

Going Native: Leveraging DB2 for z/OS SQL Procedures and UDFs

Robert Catterall

IBM

Session Code: E07

May 14, 2014, 10:30 - 11:30 AM | Platform: DB2 for z/OS



For starters, a prediction

SQL Procedure Language (SQL PL) will come to be the dominant language for development of DB2 for z/OS stored procedures and user-defined functions (UDFs).

You should know about SQL PL, and your organization should use it.

Agenda

- Where we've been, and where we're going
- The case for native SQL procedures and UDFs – performance
- The case for native SQL procedures and UDFs – scalability and security
- Native SQL procedure and UDF development and management
- Managing the shift to SQL PL

Where we've been, and where we're going

First: SQL Procedure Language

- Introduced with DB2 for z/OS Version 7
- Enabled coding of DB2 stored procedures using only SQL
 - Made possible through the 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
```

From the
DB2 V8 SQL
Reference



DB2 V7 SQL procedures – the good, and the not so good

- Good:
 - Expanded the pool of people who could develop DB2 for z/OS stored procedures
- Not so good:
 - As part of program preparation, SQL PL routine was converted into a C language program with embedded SQL DML statements
 - C language programs are generally not as CPU-efficient as programs written in COBOL
 - The CPU cost of the C language executable discouraged some organizations from using SQL procedures

Big breakthrough: native SQL procedures

- Introduced with DB2 9 for z/OS in new-function mode (or DB2 10 NFM for organizations that migrated to DB2 10 from DB2 V8)
- Key differences (vs. original SQL procedures, which are now known as external SQL procedures):
 - No external-to-DB2 executable (no object module, no load module) – a native SQL procedure’s package is that procedure’s one and only executable
 - Executes in DBM1, the DB2 database services address space (as do all packages) – not in a WLM-managed stored procedure address space
 - Runs under caller’s task (versus a TCB in a stored procedure address space)
 - Provides superior functionality (example: nested compound SQL statements – great for multi-statement condition handlers)
 - And, native SQL procedures – not external SQL procedures – are where you will see advances in SQL PL functionality (more on this to come)

Another breakthrough: “native” SQL user-defined functions

- Officially called “compiled SQL scalar functions”
- Introduced with DB2 10 for z/OS in new-function mode

```
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



“Native” (i.e., compiled) SQL UDFs – how they’re new

- They allow a SQL UDF, for the first time, to have a compound statement in the RETURNS clause of CREATE FUNCTION
 - Compound statement: a group of statements, generally set off by BEGIN and END, that comprise a SQL routine (native SQL procedures typically contain compound statements)
 - What this means: you can write a UDF in SQL PL, and the associated package is the UDF’s one and only executable (same as for a native SQL procedure)
 - Also like a native SQL procedure, a “native” UDF executes in the DB2 DBM1 address space, and executes under the invoker’s task
 - Previously, having a UDF containing variable declarations and logic flow control statements meant writing an external UDF in a language such as COBOL

More new DB2 10 stuff pertaining to SQL UDFs

- Starting with DB2 10 NFM, there are three types of CREATE FUNCTION statements for SQL UDFs (versus one with DB2 9)
 - *Compiled SQL scalar functions*, aka non-inline SQL scalar functions (new functionality – what I’ve called “native” SQL UDFs) ← Has its own package
 - *Inlined SQL scalar functions*
 - *SQL table functions* (new functionality) } Don't have their own packages - incorporated into package of invoker
- The RETURN statement in a compiled SQL scalar UDF can contain a scalar fullselect (previously, a SQL scalar UDF couldn't even reference a column)
- Via a SQL table UDF, you can return a set of rows to an invoker
 - Table UDFs formerly had to be external
- **What this all means:** starting with DB2 10 NFM, you can code high-function UDFs using only SQL

DB2 10 new functionality for native SQL procedures

Not applicable to external stored procedures - SQL or otherwise

- XML data type valid for SQL procedure input and output parameters, and for variables declared in a SQL procedure (DB2 10)
 - Also, XML data type can be passed to or received from, and/or be used in a variable declaration in, a SQL UDF (scalar or table)
 - Benefit: you no longer have to serialize an XML value into a character string or a CLOB in order to work with it in a SQL procedure or UDF


DB2 11 new functionality for native SQL procedures (1)

- Array parameters can be passed to and/or received from, and array variables can be declared in, native SQL procedures (and the same is true for compiled SQL scalar UDFs)
 - Call to SQL procedure with an array input or output parameter must come from another SQL PL routine or from a **Java program** (via the IBM Data Server Driver for JDBC and SQLJ type 4 driver) *← This is a big deal*
 - An array in this context is a form of a DB2 for z/OS user-defined data type (i.e., a UDT) – you create it, then you use it
 - Two array types: ordinary (elements are addressed by their ordinal position in the array) and associative (elements ordered and referenced by array index values)
 - For simplicity's sake, I'd recommend using ordinary arrays when you can

```
CREATE TYPE PHONENUMBERS AS DECIMAL(10,0) ARRAY[50];
```

Data type of values in the array 



 Max number of elements in array (defaults to about 2 billion)

DB2 11 new functionality for native SQL procedures (2)

- A SQL procedure can function as an autonomous transaction
 - How it's done: AUTONOMOUS option specified in CREATE PROCEDURE (or ALTER PROCEDURE) statement
 - Specified instead of COMMIT ON RETURN YES/NO
 - What it means:
 - The autonomous SQL procedure commits on returning to the calling program, but (unlike the case when COMMIT ON RETURN YES is in effect) that commit does NOT affect the calling program's unit of work
 - The autonomous SQL procedure's unit of work is independent of the calling program's unit of work – if the calling program's unit of work is rolled back, data changes made by the autonomous SQL procedure will not be rolled back
 - An autonomous SQL procedure does not share locks with its caller, and could conceivably get into a lock contention situation with its caller
 - An autonomous SQL procedure can be cancelled by cancelling its caller
 - One autonomous SQL procedure can't call another autonomous SQL procedure

The case for native SQL procedures and UDFs – performance

Native SQL procedures (and UDFs) and zIIP offload

- Some people think that native SQL procedures are zIIP eligible, period – that is NOT the case
- A native SQL procedure is zIIP-eligible ONLY when it is called by a DRDA requester (i.e., when the call comes through DDF)
 - Why this is so: a native SQL procedure runs under the task of the calling process, and if the caller is a DRDA requester its task is an enclave SRB in the DDF address space, and that makes the SQL procedure zIIP-eligible
- An external DB2 stored procedure (SQL or otherwise) always runs under a TCB in a WLM-managed stored procedure address space, and for that reason it's never zIIP-eligible, regardless of the caller
- So, *when DRDA requesters call external DB2 stored procedures*, switching to native SQL procedures will boost zIIP utilization

Eliminating task switch delays


- When an external DB2 stored procedure is called (or an external UDF invoked), the caller's thread has to be switched from the caller's task to the stored procedure's task
- Maybe not a big deal for a stored procedure, but an external UDF could be driven many times in the execution of a single query, if (for example) the UDF appeared in the SELECT of a correlated subquery
 - In that case, all the thread switching from the application's task to the external UDFs task (and back) could really add up
 - At one site (DB2 10 NFM), a given query drove over 100,000 invocations of an external UDF in a single execution, and a query monitor showed a large amount of "UDF TCB wait" time
 - The external UDF was changed to a SQL UDF, and the query's elapsed time decreased substantially

DB2 10: better SQL PL performance

(a package)

- The compiled form of a native SQL procedure executes with improved CPU efficiency in a DB2 10 environment versus DB2 9 (when the SQL procedure is regenerated in the DB2 10 system)
- A couple of reasons why this is so:
 - The path length for execution of IF statements (very common in SQL PL routines) is reduced
 - SET statements that reference built-in functions (e.g., the CHAR function) are more CPU-efficient
- You can get additional CPU savings by leveraging the new (with DB2 10 NFM) support for “chained” SET statements:

```
SET x=1 ;  
SET y=2 ;
```



```
SET x=1 , y=2 ;
```

A word about REGENERATE versus REBIND PACKAGE

- Referring to REBIND PACKAGE vs. ALTER PROCEDURE (or FUNCTION) with the REGENERATE option for a SQL PL routine
- Key difference:
 - REBIND PACKAGE affects only the non-control statements in a routine (e.g., SQL DML statements like SELECT), while REGENERATE reworks a package in its entirety
 - So, if you want improved CPU efficiency for IF statements, and for SET statements that reference built-in functions, you need to REGENERATE SQL PL routines in a DB2 10 (or later) environment – REBIND PACKAGE won't do it
 - Additionally, if you want the bulk of the control statement section of a SQL PL routine's package to go above the 2 GB bar in DBM1 when the package is allocated to a thread, you'll need to REGENERATE the package

Another thing about REGENERATE vs. REBIND PACKAGE

- As is true for REBIND PACKAGE, REGENERATE can lead to access path changes for SQL DML statements in a package
 - However, APREUSE (reuse access paths when possible) is not an option for REGENERATE
 - Additionally, plan management (which enables reactivation of a previous copy of a package via REBIND SWITCH) does not apply to REGENERATE
 - So, use REBIND PACKAGE instead of REGENERATE if there's not a need to rework the control statement section of a 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 didn't change (or changed for the better), you should be safe with a REGENERATE (access paths likely to stay the same if the REGENERATE closely follows the REBIND PACKAGE)

Some other performance boosters not limited to SQL PL...

- These are things that can improve the CPU efficiency of DB2 stored procedures in general – both native SQL and external
 - `RELEASE(DEALLOCATE)` for packages associated with frequently executed SQL PL routines (delivers CPU savings when paired with persistent threads)
 - DB2 10 provided way more virtual storage “head room” for use of `RELEASE(DEALLOCATE)` by shifting thread-related virtual storage above the 2 GB bar in DBM1 (for packages bound or rebound in DB2 10 system)
 - Also new with DB2 10: `RELEASE(DEALLOCATE)` is respected for packages executed via DBATs (i.e., DDF threads) – these threads become high-performance DBATs
 - DB2 11 provides relief for `BIND/REBIND`, `DDL`, and utility concurrency issues related to `RELEASE(DEALLOCATE)` + persistent threads (DB2 10 provides `-MODIFY DDF` command to temporarily “turn off” high-performance DBAT functionality)
 - New (with DB2 10 NFM) `RETURN TO CLIENT` option for `DECLARE CURSOR`
 - Especially for larger result sets, much more CPU-efficient way to make result set rows available to a “top-level” calling program from a nested stored procedure, as compared to temporary table approach

...and other performance benefits of stored procs in general

- From a performance perspective, the “sweet spot” for stored procedure usage is for a DB2 client-server (i.e., DDF) workload
- Stored procedures provide a means of packaging static SQL in a form that can be dynamically invoked

Client-side developers may want to call stored procedures via JDBC or ODBC

For optimal CPU efficiency

- Stored procedures also help to loosen coupling between client-side programs and database design
 - Suppose you make a performance-improving logical database design change (i.e., one that would normally be visible to data-accessing applications)
 - If affected data is accessed via stored procedures, likely that client-side code changes will not be required – just change stored procedures as needed

The case for native SQL procedures and UDFs – scalability and security

About this section of the presentation...

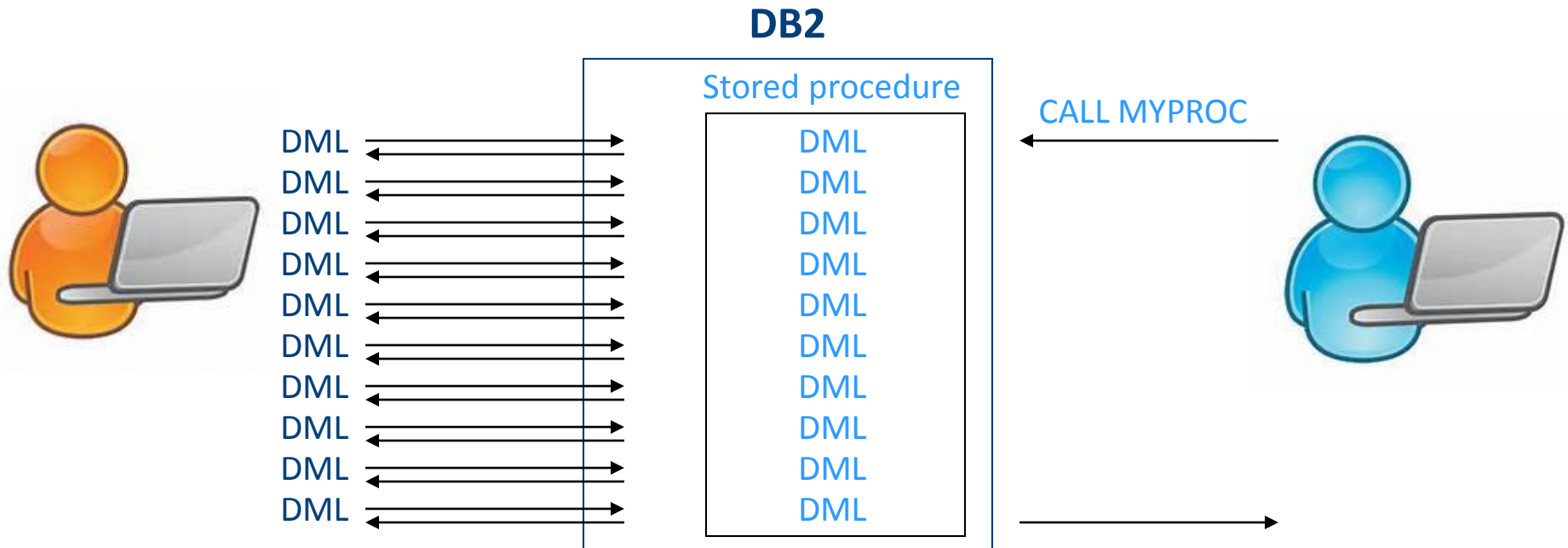
- As I see it, the scalability and security advantages of DB2 stored procedures pertain to stored procedures in general – external as well as native SQL
- That being the case, the particular advantages of native SQL procedures (and UDFs) versus external stored procedures in this area can be described as follows:

Native SQL procedures provide the security and scalability advantages that pertain to DB2 stored procedures in general, with these added advantages:

- They can be developed by SQL-knowledgeable people who aren't COBOL or Java programmers
- They provide server-side CPU cost savings for DB2 client-server (DDF) applications, thanks to significant zIIP offload

Scalability: reduced network “chattiness” for DDF trans

- For a DDF transaction that involves issuance of 10 (for example) SQL DML statements, packaging those statements in a stored procedure replaces 10 network request/response pairs with 1



Scalability: the DB2 MQListener

- If some of your transactional work can drive asynchronous versus synchronous DB2 processing, peak workloads can be spread out, time-wise, helping to smooth out server utilization profiles
 - Data provided by a transaction could be placed on a WebSphere MQ queue, at which point the transaction, from the client perspective, is done
 - The DB2 MQListener ships with DB2 for z/OS, and enables you to associate an MQ queue with a DB2 stored procedure
 - When a message arrives on the queue, the associated stored procedure will be automatically invoked and the message will be passed as input to the procedure
 - You could set this up for several queues that would be for different message types, and each queue could be associated with a particular stored procedure
 - With this approach, a transaction surge could show up as an increase in a queue's message depth, versus an increase in server utilization
 - Added bonus: a less-brittle application infrastructure (if the database is unavailable, messages build up on queues, versus transactions failing)

Security: static SQL

- As noted previously, stored procedures provide a means of packaging static SQL in a form that can be dynamically invoked
- In some cases, the contribution of static SQL toward more-robust security is more attractive even than the performance advantage
 - When SQL DML statements are static and issued by way of a stored procedure, the invoking application's authorization ID does not require table access privileges (SELECT, INSERT, UPDATE, DELETE)
 - Instead, the application's ID requires only the EXECUTE privilege on the stored procedure
 - Even tighter security: create a DB2 role, grant execute on the stored procedure to the role, and create a trusted context that restricts use of the role's privileges to a particular application ID connecting from a particular IP address

Security: database schema abstraction

- If someone wants to hack into your database, he'll 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 coders of calling programs do not
 - Data security is boosted by limiting the number of people with detailed knowledge of the database schema



Native SQL procedure and UDF development and management

IBM Data Studio

- A great tool for developing and debugging SQL PL routines
 - Free, and downloadable – current release is Version 4.1 (<http://www-03.ibm.com/software/products/en/data-studio/>)
 - Eclipse-based, GUI client that runs under Windows or Linux
 - Among other things, the Data Studio Debug view enables you to:
 - Set breakpoints (on both a line and variable basis)
 - Step into or over a line of code
 - Inspect variables, and change variable values
 - Also, beats the tar out of SPUFI when it comes to testing individual SQL statements, especially in these categories:
 - XML data access (even formats it for you on retrieval)
 - LOB data access
 - Stored procedure calls

What about SQL PL source code management?

- A lot of the popular vendor-supplied tools used for source code management (SCM) don't yet offer support for SQL PL
- Probably, more SCM tool vendors will offer SQL PL support as use of the language grows, but what are folks doing in the meantime?
- One SCM option for SQL PL: “roll-your-own”
 - This approach can be aided by the use of some sample REXX routines provided via the fix for APAR PM29226 (DB2 9 and 10 for z/OS)
 - DB2 sample job DSNTEJ67 illustrates use of these routines, which include:
 - A service that extracts the source for a SQL PL routine from the DB2 catalog and places it in a file (or into a string)
 - A service that invokes the SQL PL precompiler to generate a listing of a routine
 - A service that can be used to change various elements of the SQL PL source for a routine, such as schema, version ID, and CREATE PROCEDURE options
 - A service that can be used to deploy SQL PL source code

Another SCM option for SQL PL

- Use the open-source Apache Subversion software versioning and revision control system, in conjunction with Data Studio (<http://subversion.apache.org/>)
- Data Studio is built on Eclipse, and Subversion (also known as SVN) integrates with Eclipse
 - SQL PL routines developed using Data Studio are Eclipse resources
 - These resources can be managed by an Eclipse “Team” component plug-in such as Subversion
 - Subversion can be installed into Data Studio

Some native SQL procedure deployment considerations

- With the procedure's package being the only executable, deployment is different versus external procedures
- To get a native SQL procedure from a test system into production, you can use `BIND PACKAGE` with the `DEPLOY` option
 - The non-control section of the procedure's package will be reoptimized on the production system, but the control section will not be changed
- If a previous version of the SQL procedure is already running in production, you can control when the new version becomes active with the `ACTIVATE VERSION` option of `ALTER PROCEDURE`
 - Selective use of the new version before then can be enabled via the `CURRENT ROUTINE VERSION` special register
- *Get comfortable with these deployment mechanisms before going big into native SQL procedures and non-inline SQL scalar UDFs*

ALTER versus DROP/re-CREATE for native SQL procedures

- Sometimes – especially when an organization is just getting into native SQL procedures – DBAs will go with DROP and re-CREATE versus ALTER PROCEDURE to make changes to a procedure
- That approach can be problematic when you get into nested procedure calls (i.e., one stored procedure calls another)
 - If PROC_A calls PROC_B, and PROC_A is a native SQL procedure, an attempt to drop PROC_B will fail
 - You can check to see if any native SQL procedures are dependent on a given stored procedure by querying the SYSPACKDEP catalog table (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?
 - Bottom line: if you can accomplish a needed change with ALTER PROCEDURE, do that versus DROP and re-CREATE

Managing the shift to SQL PL

First, about your existing external stored procedures...

- Some organizations have hundreds, even thousands, of DB2 for z/OS stored procedures written in COBOL
 - It would probably be counter productive to try to change these to native SQL procedures en masse
 - On top of that, some of those COBOL stored procedures might access non-DB2 data (VSAM data, for example)
 - Low-hanging fruit for replacement with native SQL procedures: COBOL procedures that are relatively simple, access DB2 data, are frequently executed, and are primarily called by DRDA requesters (want zIIP offload)
- As for external SQL procedures, converting those to native SQL procedures is sometimes pretty straightforward, other times less so
 - Lots of very useful information in this brief IBM “technote” (just a few pages):
<http://www-01.ibm.com/support/docview.wss?uid=swg21297948>

Question: who writes SQL PL routines?

- One might think DB2 DBAs, but you could have a bandwidth problem with that approach
 - DB2 for z/OS DBA teams are often pretty lean and mean, and they're usually pretty busy managing the database system
- Organizations typically have many more application developers than DB2 DBAs
 - But SQL PL, and all kinds of aspects of the CREATE and ALTER PROCEDURE (and FUNCTION) statements, may be unfamiliar to these folks
- There's a third way...



One organization's approach

- Created a new position, “procedural DBA”
 - DB2 database-centric, but with an emphasis on developing and managing stored procedures (and functions)
 - Advertised the openings internally, and got a mix of applicants
 - Some who'd been traditional DB2 DBAs and wanted more of an application focus
 - Some who'd been traditional application programmers and wanted a more data-oriented role
 - All seemed pretty pumped about a new career challenge
- Something to keep in mind: SQL PL is virtually identical in DB2 for z/OS and DB2 for LUW systems
 - So, people writing SQL procedures and UDFs could create routines for both platforms

Robert Catterall

IBM

rfcatter@us.ibm.com

Session E07

Going Native: Leveraging DB2 for z/OS SQL
Procedures and UDFs



*Please fill out your
session evaluation before
leaving!*

