 &                        /* prevent fetch more   */
    (ROWS_FETCH - RECNT < FETCH_NUM)) THEN     /* then                 */
       FETCH_NUM = ROWS_FETCH -RECNT;          /*ROWS_FETCH            */

  EXEC SQL FETCH NEXT ROWSET FROM C1 FOR :FETCH_NUM ROWS
           INTO DESCRIPTOR :SQLDA;             /* GET DATA             */
  FETCH_CNT = SQLERRD(THREE);
  RECNT= RECNT+FETCH_CNT;              /* COUNT THE # OF RECORDS FETCHED */
  IF TOLWARN ^= QUIET                  /* If not suppress SQLWARNs *@44. */
    & ( SQLWARN0 ^= ' '                /* and WE GET A WARNING           */
      | ( SQLWARN0 = ' '               /* ... OR WE DIDN'T GET A WARNING */
        & SQLCODE > 0                  /* ... BUT SQLCODE IS POSITIVE.   */
        & SQLCODE ^= 100               /* ... AND NOT AT END OF DATA     */
        )
      ) THEN                           /* SAY SO                         */
         DO;                           /* KEF0059                        */
                /* Print SQLCA */
         END;                          /* END KEF0059                    */
   CALL CONROW;                        /* CONVERT FETCHED RECORD INTO @27*/
END;                                   /* END DO WHILE                   */
```

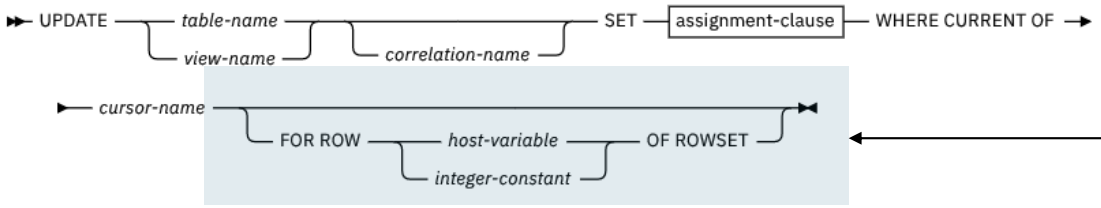
What is a ROWSET

- A group of rows for the result table of a query which are returned by a single FETCH statement
- A program controls how many rows are returned (i.e., size of the rowset)
  - Can be specified on the FETCH statement (maximum rowset size is 32767)
- Each group of rows are operated on as a rowset
  - Any locks held, are held across all rows of the rowset
  - UPDATE/DELETE WHERE CURRENT of can
    - Process a row within the row within rowset
    - Process all rows of the rowset
- Applications have the ability to intermix row positioned and rowset positioned fetches when a cursor is declared WITH ROWSET POSITIONING

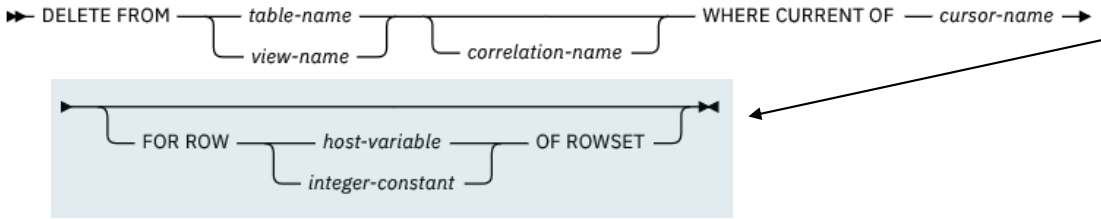
UPDATE/DELETE WHERE CURRENT OF

- When using ROWSET positioned FETCHES
  - The cursor is positioned on all rows of the ROWSET
  - UPDATE or DELETE WHERE CURRENT OF operates on:
    - All the rows of the ROWSET or
    - A single ROW of the ROWSET

positioned update:



positioned delete:



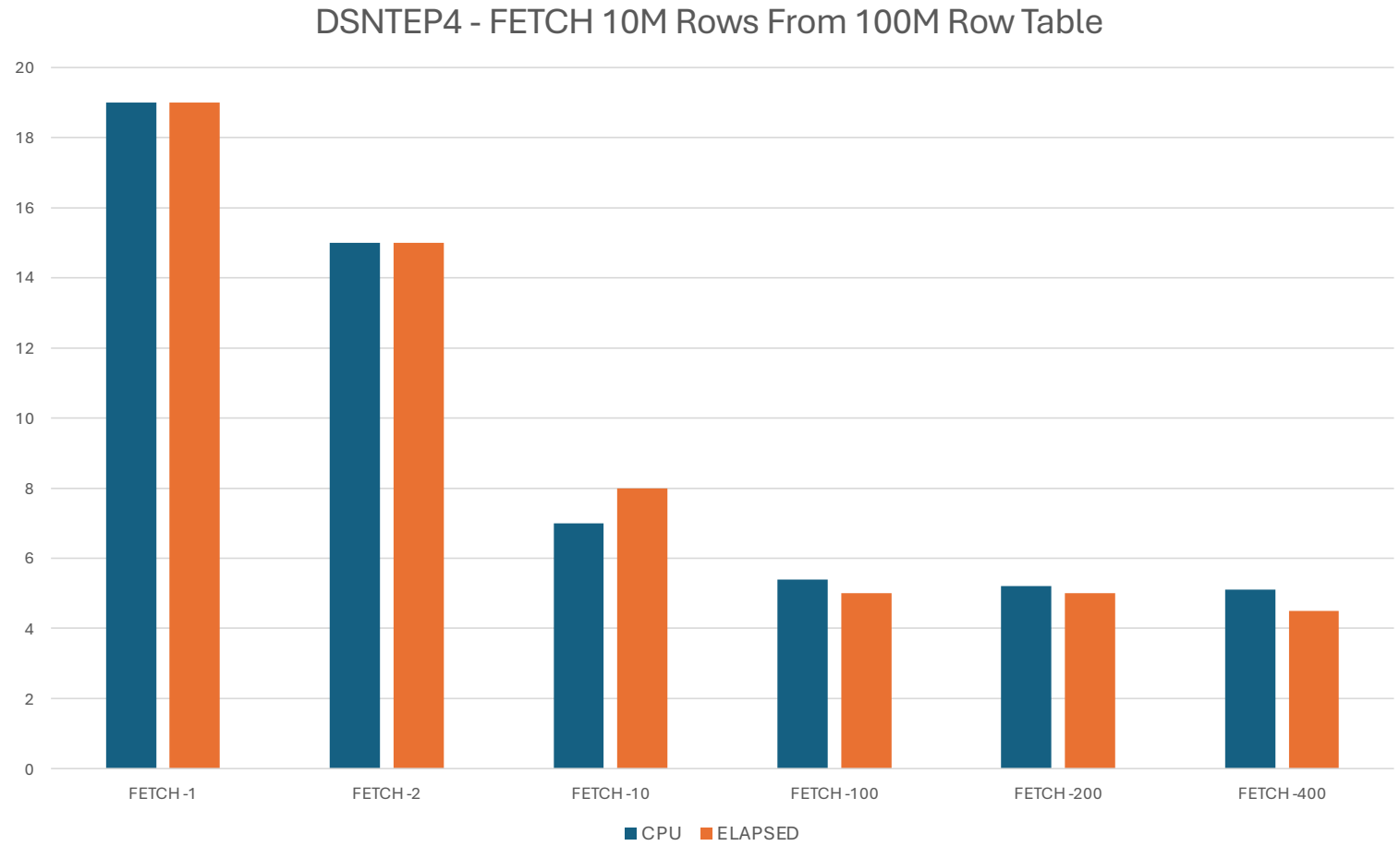
If the "FOR ROW m OF ROWSET" clause is not specified, all rows of the ROWSET are processed by the UPDATE or DELETE statement

Limiting Rows

- Newer Applications are written to provide data to multiple screen sizes
  - iPhone, iPad, Desktop
- Prior to Db2 12
  - FETCH FIRST n ROWS ONLY clause only supports literal
    SELECT LASTNAME FROM EMP FETCH FIRST 5 ROWS ONLY;
    - Requires one statement in statement cache for each screen size you want to support
  - ROWSET POSITIOINING
    FETCH NEXT ROWSET FOR m ROWS
- With Db2 12
  - FETCH FIRST m ROWS ONLY
  - Support specification of a Host Variable or Parameter Marker that is castable to BIGINT

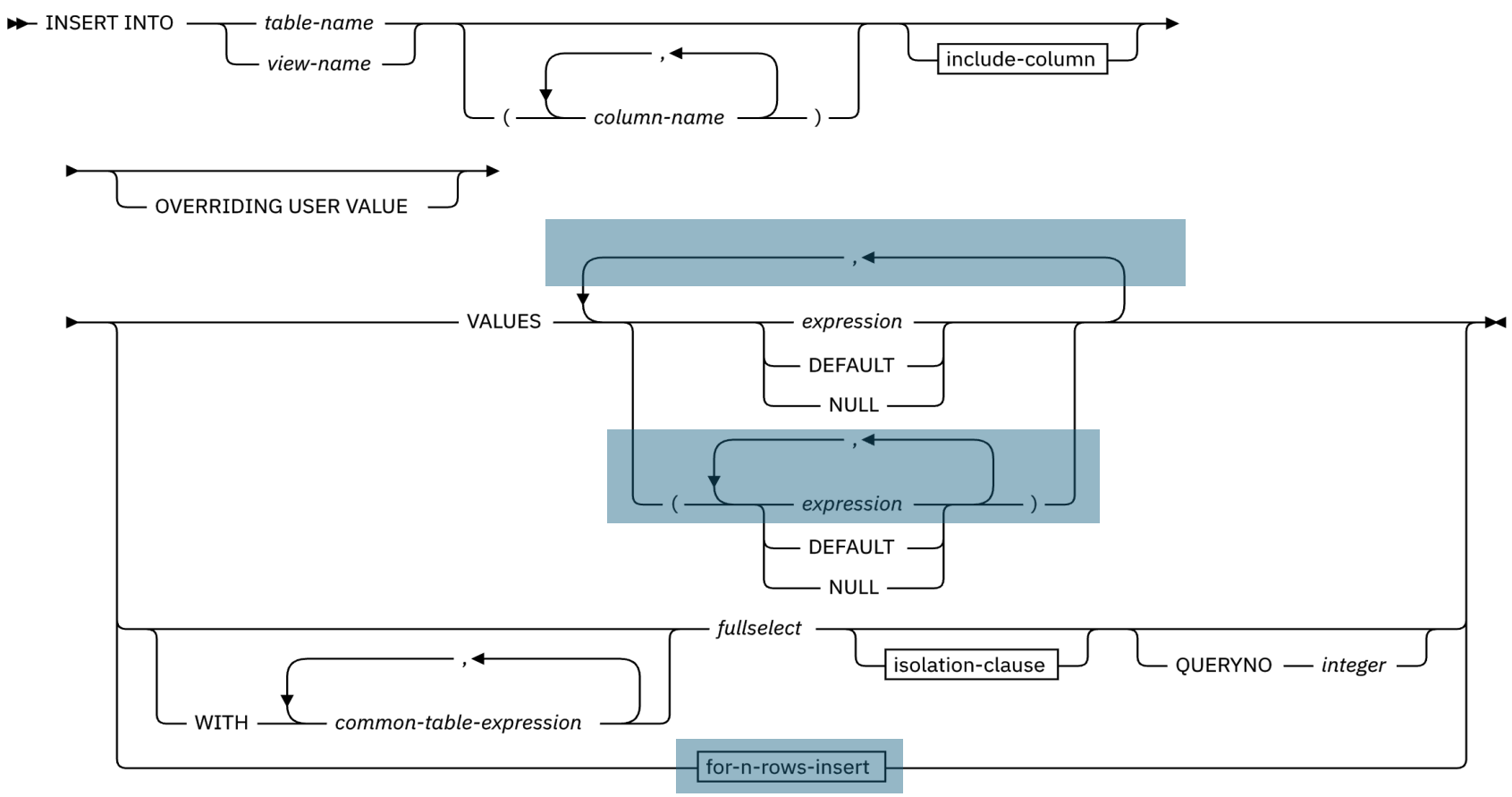
SELECT LASTNAME FROM EMP FETCH FIRST ? ROWS ONLY;

Multi-Row FETCH Performance

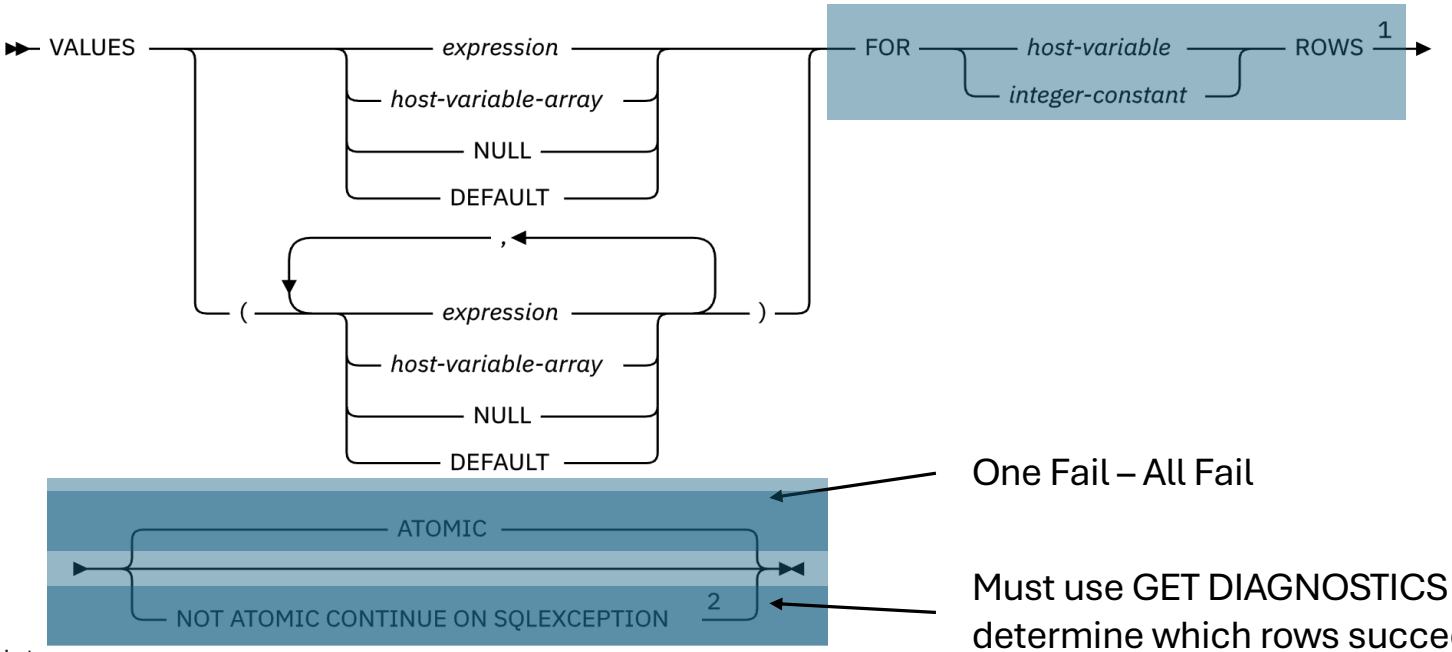



Multi-Row INSERT
IDUG
2026 Australia Db2 Tech Conference

INSERT


INSERT INTO
table-name
view-name
,
(column-name)
include-column
OVERRIDING USER VALUE
,
VALUES
expression
DEFAULT
NULL
,
(expression)
DEFAULT
NULL
fullselect
,
WITH common-table-expression
isolation-clause
QUERYNO integer
for-n-rows-insert

INSERT (cont)



One Fail – All Fail

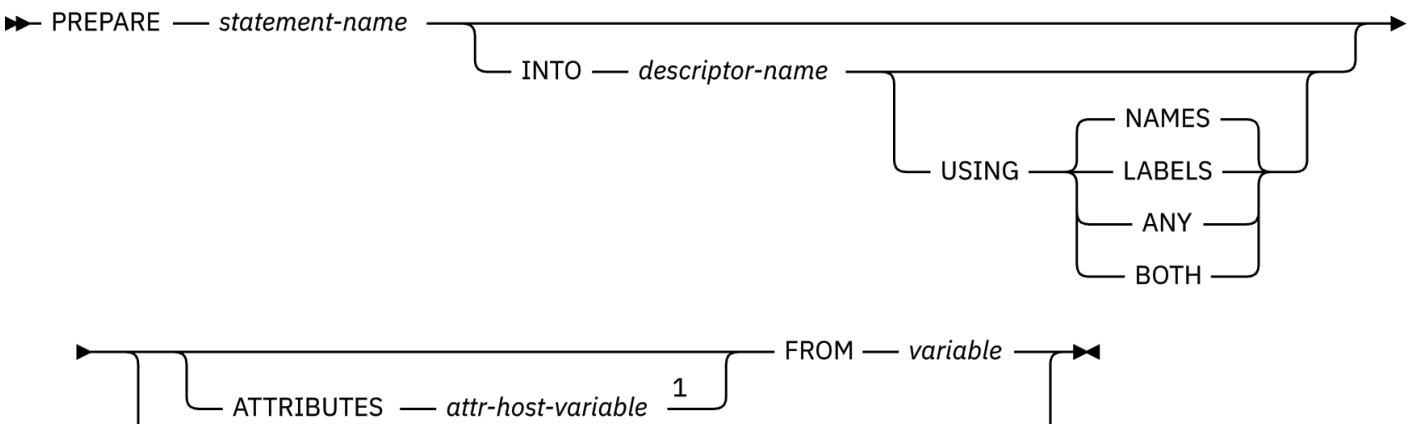
Must use GET DIAGNOSTICS to determine which rows succeeded/failed

Notes:

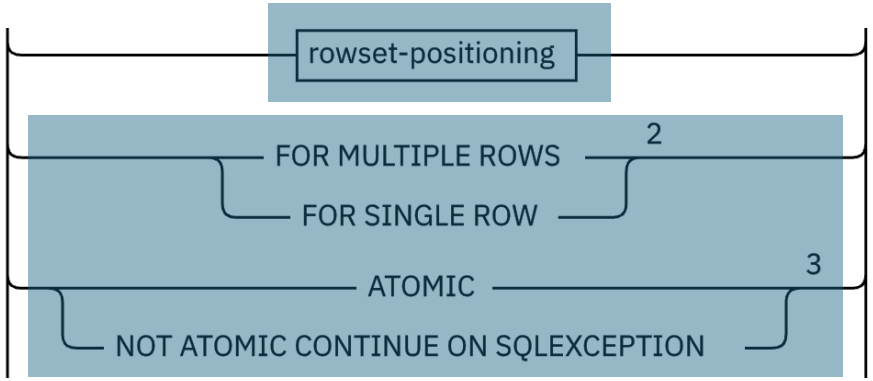
[1] The FOR *n* ROWS clause must be specified for this form of a static INSERT statement. However, this clause is optional for a dynamic INSERT statement. For a dynamic statement, the FOR *n* ROWS clause can be specified on the EXECUTE statement.

[2] The ATOMIC or NOT ATOMIC CONTINUE ON SQLEXCEPTION clauses can be specified for a static multiple-row-insert. However, this clause must not be specified for a dynamic INSERT statement. For a dynamic statement, the ATOMIC or NOT ATOMIC CONTINUE ON SQLEXCEPTION clause is specified as an attribute on the PREPARE statement.

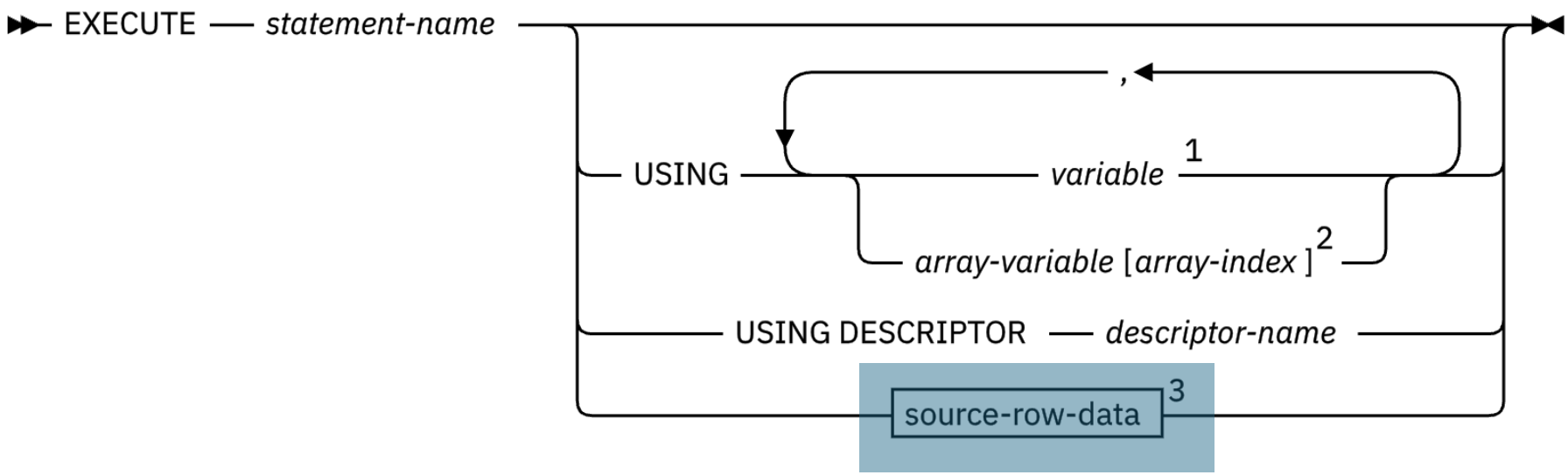
PREPARE



attribute-string



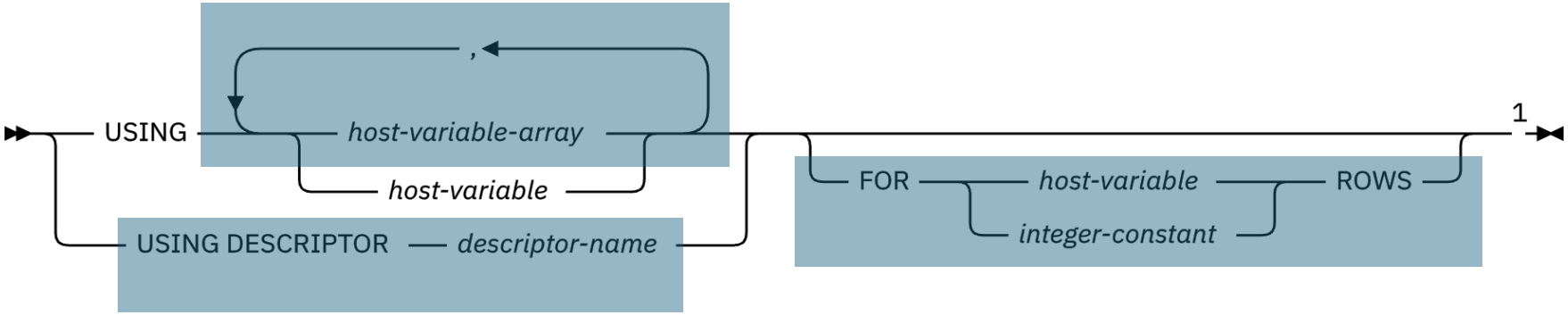
Execute



[3] This option can be specified only when *statement-name* refers to a dynamic INSERT or MERGE statement that is prepared with FOR MULTIPLE ROWS and is specified as part of the ATTRIBUTES clause on the PREPARE statement.

Execute (cont)

source-row-data:



Notes:

[1] The FOR n ROWS clause is required on the EXECUTE statement if it is not specified as part of the MERGE statement and a host-variable array is specified. The FOR n ROWS clause is also required if MERGE is used with multiple rows of source data. For an INSERT statement, the FOR n ROWS clause can only be specified for a dynamic statement that contains only a single multiple-row INSERT statement.

Multi-Row INSERT Syntax

- Static

  INSERT INTO T1 (C1,C2) VALUES (:HV_ARR1,:HV_ARR2) FOR :HV ROWS

- Dynamic

  nrows = 100

  attr_str = ‘FOR MULTIPLE ROWS NOT ATTOMIC CONTINUE ON SQLEXCEPTION’

  stmt_txt = ’ INSERT INTO T1 (C1,C2) VALUES (?,?)’

  PREPARE stmt ATTRIBUTES :attr_str FROM :stmt_txt;

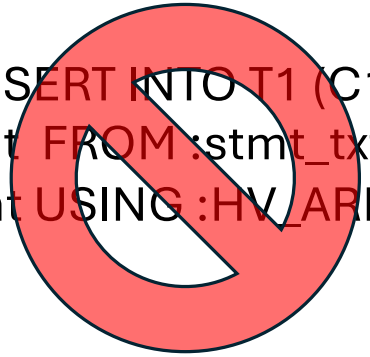
  EXECUTE stmt FOR :nrows USING :HV_ARR1, :HV_ARR2;

What About?

stmt_txt = ’ INSERT INTO T1 (C1,C2) VALUES (?,?) FOR 100 ROWS’

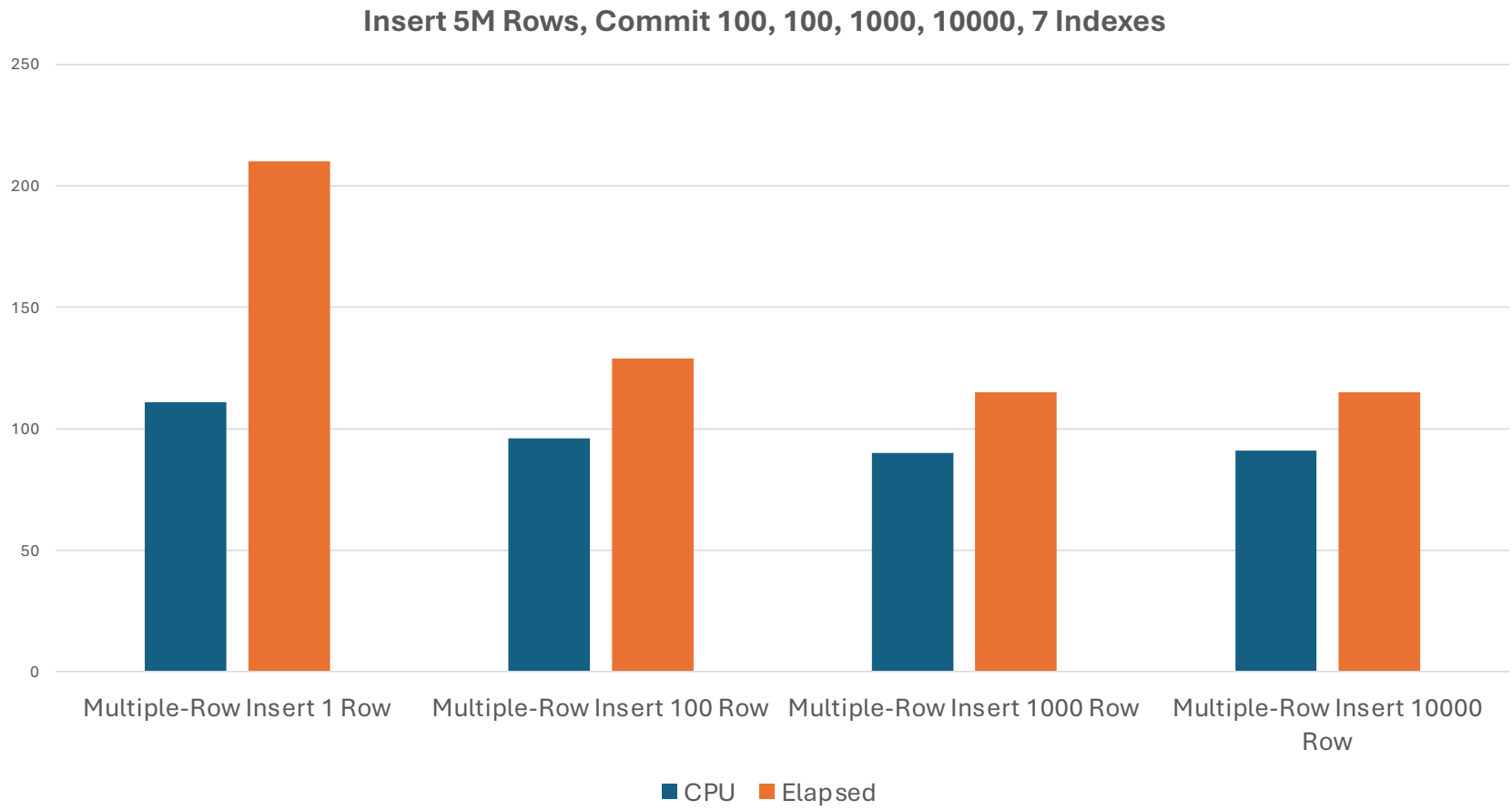
PREPARE stmt FROM :stmt_txt;

EXECUTE stmt USING :HV_ARR1, :HV_ARR2;

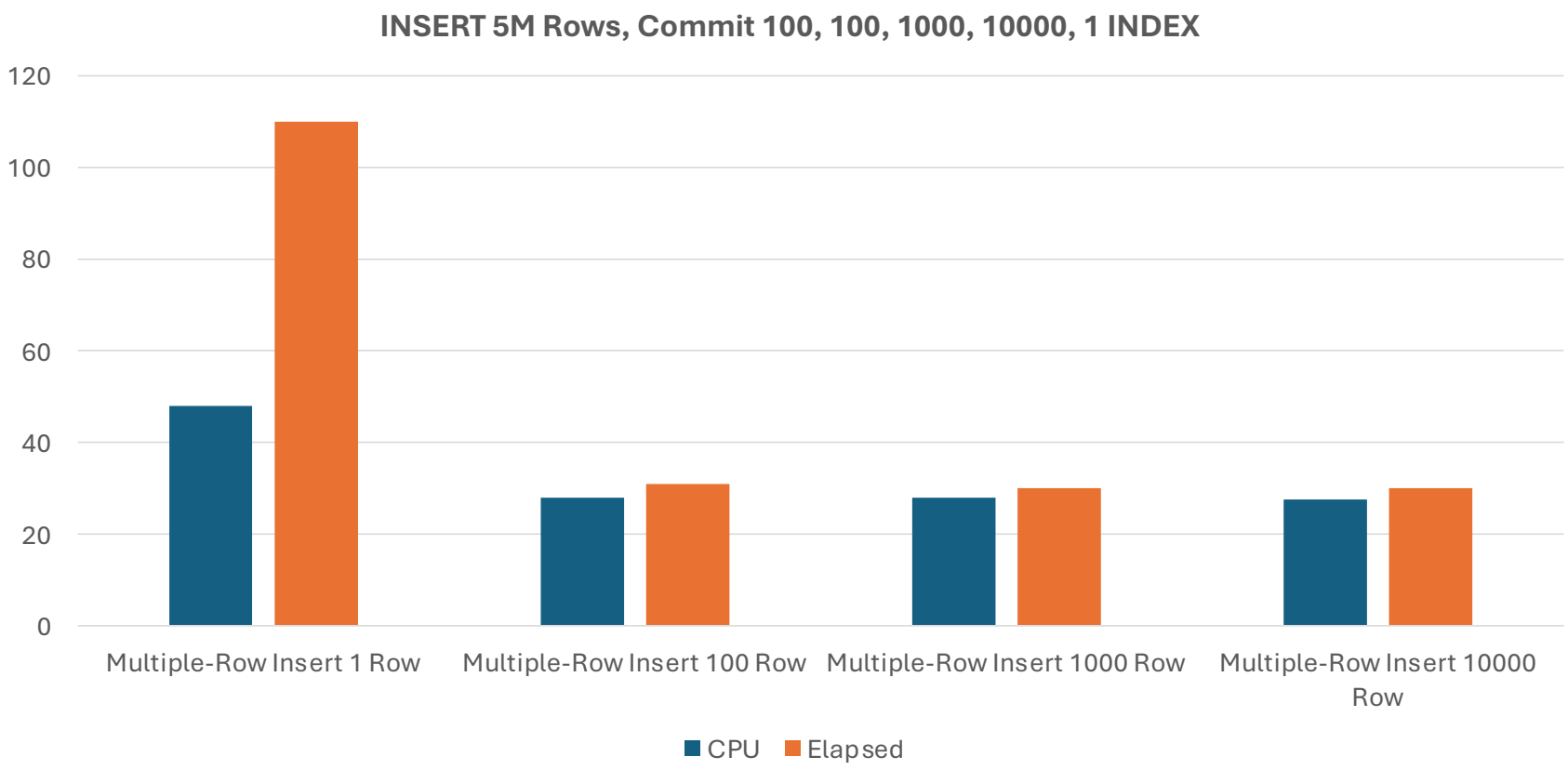


-**20186** A CLAUSE SPECIFIED FOR THE DYNAMIC SQL STATEMENT BEING PROCESSED IS NOT VALID Explanation A clause was not valid for one of the following reasons: **On PREPARE statements: A FOR SINGLE ROW or FOR MULTIPLE ROWS clause was specified.**

Multi-Row Array Insert



Multi-Row Array Insert



Multi-Rows VALUES INSERT (APPLCOMPAT V13R1M506)

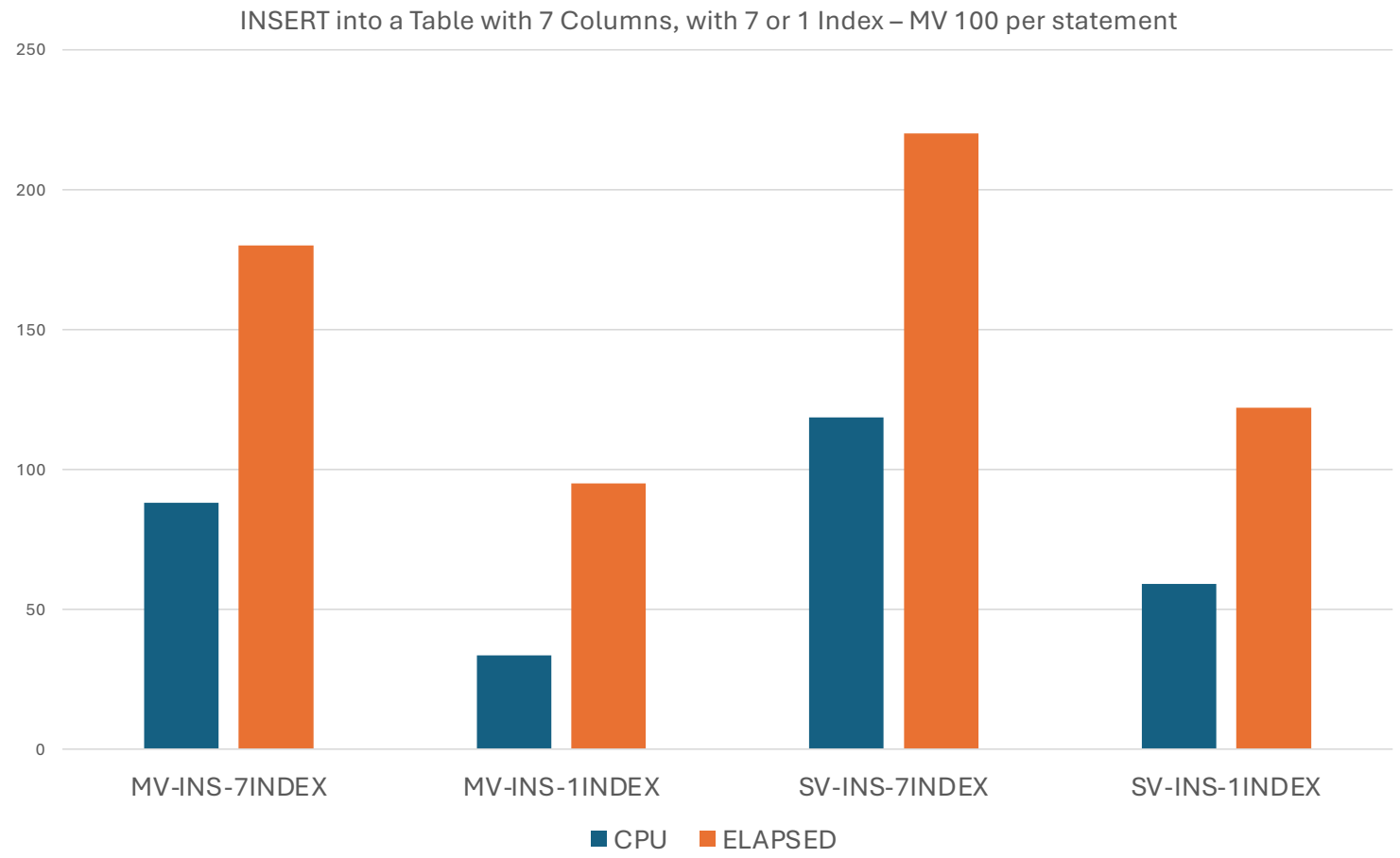
- Single Row INSERT

```
INSERT INTO DSN8D10.EMP
(EMPNO,FNAME,MI,LNAME,WORKDEPT,PHONENO, HIREDATE,JOB,EDLEVEL,SEX,BDATE,
SALARY,BONUS,COMM)
  VALUES
('000205','MARY','T','SMITH','D11','2866', '1981-08-10','ANALYST',16,'F','1956-05-22',16345,500,2300);
```

- Multi-Row INSERT

```
INSERT INTO DSN8D10.EMP
(EMPNO,FNAME,MI,LNAME,WORKDEPT,PHONENO, HIREDATE,JOB,EDLEVEL,SEX,BDATE,
SALARY,BONUS,COMM)
  VALUES
('000206','ELIZABETH','T','GRACE','D11','2866', '2023-02-17','ANALYST',16,'F','1975-06-28',
16345,500,2300), ('000207','JACK','Q','JOHNSON','D11','2867', '2023-08-10','ANALYST',16,'M','1979-07-
22', 16345,500,2300), ('000208','JENNIFER','K','WHITE','D11','2868', '2023-08-10','ANALYST',16,'F','1980-
08-13', 16345,500,2300);
```

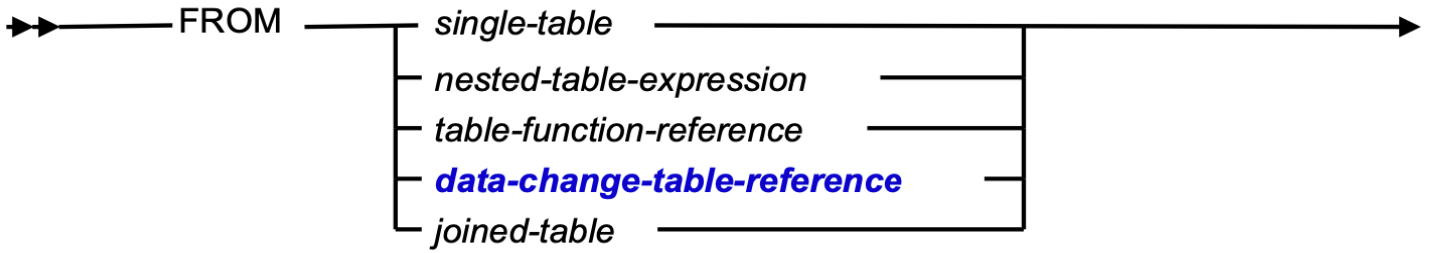
INSERTING 5M Rows

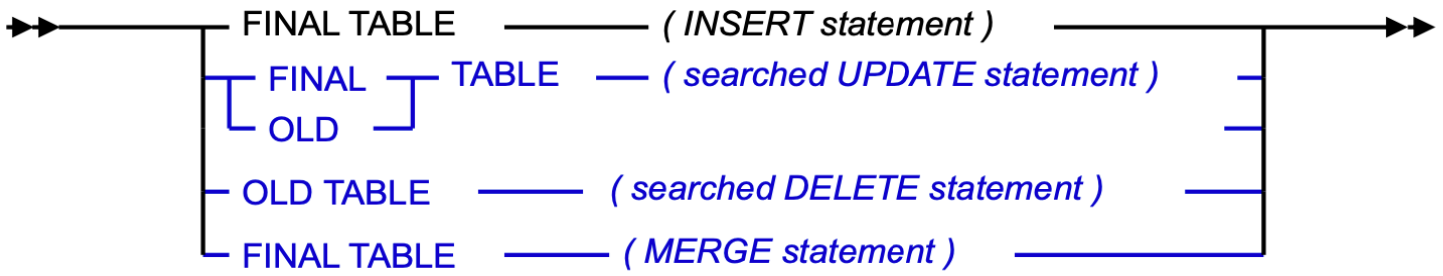



SELECT FROM – Multi-Row
IDUG
2026 Australia Db2 Tech Conference

SELECT FROM INSERT, UPDATE, DELETE, MERGE

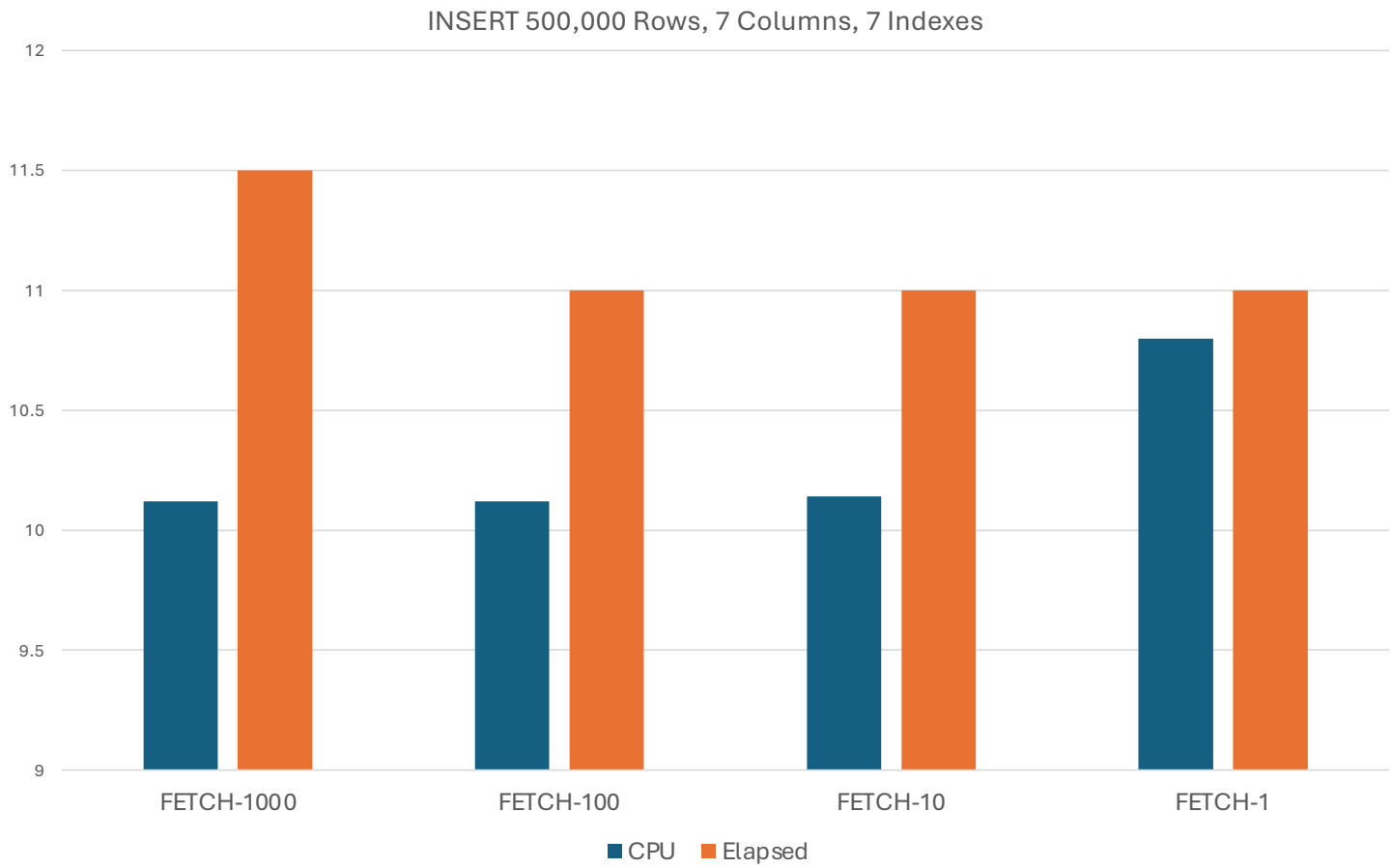
The FROM clause of a SELECT statement will allows an INSERT, UPDATE, DELETE, or MERGE statement:



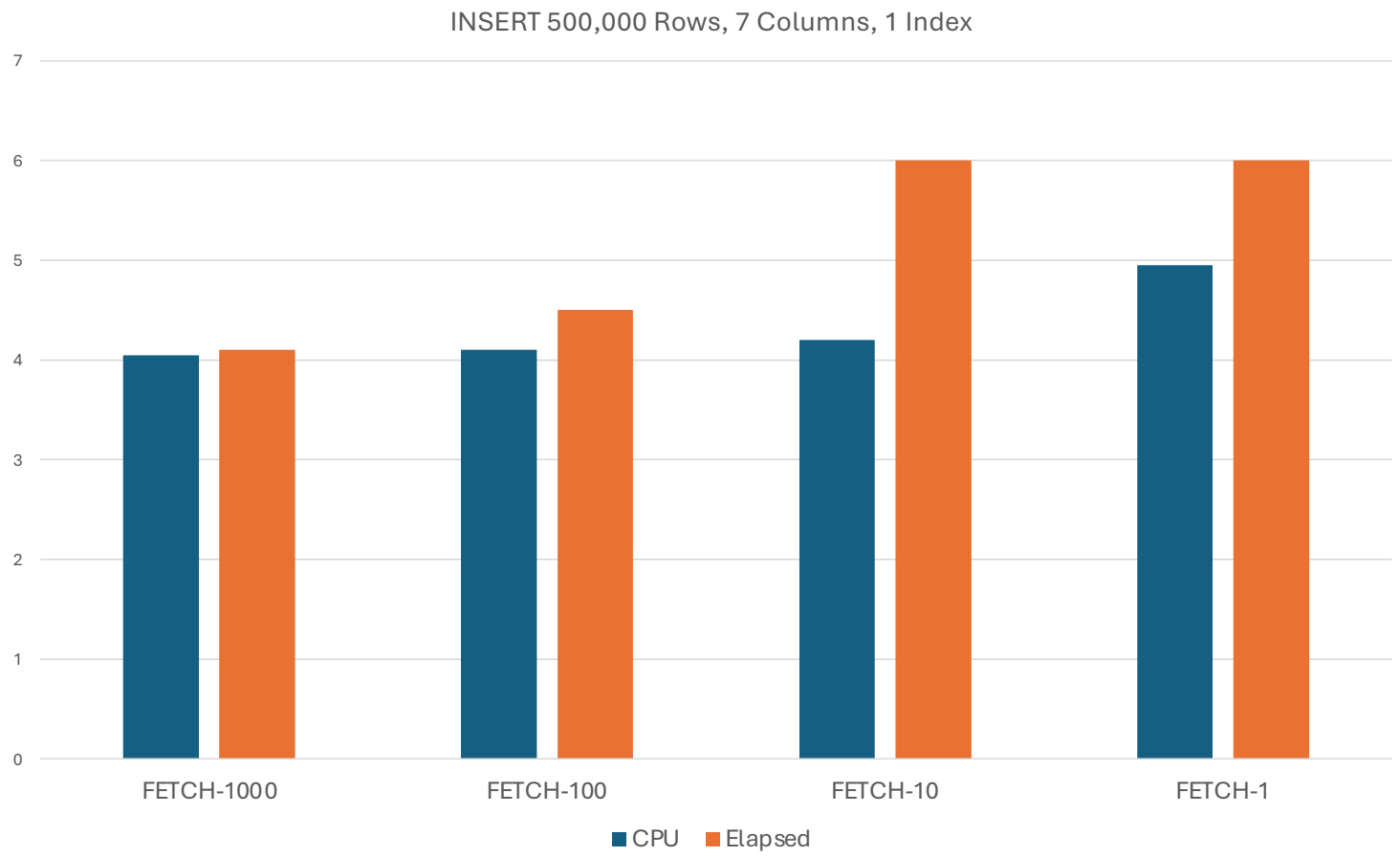



```
EXEC SQL DECLARE CS1 CURSOR WITH ROWSET POSITIONING FOR
 SELECT C1, C2, C3, C4, C5, C6, C7   FROM FINAL TABLE
  (INSERT INTO DATA2  (C1, C2, C3, C4, C5, C6, C7)
    SELECT * FROM DATA1);
```

SELECT FROM … FINAL TABLE (INSERT…)



SELECT FROM … FINAL TABLE (INSERT…)



SELECT from UPDATE - Example

If a user would like to know the new salary of each employee who is at level 'OPERATOR' and received a salary increase, they could use FINAL TABLE with a searched UPDATE:

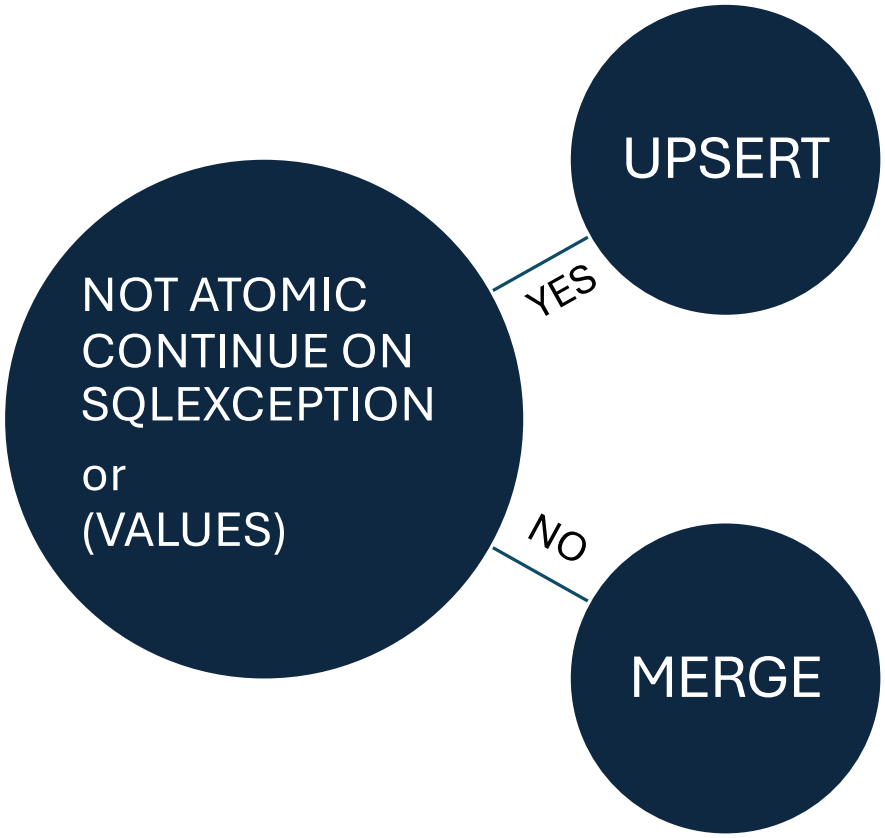
```
DECLARE CS1 CURSOR WITH ROWSET POSITIONING FOR
 SELECT salary
 FROM FINAL TABLE
    (UPDATE emp
     SET salary = salary * 1.05
 WHERE level = 'OPERATOR');

FETCH NEXT ROWSET FOR 10 ROWS FROM CS1 INTO :salary_arr;
```

MERGE (aka UPSERT)

MERGE

- Original MERGE – aka UPSERT
- Extended MERGE – ANSI SQL Standard based MERGE
  - table-reference as an additional way of specifying source data for the MERGE statement
  - Multiple MATCHED clauses
  - Enhanced predicates with MATCHED or NOT MATCHED
  - DELETE operations
  - IGNORE and the SIGNAL statement as actions

Original MERGE and Enhanced MERGE



Example Db2 9 MERGE – aka UPSERT

MERGE Data from an Application into a Table

```
MERGE INTO ACCOUNT AS TRG
    USING (VALUES (:acct_hv, :amount_hv) FOR :hv ROWS) AS SRC (ACCT, AMT)
    ON (TRG.ACCT = SRC.ACCT)
WHEN MATCHED THEN
        UPDATE SET TRG.AMT = TRG.AMT + SRC.AMT
WHEN NOT MATCHED THEN
        INSERT (ACCT, AMT) VALUES (SRC.ACCT, SRC.AMT)
NOT ATOMIC CONTINUE ON SQLEXCEPTION;
```

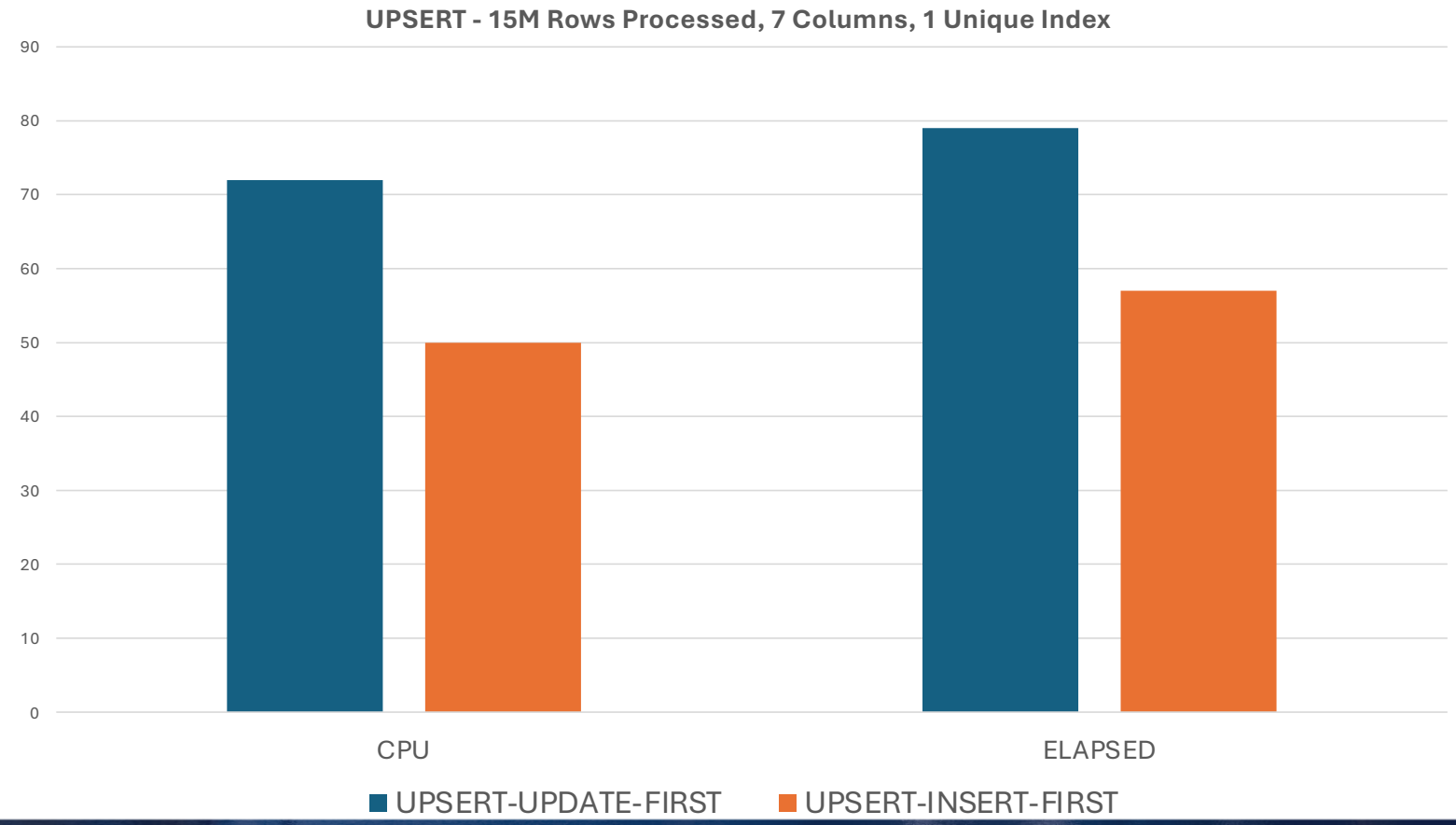
SRC.ACCT	SRC.AMT
13579	1000
24680	1500
24680	2500
95136	835
94578	5996

TRG.ACCT	TRG.AMT
10203	3000
24680	2000
95223	2000
96136	3000
94578	10000

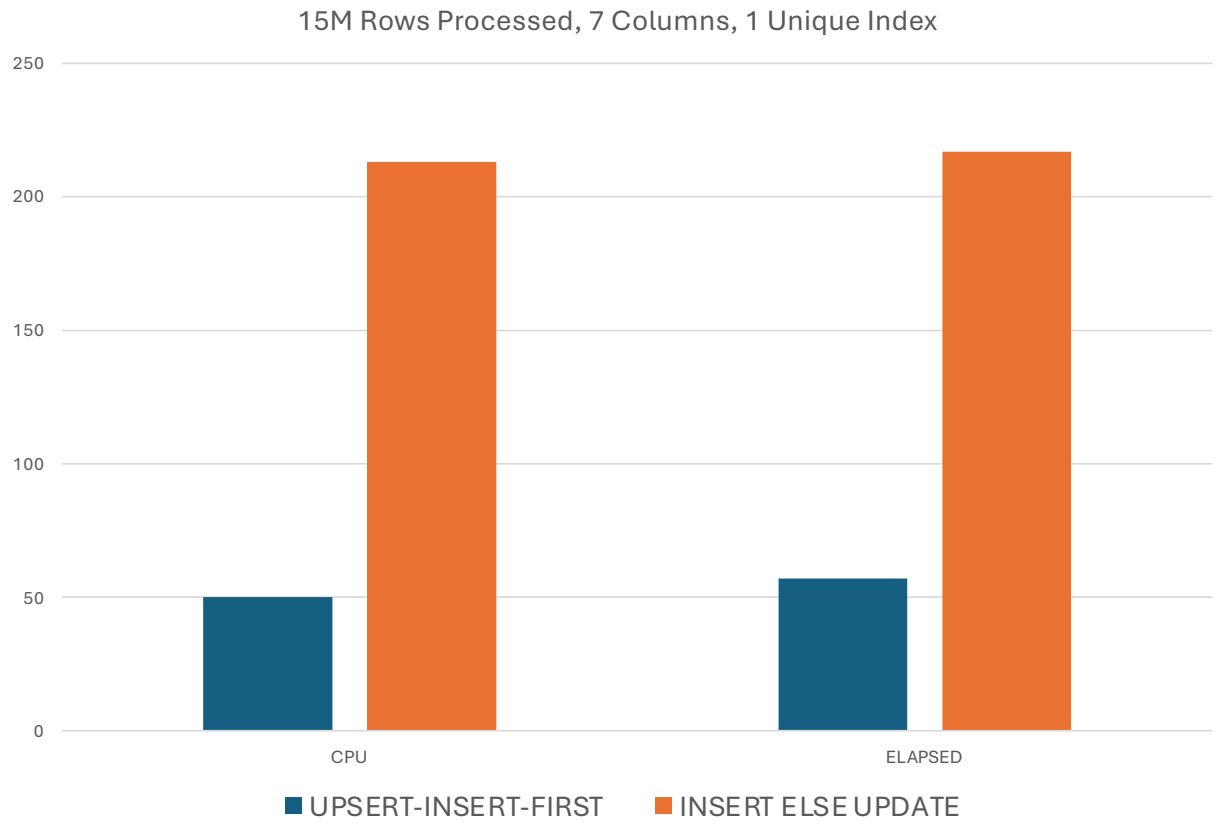
MERGE

TRG.ACCT	TRG.AMT
10203	3000
13579	1000
24680	6000
95223	2000
95136	3835
94578	15996

MERGE (UPSERT) - Order Matters



UPSERT –vs- Application



MERGE with FULLSELECT (not multi-row with an application)

MERGE Data from a Staging Table into another Table with additional logic

```
MERGE INTO TRG AS TRG
  USING ( SELECT ACCT, AMT FROM SRC) AS SRC ON TRG.ACCT = SRC.ACCT
WHEN MATCHED AND GV = '' THEN
 UPDATE SET TRG.AMT = TRG.AMT + SRC.AMT
WHEN MATCHED AND GV = 'REPLACE' THEN
 UPDATE SET TRG.AMT = SRC.AMT
WHEN MATCHED AND GV = 'DELETE' THEN
 DELETE
WHEN MATCHED AND GV = 'SS' THEN
 SIGNAL SQLSTATE '75000' SET MESSAGE_TEXT = 'SS Failure'
WHEN NOT MATCHED THEN
 INSERT (ACCT, AMT) VALUES (SRC.ACCT, SRC.AMT)
ELSE IGNORE;
```

Example MERGE with FULLSELECT

MERGE Data from a Staging Table into another Table

```
MERGE INTO TRG AS TRG
   USING (SELECT ACCT, AMT FROM SRC) AS SRC ON TRG.ACCT = SRC.ACCT
WHEN MATCHED THEN UPDATE
   SET TRG.AMT = TRG.AMT + SRC.AMT
WHEN NOT MATCHED THEN
   INSERT (ACCT, AMT) VALUES (SRC.ACCT, SRC.AMT);
```

SRC.ACCT	SRC.AMT
13579	1000
24680	1500
95223	2500
95136	835
94578	5996

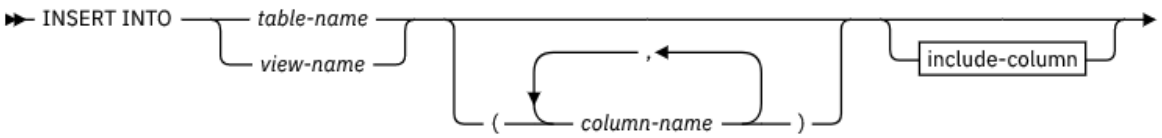
TRG.ACCT	TRG.AMT
10203	3000
24680	2000
95223	2000
96136	3000
94578	10000

MERGE

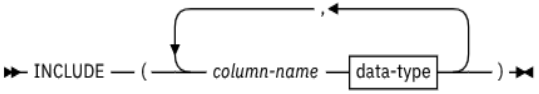
TRG.ACCT	TRG.AMT
10203	3000
13579	1000
24680	3500
95223	4500
95136	3835
94578	15996

INCLUDE columns

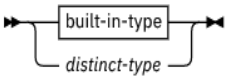
- A list of column(s) to be included in the result table of an INSERT, UPDATE, DELETE or MERGE statement.
- The include columns are only available if the
  - DELETE / INSERT / UPDATE / MERGE statement is nested in the FROM clause of a select-statement or SELECT INTO statement.
- Example Syntax for INSERT



include-column:



data-type:



SELECT … FROM … MERGE with INCLUDE

Source

S.id	S.amt
1	30
5	10
10	40
5	20
1	50

include column

Returned rows

T.id	balance	status
1	1030	upd
5	10	ins
10	540	upd
5	30	upd
1	1080	upd

Account – target table

T.id	balance
1	1000
10	500
200	600
300	300
315	100
500	4000
…	

```
SELECT id, balance, status
FROM FINAL TABLE (
MERGE  INTO account AS T    INCLUDE (status, char(3) )
USING VALUES (:hv_id, :hv_amt) FOR 5 ROWS  AS S (id,amt)
ON T.id = S.id
WHEN MATCHED THEN
   UPDATE SET balance = T.balance + S.amt, status = 'upd'
WHEN NOT MATCHED THEN
   INSERT (id, balance) VALUES (S.id, S.amt, 'ins' )
NOT ATOMIC CONTINUE ON SQLEXCEPTION      )
```

Account - after

T.id	balance
1	1080
5	30
10	540
200	600
300	300
315	100
500	4000
…	


Conclusion
IDUG
2026 Australia Db2 Tech Conference

Conclusion

- Multi-Row Processing Significantly improves performance
  - CPU and ELAPSED Time Performance improves
  - Multi-Row INSERT has less significant improvement
  - Both MRI and MRF are affected by the number of columns processed
  - Performance gains are Db2 processing only
    - An application that spends 50% in Db2 and 50% in Application logic/data processing will see gains only in the Db2 side
- Multi-Row Processing can either simplify or complicate programming
  - MERGE usually simplifies programming
  - Multi-Row INSERT and FETCH typically require slightly more complicated programming


THANK YOU
Chris Crone
cjc@broadcom.com
BROADCOM