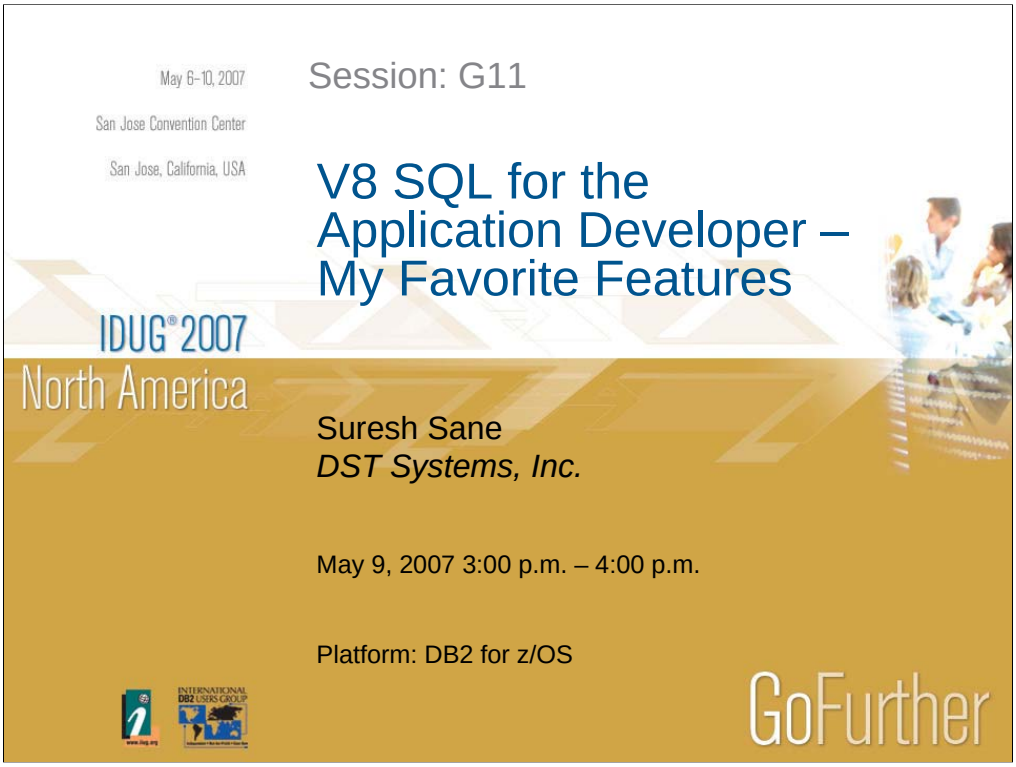
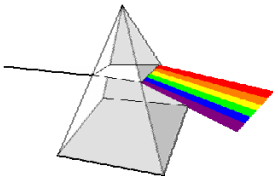


This presentation provides an in-depth look at the top few important DB2 V8 SQL features. We present a clear and detailed explanation of each feature showing the pros and cons along with benchmarks. We present real-life case studies that can serve as models for your shop. We will not just scratch the surface but cut through the hype and explain what works (and what does not!) based on hands-on experience.

Session Outline

1. **Rev it up!** ➡ **(Misc performance enhancements)**
2. **Bigger bite** ➡ **(Multi-row Fetch & Insert)**
3. **What's wrong?** ➡ **(GET DIAGNOSTICS)**
4. **Next Please!** ➡ **(Sequences, Identity columns etc)**
5. **Play it again, Sam!** ➡ **(Recursive SQL)**



2

An outline of what we will discuss in this session:

Rev it up! (Misc Performance features)

- SELECT INTO with ORDER BY
- Non-correlated EXISTS
- IN list processing

Bigger bite (Multi-row fetch/insert)

- Host language issues
- Error handling
- Benchmarks

What's wrong? (GET DIAGNOSTICS)

- Obtaining basic info
- Dealing with multi-row insert

Next please! (Sequences, Identity columns and get_unique)

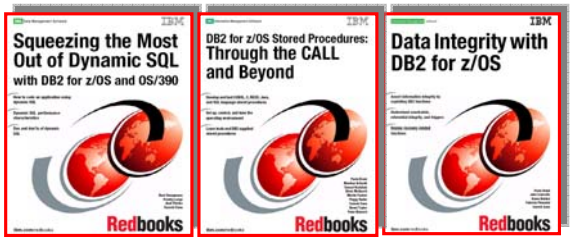
- IDENTITY vs. Sequences
- Caching and order
- Benchmarks

Play it again, Sam! (Recursive SQL)

- Business case studies
- Potential "gotcha's

About the Instructor

Suresh Sane



- **Co-author-IBM Redbooks**
  - SG24-6418, May 2002
  - SG24-7083, March 2004
  - SG24-7111, July 2006
- **Educational seminars and presentations at IDUG North America, Asia Pacific, Canada and Europe**
- **IDUG Solutions Journal article – Winter 2000**
- **Numerous DB2 courses at various locations**
- **IBM Certified Solutions Expert for both platforms for Application Development and Database Administration**

3

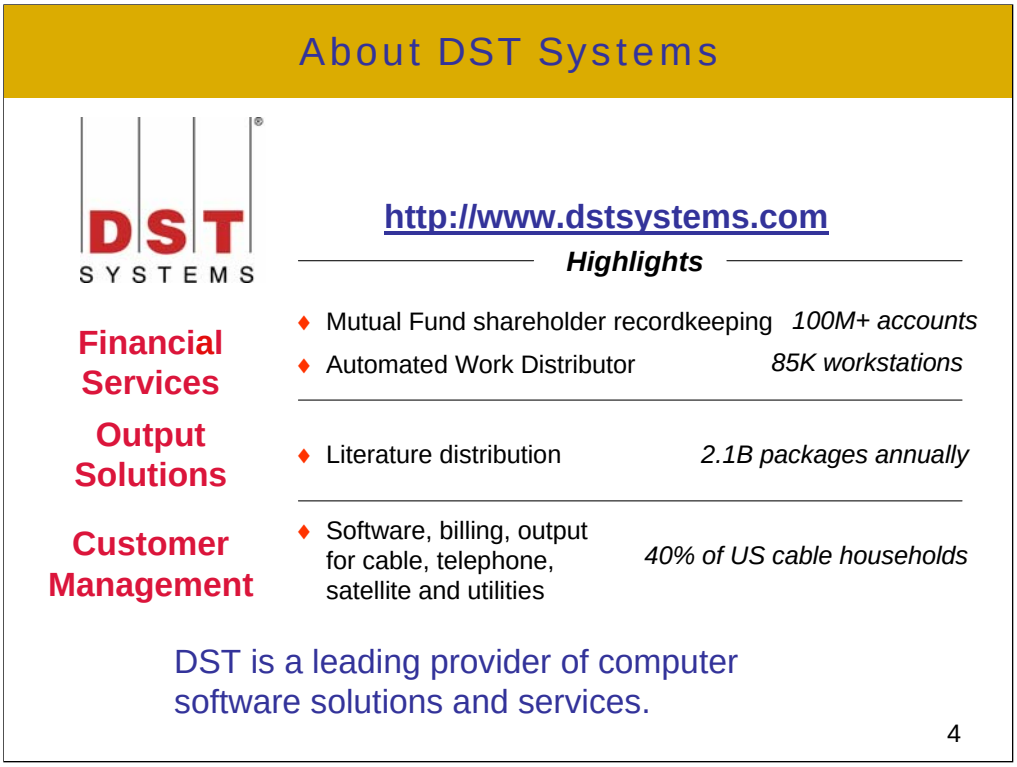
Suresh Sane works as a Database Architect at DST Systems in Kansas City, MO. He provides strategic direction for deployment of database technology at DST and has overall responsibility for the DB2 curriculum for about 1,500 technical associates.

Contact Information:

sssane@dstsystems.com

Suresh Sane
DST Systems, Inc.
1055 Broadway
Kansas City, MO 64105
USA

(816) 435-3803



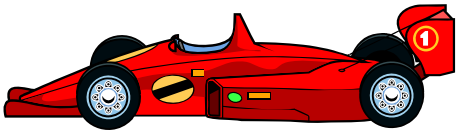
If you have ever invested in a mutual fund, have had a prescription filled, or are a cable or satellite television subscriber, you may have already had dealings with our company.

DST Systems, Inc. is a publicly traded company (NYSE: DST) with headquarters in Kansas City, MO. Founded in 1969, it employs about 12,000 associates domestically and internationally.

The three operating segments - Financial Services, Output Solutions and Customer Management - are further enhanced by DST's advanced technology and e-commerce solutions.

Where Are We?

1. **Rev it up!**	**(Misc performance enhancements)**
2. **Bigger bite**	**(Multi-row Fetch & Insert)**
3. **What's wrong?**	**(GET DIAGNOSTICS)**
4. **Next Please!**	**(Sequences, Identity columns etc)**
5. **Play it again, Sam!**	**(Recursive SQL)**



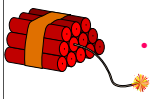
5

The first topic of this session – we will cover various enhancements that improve performance. Specifically, we will discuss SELECT INTO with ORDER BY, Non-correlated EXISTS and IN list processing.

SELECT INTO with ORDER BY

- Up to V6 – SELECT INTO could not guarantee that a single row would be returned
  - Checking for sqlcode -811 (“more than 1 row”), followed by opening a cursor
  - Using the row returned with -811 was dangerous – contents of host variables “un-predictable” but generally returned the first row
- V7 – FETCH FIRST 1 ROW ONLY introduced
  - guarantee that a single row would be returned
  - But no way influence which row
- V8 – SELECT INTO with ORDER BY introduced
  - Provides control over which row is returned
  - Up to 30% faster than cursor open-fetch-close
  - FETCH FIRST 1 ROW ONLY –a must!
    - *Specifying anything else other that 1 results in DSNH490I in pre-compile error*
    - *Leaving it out results in -811 and the 2nd row (usually) – **beware!***

```
EXEC SQL
   SELECT          COL01
   INTO            :WS-COL01
   FROM            MRF
   WHER COL01 > 100
   ORDER BY        COL01
   FETCH FIRST 1 ROW ONLY
END-EXEC.
```



6

This feature provides the ability to get a single row fast and still control which of many qualifying rows is returned – all without using a cursor.

Non-correlated EXISTS

- Correlated EXISTS subquery always stops when one qualifying row is found
- Up to V7 – Non-correlated EXISTS subquery built the set of all qualifying rows into a workfile
  - More processing time
  - More space for workfile
- V8 – Non-correlated EXISTS subquery stops as soon as one qualifying row is found
- Performance improvement depends on
  - Number of qualifying rows in the subselect (many → more improvement)
  - How quickly first row can be found (indexed → more improvement)

```
EXEC SQL
    SELECT 'Y'
    INTO   :WS-JUNK
    FROM   MRF
    WHERE  EXISTS
       (SELECT COL01
          FROM   MRF
          WHERE  COL01 > 100)
END-EXEC.
```

7

Non-correlated exists will also stop earlier now (previously needed a correlated subselect).

IN list processing

- 2 enhancements available in V7 via APARs but activated by INLISTP ZPARM
  - Default in V7 is 0 (both disabled)
  - Default in V8 is 50 (both enabled)
- APAR PQ73454 – Predicate pushdown
  - Pushdown of predicates into Nested Table Expression (NTE) or Materialized View (MV)
  - Potential index usage for NTE or MV
  - Example follows
- APAR PQ73749 – Correlated subquery transformation
  - Predicate transitive closure to allow predicate "pull up"
  - Better filtering in outer query
  - Example follows
- APAR PQ68662 – (**not new in V8**)
  - Choice of index vs. TS scan – now accounts for bufferpool residency of needed pages from previous INLIST processing

8

We will explore the first two in detail in the following slides. The third feature is not new in V8 but shown here for reference purposes.

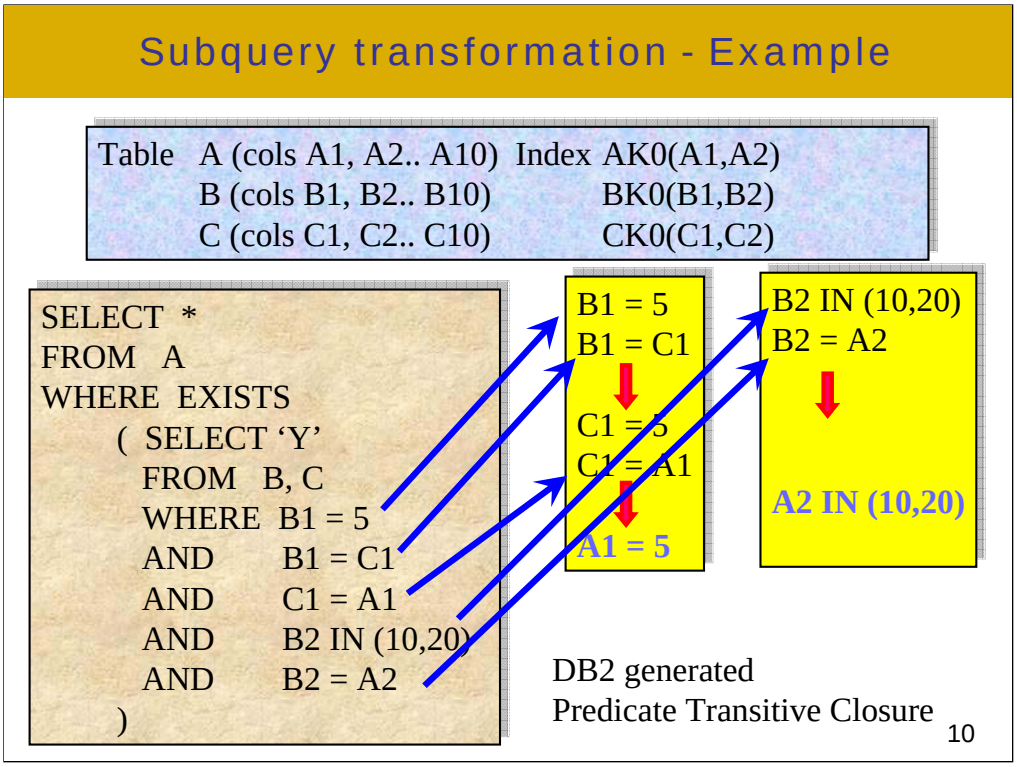
Predicate pushdown - Example

```
SELECT * FROM
  (SELECT COL01, COL02
   FROM MRF  A
   UNION
   SELECT COL01, COL02
   FROM MRI  B ) X
WHERE X.COL01 IN (100,200,300)
```

```
SELECT * FROM
   (SELECT COL01, COL02
   FROM MRF  A
   WHERE A.COL01 IN (100,200,300)
   UNION
   SELECT COL01, COL02
   FROM MRI  B
   WHERE B.COL01 IN (100,200,300)
   ) X
WHERE X.COL01 IN (100,200,300)
```

9

An example of how predicates outside a UNION can be “pushed down” to inside the UNION.



An example of a query can be transformed by repeated application of transitive closure.

Subquery transformation – Rewritten Query

```
SELECT  *
FROM   A
WHERE  A1 = 5
AND      A2 IN (10,20)
AND    EXISTS
       (  SELECT 'Y'
          FROM   B, C
          WHERE  B1 = 5
          AND       B1 = C1
          AND       C1 = A1
          AND       B2 IN (10,20)
          AND       B2 = A2
       )
```

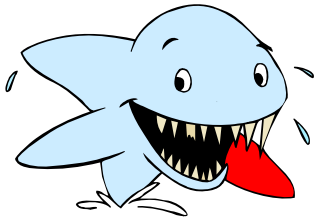
Potentially a great access path!

11

The derived predicates on the outer table now allow a more efficient access path as shown in the slide that follows.

Where Are We?

1. **Rev it up!** **(Misc performance enhancements)**
2. **Bigger bite** **(Multi-row Fetch & Insert)**
3. **What's wrong?** **(GET DIAGNOSTICS)**
4. **Next Please!** **(Sequences, Identity columns etc)**
5. **Play it again, Sam!** **(Recursive SQL)**



12

Lets' continue with two important features – the ability to fetch and insert multiple rows in one SQL statement.

What is multi-row fetch/insert?

- Multi-row FETCH
  - A single FETCH retrieving multiple rows as a rowset
  - Supports static and dynamically prepared statements (fetch is always static)
- Multi-row INSERT
  - A single INSERT adding multiple rows
  - Supports static and dynamically prepared statements
- Benefit
  - Until V7 – most SQL operations were set oriented but fetch/insert were row-at-a-time
  - Makes SQL more usable and powerful
  - Improves performance by reducing trips between application and DBMS
  - For distributed applications, also reduces network traffic

13

The basics – what are they and why should I care?

While SQL is founded on the ability to use set-oriented processing, the inability of the host language to process more than one row (until V7) has been a major bottleneck. This feature overcomes this “impedance mismatch” between SQL and the host language.

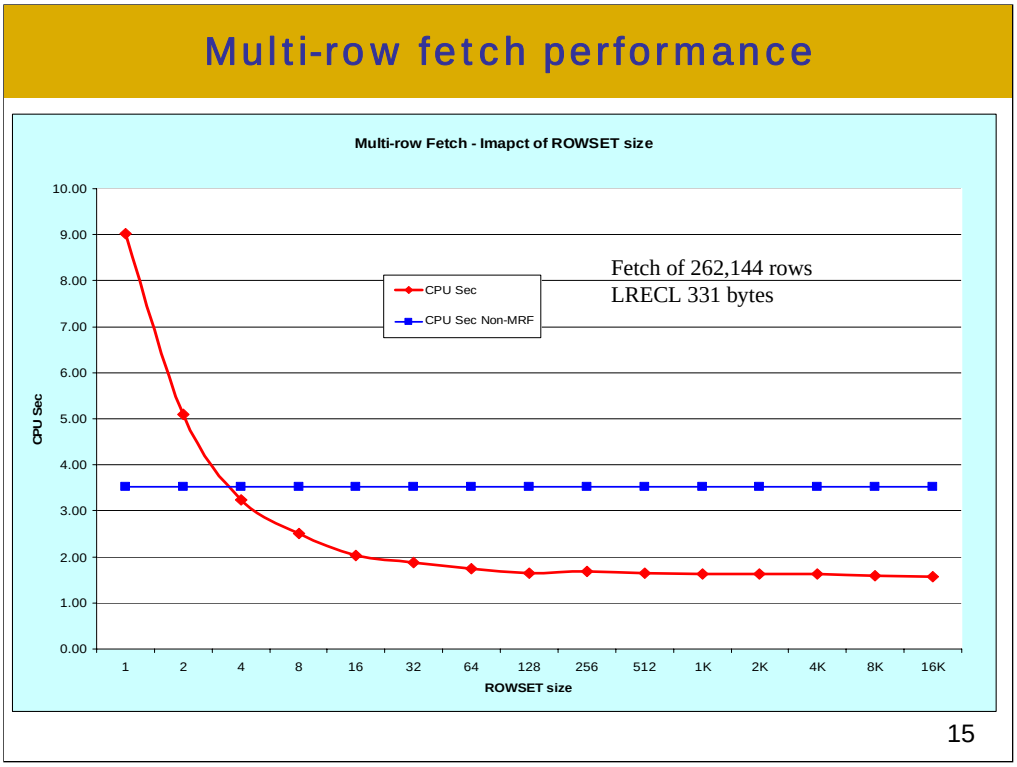
COBOL example for FETCH

```
EXEC SQL DECLARE C1 CURSOR
WITH ROWSET POSITIONING FOR
       SELECT COL01, ...
       FROM MRF
       ORDER BY COL01
END-EXEC.
EXEC SQL
    FETCH NEXT ROWSET FROM C1
    FOR :WS-MAX ROWS
    INTO :WS-COL01, …
END-EXEC.
```

14

Implementing multi-row fetch in a COBOL program.

You declare the cursor using the “WITH ROWSET POSITIONING” clause and during the fetch, you fetch the “NEXT ROWSET” for the specified number of rows.



Benchmarks run in my "sandbox" non-data-sharing DB2 subsystem. Applies to this and all benchmarks presented here – YMMV – Your Mileage May Vary!

Note the log scale in the x-axis.

When fetching even a handful of rows (as few as 4), multi-row-fetch seems to perform better than the traditional V7-style fetch. Various other factors, especially the LRECL may impact this break-even point. Also notice that using a multi-row fetch when you know you will return only 1 or 2 rows can be expensive!

Overall, it is clear that up to 50% reduction in fetch cpu can be achieved by deploying multi-row fetch when appropriate.

Compare this with the 5-7% degradation possible with V8 – this is the cost of doing nothing (see ref #2).

Declare cursor

- WITHOUT ROWSET POSITIONING (default)
  - Behaves like a cursor in V7
  - Cursor can be used to return a single row from FETCH
  - FOR n ROWS clause not allowed on FETCH
  - If FOR n ROWS is specified, SQLCODE -249 (SQLSTATE 24523)
  - Does **not** mean no DRDA blocking during a distributed access – this clause only decides number of rows returned to the application
- WITH ROWSET POSITIONING
  - Cursor can be used for single row or multiple rows
  - FOR n ROWS of FETCH determines the max number of rows returned

16

Choices on DECLARE CURSOR.

Clarifying the impact of specifying without (default) or with row positioning. Note that sqlcode -249 occurs at run time (not during bind!).

Also note that DRDA block fetch is independent of multi-row fetch.

Fetch

- ROWSET-POSITIONED block
  - Similar to V7 row-positioned block
  - Allows NEXT ROWSET, PRIOR ROWSET, FIRST ROWSET, LAST ROWSET, CURRENT ROWSET, ROWSET STARTING AT ABSOLUTE and ROWSET STARTING AT RELATIVE
- MULTIPLE-ROW-FETCH block
  - Similar to V7 single-row-fetch block
  - Additional clause FOR n ROWS
    - *Specifies the number of rows fetched in a rowset*
    - *Can be an integer or a host variable*

17

…and choices during FETCH.

Host variable arrays

- A host language structure in which each element holds a value for the same column
- Can be referenced only in a multi-row FETCH or INSERT
- Supported in:
  - COBOL
  - PL/I
  - C and C++
- Assembler support limited (allowed where USING DESCRIPTOR is allowed)
- May not be arrays of structures (what a pity!) – nor an “OCCURS DEPENDING ON”
- No DCLGEN support
- JDBC/ODBC – some restrictions

18

How host variable arrays are supported in various languages.

Especially note the lack of support for DCLGEN and that it cannot be array of structures (see next slide).

Host variable arrays - Example

Valid:

```
01  WS-MRF.
   10 WS-COL01 PIC S9(9) USAGE COMP OCCURS 1024 TIMES.
   10 WS-COL02 PIC X(10) OCCURS 1024 TIMES.
…
   10 WS-COL30 PIC X(10) OCCURS 1024 TIMES.
```

Invalid:

```
01  WS-MRF OCCURS 1024 TIMES.
   10 WS-COL01 PIC S9(9) USAGE COMP.
   10 WS-COL02 PIC X(10).
…
   10 WS-COL30 PIC X(10).
```

19

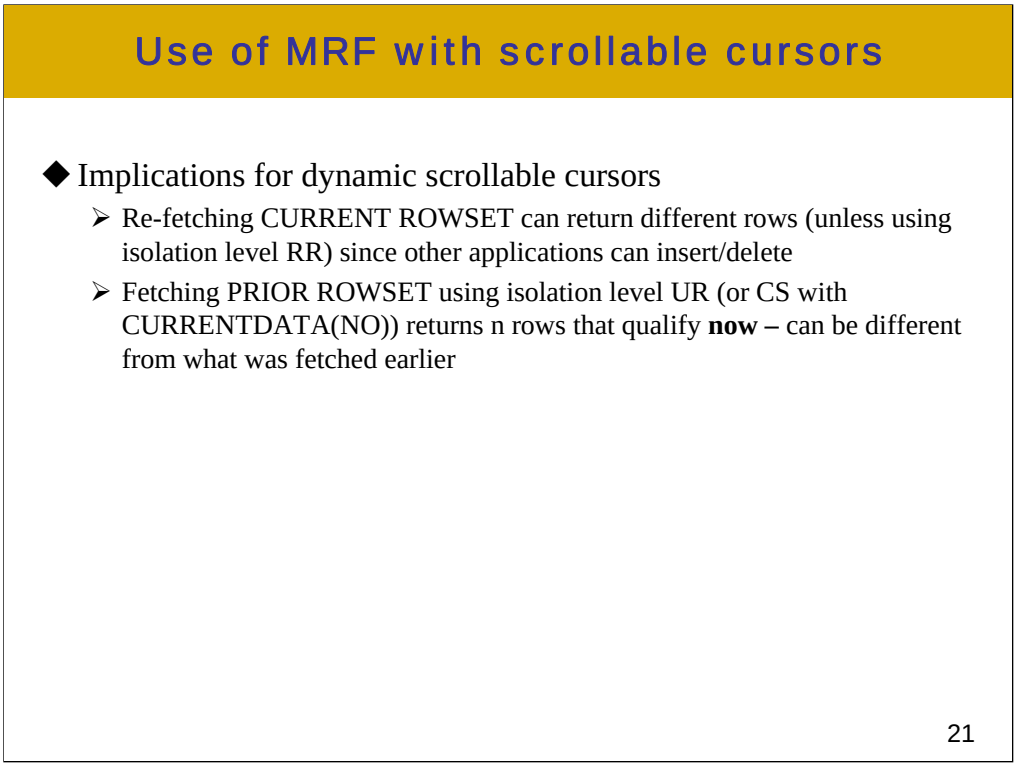
How host variable arrays are supported in COBOL. Notice that the OCCURS cannot be at the group level. OCCURS DEPENDING ON is also **not** supported.

Use of MRF with scrollable cursors

- Need to deal with “holes”
  - Must provide an indicator variable array for at least 1 column (even if all columns fetched are not null)
  - When provided, the indicator set to -3 (-1 = null as before, -2 = conversion error as before) to indicate a hole and SQLSTATE set to 02502 and SQLCODE set to +222
  - When not provided, an error is returned – SQLSTATE 24519, SQLCODE -247
- Implications for static scrollable cursors
  - With insensitive, your application sees updates/deletes made only by you
  - With sensitive, your application sees updates/deletes made by others also
  - Holes are possible in either case

20

Implications of using multi-row fetch with scrollable cursors.



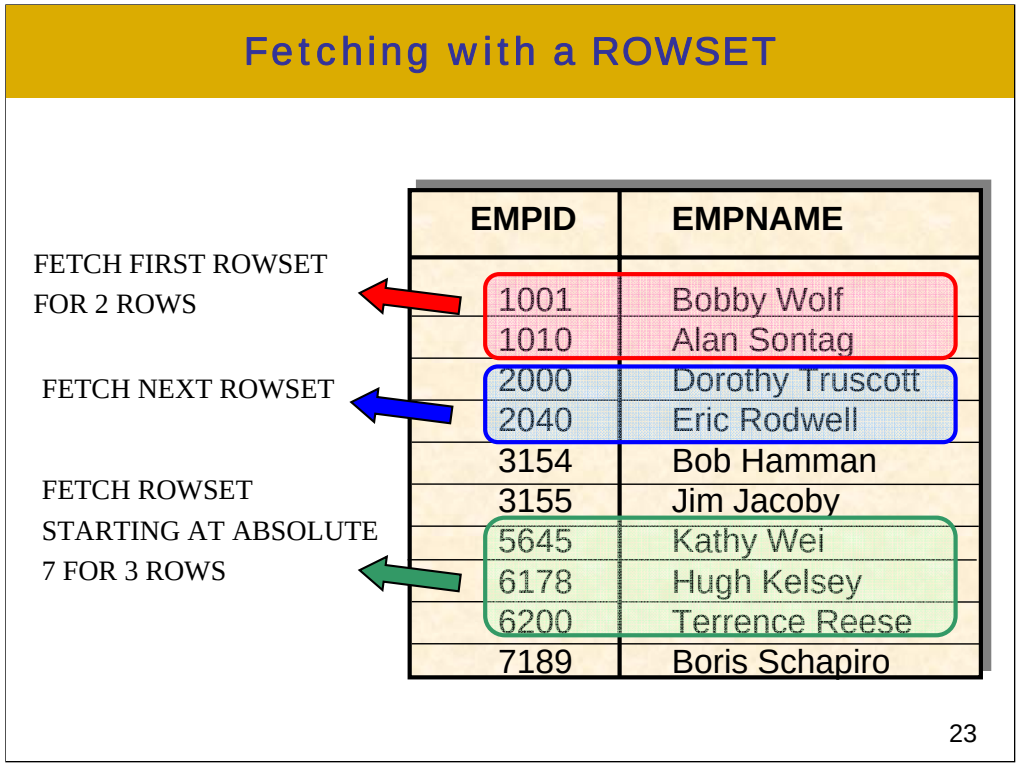
Implications of using multi-row fetch with dynamic scrollable cursors - continued.

What is a ROWSET?

- A group of rows returned by a single FETCH statement or inserted by a single multi-row INSERT statement
- Controlled by the application – FETCH… FOR n ROWS
- Minimum 1, maximum 32,767
- Performance implications – see benchmarks (covered earlier)
- Each group is operated on as a set
- Can mix and match rowsets with rows (danger!) – (covered in following slides)

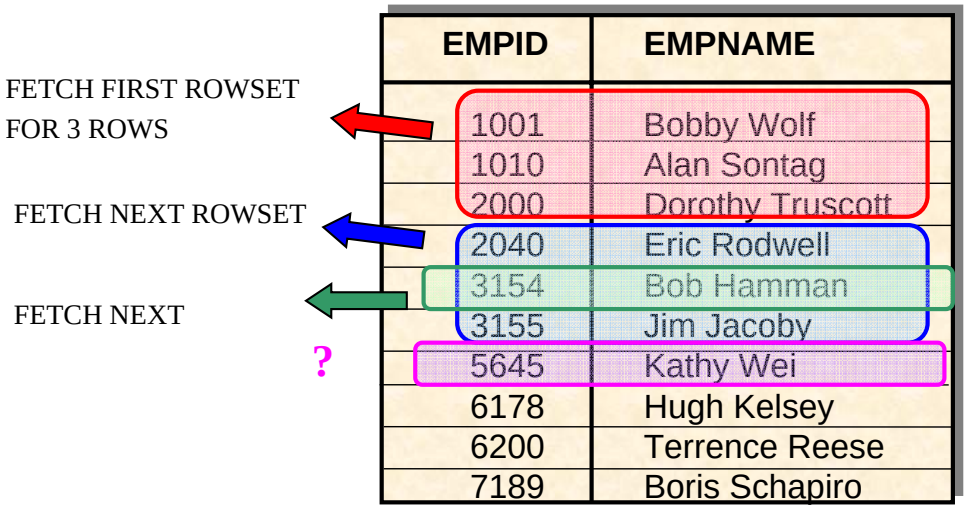
22

Formalizing the concept of a rowset.



Example of how fetching with a rowset works.

Mixing ROWSETs and ROWs



24

Danger – the next normal fetch does **not** return what you would expect!

Partial ROWSETs (forward)

FETCH ROWSET
STARTING AT 5
FOR 4 ROWS

FETCH NEXT ROWSET
FOR 3 ROWS

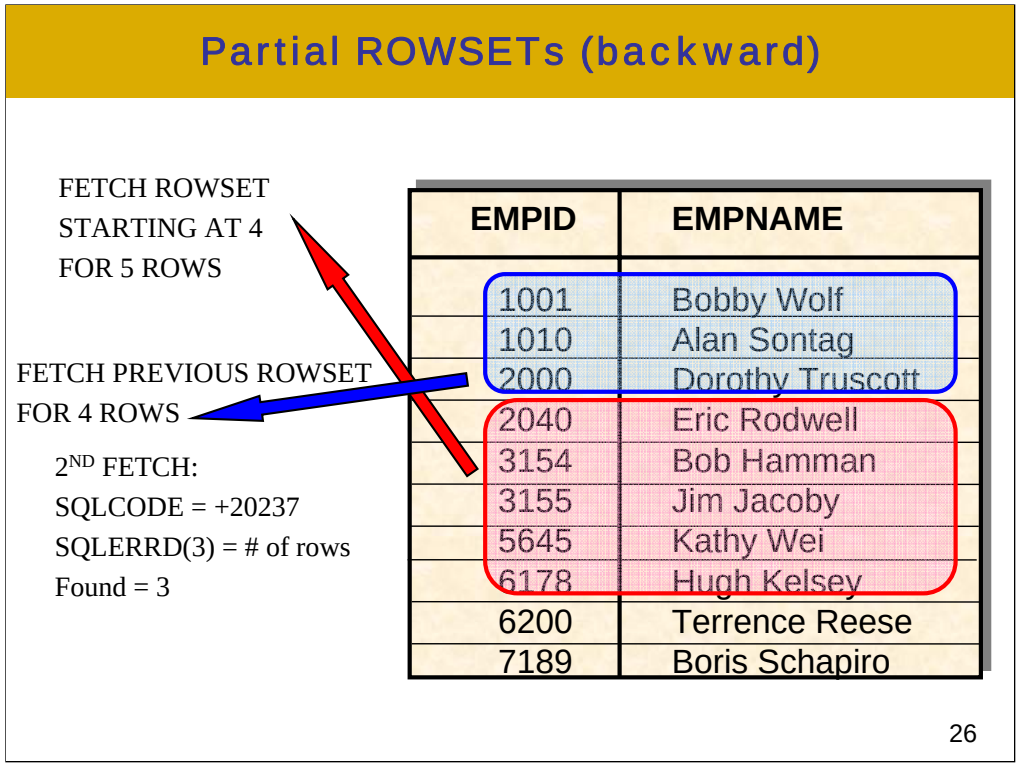
2ND FETCH:
SQLCODE = +100
SQLERRD(3) = # of rows
Found = 2

EMPID	EMPNAME
1001	Bobby Wolf
1010	Alan Sontag
2000	Dorothy Truscott
2040	Eric Rodwell
3154	Bob Hamman
3155	Jim Jacoby
5645	Kathy Wei
6178	Hugh Kelsey
6200	Terrence Reese
7189	Boris Schapiro

25

Scrolling forward with a partial rowset available.

Contents of host variables from previous fetch are un-affected (e.g. if 10 rows successfully fetched followed by 6, host variable for rows 7 thru 10 are not changed).



Similarly, scrolling backward with a partial rowset available.

Fetching past boundaries with ABSOLUTE or RELATIVE

- No data returned
- SQLCODE = +100
- SQLERRD(3) = 0
- Cursor positioning depends on the direction of fetch
  - **After** fetched row if fetching forward
  - **Before** fetched row if fetching backward

27

Impact of fetching past boundaries.

Positioned update/delete of MRF

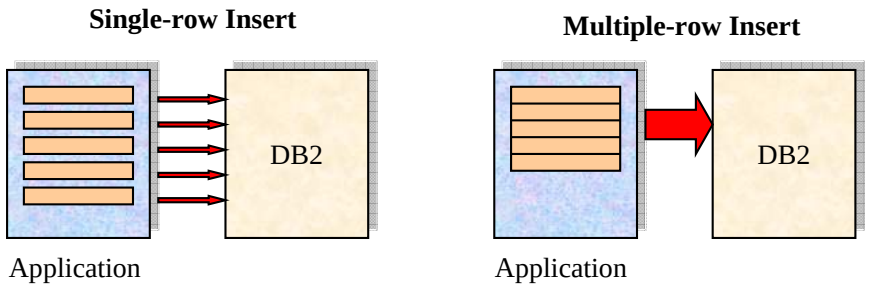
- UPDATE/DELETE… WHERE CURRENT OF..
  - If positioned on a single row (previous fetch was not FOR n ROWS), updates/deletes single row
  - If positioned on a rowset (previous fetch was FOR n ROWS), updates/deletes entire set of rows
  - If you mix and match row and rowset, could accidentally update/delete a rowset instead of a single row!
- UPDATE/DELETE WHERE CURRENT OF..FOR ROW x OF ROWSET
  - Updates/Deletes the specific row of the rowset

28

Positioned update/delete using a cursor that has fetched multiple rows.

Multi-row insert

- A single SQL statement can insert multiple rows using a host variable array
- For static SQL, uses the FOR n ROWS clause on insert
- For dynamic SQL, FOR n ROWS is specified on the EXECUTE
- Host variable arrays contain multiple rows of a single column
- Reduces trips into DB2 and for remote applications, reduces network traffic



29

Multi-row insert – the basics.

COBOL example for INSERT

```
01  WS-MRF.
    10 WS-COL01 PIC S9(9) USAGE COMP OCCURS 1024 TIMES.
    10 WS-COL02 PIC X(10) OCCURS 1024 TIMES.
...
    10 WS-COL30 PIC X(10) OCCURS 1024 TIMES.
```

```
EXEC SQL
    INSERT INTO MRI
    VALUES (:WS-COL01
                  ,:WS-COL02
          ....
                   ,:WS-COL30)
    FOR 1024 ROWS ATOMIC
END-EXEC.
```

30

A COBOL example. Note the host variable declaration (“occurs” at the elementary level - not at the group level, as explained earlier).

How granular is multi-row insert?

- ATOMIC (default)
  - If insert fails for any row, all changes are undone ("all-or-nothing")
  - Beware of logging implications – 32K rows with row length 32K could log more than 1 GB of data!
  - Easier to restart and re-position
- NOT ATOMIC CONTINUE ON SQLEXCEPTION
  - Each row is processed independently, processing continues on failure
  - Use GET DIAGNOSTICS to see the results for each failed row (covered next section)
  - Possible SQLCODES
    - *All failed – SQLCODE = -254, SQLSTATE = 22530*
    - *All successful but warnings – SQLCODE = +252, SQLSTATE = 01659*
    - *At least one (but not all) failed –SQLCODE = -253, SQLSTATE = 22529*
  - Restart and re-positioning can be tricky

31

Granularity of a multi-row insert.

Dynamic SQL & Multi-row INSERT

◆ During PREPARE:

```
MOVE 'FOR MULTIPLE ROWS ATOMIC' TO WS-ATTR.

EXEC SQL PREPARE S1 ATTRIBUTES :WS-ATTR FROM :WS-S1
END-EXEC.
```

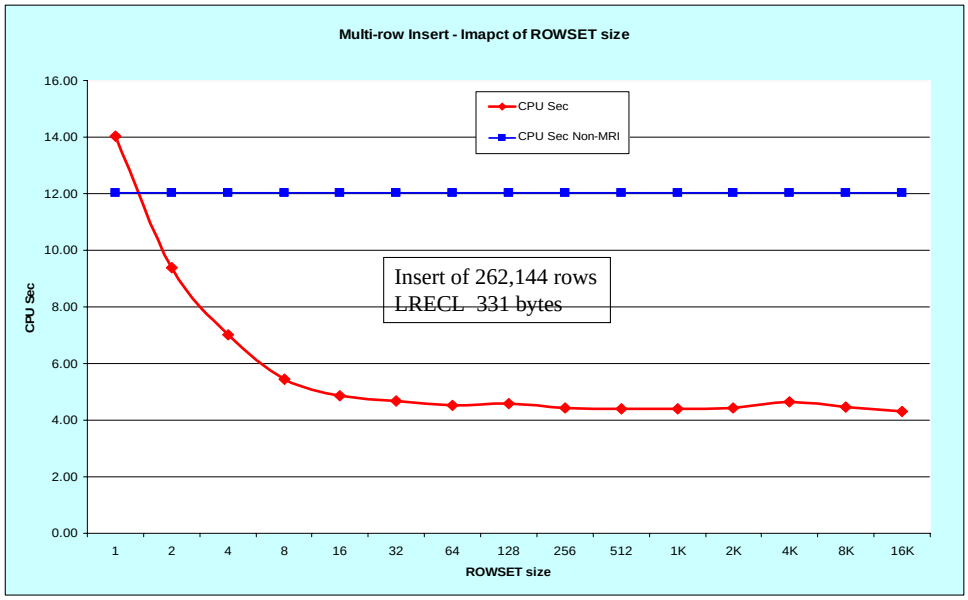
◆ During EXECUTE:

```
EXEC SQL EXECUTE S1
    USING
          :WS-COL01
        , :WS-COL02 …
 FOR :WS-NROWS ROWS
END-EXEC.
```

32

Dealing with a multi-row insert in dynamic SQL. Note the attributes on the prepare and the number of rows on the execute statement unlike static SQL.

Multi-row insert performance



33

Performance benchmarks.

Note the log scale on the x-axis.

Based on this, even when inserting very few rows (as few as 2!), multi-row insert seems to work better. The graph tapers off at about 64 rows, with limited benefit. Also note that a larger size can mean more failures and longer rollback times in case of failure.

DRDA considerations

- DRDA Version 3 is required
- Between 2 DB2 z/OS V8 systems
  - Multi-row insert does not affect blocking
  - Multi-row fetch - maximum 10 MB can be returned
  - Rowset size determines the blocking – 32KB blocks size is ignored
- Between DB2 LUW and DB2 z/OS V8 system
  - Multi-row operations are not supported for embedded SQL
  - ODBC/CLI driver provides support for multi-row insert and limited support for multi-row fetch

34

Impact of DRDA on multi-row insert.

Other factors affecting performance

- Number of columns fetched
  - Fewer columns ➔ more improvement (yet another reason for not allowing select *)
- Data type and size of columns fetched/inserted
  - Simple and smaller columns ➔ more improvement
- Complexity of SQL
  - Simple SQL ➔ more improvement
- For insert – number of indexes on the table

35

Impact of various other factors on performance of multi-row operations. Multi-row updates and deletes are affected likewise.

Where Are We?

1. **Rev it up!** **(Misc performance enhancements)**
2. **Bigger bite** **(Multi-row Fetch & Insert)**
3. **What's wrong?** **(GET DIAGNOSTICS)**
4. **Next Please!** **(Sequences, Identity columns etc)**
5. **Play it again, Sam!** **(Recursive SQL)**

36

Let's discuss next the topic of error handling, which is specially important when dealing with any multi-row fetch and multi-row insert that uses the NOT ATOMIC option.

GET DIAGNOSTICS

- Extends the diagnostic information available in SQLCA
- Supports SQL error message tokens longer than 70 bytes (SQLCA limitation) – see example 4 that follows
- Cannot be dynamically prepared – static only
- Returns diagnostic information in 3 different areas:
  - (Keyword ALL indicates all variables should be combined into one string)
  - Statement information area
    - *Information about the SQL statement*
  - Condition information area
    - *Information about conditions*
    - *Information about connections*
  - Combined information area
    - *A textual representation of all information*

37

The basics.

Statement information area contains things such as:

NUMBER (number of errors)
ROW_COUNT (number of rows for last stmt)

Condition information area contains things such as:

DB2_REASON_CODE
DB2_RETURNED_SQLCODE

as well as

DB2_AUTHORIZATION_ID
DB2_PRODUCT_ID

For complete details see DB2 manuals.

Example 1 – Information similar to SQLCA

```
EXEC SQL
  GET DIAGNOSTICS CONDITION 1
     :WS-SQLCODE      = DB2_RETURNED_SQLCODE
    , :WS-TOKEN-COUNT  = DB2_TOKEN_COUNT
    , :WS-ORDINAL-TOKEN-1 = DB2_ORDINAL_TOKEN_1
    , …
    , :WS-MESSAGE-ID  = DB2_MESSAGE_ID
    , :WS-MESSAGE-TEXT = MESSAGE_TEXT
    , :WS-MODULE-DETECTING-ERROR = DB2_MODULE_DETECTING_ERROR
   , :WS-RETURNED-SQLSTATE = RETURNED_SQLSTATE
 END-EXEC.
```

```
WS-SQLCODE         =  00000080L
WS-TOKEN-COUNT   =  000000002
WS-ORDINAL-TOKEN-1  =    GDK0
WS-ORDINAL-TOKEN-2  =    0000000A05
WS-MESSAGE-ID   =  DSN00803E
WS-MESSAGE-TEXT  =  AN INSERTED OR UPDATED  VALUE IS INVALID
     BECAUSE  INDEX IN INDEX SPACE GDK0 CONSTRAINS  COLUMNS OF
     THE TABLE SO NO TWO ROWS CAN CONTAIN DUPLICATE VALUES
     IN THOSE COLUMNS.
```

38

SQLCA-like information.

Example 2 – Handling Multi-row fetch/insert

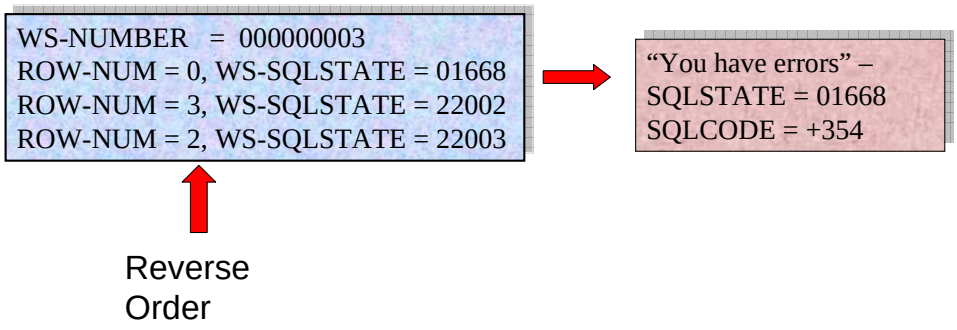
```
EXEC SQL
  GET DIAGNOSTICS
     :WS-NUMBER      =   NUMBER
END-EXEC.
PERFORM 4000-GET-EACH
THRU    4000-EXIT
VARYING WS-SUB1         FROM 1 BY 1
UNTIL   WS-SUB1 > WS-NUMBER.
...
4000-GET-EACH.
EXEC SQL
  GET DIAGNOSTICS CONDITION :WS-SUB1
     :WS-RETURNED-SQLSTATE = RETURNED_SQLSTATE
END-EXEC.
```

39

Dealing with multi-row insert.

Example 2a – Handling Multi-row fetch

Fetch 5 rows as a single rowset. Row 2 fails due to data overflow (-802) and row 3 fails since no null indicator is provided (-305).



Reverse Order

40

The number of errors is always one higher than actual and the real errors occur in reverse sequence!

Example 2b – Handling Multi-row insert

Insert 5 rows as a single rowset. Row 4 fails due to check constraint violation (-545) and row 5 fails as a unique key violation (-803).

ATOMIC

```
WS-NUMBER   =  000000001
ROW-NUM = 4, WS-SQLSTATE = 23513
```

NOT ATOMIC

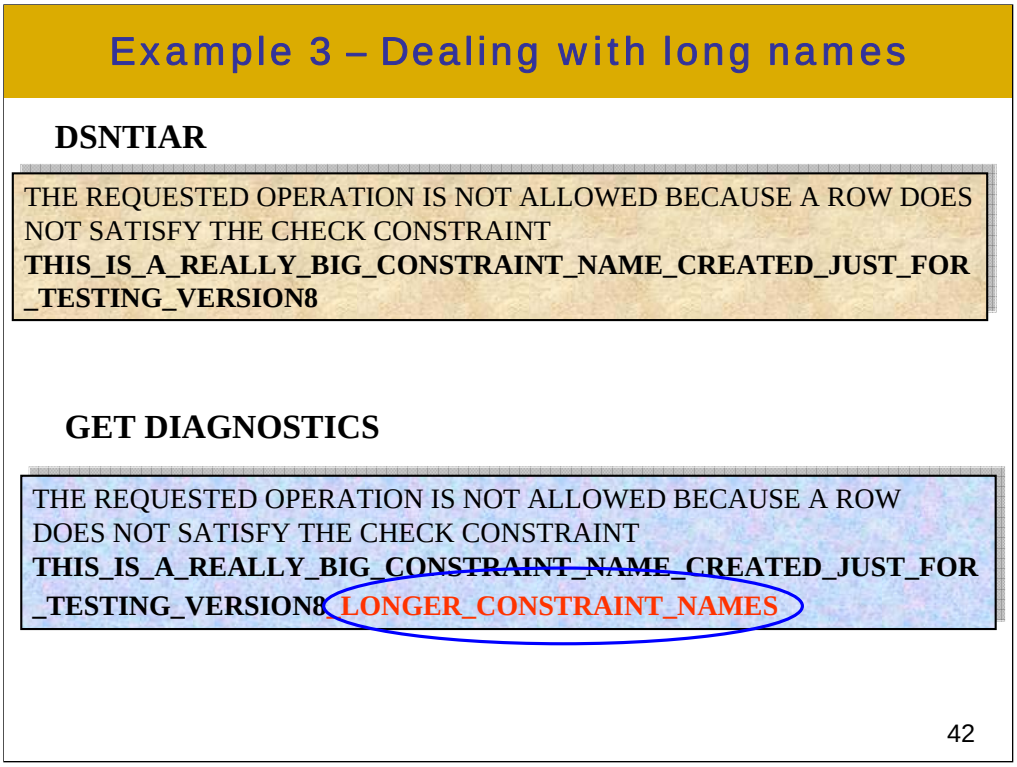
```
WS-NUMBER   =  000000003
ROW-NUM = 0, WS-SQLSTATE = 22530
ROW-NUM = 5, WS-SQLSTATE = 23505
ROW-NUM = 4, WS-SQLSTATE = 23513
```

```
"You have errors" –
SQLSTATE = 22530
SQLCODE = -254
---- or ----
SQLSTATE = 22529
SQLCODE = -253
```


Reverse Order

41

Notice that only the first error is returned for ATOMIC. When using NOT ATOMIC, the number of errors is always one higher than actual and the real errors occur in reverse sequence!



For longer object names, GET DIAGNOSTICS is the only reliable means of obtaining the full names since SQLCA limits tokens to 70 characters.

Where Are We?

1. **Rev it up!** **(Misc performance enhancements)**
2. **Bigger bite** **(Multi-row Fetch & Insert)**
3. **What's wrong?** **(GET DIAGNOSTICS)**
4. **Next please!** **(Sequences, Identity columns etc)**
5. **Play it again, Sam!** **(Recursive SQL)**

43

We gave up on IDENTITY columns introduced in V6 due to their operational limitations. We will re-visit these and also look at other options for generating the next number. This includes SEQUENCE objects.

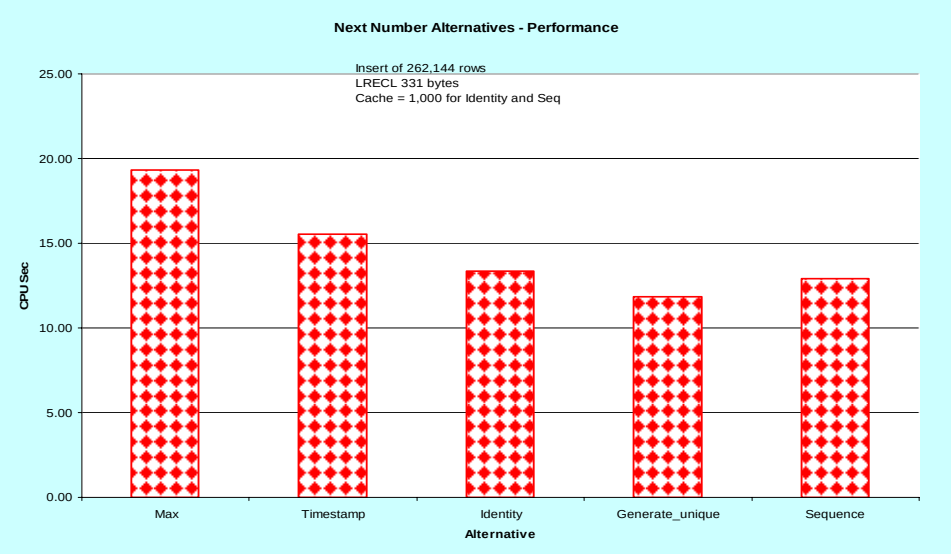
Choices for next number – an evolution

- Until V5:
  - Single control table
  - MAX() + 1 logic
  - Timestamp
- V6 introduced IDENTITY columns
  - Tied to specific table only
  - Max 1 per table
  - Operations issues (GENERATED BY DEFAULT vs. GENERATED ALWAYS)
- V7 introduced GET_UNIQUE function
  - Returns a 13 byte ascending FOR BIT DATA value
- V8 fixes Identity columns and introduces Sequence objects
  - Overcomes earlier limitations of Identity columns
  - Sequence objects compatible with DB2 for LUW

44

The basics. We will compare and contrast these options in this section. In addition, we will dig deeper into features of SEQUENCE objects since they are new in V8.

Performance



45

How expensive are sequences and identity columns? Not too much, it turns out. In fact, with the greater concurrency (not measured here), they may be good option in several cases.

Overall, if you currently use the DB2 timestamp (and possibly manipulate it before using), to obtain the next number, SEQUENCE objects can perform almost as well. Home-grown assembler routines (that obtain the store-clock value) will clearly outperform either of these, but SEQUENCEs are certainly a viable option.

The max+1 (if not using an index) will become progressively worse as the table size increases.

GENERATE_UNIQUE is also an option you should consider since it performs well and can eliminate the need for the timestamp column (it is based on the store clock value).

Features of sequence objects

- Eliminates hot-spots resulting from a single control table
- Ensures that multiple transactions can concurrently obtain a sequence number that is guaranteed to be unique
- No lockwaits for transactions
- In a data-sharing environment, failure of one member allows surviving members to continue

46

Features of SEQUENCE objects.

A comparison of options

Item	SELECT MAX() + 1	GENERATE_ UNIQUE	IDENTITY	SEQUENCE
Locking issues	Possible	Never	Never	Never
Uniqueness assured	Always (proper use)	Always within sysplex	Always within sysplex	Always within sysplex (proper use)
Performance	OK with index	Excellent	Excellent	Excellent
Max value	31 digits	13-byte bit data	31 digits	31 digits
Retrieved via	SELECT	SELECT	IDENTITY_VAL _LOCAL	NEXT VALUE
Override	Yes	Yes	Yes (by default) No (generated always)	Yes
Assign before insert	Yes	Yes	**No**	**Yes**
Max per table	Many	Many	**1**	**Many**
Possible miscoding	Yes	No	**Never**	**Yes (wrong object)**

47

A summary of each option. Depending on your needs, any one of them may be best suited for your environment.

Sequence object - options

- AS INTEGER/data-type – can be smallint, integer or decimal
- START WITH – starting value
- INCREMENT BY – how next value is determined
- MINVALUE/NO MINVALUE – minimum value allowed
- MAXVALUE/NO MAXVALUE – maximum value allowed
- CYCLE/NO CYCLE – what to when min or max is reached
- CACHE/NO CACHE – whether or not to cache
- ORDER/ NO ORDER – whether or not order is needed

48

Options. The options you are likely to use, we will cover in detail in the slides that follow.

Examples

```
CREATE SEQUENCE CACHESEQ
   AS INTEGER
        START WITH 1
   INCREMENT BY 1
   CACHE 80
   NO ORDER;

GRANT USAGE ON SEQUENCE CACHESEQ TO PUBLIC;

INSERT INTO SEQCACHE
VALUES (NEXT VALUE FOR CACHESEQ
            , :WS-COL02
            ...
            , :WS-COL30)
```

49

A simple example.

Note the grant before anyone can use the object and the fact that there is no direct link between the sequence object and the table it populates – it is only via the use of the function NEXT VALUE FOR.. that a link is established at run-time.

Ranges and Cycles

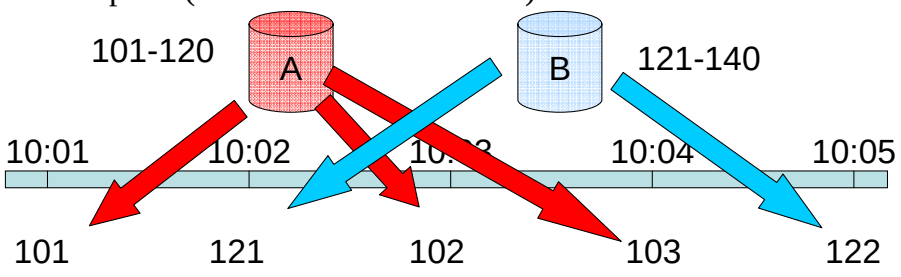
- You can specify that a sequence cycle after reaching the limit value (max for ascending, min for descending)
- First cycle always starts with START WITH value but all subsequent cycles start with MINVALUE (for ascending) or MAXVALUE (for descending)
- If NO CYCLE is specified, generation of sequence values stops when the limit value is reached (SQLCODE = -359, SQLSTATE = 23522)

50

How ranges and cycles interact. In my view, this is mostly academic – since both will hardly ever be used.

Caching

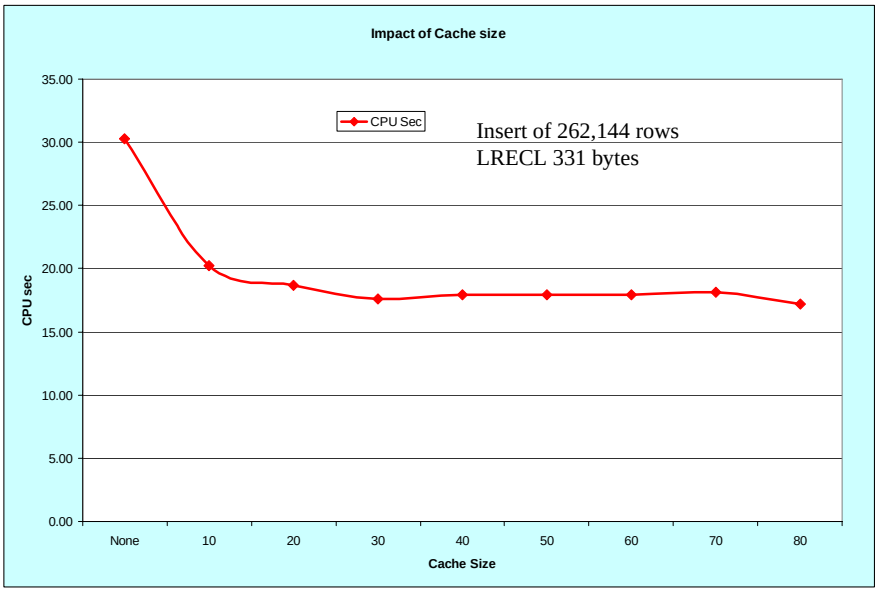
- Specifying CACHE allows DB2 to cache pre-allocated values in memory for faster access
  - Sync i/o to SYSIBM.SYSSEQUENCES is required only when cache values are exhausted
- IBM recommended value is 20
  - Larger number means more gaps
  - Actual optimal number may depend on transaction volume (30-50?)
- In a data-sharing environment, each member has its own cache – example – (cache size = 20 assumed)



51

Basics of caching.

Impact of Cache Size on Performance



52

Impact of cache size. 30-50 seems to provide the most benefit. I disagree with the categorical hard limit of 20 recommended in the redbook (ref #1).

Gaps

- It is possible to get gaps when:
  - Transaction obtains the sequence number and rolls back or abends (causing a rollback)
  - NEXTVAL is used in a select cursor in a DRDA environment where the client uses block-fetch and the application does not fetch all retrieved rows
  - DB2 or system failure (all cached values lost)
- A transaction obtaining 2 sequence numbers right after each other may not obtain consecutive values

53

How you can end up with gaps.

Caching and Order

Data Sharing	Order	Cache	Impact
YES	YES	YES	Order assured, may disable cache (OK with single application processes)
		NO	Order assured
	NO	YES	Order not assured, good performance
		NO	Order not assured but no reason to use
NO	YES	YES	Not allowed – DB2 changes to no cashing
		NO	Order is assured and integrity dictates you must sacrifice performance
	NO	YES	Order not assured, good performance
		NO	Order not assured, no reason to use

54

How caching and order interact with each other.

Recoverability

- Upon DB2 failure, sequences recovered from catalog tables and logs – unique values are guaranteed
- Unused sequence numbers from cache are lost on DB2 or system failure
- A point-in-time recovery of catalog could generate duplicate sequence numbers

55

Recovering a sequence.

Catalog impact

- SYSIBM.SYSSEQUENCES (changed table)
  - MAXASSIGNEDVAL — last possible assigned value
  - PRECISION — precision of the sequence data type
  - RESTART WITH — number to start with (used in ALTER)
- SYSIBM.SYSSEQUENCESDEP (changed table)
  - DTYPE — type of object sequence is dependent on (F=SQL function, I=identity column)
  - DSCHEMA — qualifier of the object
  - BSCHEMA — schema name of sequence
  - BNAME — sequence name
- SYSIBM.SYSSEQUENCEAUTH (new table)
  - Contains one row for each ALTER and USAGE privilege held by a user over a sequence object (similar to other auth tables).

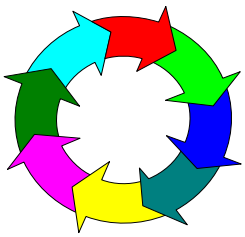
56

Where to look in the catalog.

Note that SYSSEQUENCESDEP does **NOT** show which programs use the sequence – this information is available in SYSSEQUENCESAUTH for static SQL. Dependency of programs using dynamic SQL is not tracked in the catalog.

Where Are We?

1. **Rev it up!** **(Misc performance enhancements)**
2. **Bigger bite** **(Multi-row Fetch & Insert)**
3. **What's wrong?** **(GET DIAGNOSTICS)**
4. **Next Please!** **(Sequences, Identity columns etc)**
5. **Play it again, Sam!** **(Recursive SQL)**



57

The final topic of this session – powerful but dangerous!

When used properly, it provides a very powerful mechanism for making something out of nothing. Traditional SQL can report on data present in a table – with recursive SQL we can generate data that does not exist!

We will look at two examples in this section – one to generate prime numbers and one to generate a org chart.

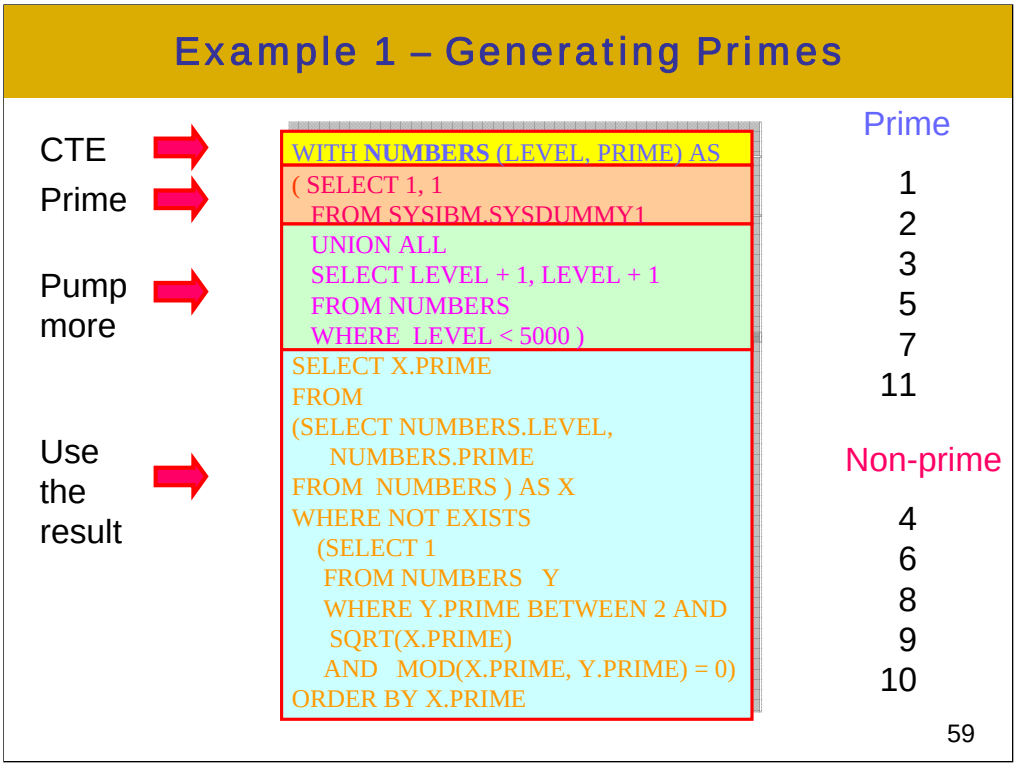
How it works

- A technique where a function calls itself to perform some part of the task
- Useful for org charts (example follows) or bill-of-material explosion – cursor within cursor is otherwise required
- Requires:
  - An initialization select (“priming the pump”)
  - An iterative select (“pump more”)
  - A main select (“use the result”)
- Prone to infinite cycles unless controlled properly (see recommendations)
- DB2 imposes no limit on the depth of recursion but issues a warning (SQLSTATE = 01605, SQLCODE=+347) for an SQL statement that does not use a control variable to limit the depth

58

The basics.

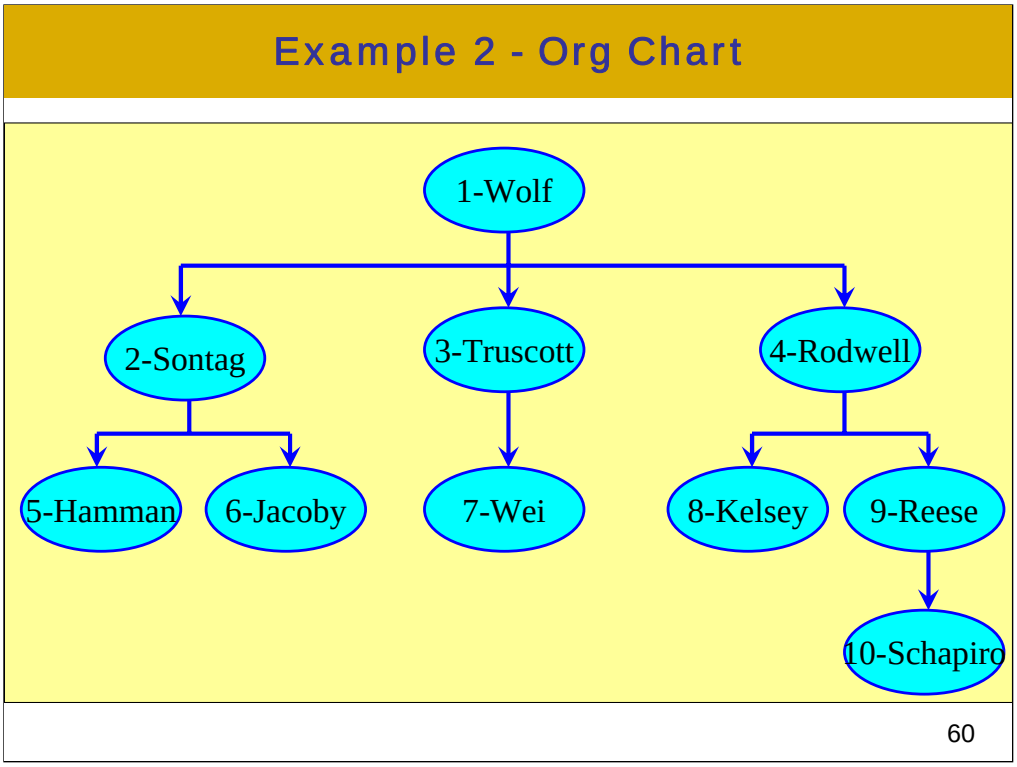
The “pump” terminology is borrowed from Tink Tysor (see ref #5). Tink provides very thorough discussion on this fascinating topic.



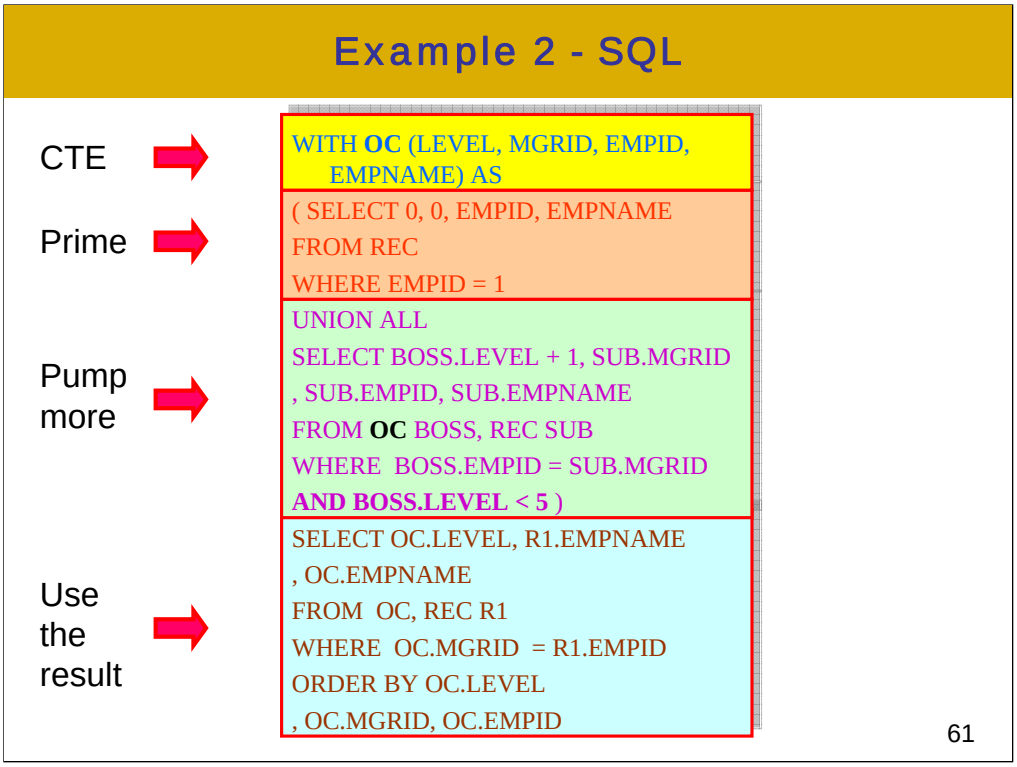
Actual SQL to generate prime numbers.

The definition – WITH NUMBERS – is an example of a Common Table Expression (CTE) which is similar to the Nested Table Expression (NTE) most of us already familiar with. It is required for using recursive SQL.

Also notice that we can stop checking for factors once we cross SQRT of that number (a big gain in performance – for example, instead of checking 400, we can stop at 20).



Example of how recursive SQL can be used effectively for traversing hierarchical structures.



Actual SQL to generate the org chart.

The definition – WITH OC – is an example of a Common Table Expression (CTE) which is similar to the Nested Table Expression (NTE) most of us already familiar with. It is required for using recursive SQL.

Example 2 - Intermediate Result

Common Table Expression
OC

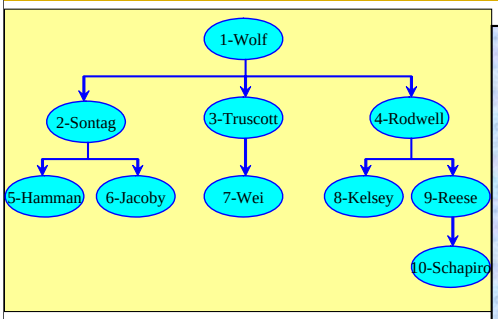
LEVEL	MGRID	EMPID	EMPNAME
0	0	1	WOLF
1	1	2	SONTAG
1	1	3	TRUSCOTT
1	1	4	RODWELL
2	2	5	HAMMAN
2	2	6	JACOBY
2	3	7	WEI
2	4	8	KELSEY
2	4	9	REESE
3	9	10	SCHAPIRO

62

Contents of the common table expression OC as the SQL is executed.

The definition – WITH OC – is an example of a Common Table Expression (CTE) which is similar to the Nested Table Expression (NTE) most of us already familiar with. It is required for using recursive SQL.

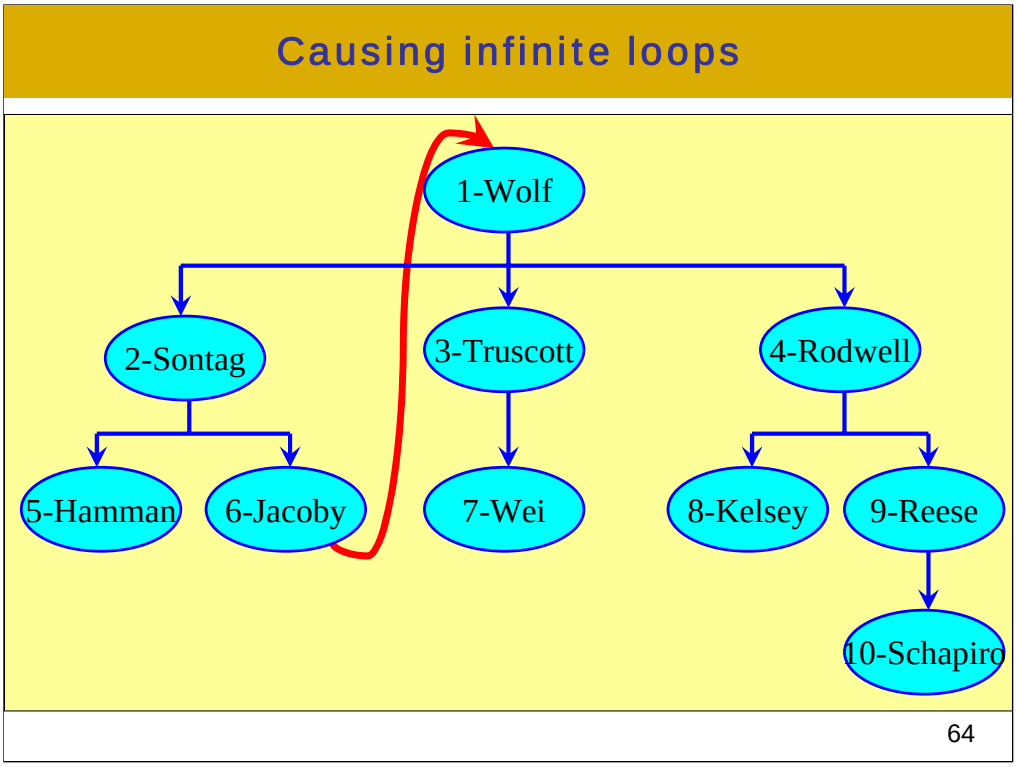
Example 2 - Result



LEVEL	EMPNAME	EMPNAME
1	WOLF	SONTAG
1	WOLF	TRUSCOTT
1	WOLF	RODWELL
2	SONTAG	HAMMAN
2	SONTAG	JACOBY
2	TRUSCOTT	WEI
2	RODWELL	KELSEY
2	RODWELL	REESE
3	REESE	SCHAPIRO

63

… and the result.



Danger lurks around the corner!

Causing infinite loops

```
WITH OC (LEVEL, MGRID, EMPID,
    EMPNAME) AS
( SELECT 0, 0, EMPID, EMPNAME
FROM REC
WHERE EMPID = 1
UNION ALL
SELECT BOSS.LEVEL + 1, SUB.MGRID
, SUB.EMPID, SUB.EMPNAME
FROM OC BOSS, REC SUB
WHERE  BOSS.EMPID = SUB.MGRID
AND BOSS.LEVEL < 5 )
SELECT OC.LEVEL, R1.EMPNAME
, OC.EMPNAME
FROM  OC, REC R1
WHERE  OC.MGRID  = R1.EMPID
ORDER BY OC.LEVEL
, OC.MGRID, OC.EMPID
```

```
DSNT404I SQLCODE = 347,
WARNING:  THE RECURSIVE COMMON
 TABLE EXPRESSION OC MAY
      CONTAIN AN INFINITE LOOP
DSNT418I SQLSTATE   = 01605
SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNXODML
SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD    = 0 0  0  1209090530
0  0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD    = X'00000000'
X'00000000'  X'00000000'  X'481141E2'
       X'00000000'  X'00000000'
SQL DIAGNOSTIC INFORMATION
    LEVEL  EMPNAME     EMPNAME
```

Removed

65

Without the extra protection offered by the level clause, the warning and possible infinite loop.

Recommendations

- Check SQL logic very carefully
- Test against small test tables
- Test against “production-like” data
- Use controls to limit the depth of recursion
- Consider possible future changes that could cause infinite loops (e.g. recursive or circular references)
- In specific instances, can lead to a huge performance gain:

“…400,000 customers in a hierarchy of about 4,000 nodes resulting in over 2.5 million queries, the application took **days** to calculate total sales… single recursive query processed in about **5 minutes**”
- Dan Luksetich, IDUG Solutions Journal, May 2004.

66

My recommendations for recursive SQL – use it but use it wisely!

My Favorite Few Features

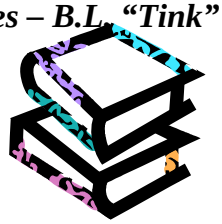
1. **Misc performance features**
   - SELECT INTO with ORDER BY
   - Non-correlated subselect
   - IN list processing
   - **Some new functionality but mostly just need to rebind!**
2. **Multi-row fetch/insert**
   - **Huge performance gain!**
3. **GET DIAGNOSTICS**
   - **Mostly for Multi-row fetch/insert**
4. **Choices for next number**
   - IDENTITY
   - SEQUENCE
   - GET_UNIQUE
   - **More choices – use Identity or Sequences as appropriate**
5. **Recursive SQL**
   - **Powerful but dangerous!**

67

My favorite few we have covered. I trust this session has empowered you with the knowledge to exploit them fully. Good Luck!

References

1. ***DB2 UDB for z/OS V8– Everything You Ever Wanted to Know, … and More – SG24-6079***
2. ***DB2 UDB for z/OS Version 8 Performance Topics – SG24-6465***
3. ***DB2 UDB for z/OS V8 Administration Guide – SC18-7413***
4. ***DB2 UDB for z/OS V8 SQL Reference – SC18-7426***
5. ***DB2 UDB for z/OS V8– Application programming and SQL Guide SC18-7415***
6. ***Recursive SQL for Dummies – B.L. "Tink" Tysor - IDUG NA 2005 - Session A1***



68

Some of the many useful references.

Acknowledgements

DST SYSTEMS®

- Terry Berman
- David Churn
- John Rader
- Mike Todd

69

Thanks to this talented group for reviewing my work and in providing the rigor required in such an endeavor – they always keep me honest!

V8 SQL for the Application Developer – My Favorite Features

70

Not really…in the unlikely event that we actually have time ….I doubt it!

Session: G11
V8 SQL for the Application Developer –
My Favorite Features

Suresh Sane
DST Systems, Inc.
sssane@dstsystems.com

GoFurther

Thank you and good luck with V8!