



Session: E08

Quit Calling DB2 So Much!

Susan Lawson and Dan Luksetich
YL&A

IDUG 2008
Europe



Experience IDUG

14 October 2008 • 16:40-17:40
Platform: z/OS

Disclaimer PLEASE READ THE FOLLOWING NOTICE



- The information contained in this presentation is based on techniques, algorithms, and documentation published by the several authors and companies, and in addition is the result of research. It is therefore subject to change at any time without notice or warning.
- The information contained in this presentation has not been submitted to any formal tests or review and is distributed on an “As is” basis without any warranty, either expressed or implied.
- The use of this information or the implementation of any of these techniques is a client responsibility and depends on the client’s ability to evaluate and integrate them into the client’s operational environment.
- While each item may have been reviewed for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere.
- Clients attempting to adapt these techniques to their own environments do so at their own risks.
- Foils, handouts, and additional materials distributed as part of this presentation or seminar should be reviewed in their entirety.

Abstract



In this presentation will look at ways reducing the number of calls you make to DB2 z/OS via exploitation of V8/V9 SQL features for SQL call reduction. The quantity of SQL calls made can sometimes be more detrimental to performance than poor quality SQL. Don't be fooled by low response times per statement and high bufferpool hit ratios! So, here we'll look at ways of using V8 and V9 SQL and application features to help reduce the number of trips we make across the address spaces in DB2 z/OS.

This presentation was developed by Susan Lawson and Dan Luksetich of YLA. They can be reached at Susan_Lawson@ylassoc.com and Dan_Luksetich@ylassoc.com repectively.

Outline



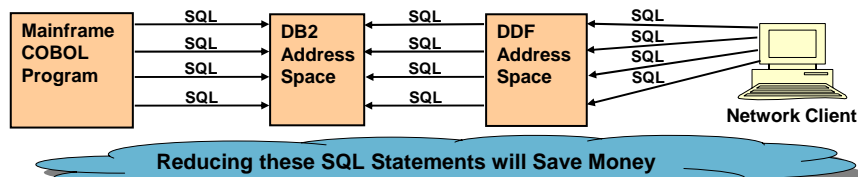
- **The Problem: Quantity of SQL calls**
- **Discussion of generic applications design**
- **V8 Features to help reduce quantity**
- **V9 Features to help reduce quantity**
- **Improving overall SQL performance**

The Challenges: Performance and Availability

Major SQL Performance Challenge



- **Generic application design has led to an increase in SQL statements issued**
 - Every SQL call represents overhead
 - Cross memory communication on the mainframe
 - Cross network traffic plus cross memory communication from remote clients
 - Modern generic techniques increases the amount of SQL
 - Use of system generated keys
 - Programmatic joins
 - Flexible object-relational design
- **Every call to DB2 represents CPU and elapsed time consumed**
 - Can we reduce this time with DB2 for z/OS V8 SQL?



Reducing these SQL Statements will Save Money

***Exploitation of V8/V9 SQL Features
for SQL Call Reduction***

SELECT from INSERT

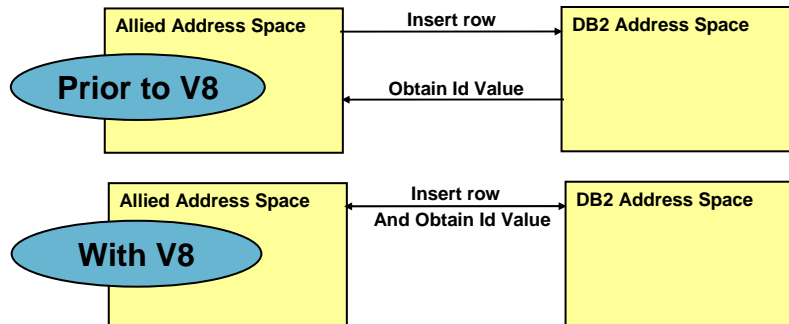


Find value of Identity Column during insert (assume table created with identity column on ACCT_ID generated always)

```
SELECT ACCT_ID
FROM FINAL TABLE
(ININSERT INTO UID1.ACCOUNT (NAME, TYPE, BALANCE)
VALUES ('Master Card', 'Credit', 50000) )
```

- This feature is useful for reducing the number of times an application must travel to the database when establishing new data
 - Quantity of SQL is a major contributor to elapsed and CPU time
- Several generated data can be selected
 - Identity column values
 - ROWID
 - Sequence values
 - Defaulted values
 - Data values resulting from triggers
- Identity values generate/selected can then be used for inserting into child tables

SELECT from INSERT Performance



- SELECT from INSERT works very well with the V8 sequence objects

```
Find value of sequence object during insert  
SELECT ACCT_ID  
FROM FINAL TABLE  
(INSERT INTO UID1.ACCOUNT (ACCT_ID,NAME, TYPE, BALANCE)  
VALUES (NEXT VALUE FOR ACCT_SEQ, 'Master Card', 'Credit', 50000) )
```

V8 - SELECT from INSERT – for Performance – Results!



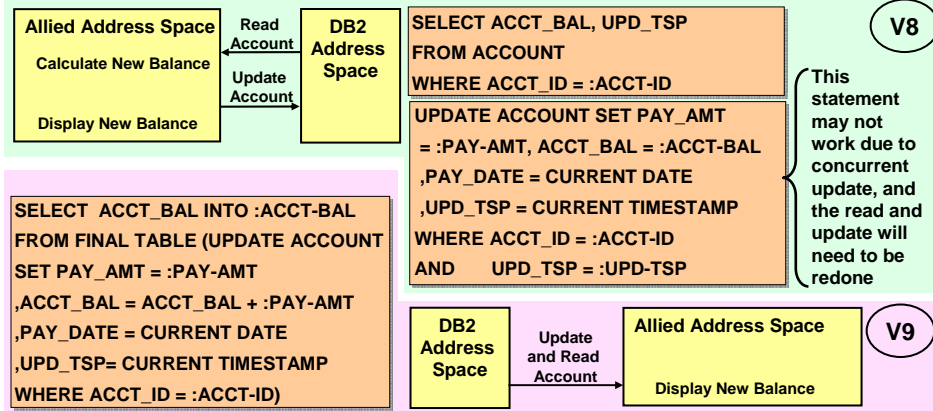
- **Is this a big deal?**
- **Maybe, if this statement is executed millions of times a day!**
- **Our tests indicate a potential savings of about 13% CPU**
- **One of our customers has saved about 40% CPU**
 - **For their generic insert program**
 - **By replacing INSERTs and SELECTs with INSERT within SELECT for one of their major applications**

V9 - SELECT from an UPDATE/DELETE



- Will allow a searched UPDATE or DELETE to be placed in the FROM clause
 - In a Cursor or in the SELECT INTO statement
 - Returning old or new data
- Reduces SQL statements issued and improves concurrency!

What if I update ACCOUNT with today's payment, and return the balance?



SELECT from an UPDATE/DELETE



- New **INCLUDE** clause in version 9 allows for more elimination of SQL statements replaced by single complex statement
 - Allows additional columns to be provided in the **SELECT** list
 - These additional columns are not included in the operation

What if I update **ACCOUNT** with today's payment, and return both the previous and the new balance?

```
SELECT ACCT_BAL, OLD_BAL
INTO :NEW-ACCT-BAL, :ACCT-OLD-BAL
FROM FINAL TABLE
(UPDATE ACCOUNT INCLUDE(OLD_BAL DEC(9,2))
SET PAY_AMT = :PAY-AMT
,ACCT_BAL = ACCT_BAL + :PAY-AMT
,PAY_DATE = CURRENT DATE
,UPD_TSP= CURRENT TIMESTAMP
,OLD_BAL = ACCT_BAL
WHERE ACCT_ID = :ACCT-ID)
```

New column returned here

New column defined here

New column value set here

SELECT from an UPDATE/DELETE



- You can **SELECT** from a **DELETE** the same as from an **INSERT** or **UPDATE**
 - The **OLD TABLE** clause gets the deleted row(s)
 - This clause also works for **UPDATE**

What if the **ACCOUNT** is being closed and deleted, but I wanted to return the account information?

```
SELECT ACCT_BAL, PAY_DATE  
INTO :ACCT-BAL, :PAY-DATE  
FROM OLD TABLE  
(DELETE FROM ACCOUNT  
WHERE ACCT_ID = :ACCT-ID)
```

Scalar Fullselect



```
SELECT COLA,  
      (SELECT COLB FROM TABLE2 WHERE COLB1 = T1.COLA),  
      (SELECT COLC FROM TABLE3 WHERE COLC1 = T1.COLA)  
FROM TABLE1 T1
```

- The SQL language is becoming more orthogonal
 - Any expression can be placed anywhere in a SQL statement
 - A SQL statement is an expression
 - A SELECT statement
 - A WHERE clause
 - A CASE statement
- Scalar fullselects within a SQL statement can be in many ways a performance advantage!
 - Reduction of SQL statements issued
 - Utilizing correlation for transaction performance
 - Creating complex SQL statements

Scalar Fullselect – Reducing SQL Calls



```
SELECT COLA  
FROM TABLE1 T1  
WHERE COLB BETWEEN  
  (SELECT MIN(COL1) FROM TABLE1)  
AND  
  (SELECT MAX(COL1) FROM TABLE1)
```

Instead of three queries to get an answer, only one query is issued. Scalar fullselect provides this solution

- We can now use SQL to reduce the number of times we go to the database
 - Quantity of SQL is a major contributor to elapsed and CPU time
 - Are you using multiple queries to get data from the database?
 - You should investigate getting those answers into a single query if you want to save CPU
- Getting answers into a single SQL statement makes the SQL more portable as well as improving performance

Scalar Fullselect – Correlation



30
secs.

```
SELECT DISTINCT A.CUST_ID,  
                B.BUNDLE_FLG,  
FROM CUST_PRDT_PRC_SUMM A  
LEFT OUTER JOIN  
(SELECT DISTINCT CUST_ID,  
 1 as BUNDLE_FLG  
FROM CUST_PRC_SUMM_CMPT) B  
ON A.CUST_ID = B.CUST_ID
```

This query performs an existence check on the CUST_PRC_SUMM_CMPT table, but the table expression is materialized due to DISTINCT and the entire table is read

<1
sec.

```
SELECT DISTINCT A.CUST_ID,  
(SELECT MAX(CUST_ID)  
FROM CUST_PRC_SUMM_CMPT B  
WHERE A.CUST_ID = B.CUST_ID) AS BUNDLE_FLG  
FROM CUST_PRDT_PRC_SUMM A;
```

This query returns the same result, but correlation is used to probe the inner table and avoid materialization and using an index on CUST_ID

- Use correlation for transactions when outer table processes relatively little data
 - Encourages nested loop join
 - Encourages index access

Scalar Fullselect – Advanced SQL



```
SELECT ACCT_NUM, ACCT_RATING_CDE,  
       (SELECT MAX(DOLLAR_AMT)  
        FROM ACCT_HIST B  
         WHERE B.ACCT_ID = A.ACCT_ID)  
FROM ACCOUNT A  
WHERE ACCT_ID = 543670001
```

This query returns both line item and aggregate data in one statement!

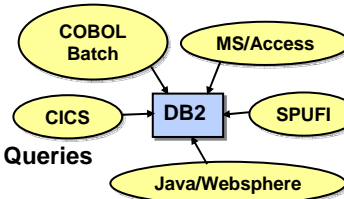
- **Advanced SQL**

- **Performance Improvement**

- Data is Filtered or Aggregated
 - Transactions Process Little Data
 - One Large Query Instead of Many Small Queries
 - Little or No Logic in SELECT List
 - Avoid UDFs, CASE Expressions, Data Conversions

- **Flexibility**

- SQL is Highly Portable
 - Relatively Easy to Code – Fast Time to Delivery
 - Write a SQL statement in one place, and run it anywhere



Common Table Expressions



Common table expressions are materialized on first reference

```
WITH DEPTOTAL (DEPT_NO, MAXSALARY) AS  
  ( SELECT DEPT_NO, SUM(SALARY+COMM)  
    FROM EMP  
    GROUP BY DEPT_NO )  
SELECT DEPT_NO  
FROM DEPTOTAL  
WHERE MAXSALARY =  
  ( SELECT MAX(MAXSALARY)  
    FROM DEPTOTAL )
```

DEPTOTAL referenced here

The WITH clause defines a table called DEPTOTAL. This table can be used within the SQL statement, including other common table expressions

- A table defined in a SQL statement that exists for the duration of that SQL statement
 - Can provide performance improvements by computing a value once, not multiple times
 - Can be used in SELECT, CREATE VIEW and INSERT
 - Can contain references to host variables
 - Simplifies complex SQL to reduce possibility of programmer error
- Common table expressions are extremely powerful and enable
 - Complex SQL
 - Reduction of SQL issued

Common Table Expressions



WITH DEPTOTAL (DEPT_NO, MAXSALARY)
AS

```
SELECT DEPT_NO, SUM (SALARY+COMM)
FROM EMP
GROUP BY DEPT_NO
```

```
SELECT DEPT_NO FROM DEPTOTAL
WHERE MAXSALARY =
```

```
(SELECT MAX(MAXSALARY)
FROM DEPTOTAL)
```

EMP

DEPT_NO	SALARY	COMM
1	10,000	1,000
1	20,000	1,000
2	10,000	1,000
2	30,000	2,000

```
DEPT_NO  MAXSALARY
1         21,000
2         32,000
```

```
DEPT_NO  Final
2         ← Result
```

```
MAXSALARY
32,000
```

Common Table Expressions



- Prior to V8 searching on multiple values was difficult

```
SELECT PERSON_ID
FROM PERSON_TBL
WHERE (LASTNAME = 'SMITH'
      OR LASTNAME = 'JONES')
AND FIRST_INIT = 'A'
```

V7

Using an index on LASTNAME, FIRST_INIT, this statement can only index match on LASTNAME

```
SELECT PERSON_ID
FROM PERSON_TBL
WHERE LASTNAME = 'SMITH'
AND FIRST_INIT = 'A'
UNION ALL
SELECT PERSON_ID
FROM PERSON_TBL
WHERE LASTNAME = 'JONES'
AND FIRST_INIT = 'A'
```

V7

- V8 Gives us more choices

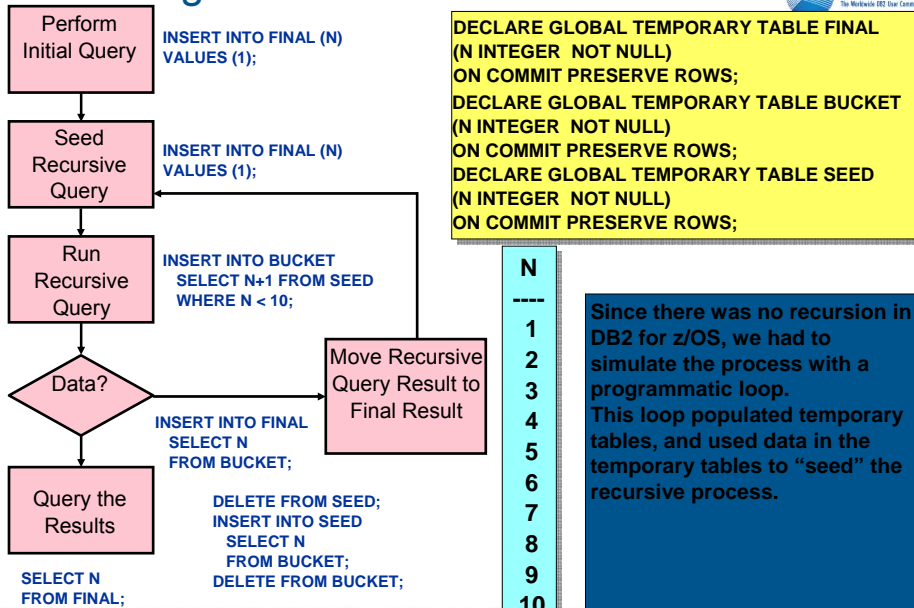
```
WITH PRSN_SEARCH(LASTNAME) AS
(SELECT 'SMITH' FROM SYSIBM.SYSDUMMY1
UNION ALL
SELECT 'JONES' FROM SYSIBM.SYSDUMMY1)
FROM PERSON_TBL A, PRSN_SEARCH B
WHERE A.LASTNAME = B.LASTNAME
AND FIRST_INIT = 'A'
```

V8

This statement matches on both indexed columns!

This query may use both columns of the index, but we still probe twice. This could also be in multiple statements!

Simulating Recursion in V7



Since there was no recursion in DB2 for z/OS, we had to simulate the process with a programmatic loop. This loop populated temporary tables, and used data in the temporary tables to "seed" the recursive process.

Recursive Solution in V8



```
WITH TEMP(N) AS  
(SELECT 1  
 FROM SYSIBM.SYSDUMMY1  
 UNION ALL  
 SELECT N+1  
 FROM TEMP  
 WHERE N < 10)  
 SELECT N  
 FROM TEMP
```

Initial select sets the starting values

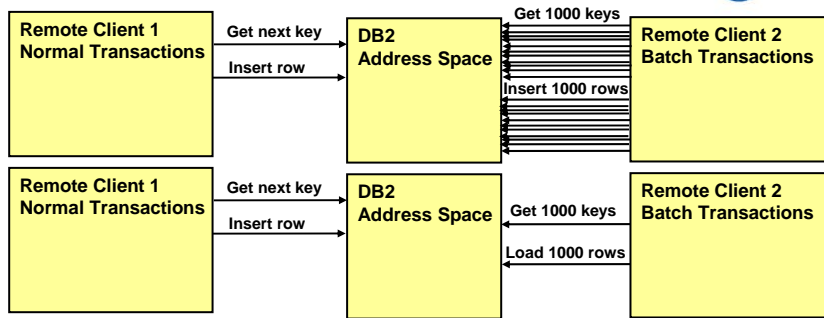
Iterative Select increments the value

Result

N
1
2
3
4
5
6
7
8
9
10

- Uses same syntax as common table expressions
- Ability to iterate in the SQL statement
- Be sure to add controls to stop
 - Else...SQL +347 The recursive common table expression "TEMP" may contain an infinite loop. SQLSTATE=01605

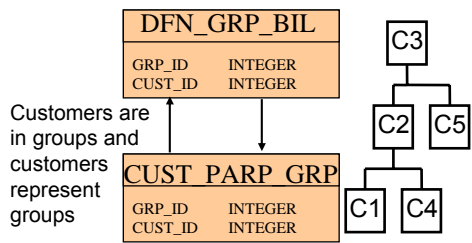
Recursion Reducing Calls



- Application needed a system generated key
 - Sequence object
 - Identity column
- Both batch and online can insert rows
 - Batch is high volume
 - message traffic was killing performance
- V8 to the rescue
 - Get 1000 keys in one message

```
WITH GET_THOUSAND (C) AS
(SELECT 1
 FROM SYSIBM.SYSDUMMY1
 UNION ALL
 SELECT C+1
 FROM GET_THOUSAND
 WHERE C < 1000)
SELECT NEXT VALUE FOR SEQ1
FROM GET_THOUSAND;
```

Recursion Reducing Calls



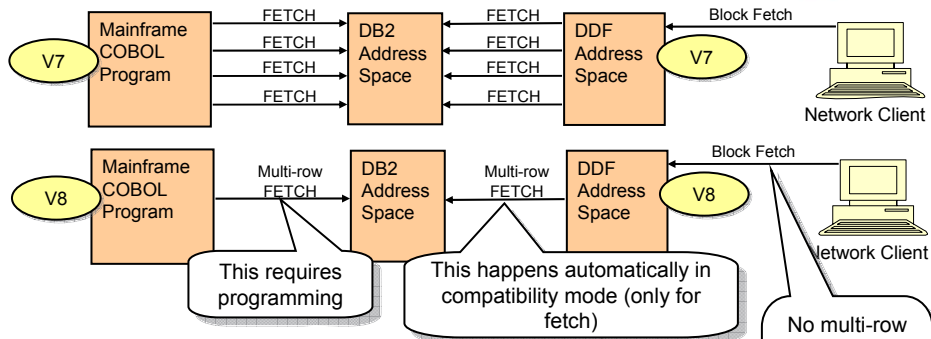
- Imagine that Customers C1, C4, and C5 have all purchase \$1.00 worth of products. The activity query returns: C1 - \$1, C4 - \$1, C5 - \$1
- Then the group query returns the activity query: C1 - \$1, C4 - \$1, C5 - \$1
- C2 - \$1 for C1 participation in C2
- C3 - \$1 for C2 participation in C3 (rolled up from C1)
- C2 - \$1 for C4 participation in C2
- C3 - \$1 for C2 participation in C3 (rolled up from C4)
- C3 - \$1 for C5 participation in C3

```

WITH GROUPS(CUST_ID, TOT_AMT) AS
(SELECT CUST_ID, TOT_AMT
 FROM ACTIVITY
 UNION ALL
 SELECT C.CUST_ID, A.TOT_AMT
 FROM GROUPS A,
      CUST_PARP_GRP_BIL B,
      DFN_GRP_BIL C
 WHERE A.CUST_ID = B.CUST_ID
 AND B.GRP_ID = C.GRP_ID)
SELECT CUST_ID,
      SUM(TOT_AMT) AS TOT_AMT
 FROM GROUPS
 GROUP BY CUST_ID
 ORDER BY CUST_ID;
  
```

- Prior to V8 this was solved with multiple statements
 - Application ran for 3 days
- After V8 this was solved in one statement
 - Application ran for 5 minutes!!!!

Multi-Row Fetch



- **Multi-row fetch is specifically designed to save CPU**
 - It is enabled for local application via specific SQL syntax
 - It is enabled for remote applications automatically
 - For block fetching cursors
- **Going across address spaces is expensive**
 - If we get multiple rows in one operation we can save CPU

Multi-Row Fetch

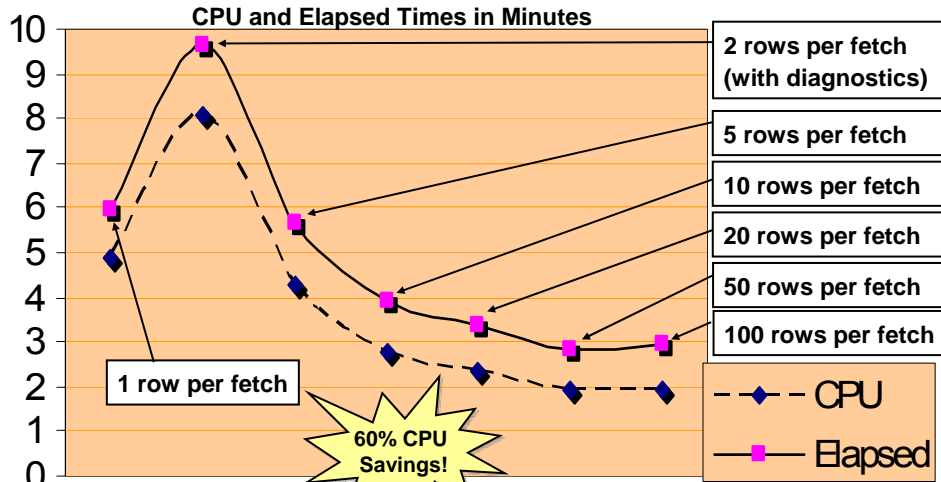


- **Ability to return multiple rows on a single API call**
 - **Potential performance improvement up to 50%**
- **Static or dynamic**
- **Scrollable or non-scrollable cursor support**
- **Wide Cursors**
 - **Multiple rows, instead of one**
- **Supports positioned UPDATE and DELETES**
 - **But there are caveats**
- **Watch out for locks!!**
 - **Every row being processed by a multi-row fetch must be locked**
 - **Cursor is position on all rows in current rowset**
 - **Locks will depend on isolation level and whether or not a result table is materialized**
- **Can mix single row and multi-row fetch in same cursor**
 - **FETCH NEXT is relative to first row in current rowset – Be careful!**

Multi-Row Fetch - Impact



Local Cobol Test Sequential Read of Approximately 39 Million Rows, 5 Columns



Local Cobol Test

50,000 Random Read Requests at about 20 Rows Each

- Application is very dependent upon I/O response in a random application
 - No elapsed time improvement should be expected
 - Should get CPU improvement
- Actual result
 - One row fetch
 - 592 seconds elapsed, 22.5 seconds CPU
 - Twenty row fetch
 - 626 seconds elapsed, 16.7 seconds CPU
 - 25% CPU savings!

Remote Application

- Remote clients
 - SQLJ client experienced no CPU increase when moved from V7 to V8.
Multi-row fetch offset additional V8 CPU cost

Multi-Row Insert



- Ability to insert multiple rows with one API call
 - Potential 25% CPU reduction
- Static or Dynamic Support
 - For static, FOR 'n' ROWS on INSERT statement
 - For dynamic, FOR 'n' ROWS on EXECUTE statement
- Input values provided by host variable arrays
- Supports host language arrays
 - Assembler, C, C++, Cobol, PL/I
- Helps with application portability
- Great performance over single row inserts over a network
 - Rowset size becomes block size for query block
 - If 10 rows in rowset that is the block – 32K block size is turned off
- Insert up to 32767 rows per API call
- TEST PERFORMANCE OF THIS
 - Verses single row loop

```
INSERT INTO TABLE1
VALUES (:valuearray1,
       :valuearray2)
FOR hv: ROWS
ATOMIC
```

V9 - MERGE Statement



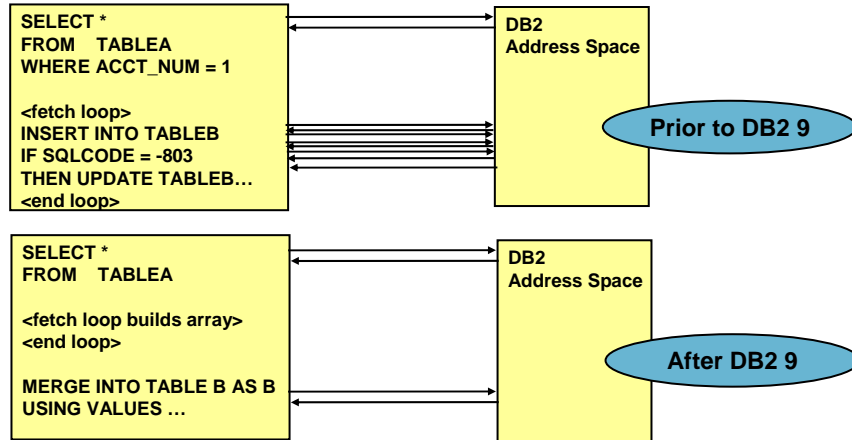
- **New statement combining conditional INSERT and UPDATE operations on a target using values from a source**
- **Given a set of input rows**
 - **Update the target table if the key exist**
 - **Insert the row for keys that do not exist**
- **Previously may have been accomplished with a conditional INSERT or UPDATE**
- **Good for OLTP applications**

```
MERGE INTO main_account AS M
USING VALUES (:hv_acnum, :hv_amnt)
FOR 4 ROWS AS B(acct_num, amount)
ON M.acct_num = B.acct_num
WHEN MATCHED THEN
UPDATE SET balance = M.balance + B.amount
WHEN NOT MATCHED THEN
INSERT (acct_num, balance) VALUES (b.acct_num, b.amount)
NOT ATOMIC CONTINUE ON SQLEXCEPTION
```

V9 - MERGE for Data Replication



- In many situations applications are written to replicate data from one database to another
 - Use of MERGE can dramatically reduce the quantity of SQL statements required to do this!



ORDER BY and FETCH FIRST



- In a **SUBSELECT** or **FULLSELECT**
- Can help reduce I/O or CPU

List the accounts and status for top 5 outstanding bad accounts

```
SELECT ACCT_ID, ACCT_STATUS
FROM ACCOUNT
WHERE ACCT_ID IN
  (SELECT ACCT_ID
   FROM BAD_ACCOUNTS
   ORDER BY AMOUNT_OWED DESC
   FETCH FIRST 5 ROWS ONLY)
```

← Sorted in memory

- Improvement to performance by also doing the sorting in memory
 - Only final results are written to a workfile
 - (Data size + key size) x # rows must be ≤ 32704 (must fit on 32Kpage)
 - Otherwise a normal tournament sort is performed

ORDER BY and FETCH FIRST



- This is an exciting and powerful feature
 - It allows for all sorts of powerful existence checking

List the balance and most recent status for account 'A000010'

```
SELECT A.ACCT_BAL, H.ACCT_STATUS
FROM ACCOUNT A, ACCT_STS_HIST H
WHERE A.ACCT_ID = H.ACCT_ID
AND A.STATUS_DTE =
  (SELECT MAX(STATUS_DTE)
   FROM ACCT_STS_HIST X
   WHERE X.ACCT_ID = H.ACCT_ID)
AND A.ACCT_ID = 'A000010';
```

Status is stored in a history table

Which will be the performance winner?

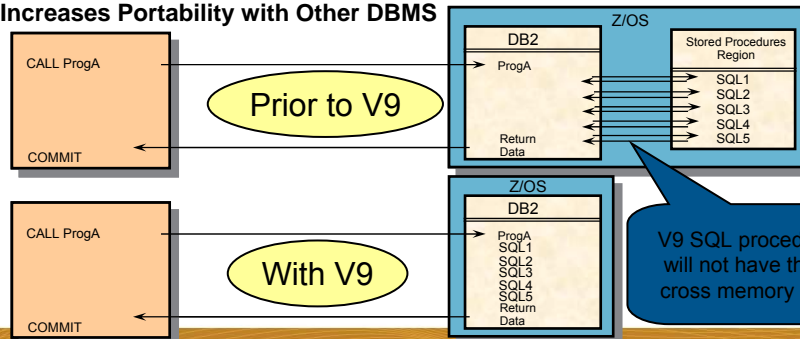
```
SELECT A.ACCT_BAL, H.ACCT_STATUS
FROM ACCOUNT A,
TABLE (SELECT ACCT_STATUS
       FROM ACCT_STS_HIST X
       WHERE A.ACCT_ID = X.ACCT_ID
       ORDER BY STATUS_DTE DESC
       FETCH FIRST 1 ROW ONLY) AS H
WHERE A.ACCT_ID = 'A000010';
```

FETCH FIRST and ORDER BY gives the same result with one probe of ACCT_STS_HIST

Native SQL Procedure Support – V9



- **SQL Stored Procedures Prior to V9**
 - SQL procedures ultimately become C programs
 - Procedures execute in WLM managed stored procedure address space
- **Native SQL Procedure Language Support**
 - Does not require a C compiler
 - Instead of a C program you get a runtime DB2 structure
 - Dramatic performance improvement since no cross-memory call to the stored procedure address space
- **Increases Portability with Other DBMS**



Conclusion



- **Reduce SQL calls**
- **Reduce quantity of calls with V8 SQL features**
- **Reduce quantity of calls with V9 SQL features**
- **Be prepared to detect SQL problems of this nature**
- **Improve overall SQL performance via performing less trips across address spaces**

DB2 V9 Manuals



- DB2 V9.1 for z/OS Administration Guide, SC18-9840-00
- DB2 V9.1 for z/OS Application Programming and SQL Guide, SC18-9841-00
- DB2 V9.1 for z/OS Application Programming Guide and Reference for JAVA SC18-9842-00
- DB2 V9.1 for z/OS Codes, GC18-9843-00
- DB2 V9.1 for z/OS Command Reference, SC18-9844-00
- DB2 V9.1 for z/OS Data Sharing: Planning and Administration, SC18-9845-00
- DB2 V9.1 for z/OS Diagnosis Guide and Reference, LY37-3218-00
- DB2 V9.1 for z/OS Diagnostic Quick Reference, LY37-3219-00
- DB2 V9.1 for z/OS Installation Guide, GC18-9846-00
- DB2 V9.1 for z/OS Introduction to DB2, SC18-9847-00
- DB2 V9.1 for z/OS Licensed Program Specifications, GC18-9848-00
- DB2 V9.1 for z/OS Messages, GC18-9849-00
- DB2 V9.1 for z/OS ODBC Guide and Reference, SC18-9850-00
- DB2 V9.1 for z/OS Performance Monitoring and Tuning Guide, SC18-9851-00
- DB2 V9.1 for z/OS RACF Access Control Module Guide, SC18-9852-00

DB2 Information on the Web



- IBM
 - ibm.com
- IBM Software
 - ibm.com/software
- DB2 Family
 - ibm.com/software/db2
- DB2 Solutions Directory Applications
 - ibm.com/developerworks/db2
- "Red Books"
 - ibm.com/redbooks
- DB2 for z/OS
 - ibm.com/software/db2zos
- DB2 Support
 - ibm.com/software/db2zos/support.html
- DB2 for z/OS Papers
 - <ftp://ftp.software.ibm.com/software/data/db2zos>
- DB2 Magazine
 - <http://www.db2mag.com>
- DB2 Certification
 - <http://www.ibm.com/certify>
- DB2 Experts
 - www.db2expert.com

Session E08



Quit Calling DB2
So Much!

Susan Lawson and Dan Luksetich

YL&A

Susan_Lawson@ylassoc.com

Dan_Luksetich@ylassoc.com