Stored Procedures in DB2 Version 4

BY CRAIG S. MULLINS

The Basics

Stored procedures are specialized pieces of code that are stored in a Relational DataBase Management System (RDBMS). The motivating reason for stored procedure support is to move SQL code off of the client and onto the database server. This results in less overhead because one client request can invoke multiple SQL statements.

Stored procedures can be thought of as similar to other database objects such as tables, views and indexes because they are controlled by the DBMS. Depending upon the particular implementation, stored procedures may also physically reside in the RDBMS. However, a stored procedure is not "physically" associated with any other object in the database. It can access and/or modify data in one or more tables. Basically, stored procedures can be thought of as "programs" that "live" in the RDBMS.

A stored procedure must be directly and explicitly invoked before it can be executed. In other words, stored procedures are not event-driven. Contrast this with the concept of database triggers, which are event-driven and never explicitly called. Instead, triggers are automatically executed (sometimes referred to as "fired") by the RDBMS as the result of an action. Stored procedures are never automatically invoked.

DB2's Stored Procedure Implementation

DB2 for MVS Version 4 provides stored procedure support. However, if you have experience using stored procedures in another RDBMS, you will notice immediately that DB2's implementation is quite different. Both Sybase SQL Server and Oracle 7 enable users to code stored procedures using procedural extensions to SQL: Sybase provides Transact SQL and Oracle provides PL/SQL. DB2, on the other hand, enables stored procedures to be written in traditional languages. Any LE/370-supported language can be used to code stored procedures. The current list of supported languages therefore is:

- Assembler;
- C;
- COBOL; and
- PL/I.

DB2 stored procedures can issue both static and dynamic statement SQL statements with the exception of CALL, COMMIT, ROLLBACK, CONNECT, SET CONNECTION and RELEASE. Additionally, a stored procedure can issue DB2 commands and IFI calls. Stored procedures can access flat files, VSAM files and other files as well as DB2 tables. Additionally, because stored procedures utilize the Call Attach Facility (CAF), they can access resources in CICS, IMS and other MVS address spaces.

Once coded, stored procedures must be registered in the DB2 Catalog. This is in sharp contrast to the manner in which other database objects are recorded in the DB2 Catalog. Typically, when an object is created, DB2 automatically stores the meta data description of that object in the appropriate DB2 Catalog tables. For example, when a new table is created, DB2 automatically records the information in SYSIBM.SYSTABLES, SYSIBM.SYSCOLUMNS and possibly SYSIBM.SYSFIELDS. Since stored procedures are not created within DB2, nor are they created using DDL, an administrator must use SQL INSERT statements to populate a new DB2 Catalog table, SYSIBM.SYSPROCEDURES, with the meta data for the stored procedure (refer to Table 1 for a description of this table).

After the stored procedure has been registered to DB2, it must be started using a new DB2 command:

```
-START PROCEDURE (procedure name)
```

A stored procedure cannot be executed until it has first been started. Two additional administrative commands for stored pro-cedures have been added:

```
-STOP PROCEDURE(procedure name)
ACTION(REJECT I QUEUE)
```

The stop command will disable subsequent executions of the named stored procedure. The ACTION parameter can be specified to indicate whether future attempts to run the stored procedure will be entirely rejected or queued to be run when the stored procedure is started again.

```
-DISPLAY PROCEDURE(procedure
name)
```

The display command can be used to monitor the status of stored procedures. This command will show:

- whether the named procedure is currently started or stopped;
- how many requests are currently executing;
- the high water mark for concurrently running requests;
- how many requests are currently queued;
- the high water mark for concurrently running requests; and
- how many times a request has timed-out.

To run a stored procedure the user must explicitly issue a CALL statement. For example, the following statement calls a stored procedure named SAMPLE sending a literal string as a parameter:

```
EXEC SQL
  CALL SAMPLE('ABC')
END-EXEC.
```

Stored procedures run in a new DB2 address space known, appropriately enough, as the Stored Procedure Address Space (SPAS). IBM made a wise move in forcing stored procedures to run in their own address space because it will eliminate the possibility of potentially bug-ridden stored procedure code "stepping on" the DB2 address spaces.

Table 1: SYSIBM.SYSPROCEDURES Columns

PROCEDURE	The name of the stored procedure.
AUTHID	The authorization id of the user running the SQL application that issued the CALL. If the column is blank then the row applies to all AUTHIDs.
LUNAME	LUNAME of the system that issued the CALL. The LUNAME can refer to a local or remote system. If the column is blank then the row applies to all systems.
LOADMOD	The MVS load module corresponding to the application program that is to be used.
LINKAGE	Indicates whether the parameters passed to this stored procedure can contain NULLS.
COLLID	The collection id of the package for this stored procedure.
LANGUAGE	The programming language used to code this stored procedure.
ASUTIME	The number of service units permitted for the execution of this stored procedure before cancellation. If ASUTIME is zero, there is no limit.
STAYRESIDENT	Indicates if a module is to remain in memory after the stored procedures finishes.
IBMREQD	IBM supplied information.
RUNOPTS	A list of run-time options for this stored procedure.
PARMLIST	The parameter list expected by the stored procedure when it runs.

Why Use Stored Procedures?

The predominant reason for using stored procedures is to promote code reusability. Instead of replicating code on multiple servers, stored procedures enable code to reside in a single place: the database server. Stored procedures then can be called from client programs to access DB2 data. This is preferable to cannibalizing sections of program code for each new application that must be developed. By coding a stored procedure, the logic can be invoked from multiple processes instead of being re-coded into each new process every time the code is required.

An additional benefit of stored procedures is increased consistency. If every user with the same requirements is calling the same stored procedures, then the DBA can be assured that everyone is running the same exact code. If each individual user used his or her own individual and separate code, no assurance could be given that the same logic was being used by everyone. In fact, it is almost a certainty that inconsistencies will occur.

Stored procedures are particularly useful for reducing the overall code maintenance effort. Because the stored procedure exists in one place, changes can be made quickly without requiring propagation of the change to multiple workstations.

Another common reason to employ stored procedures is to enhance performance. A stored procedure may result in enhanced performance because it is typically stored in parsed (or compiled) format thereby eliminating parser overhead. Additionally, in a client/server environment, stored procedures will reduce network traffic because multiple SQL statements can be invoked with a single execution of a procedure instead of sending multiple requests across the communication lines.

Additionally, stored procedures can be coded to support database integrity constraints, implement security requirements, reduce code maintenance efforts and support remote data access.

Procedural SQL?

How is DB2's stored procedure support different than the other RDBMS vendors? Well, the biggest difference is the manner in which the stored procedure is coded. As mentioned at the beginning of this article, Oracle and Sybase utilize procedural dialects of SQL for stored procedure creation.

But what is procedural SQL? One of the biggest benefits derived from SQL (and RDBMS products in general) is the ability to operate on sets of data with a single line of code. Using a single SQL statement, multiple rows can be retrieved, modified or removed in one fell swoop! However, this very capability also limits SQL's functionality. A procedural dialect of SQL eliminates this drawback through the addition of looping, branching and flow of control statements. Procedural SQL has major implications on database design.

Procedural SQL will look familiar to anyone who has ever written any type of SQL or coded using any type of programming language. Typically, procedural SQL dialects contain constructs to support looping (while), exiting (return), branching (goto), conditional processing (if...then...else), blocking (begin...end) and variable definition and usage.

The Benefits of Procedural SQL

The most useful procedural extension to SQL is the addition of procedural flow control statements. Flow control within a procedural SQL is handled by typical programming constructs that can be mixed with standard SQL statements. These typical constructs enable programmers to:

- embed SQL statements within a loop;
- group SQL statements together into executable blocks;
- test for specific conditions and perform one set of SQL statements when the condition is true, another set when the condition is false (if ... else);
- suspend execution until a pre-defined condition occurs or a preset amount of time expires; and
- perform unconditional branches to other areas of the procedural code.

The addition of procedural commands to SQL provides a more flexible environment for application developers. Oftentimes, major components of an application can be delivered using nothing but SQL. Stored procedures and complex triggers can be coded using procedural SQL, thereby reducing the amount of host language (COBOL, C, PowerBuilder, etc.) programming required. This is a major benefit.

Additionally, when stored procedures (and triggers) can be written using just SQL, more users will be inclined to use these features. DB2 Version 4 requires stored procedures to be written in a host language. This may scare off many potential developers. Most DBAs I know avoid programming like the plague.

In addition to SQL-only stored procedures, procedural SQL extensions also enable more complicated business requirements to be coded using nothing but SQL. For example, ANSI SQL provides no mechanism to examine each row of a result set during processing. A procedural SQL can accomplish this quite handily using cursors and looping.

The Drawbacks of Procedural SQL

The biggest drawback to

procedural SQL is that it is not currently in the ANSI SQL standard. DB2's stored procedure support is based upon the ANSI SQL3 standard. Lack of ANSI support can result in each DBMS vendor supporting a different flavor of procedural SQL. If your shop has standardized on one particular DBMS or does not need to scale applications across multiple platforms, then this may not be a problem. But, then again, how many shops does this actually describe? Not many, I'd venture to guess!

The bottom line is that scalability will suffer when applications are coded using non-standard extensions—like procedural SQL. It is a non-trivial task to re-code applications that were designed to use stored procedures and triggers using procedural SQL constructs. If an application needs to be scaled to a platform that utilizes a DBMS that does not support procedural SQL, this is exactly what must be done.

Performance drawbacks can be realized when using procedural SQL if the developer is not careful. For example, improper cursor specification can cause severe performance problems. But, this can happen just as easily when cursors are used inside a host language. The problem is more inherent to application design than it is to procedural SQL.

The final drawback is that even procedural SQL dialects are not computationally complete. Most dialects of procedural SQL lack programming constructs to control the users screen and mechanisms for data input/output (other than to relational tables).

Conclusion

Although not new to the RDBMS industry, stored procedures are a powerful new feature of DB2 Version 4. They enable multiple data access statements to be executed with a single request. Additionally, they are controlled and managed by DB2 providing a consistent and reusability point of reference for frequently executed database code. As such, they promise to be one of the most exciting and useful features of DB2 Version 4.

Craig S. Mullins is a member of the Technical Advisory Group at PLATINUM technology, inc. He has more than seven years of experience in all facets of data base systems development, including developing and teaching DB2 classes, systems analysis and design, data base and system administration and data analysis.

Was this article of value to you? If so, please let us know by circling Reader Service No. 34.

Would You Like To Be Published in

IDUG SOLUTIONS JOURNAL ?

All the readers of the IDUG Solutions Journal have valuable experiences they can share within the pages of our magazine.

If you are interested in submitting an article, please mail to:
Lisa Ryan, Managing Editor
IDUG Solutions Journal
19380 Emerald Drive
Brookfield, WI 53045-3617

Send a copy of the article along with a 5.25" or 3.5" diskette in either ASCII or WordPerfect 5.1 format. All of our articles go through a review process by our technical reviewers so please allow several weeks for our reply. Deadlines are four months prior to the targeted issue.