Going Native: Leveraging DB2 for z/OS Native SQL Procedures and User-Defined Functions

Dallas-area DB2 users group

April 19, 2017
Agenda

- Where we’ve been, and where we’re going
- Native SQL procedures and UDFs – performance
- 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
SQL Procedure Language

- Initially delivered with DB2 for z/OS 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

```sql
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
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, 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 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 (vs. TCB in stored proc. address space)
  - Superior functionality
    - Native SQL procedures – not external stored procedures – are where you will see greatest advances in SQL PL functionality
DB2 10 “native” SQL user-defined functions

- These are basically UDFs written in SQL PL
  - 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” UDF executes in the DB2 DBM1 address space, and under the invoker’s task

- Also new with DB2 10: compiled SQL scalar function can have scalar fullselect in RETURN part of CREATE FUNCTION
  - Previously, couldn’t even reference a column in RETURN of SQL UDF

- Also new with DB2 10: SQL table functions
  - Table UDF returns a set or rows to an invoker
  - Could have table UDFs before, but they could not be written in SQL PL
What a pre-DB2 10 SQL UDF had to look like (and still can, in the form of an inlined SQL scalar function)

```
CREATE FUNCTION KM_MILES(X DECIMAL(7,2))
RETURNS DECIMAL(7,2)
LANGUAGE SQL
CONTAINS SQL
NO EXTERNAL ACTION
DETERMINISTIC
RETURN X*0.62;
```

Here’s your logic
What a “native” SQL UDF can look like

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#
And here’s a SQL UDF with a scalar fullselect

```sql
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);
```

 Couldn’t have this in RETURN statement before DB2 10
DB2 10: new native SQL procedure functionality

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
  - Also, XML data type can be passed to or received from, and/or be used in a variable declaration in, a SQL UDF
  - 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 native SQL procedure functionality

- **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 array input or output parameter can come from SQL PL routine, from Java program, or from .NET client (for latter two, via IBM Data Server Driver type 4 driver)
    - .NET client can call native SQL procedure with array input parameter
  - 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)

This is an ordinary array
```sql
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)

Associative array would have data type of index values here (INTEGER or VARCHAR)
DB2 11 new **native** SQL procedure functionality

- 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
    - An 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
    - Autonomous SQL procedure’s unit of work (UOW) is independent of calling program’s UOW – if calling program’s UOW rolled back, data changes made by autonomous SQL procedure will not be rolled back
    - An autonomous SQL procedure does not share locks with caller, and could conceivably get into a lock contention situation with its caller
    - One autonomous SQL procedure can’t call another
CREATE TRIGGER MYTRIG01
BEFORE INSERT ON MYTAB
REFERENCING NEW AS N
FOR EACH ROW
ALLOW DEBUG MODE
QUALIFIER ADMF001
WHEN(N.ending IS NULL OR n.ending > '21:00')
L1: BEGIN ATOMIC
    IF (N.ending IS NULL) THEN
        SET N.ending = N.starting + 1 HOUR;
    END IF;
    IF (N.ending > '21:00') THEN
        SIGNAL SQLSTATE '80000'
        SET MESSAGE_TEXT = 'Class ending time is beyond 9 pm';
    END IF;
    SET GLOBAL_VAR = NEW.C1;
END L1#

With advanced trigger, SET is not restricted to transition variables – it can also be used with global variables and SQL variables (latter refers to variables declared in body of trigger)

As with other SQL PL routines, debug with Data Studio

These are among new CREATE TRIGGER options

SQL PL logic flow control statements

Trigger body contains this logic:
If class end time is null, value is set to 1 hour after start of class; otherwise, if class ends after 9pm then an error is returned
Other DB2 12 SQL PL enhancements

- Compiled SQL scalar functions (i.e., “native” SQL UDFs) can issue dynamic SQL statements
  - Pre-DB2 12: do-able from native SQL procedure, not “native” SQL UDF

- Variable declared in SQL PL routine can be declared as a constant

```sql
... DECLARE VAR2 INTEGER;
DECLARE cMAXVAL INTEGER CONSTANT 2000;
SELECT 1 INTO VAR2 FROM TEST WHERE VAR1 > cMAXVAL;
IF VAR1 > cMAXVAL THEN
    ...
ELSE
    ...
END IF;
```

- Obfuscate SQL PL logic in SYSROUTINES, SYSTRIGGERS
  - Logic can be “scrambled” to protect intellectual property
  - Done with new built-in function, called WRAP
Native SQL procedures and UDFs – performance
Native SQL procedures (and UDFs): zIIP offload

- Some people think that native SQL procedures are always zIIP-eligible – NOT the case

- A native SQL procedure is zIIP-eligible ONLY when called by a DRDA requester (i.e., when the call comes through DDF)
  - A native SQL procedure runs under the task of its caller – if caller is a DRDA requester, task is an enclave SRB in the DDF address space, and that makes the SQL procedure zIIP-eligible (up to 60%)

- An external DB2 stored procedure (SQL or otherwise) runs under a TCB in a stored procedure address space, and for that reason it’s not 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.

- Not so important for a stored procedure, but a UDF could be driven many times in the execution of 1 query, if (for example) the UDF appears in the SELECT of a correlated subquery.
  - In that case, all the thread switching from the application’s task to an external UDF’s task (and back) could really add up.
  - At one site (DB2 10 NFM), a given query drove over 1,000 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 delivered better SQL PL performance

(a package)

- The compiled form of a native SQL procedure executes with improved CPU efficiency in a DB2 10 environment vs. DB2 9 (when 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:

  ```sql
  SET x=1;
  SET y=2;
  SET x=1, y=2;
  ```
REGENERATE versus REBIND PACKAGE

- Referring to REBIND PACKAGE vs. ALTER PROCEDURE (or FUNCTION) with REGENERATE 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
  - To get improved CPU efficiency for IF statements, and for SET statements that reference built-in functions, must REGENERATE SQL PL routines in DB2 10 (or later) environment – REBIND PACKAGE won’t do it
  - Plus, if you want the bulk of the control statement section of a SQL PL routine’s package to go above 2 GB bar in DBM1 when the package is allocated to a thread, you’ll need to REGENERATE the routine
More on REGENERATE vs. REBIND PACKAGE

- Like REBIND PACKAGE, REGENERATE can lead to access path changes for SQL DML statements in a SQL PL routine
  - APREUSE (reuse access paths) not an option for REGENERATE
  - Also, plan management (reactivate previous instance of package via REBIND SWITCH) not applicable to REGENERATE
  - So, use REBIND PACKAGE instead of REGENERATE if there’s not a need to rework 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 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)
Other performance boosters

These are things that can improve the CPU efficiency of DB2 stored procedures in general – both native SQL and external

- RELEASE(DEALLOCATE) for packages of frequently executed stored procedures (saves CPU when used with persistent threads)
  - DB2 10 provided more “head room” for RELEASE(DEALLOCATE) by shifting thread-related virtual storage above the 2 GB bar in DBM1 (for packages bound or rebound in DB2 10 system)
  - More DB2 10: RELEASE(DEALLOCATE) respected for packages executed via DDF threads – these become high-performance DBATs
  - DB2 11: relief for BIND/REBIND, DDL, utility concurrency issues related to RELEASE(DEALLOCATE) + persistent threads (still might need to temporarily “turn off” high-performance DBATs via -MODIFY DDF)
- DB2 10 NFM: DECLARE CURSOR WITH RETURN TO CLIENT
  - Especially for larger result sets, a more CPU-efficient way to make result set rows available to a “top-level” calling program from a nested stored procedure, versus temporary table approach
Performance benefits of stored procedures

- 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.

  - For optimal CPU efficiency, client-side developers may want to call stored procedures via JDBC or ODBC.

- Stored procedures also help to loosen coupling between client-side programs and database design.
  - Consider a logical database design change (visible to data-accessing programs) that would improve application performance.
  - If affected data is accessed via stored procedures, client-side code likely won’t need changes – just change stored procedures as needed.
Native SQL procedures and UDFs – scalability and security
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 a DDF transaction that involves issuance of, say, 10 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 vs. synchronous DB2 processing, peak workloads can be spread out, time-wise, smoothing out mainframe utilization profiles
  - Input data could be placed on a WebSphere MQ queue, at which point a 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 message arrives on queue, associated stored procedure is automatically invoked – message is passed as input
    - You could set up several queues for different message types, and each queue could be associated with a particular stored procedure
  - With this approach, transaction surge would increase a queue’s message depth, vs. increasing server utilization
  - Added bonus: a less-brittle application infrastructure (if the database is unavailable, messages build up on queues)
Security: static SQL

- Recall that stored procedures provide a means of packaging static SQL in a form that can be dynamically invoked.

- Sometimes, the contribution of static SQL toward more-robust security is more attractive than the performance advantage.
  
  - 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 the EXECUTE privilege on the stored procedure.
    
    • Even more secure: 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 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 4.1.2
  - 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, much better than 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?

- There are solutions available from IBM and other vendors for management of SQL PL source code

- Another option: “roll-your-own”
  - This approach can be aided by some DB2-provided sample REXX routines (provided for DB2 10 via fix for APAR PM29226 – part of DB2 11 base)
  - DB2 sample job DSNTEJ67 illustrates use of these routines
    - One service extracts the source for a SQL PL routine from the DB2 catalog and places it in a file (or into a string)
    - Another invokes SQL PL precompiler, generates a listing of a routine
    - Another can be used to change elements of SQL PL source for a routine (e.g., schema, version ID, CREATE PROCEDURE options)
    - Another can be used to deploy SQL PL source code
Another free 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
Native SQL procedure deployment

- With the procedure’s package being the only executable, deployment is different versus external procedures.

- To get native SQL procedure from test system into production, you can use BIND PACKAGE with the DEPLOY option:
  - Non-control section of procedure’s package will be reoptimized on production system, but the control section will not be changed.

- If previous version of the SQL procedure is already running in production, control when the new version becomes active via 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** making major use of native SQL procedures and UDFs.
ALTER vs. drop/re-create for native SQL procs

- Sometimes – especially early in an organization’s use of SQL procedures – DBAs will use 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 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?
  - If you can accomplish change with ALTER PROCEDURE, go that route
    - 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
Managing the shift to SQL PL
About 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: simple, frequently executed COBOL procedures that access DB2 data and are mostly called by DRDA requesters (want zIIP offload)

- As for external SQL procedures, converting those to native SQL procedures may be very straightforward – or not
  - Lots of useful information in this 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

▪ Maybe application developers – organizations usually have more of these folks 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 people

▪ There’s another way…
What about creating a new role?

- Might be called something like “procedural DBA”
  - DB2 database-centric, but with an emphasis on developing and managing stored procedures (and functions)
  - With such a position available, you could get a mix of applicants
    - Some who have been traditional DB2 DBAs and want more of an application focus
    - Some who have been traditional application programmers and want a more data-oriented role
    - In either case, could be an appealing 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
rfcatter@us.ibm.com