DB2 for z/OS Stored Procedures Update

Michigan DB2 Users Group

May 15, 2013
Agenda

- A brief review of DB2 for z/OS stored procedure enhancements over time
- Native SQL procedures
- Converting external SQL procedures to native SQL procedures
- Hints, tips, etc.
A brief review of DB2 for z/OS stored procedure enhancements over time
## Stored procedure enhancements (1)

<table>
<thead>
<tr>
<th>DB2 for z/OS V4</th>
<th>Stored procedure functionality introduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2 for z/OS V5</td>
<td>Caller of stored procedure can fetch rows from cursor declared and opened in stored procedure</td>
</tr>
<tr>
<td></td>
<td>WLM-managed stored procedure address spaces introduced – recommended over DB2-managed stored procedure address space</td>
</tr>
<tr>
<td></td>
<td>Support for Java stored procedures</td>
</tr>
<tr>
<td>DB2 for z/OS V6</td>
<td>DDL support: CREATE, ALTER, DROP PROCEDURE</td>
</tr>
<tr>
<td></td>
<td>SYSPROCEDURES catalog table replaced by SYSROUTINES</td>
</tr>
</tbody>
</table>
## Stored procedure enhancements (2)

| DB2 for z/OS V7 | ▪ SQL procedure language introduced (stored procedures written in SQL PL executed as C language programs in WLM-managed address spaces)  
▪ COMMIT and ROLLBACK can be issued from a stored procedure |
| DB2 for z/OS V8 | ▪ Abend limit can be set at individual stored procedure level (versus DB2 subsystem level)  
▪ DB2, z/OS work together to optimize the number of tasks in a stored procedure address space |
## Stored procedure enhancements (3)

<table>
<thead>
<tr>
<th>DB2 9 for z/OS</th>
<th>Native SQL procedures introduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2 10 for z/OS</td>
<td>Improved SQL PL performance</td>
</tr>
<tr>
<td></td>
<td>RETURN TO CLIENT CURSORS</td>
</tr>
<tr>
<td></td>
<td>User-defined functions can be written in SQL PL</td>
</tr>
<tr>
<td></td>
<td>(I call these “native” SQL UDFs)</td>
</tr>
<tr>
<td></td>
<td>XML data type supported for parameters of native SQL procedures (and “native” SQL UDFs)</td>
</tr>
</tbody>
</table>
DB2 10 RETURN TO CLIENT cursors

- Before DB2 10: stored procedure can return result set 1 level up in chain of nested procedures (WITH RETURN TO CALLER)
  - For example, if program PROG_A calls stored procedure PROC_B, and PROC_B calls PROC_C, PROC_B can fetch from a cursor declared and opened in PROC_C, but PROG_A cannot
    - If PROG_A needs that result set, PROC_C can put it in a temporary table, and PROG_A can get the rows from that temp table

- A DB2 10 stored procedure can declare a cursor WITH RETURN TO CLIENT (as is the case with DB2 for LUW)
  - “Top-level” program (caller of first stored procedure in a chain of nested calls) can fetch rows in result set associated with that cursor
  - The cursor’s result set will be invisible to stored procedures between cursor-declaring procedure and top-level program
Previous slide’s point, in a picture…

- **DB2 9:**
  - Program XYZ
  - Stored proc A
  - Stored proc B
    - DECLARE C1 CURSOR...
    - WITH RETURN TO CALLER

- **DB2 10:**
  - Program XYZ
  - Stored proc A
  - Stored proc B
    - DECLARE C1 CURSOR...
    - WITH RETURN TO CALLER
    - WITH RETURN TO CLIENT
SQL user-defined functions – before DB2 10

- DB2 9 and earlier: “logic” in a SQL scalar UDF is what you can code in the RETURN statement, and that’s pretty limited
  - RETURN can’t contain a SELECT statement
  - RETURN can’t include a column name

- Basically limited to receiving a value (or values) as input, transforming that value (or values) arithmetically and/or with scalar functions, and returning result of that transformation
  - Example:

```sql
CREATE FUNCTION KM_MILES(X DECIMAL(7,2))
RETURNS DECIMAL(7,2)
LANGUAGE SQL
...
RETURN X*0.62;
```
DB2 10: “native” SQL UDFs

- DB2 10: SQL scalar UDFs can do MUCH more than before
  - RETURN can contain a scalar fullselect
    
    ```
    RETURN (SELECT WORKDEPT FROM EMP WHERE EMPNO = P1);
    ```

  - RETURN can be at the end of a compound SQL statement, in which variables can be declared and SET, and SQL control statements used to generate the value to be returned

    ```
    BEGIN
      DECLARE VAR1, VAR2 CHAR(10);
      SET VAR1 = ...;
      IF P1 = ...;
      RETURN VAR2;
    END@
    ```

- Support for SQL table UDFs, too
  - Can code fullselect in RETURN statement
Native SQL procedures
The big difference

- **External SQL procedure**: under the covers, turned into a C language program with embedded SQL
  - As is true of other external stored procedures, runs in a WLM-managed stored procedure address space
  - A concern for some folks: not-in-DB2 part of C program generally consumes more CPU than does equivalent COBOL code

- **Native SQL procedure** (introduced with DB2 9 in new-function mode): the procedure’s package is the only executable – there’s no external-to-DB2 load module
  - So, a native SQL procedure runs entirely in the DB2 database services address space (DBM1)
Native SQL procedure efficiency (1)

- An external stored procedure runs under its own TCB
  - Caller’s task (TCB or SRB) is suspended, and stored proc task uses caller’s thread for communication with DB2
    * In some cases, there can be processing delays and a build-up of DBM1 virtual storage consumption associated with the switching of threads from calling-program tasks to stored procedure tasks

- Native SQL procedure runs under the calling program’s task
  - No queuing, no delays related to thread-switching

"Use my thread."
Native SQL procedure efficiency (2)

- For every SQL statement in an external stored procedure (or any other external-to-DB2, SQL-issuing program), an “addressability round trip” is required
  - Program’s task switches addressability from “home” address space (in this case, a WLM-managed stored procedure address space) to DB2 DBM1 for SQL execution, then switches back
  - Each round trip consumes a few thousand instructions, and that’s just the back-and-forth – not the execution of the SQL in DBM1

- Native SQL procedures eliminate this extra path length
  - With the CALL to the native SQL procedure, you’re already in DBM1, and you stay there until stored procedure completes
External procedure vs. native SQL procedure

**External**

Client program

```
CALL MYPROC (parm1, parm2)
```

**z/OS server**

```
DDF  DBM1  WLM SPAS
   ←   ←   ←
   SQL1 SQL2 end
```

Load library

```
Stored proc program
```

```
DB2 directory
```

**Native**

Client program

```
CALL MYPROC (parm1, parm2)
```

**z/OS server**

```
DDF  DBM1  end
   ←   ←   ←
   SQL1 SQL2
```

```
DB2 directory
```

```
Stored proc package
```

```
``
The zIIP factor

- A native SQL procedure is zIIP-eligible if it is invoked via a remote call through the DB2 Distributed Data Facility (DDF)
  - This is directly related to the fact that (as mentioned) as native SQL procedure runs under the task of the process that issued the CALL
  - If caller is a DRDA requester, the task under which the native SQL procedure executes is a preemptable SRB, and that makes the native SQL procedure zIIP-eligible (as is all SQL executed via DDF)
  - Amount of CPU processing directed to a zIIP engine tends to be 55-60% for native SQL procedures called through DDF (same as for all SQL executed via DDF)

- Native SQL procedure probably consumes a little more CPU than equivalent COBOL stored procedure, but with zIIP offload, *cost* of CPU resource consumed could be less for native SQL procedure than for COBOL *for DRDA callers*
A native SQL procedure functionality benefit

- Nested compound statements (compound SQL statement within a compound SQL statement) allowed in native SQL procedures, not allowed in external SQL procedures
  - Compound statement: a group of SQL statements, bounded by BEGIN and END
    - Within a compound statement, variables, cursors, and condition handlers can be declared
    - A SQL procedure will very often contain a compound SQL statement
  - With nested compound statements, condition handlers can have their own compound statements (enables more sophisticated error handling)
  - Also good in terms of DB2 Family compatibility (DB2 for LUW already supported nested compound statements in SQL procedures)

*Important for cross-platform development*
A native SQL procedure lifecycle benefit

- Simpler creation, management, maintenance versus external stored procedures
  - No external-to-DB2 resources involved (e.g., no source, object, or load libraries)
    - The native SQL procedure package is the executable, and it is stored in the DB2 directory
  - No external-to-DB2 processes involved (e.g., no need for compile and link processes)

- No worries about mismatch between external program and DB2 package when you execute a native SQL procedure
  - The SQL procedure’s package is the sole executable
Native SQL procedures and scalability

Question: “Native SQL procedures run in one address space: DB2’s DBM1. Will that have a constraining effect on throughput?”

Answer: NO

- A native SQL procedure’s executable is its package, and packages always run in DBM1
  - If you’re running 1000 CICS-DB2 transactions per second from multiple CICS regions, each one has a package, and that code (compiled SQL) runs in DBM1
- No worries about DBM1 “running out of tasks” – a native SQL procedure runs under the caller’s task, which is external to DBM1
Native SQL procedures and system stability

- Question: “Given that all native SQL procedures run in DBM1, should I be concerned that a ‘bad’ native SQL procedure will take DB2 down?”

- Answer: NO
  - Everything that executes in DBM1 is DB2-generated, DB2-managed code
    - Multiple address spaces for external stored procedures help to protect the system from a problem that might arise in user-written code – that’s not a problem with native SQL procedures
Converting external SQL procedures to native SQL procedures
If you’re using external SQL procedures now…

- Get familiar and comfortable with the different lifecycle processes of native SQL procedures
  - New DEPLOY option of BIND PACKAGE
  - New ACTIVATE VERSION option of ALTER PROCEDURE
    - Use of correct version of native SQL procedure’s package not driven by use of correct stored procedure load module – there is no load module

- Maybe convert external SQL procedures to native SQL procedures when upgrading existing application
  - Sometimes simple: drop and recreate procedure without FENCED and EXTERNAL options, and without a WLM ENVIRONMENT specification
    - May need WLM ENVIRONMENT FOR DEBUG MODE
  - Sometimes, not so simple (more on this to come)

- Consider using native SQL procedures for new DB2 application development
External-to-native conversion (1)

- SQL PL source code changes may be required

- One reason for that: some error-handling logic that worked for an external SQL procedure won’t produce the desired behavior in a native SQL procedure
  - As previously noted, native SQL procedures allow nested compound statements, providing a means of coding multi-statement error handlers
    - Lacking that option, people coding external SQL procedures would sometimes use an IF block to implement the multi-statement handler
    - Problem: an “always true” condition that is used to enter an IF-based handler (IF 1=1 THEN…) will clear the diagnostics area (oops) when used in a native SQL procedure
    - In “going native”, change these condition handlers to nested compound SQL statements set off by BEGIN and END
Another potential reason for SQL PL source code changes when converting from external to native: differences in unqualified column/variable/parameter name resolution

Suppose you have in a SQL procedure an unqualified variable, parameter, or column name

- If it’s an external SQL procedure: DB2 will check first to see if a variable of that name has been declared, then if it’s the name of one of the procedure’s parameters – if neither is true, assumption is that it’s a column name
- If it’s a native SQL procedure: DB2 will check first to see if name is that of a column of a table referenced by the procedure, then if a variable of that name has been declared, then if it’s the name of a parameter

To avoid unexpected behavior, do either one of the following:

- Use qualifiers (table name for a column, procedure name for a parameter, compound statement label for a variable)
- Use a naming convention that identifies parameters and variables (e.g. prefix parameters with p_ and variables with v_)
External-to-native conversion (3a)

- What about the external SQL procedure’s collection?
  - The package of an external SQL procedure can be bound into any collection, and that collection name can be specified via the COLLID option of CREATE PROCEDURE
    - By default, calling program will search in COLLID collection for external procedure’s package
  - When a native SQL procedure is created, the default collection name for the package will be the same as the procedure’s schema name
External-to-native conversion (3b)

- If the package of a to-be-converted external SQL procedure was bound into a collection with a name other than the procedure’s schema name:
  - Ensure that the collection with the same name as the procedure’s schema will be searched when the native SQL procedure is called,
  - or-
  - Put a SET CURRENT PACKAGESET in the body of the native SQL procedure, referencing external procedure’s collection name, and bind a copy of the native SQL procedure’s “root” package into that collection
    - So, if the external SQL procedure has a schema of PROD, and if it was bound into a collection called COLL_A, on conversion put a SET CURRENT PACKAGESET = ‘COLL_A’ in the body of the SQL procedure, and bind a copy of the native SQL procedure’s package (from collection PROD) into collection COLL_A
External-to-native conversion (4)

- For more conversion information, check out the brief (just a few pages) but highly informative IBM “Technote” at this URL:

  http://www-01.ibm.com/support/docview.wss?uid=swg21297948
External-to-native: something else to consider

- Suppose stored procedure PROC_A calls stored procedure PROC_B, and you want to drop PROC_B
  - If PROC_A is an external procedure, you can drop PROC_B, and PROC_A’s package will be invalidated
  - If PROC_A is a native SQL procedure, its dependence on PROC_B will prevent successful execution of the drop of PROC_B
  - At some sites, changes to external stored procedures are routinely accomplished via DROP and re-create of the procedure
    - Add native SQL procedures to the mix, and DROP/re-create may not be the right approach in some cases
    - If DROP PROCEDURE is blocked by a dependency, try to make changes via ALTER PROCEDURE, instead
    - Dependencies of native SQL procedures on called stored procedures can be checked via SYSPACKDEP catalog table (DTYPE = ‘N’)
Hints, tips, etc.
Stored procedures and CICS transactions

- Doesn’t have to be an “either/or” thing – they can be used very effectively together
  - An interesting real-life situation: an organization coded a DB2 stored procedure to access data in a VSAM file
  - Worked fine until volume ramped up, then throughput was really bad
  - Stored procedure was changed to invoke a CICS transaction program that accessed the VSAM data, and throughput got much better
    - Here’s why: after the change, the VSAM file, once opened, stayed allocated to CICS
    - Throughput was initially negatively impacted by the file open and close overhead incurred every time the stored procedure that directly accessed the VSAM file was executed
External procedures: TYPE SUB vs. MAIN

- An option on the CREATE (or ALTER) PROCEDURE statement 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
  - However, 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
External procedures: STAY RESIDENT YES/NO

- Another option of 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)
Native SQL procedure source code management (SCM)

- Existing vendor-supplied SCM tools applicable when there is an external-to-DB2 executable and associated source code
  - What if you’re working with native SQL procedures?
    • Language is SQL PL, “source” is the CREATE PROCEDURE statement
    • SQL PL support currently lacking in many popular SCM tools

- In response to user requests for help in this area, APAR PM29226 (DB2 9 and 10 PTFs available in September, 2011) modified sample job DSNTEJ67
  - The job facilitates conversion of an external SQL procedure to a native SQL procedure, but don’t get hung up on that if you don’t have any external SQL procedures!
    • The new functionality is intended to assist with native SQL procedure source code management
More on APAR PM29226

- The external-to-native SQL procedure conversion accomplished via modified DSNTEJ67 illustrates use of new services (implemented via REXX routines)
  - One service extracts SQL procedure source from catalog, places it in a file (or, if you wish, into a string)
  - Another service invokes the SQLPL precompiler, and produces a listing of a SQL procedure
  - Another service enables one to change various elements of the SQLPL source for a procedure: schema, version ID, all the options
    - Can specify new values, or have values removed
  - Also a service to deploy SQLPL source

- Bottom line: lots of organizations are handling SQL PL SCM in a “roll your own” way – the PM29226 routines can help
Native SQL procedure debugging: Data Studio

- Workstation-based tool, downloadable at no charge from IBM’s Web site:

- A very useful reference document: IBM “red paper” titled “Data Studio and DB2 for z/OS Stored Procedures”

- Note: Version 3.2 of Data Studio (current version) includes functionality of what had been Optim Development Studio, Optim Database Administrator, and Data Studio Health Monitor
  - So, references to the functionality of these former tools in documentation (including above-referenced “red paper”) now pertain to Data Studio

- Data Studio also great for native SQL procedure development
Data Studio: getting into debug mode

1. Deploy stored procedure with "Enable debugging"

2. Select stored procedure for Debug...
Data Studio: the debug perspective

- **Resume**: to continue execution from the current location.
- **Terminate**: to stop the execution.
- **Step into**: to enter an enclosing procedure.
- **Step over**: to execute a procedure without entering it.
- **Step return**: to return from a procedure.
- **Variables**: to view and change variable values.
- **Current value**: to view the current value of a variable.
- **Current location**: to view the current execution location.
- **Breakpoint**: to set a breakpoint in the code.
Data Studio debug perspective – breakpoints

Clicking resume will stop at next break point

Debugger stops at line 11

All breakpoints in this session
Thanks for your time

Robert Catterall
rfcatter@us.ibm.com