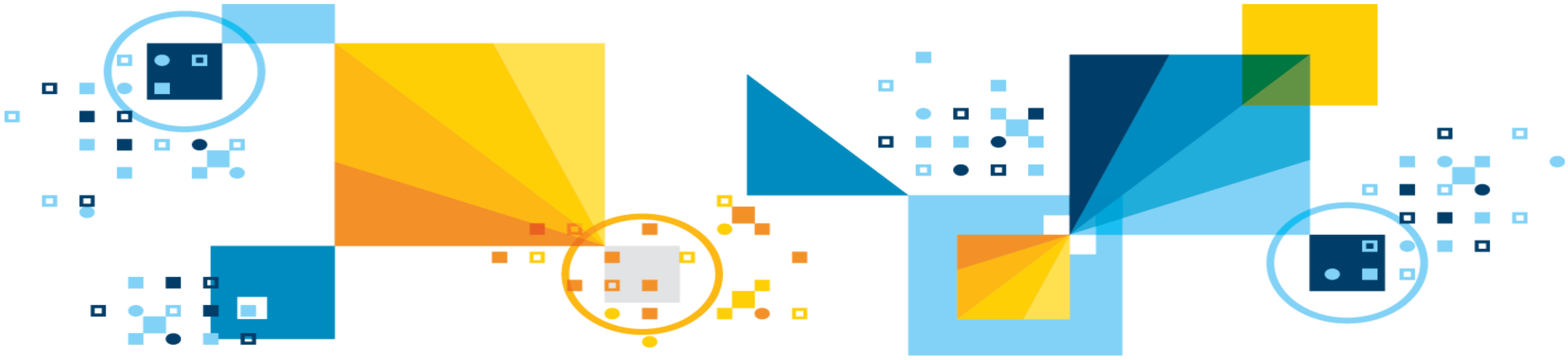
IBM Analytics

Robert Catterall
IBM Senior Consulting Db2 for z/OS Specialist
rfcatter@us.ibm.com

Db2 for z/OS Stored Procedures: the System Perspective

TRIDEX
June 11, 2019



Agenda

- A brief history of Db2 for z/OS stored procedure functionality
- The performance advantages of native SQL procedures
- Security advantages of stored procedures
- Managing native SQL procedures
- External Db2 stored procedures
- Can stored procedures be over-used in a Db2 for z/OS environment?

A brief history of Db2 for z/OS stored procedure functionality

Introduction and early enhancements

Db2 for z/OS V4	▪ Stored procedure functionality introduced
Db2 for z/OS V5	▪ Caller of stored procedure can fetch rows from cursor declared and opened in stored procedure ▪ WLM-managed stored procedure address spaces introduced (the Db2-managed stored procedure address space went away with Db2 9) ▪ Support for Java stored procedures
Db2 for z/OS V6	▪ DDL support: CREATE, ALTER, DROP PROCEDURE ▪ SYSPROCEDURES catalog table replaced by SYSROUTINES

SQL Procedure Language (SQL PL) introduced with Version 7

- A way to code Db2 stored procedures using only SQL
  - Enabled via introduction of a new category of SQL statements, called control statements (referring to logic flow control)
    - Examples: IF, WHILE, ITERATE, LOOP, GOTO

```
CREATE PROCEDURE divide2
   (IN numerator INTEGER, IN denominator INTEGER,
    OUT divide_result INTEGER)
   LANGUAGE SQL
   BEGIN
     DECLARE overflow CONDITION FOR SQLSTATE '22003';
     DECLARE CONTINUE HANDLER FOR overflow
       RESIGNAL SQLSTATE '22375';
     IF denominator = 0 THEN
       SIGNAL overflow;
     ELSE
       SET divide_result = numerator / denominator;
     END IF;
   END
```

This is a compound SQL statement – the "body" of a stored procedure coded in SQL PL

From the Db2 V8 *SQL Reference*

Db2 V7 SQL procedures – pro and con

- Pro:
  - Expanded the pool of people who could develop Db2 for z/OS stored procedures
- Con:
  - As part of program preparation, a Db2 Version 7 (or 8) stored procedure written in SQL PL was converted into a C language program with embedded SQL DML statements
    - C language programs generally not as CPU-efficient as COBOL programs in a z/OS system, so Db2 for z/OS users tended to favor COBOL stored procedures over “external” SQL procedures

Important breakthrough: Db2 9 native SQL procedures

- Key characteristics:
  - No external-to-Db2 executable (no object or load module) – a native SQL procedure's package is its one and only executable
  - Executes in DBM1, the Db2 database services address space (as do all packages) – not in a stored procedure address space
  - Runs under caller's task (an external stored procedure runs under its own TCB in a stored procedure address space)
  - Superior functionality
    - Native SQL procedures – not external stored procedures – are where we have seen greatest advances in stored procedure functionality

More enhancements: Db2 10 SQL PL user-defined functions

- Officially called compiled SQL scalar functions, you can think of them as “native” SQL UDFs
  - As is true for a native SQL procedure, a “native” SQL UDF’s package is its one and only executable
  - Also like a native SQL procedure, a “native” SQL UDF executes in the Db2 DBM1 address space, and under the task of the invoking application process
- Also new with Db2 10: compiled SQL scalar function can have scalar fullselect in RETURN part of CREATE FUNCTION
  - Previously, could not even reference a column in RETURN part of SQL UDF
- Also new with Db2 10: SQL table functions
  - A table UDF returns a set or rows to an invoking application process
  - You could have table UDFs before Db2 10, but they could not be written in SQL PL

Example of a “native” SQL UDF

```
CREATE FUNCTION REVERSE(INSTR VARCHAR(4000))
  RETURNS VARCHAR(4000)
  DETERMINISTIC NO EXTERNAL ACTION CONTAINS SQL
  BEGIN
    DECLARE REVSTR, RESTSTR VARCHAR(4000) DEFAULT '';
    DECLARE LEN INT;
    IF INSTR IS NULL THEN
     RETURN NULL;
    END IF;
    SET (RESTSTR, LEN) = (INSTR, LENGTH(INSTR));
    WHILE LEN > 0 DO
     SET (REVSTR, RESTSTR, LEN)
       = (SUBSTR(RESTSTR, 1, 1) CONCAT REVSTR,
       SUBSTR(RESTSTR, 2, LEN - 1),
       LEN - 1);
    END WHILE;
    RETURN REVSTR;
  END#
```

From the Db2 10 *SQL Reference*

Body of “native” SQL UDF

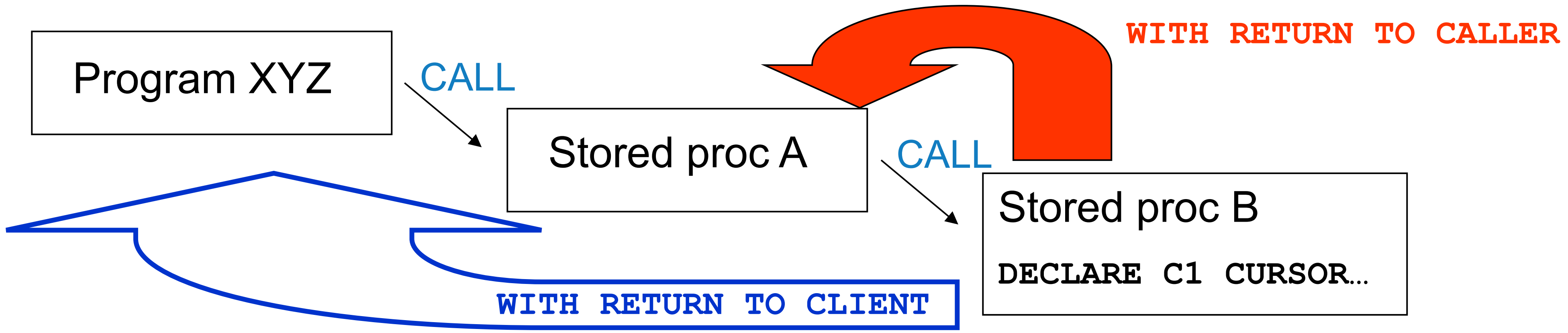
A SQL UDF with a scalar fullselect

```
CREATE FUNCTION UNIT_SALES_COUNT(X CHARACTER(6))
RETURNS INTEGER
LANGUAGE SQL
CONTAINS SQL
NO EXTERNAL ACTION
DETERMINISTIC
RETURN
 (SELECT SUM(UNIT_SALES)
  FROM PRODUCT_SALES
  WHERE PRODUCT_ID = X);
```

Could not have this in RETURN statement before DB2 10

Another Db2 10 enhancement: RETURN TO CLIENT cursors

- Before Db2 10: stored procedure could return query return result set 1 level up in chain of nested stored procedures (cursor declared WITH RETURN TO CALLER)
  - Making stored procedure cursor's result set available more than one level up often required use of temporary table for intermediate storage of rows
- Db2 10 introduced WITH RETURN TO CLIENT cursor specification
  - Enables "top-level" program to directly fetch result set rows
  - Better performance, simpler coding versus use of intermediate temporary table



Some Db2 11 and Db2 12 stored procedure enhancements

- Db2 11
  - Db2 array (a Db2 user-defined data type) can be input to or output from a stored procedure (if it is a native SQL procedure and the caller is another SQL PL routine, or a Java or .NET client using a type 4 driver to access Db2)
  - Autonomous procedure: type of native SQL procedure that has separate unit of work versus caller
    - What that means: if transaction program calls autonomous procedure, and that procedure inserts row into table and returns control to calling program, and calling program fails, Db2 will back out all data-change work done by or on behalf of failing program, *but not updates done by autonomous procedure*
    - Can make it much easier to implement, for example, a "transaction audit trail"
- Db2 12
  - Advanced trigger: a new type of trigger, the body of which can contain a SQL PL routine
  - Makes it much easier to create and implement triggers with advanced functionality (previously, enabling advanced functionality via a trigger often required having trigger call a stored procedure)

The performance advantages of native SQL procedures

zIIP offload

- A native SQL procedure's execution is up to 60% zIIP-offload-able when the procedure is called through the Db2 distributed data facility (DDF)
  - CALL could be issued by client program through IBM Data Server Driver (or Db2 Connect), or could be executed by client program using Db2's REST interface
  - SQL procedure is up to 60% zIIP eligible when called through DDF because SQL that runs under a preemptable SRB in the DDF address space gets that level of zIIP offload
    - External stored procedure (e.g., one written in language such as COBOL or C) always executes under a TCB in a stored procedure address space, and so is NOT zIIP-eligible even when called via DDF
    - **Note:** if DDF-using application called external stored procedure and that procedure calls native SQL procedure, latter will NOT be zIIP-eligible because it will execute under external stored procedure's TCB (a native SQL procedure never has its own task – it always runs under the task of its caller)
- That zIIP offload advantage is shared by other SQL PL routines (compiled SQL scalar functions or advanced triggers) executed by DDF-using applications

Elimination of TCB wait time

- When Db2-accessing program calls an external stored procedure, the external stored procedure has to be scheduled for execution and calling program's Db2 thread has to be switched from caller's task to external stored procedure's task
  - That can lead to two performance-impacting issues:
    1. If the z/OS system is really busy, there can be a delay in scheduling external stored procedure for execution (especially if stored procedure address space has a too-low priority – more on that to come)
    2. Switching Db2 thread from caller's task to external stored procedure's task can introduce some delay
- No scheduling, no thread-switch delays for native SQL procedure, because it never has its own task (always runs under task of caller – it is just a package)
  - This advantage of native SQL procedures extends to compiled SQL scalar functions
    - Consider UDF that is referenced in inner query of a correlated subquery – if outer query qualifies a large number of rows, UDF could be invoked thousands of times, leading to thousands of task switches and elevated "TCB wait" time if UDF is external (no "TCB wait" time if UDF written in SQL PL)

Another performance booster: high-performance DBATs

- These DBATs improve performance for external <u>and</u> native SQL procedures
- A DBAT becomes a high-performance DBAT when a package bound with RELEASE(DEALLOCATE) is allocated for execution to the DBAT
  - The package in question could be associated with a stored procedure
- At transaction completion, a high-performance DBAT will stay dedicated to the connection through which it was instantiated, and can be reused 200 times
  - CPU efficiency advantage: RELEASE(DEALLOCATE) packages (and associated table space locks) remain allocated to DBAT until it is terminated (versus being released at each COMMIT)
    - Table space-level locks are almost always non-exclusive
  - In addition to RELEASE(DEALLOCATE) packages, need DDF to be enabled for high-performance DBATs (can be done via Db2 command -MODIFY DDF PKGREL(BNDOPT))
    - -MODIFY DDF PKGREL(COMMIT) can be used to temporarily turn high-performance DBATs off – might need to do that if high-performance DBATs could interfere with BIND/REBIND/ALTER work

Security advantages of stored procedures (native and external)

Security: static SQL

- Stored procedures provide a means of packaging static SQL in a form that can be dynamically invoked by client programs
- When SQL DML statements are static and issued by way of a stored procedure, invoking application's authorization ID does not require table access privileges (SELECT, INSERT, UPDATE, DELETE)
  - Instead, the application's ID requires only EXECUTE privilege on stored procedure
  - Even more secure: do not grant package EXECUTE privilege to application's authorization ID
    - Instead, create role, grant execute on stored procedure to role, and create trusted context that restricts use of role's privileges to a particular application ID connecting from the IP address (or addresses) of the server on which the application runs
    - That way, if someone tries to use the application's Db2 credentials (ID and password) from a PC (for example), they will not be able to access any data because the ID has no Db2 privileges

Security: database schema abstraction

- If someone wants to hack into your database, he will have an easier time of it if he knows names of tables and columns
- When “table-touching” SQL statements are packaged in stored procedures, stored procedure developers require knowledge of database details but developers of calling programs do not
  - Data security is boosted by limiting the number of people with detailed knowledge of the database schema

Managing native SQL procedures

ALTER REGENERATE versus REBIND PACKAGE

- Should you REBIND a native SQL procedure's PACKAGE, or execute ALTER PROCEDURE with REGENERATE to regenerate the package?
  - Similarly, for compiled SQL scalar function can execute ALTER FUNCTION with REGENERATE
- Key difference:
  - REBIND PACKAGE affects <u>only non-control</u> part of SQL PL routine's package (non-control SQL statements include DML statements such as SELECT and INSERT), while ALTER PROCEDURE with REGENERATE regenerates a package in its entirety
    - Control SQL statements control SQL PL routine's logic flow – they include IF, WHILE, ITERATE, LOOP
- Sometimes, you just want to REBIND a native SQL procedure's package
  - Example: REBIND to get new access path for query after creating new index on a table
- Sometimes, you want to regenerate native SQL procedure's package in its entirety
  - Example: after migrating to new version of Db2 (regenerated package code more CPU-efficient)

More on ALTER PROCEDURE with REGENERATE

- Like REBIND PACKAGE, REGENERATE can lead to access path changes for SQL DML statements in a SQL PL routine
  - REBIND option APREUSE (reuse access paths) is not an option for REGENERATE
  - Also, plan management (enabling restoration of previous instance of package via REBIND SWITCH) not applicable to REGENERATE
  - Recommendation: use REBIND PACKAGE instead of ALTER PROCEDURE with REGENERATE if there is not a need to regenerate control section of SQL PL routine’s package
  - If you do want to REGENERATE a SQL PL routine’s package, consider doing a REBIND PACKAGE first
    - After the REBIND PACKAGE, check to see if access paths changed (APCOMPARE on REBIND can help here)
    - If access paths did not change (or changed for the better), you should be able to REGENERATE package without a negative performance surprise (access paths likely to stay the same if the REGENERATE closely follows the REBIND PACKAGE)

Changing native SQL procedure: ALTER or drop/re-create?

- Sometimes DBAs will use drop and re-create versus ALTER PROCEDURE to make changes to a native SQL procedure
- That approach can be problematic when there are nested stored procedure calls (i.e., one stored procedure calls another)
  - If stored procedure PROC_A calls PROC_B, *and PROC_A is a native SQL procedure*, an attempt to drop PROC_B will fail
    - You can check SYSPACKDEP catalog table to see if any native SQL procedures are dependent on a given stored procedure (DTYPE = ‘N’)
  - You could try dropping PROC_A, then dropping PROC_B, then changing PROC_B via re-create, then re-create PROC_A, but what if PROC_A is called by a native SQL procedure?
  - Recommendation: if you can accomplish change with ALTER PROCEDURE, do that instead of drop/re-create
    - Drop/re-create needed if you need to change number or data type of procedure’s parameters

More on ALTER PROCEDURE

- When ALTER PROCEDURE is used to change one or more options (e.g., ASUMTIME) for the active version of a native SQL procedure, some package invalidation is likely to occur
  - Change might invalidate the package of the named SQL procedure, or the packages of programs that call the SQL procedure, or both
- With that in mind, consider making native SQL procedure changes via a 2-step process – neither active version of the target SQL procedure nor calling programs will be impacted
  1. Make the change using ALTER PROCEDURE with ADD VERSION
     - On that statement you'll include the list of parameters for the procedure, any options that will have non-default values, and the body of the procedure
  2. Activate this new version using ALTER PROCEDURE with ACTIVATE VERSION

Native SQL procedure deployment

- With a native SQL procedure’s package being its only executable (i.e., no object or load module), deployment is different versus external stored procedures
- To get native SQL procedure from one environment to another (e.g., test to prod), recommended action is BIND PACKAGE with DEPLOY versus re-create on target
  - Non-control section of procedure’s package will be reoptimized on production system, but control section will not be regenerated – no need for that
- If a previous version of the native SQL procedure is already running in production, control when the new version becomes active via ACTIVATE VERSION option of ALTER PROCEDURE
  - Selective execution of the new version before then (to verify that it functions properly) can be enabled via the CURRENT ROUTINE VERSION special register
  - Example, if new version is 5, have some users check out version 5, then make 5 active version

Ensuring that stored procedure coding standards are followed

- One factor: strictly limit authorization IDs that can issue CREATE PROCEDURE on the production system, and IDs that can issue BIND PACKAGE with DEPLOY (to move a native SQL procedure to production from a test/development system)
- Question: if a given stored procedure does not follow a coding standard, can we determine who wrote the procedure?
  - Answer: yes – primary authorization ID of process that created stored procedure (native SQL or external) is recorded in CREATEDBY column of SYSIBM.SYSROUTINES table in Db2 catalog
- With CREATE PROCEDURE (in production) and BIND PACKAGE authorization restricted to a few IDs, stored procedure code can be reviewed before promotion to production
  - Code review can be largely automated through use of IBM DevOps Experience for z/OS (new offering), in conjunction with IBM UrbanCode Deploy

External Db2 stored procedures

Are there still good use cases for external stored procedures?

- Yes – two examples:
  - Need for functionality beyond what can be provided by a native SQL procedure
    - That would have to be pretty advanced functionality – how advanced does data layer have to be?
  - Access to data outside of Db2 (native SQL procedure can only issue SQL statements)
    - If stored procedure will access data in VSAM file, and volume of execution will be fairly high, good idea to have the stored procedure access the VSAM file through a CICS transaction (if you have CICS)
    - Why? Because if a stored procedure directly accesses VSAM data, the VSAM file will be opened and closed for each execution of the stored procedure – not good for scalability
    - When VSAM data is accessed by way of a CICS transaction, the file is opened and allocated to CICS, *and it stays open and allocated to CICS*
  - Note that there are ways in which a native SQL procedure can access data outside of Db2
    - A native SQL procedure can call another stored procedure
    - A native SQL procedure can interact with MQ via Db2 functions such as MQSEND and MQRECEIVE
    - A native SQL procedure can consume Web services via Db2 functions such as SOAPHTTPNV

COBOL stored procedures versus COBOL subroutines

- Suppose that you have a COBOL program that needs to invoke another COBOL program – should the second COBOL program be invoked as a subroutine (COBOL CALL) or as a stored procedure (SQL call)?
  - From a CPU efficiency perspective, right choice is COBOL CALL
    - If COBOL stored procedure is invoked via SQL call, that involves a separate TCB (to which first COBOL program's Db2 thread must be switched) in separate address space (stored procedure address space)
    - COBOL subroutine would run under caller's task in caller's address space
  - Could be a code re-use argument for using SQL CALL to invoke COBOL stored procedure
    - The COBOL stored procedure could be invoked by any process that can issue a SQL statement
  - If secondary COBOL program will be invoked very frequently from other COBOL programs and invoked in some cases by non-COBOL programs, may want to maintain it in both subroutine and Db2 stored procedure form
    - Oftentimes, very little code difference between those two forms

Java stored procedures

- Certainly a viable option
  - Java has been supported on z/OS systems for 20 years
  - Huge improvement in Java performance in z/OS environment, especially over past 10 years
- Important enhancement delivered with Db2 11 for z/OS: a single, multi-threaded, 64-bit JVM serves all Java stored procedures running in a stored procedure address space
  - Improved scalability, more efficient use of CPU and memory resources versus pre-Db2 11 situation (single-threaded, 31-bit JVM for each Java stored procedure running in address space)
- zIIP eligibility: Java code is zIIP-eligible in a z/OS system, so Java part of Java stored procedure is zIIP-eligible; however, SQL is not Java, and Java stored procedures run under TCBs
  - What that means: SQL statements issued by Java stored procedures are not zIIP-eligible

Recommendations for external stored procedures (1)

- Stored procedure address spaces should have same priority (per z/OS WLM policy) as Db2 MSTR, DBM1 and DIST address spaces…
  – …and those address spaces should have priority below SYSSTC but higher than CICS application regions or IMS message regions (IRLM should be assigned to SYSSTC service class)
- Why: if stored procedure address space has too-low priority, and system gets really busy, there could be significant delays in scheduling called stored procedures for execution – could even lead to calling program receiving SQL error code
- Question: will external stored procedures run at the priority of the address space in which they execute?
  – Answer: no – priority of stored procedure address space applies only to address space's main task (a stored procedure – external or native SQL – always inherits the priority of its caller)

Recommendations for external stored procedures (2)

- PROGRAM TYPE: SUB versus MAIN
  - PROGRAM TYPE is an option for CREATE (or ALTER) PROCEDURE for an external stored procedure
- PROGRAM TYPE SUB has been observed to reduce CPU consumption associated with a stored procedure by 10% in some cases
  - <u>However</u>, TYPE SUB means that the program is responsible for initialization of work areas
  - Some users tried TYPE SUB, then went back to TYPE MAIN because former led to “unpredictable results,” due to stored procedure programs not effectively initializing work areas
- TYPE SUB is good for performance, but ensure that your stored procedure programs are well suited to run as subroutines

Recommendations for external stored procedures (3)

- STAY RESIDENT: YES versus NO
- STAY RESIDENT is another option for CREATE (or ALTER) PROCEDURE for an external stored procedure
- YES can improve stored procedure CPU efficiency, but it should NOT be used for stored procedure programs compiled and linked as non-reentrant and non-reusable
  - Go with STAY RESIDENT NO for stored procedure programs that are non-reentrant and non-reusable
  - If STAY RESIDENT NO is specified for a frequently-executed stored procedure, module load time can be reduced by loading from the z/OS Virtual Lookaside Facility (VLF)

Can stored procedures be over-used in a Db2 for z/OS environment?

Over-use is possible – depends on priorities

- If maximum performance is a priority, stored procedures could be over-used
- Comparing performance of SQL DML statements issued by client programs versus packaging same statements in server-side stored procedures:
  - Very similar if transaction involves execution of 3-4 SQL DML statements
  - If > 4 SQL DML statements/tran, stored procedures will probably provide better performance
  - If only 1 or 2 SQL DML statements/tran, performance probably best with client-issued SQL
- Using stored procedures even for transactions that involve execution of only 1 or 2 SQL DML statements could be considered over-use from performance perspective
  - That said, performance difference may not be very substantial
  - If stored procedures will be used even for simplest transactions, use of native SQL procedures becomes more important (avoid thread task switch overhead of external stored procedures)
  - Note that performance benefit of high-performance DBATs <u>increases</u> as in-Db2 CPU time per transaction <u>decreases</u>

On the other hand…

- If stored procedures are used to ensure that all ”table-touching” SQL issued for an application is static (perhaps for data-security reasons), it may be that stored procedures will never be thought of as being over-used
  - May not be a viable alternative for ensuring that data access is via static SQL
  - Yes, client programs could issue static SQL DML statements (for example: SQLJ in Java programs), but client-side developers often prefer to invoke stored procedures via interface such as JDBC versus using something like SQLJ
- Again, the more that stored procedures are used…
  - …the more important it is to make them native SQL procedures when possible
  - …the more important it is to get high-performance DBAT functionality by binding high-use stored procedure packages with RELEASE(DEALLOCATE)

The bottom line on extent of stored procedure usage

- Determine first what your organization’s priorities are for client-server Db2 for z/OS applications
- <u>Then</u> make a decision on extent of stored procedure usage that will conform to your organization’s priorities

Thanks for your time.

Robert Catterall
rfcatter@us.ibm.com