*Craig S. Mullins*

# Stored Procedure Guidelines for DB2

*Craig shows you how to implement stored procedures intelligently to decrease your overhead costs.*

BOTH OF THE MAJOR IMPLEMENTations of DB2 (DB2 for MVS and DB2 for Common Servers) provide support for stored procedures. In this article, I discuss tips and tricks for implementing useful stored procedures, as well as guidelines for proper stored procedure administration.

## WHAT IS A STORED PROCEDURE?

Stored procedures are specialized programs that are stored in a relational database management system. The motivating reason to use stored procedures is to enable developers to relocate application code from the client to the database server. One client request can therefore invoke multiple SQL statements and, as a result, overhead costs will decrease.

Stored procedures are similar to other database objects such as tables, views, and indexes in that they are managed and controlled by (or within) the RDBMS. Stored procedures may also physically reside in the RDBMS (although this structure is not always the case for DB2). However, stored procedures are not "physically" associated with any other object in the database. Stored procedures can access and/or modify data in one or more tables; think of them 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. In contrast, consider the concept of database triggers, which are event-driven and never explicitly called. Triggers are automatically executed — a process sometimes referred to as "firing" — by the RDBMS as the result of an action. Stored procedures, however, are never automatically invoked.

## WHY USE STORED PROCEDURES?

**Reuse.** 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 can then be called from client programs to access DB2 data. This process is preferable to cannibalizing sections of program code for each new application you must develop. By coding a stored procedure, the logic can be invoked by multiple programs instead of being recoded into each new process every time the code is required. When implemented wisely, stored procedures are 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.

However noble the goal of reusable components, simply mandating the use of stored procedures will not ensure that goal. Documentation and management support (perhaps even coercion) is necessary to ensure successful reuse. The basic maxim applies: "How can I reuse it if I don't know it exists or I don't know what it does?"

**Consistency.** 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 is using his or her own individual and separate code, no assurance can be given that everyone is using the same logic. In fact, inconsistencies will almost certainly occur.

**Data Integrity.** In addition, stored procedures can be coded to support database integrity constraints. Column validation routines can be coded into stored procedures that are called each time an attempt is made to modify the column data. Of course, this setup will only catch planned changes that are issued through applications that utilize the stored procedure. Ad hoc changes will not be checked; ad hoc changes require triggers that can call the stored procedure (DB2 for MVS does not yet support triggers).

**Performance.** 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. In addition, 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. Figure 1 shows a call to a stored procedure in DB2 for MVS. The passing of SQL and results occurs within the SPAS, instead of over the network — as would be necessary without the stored procedure. Only two network requests are required: one to request that

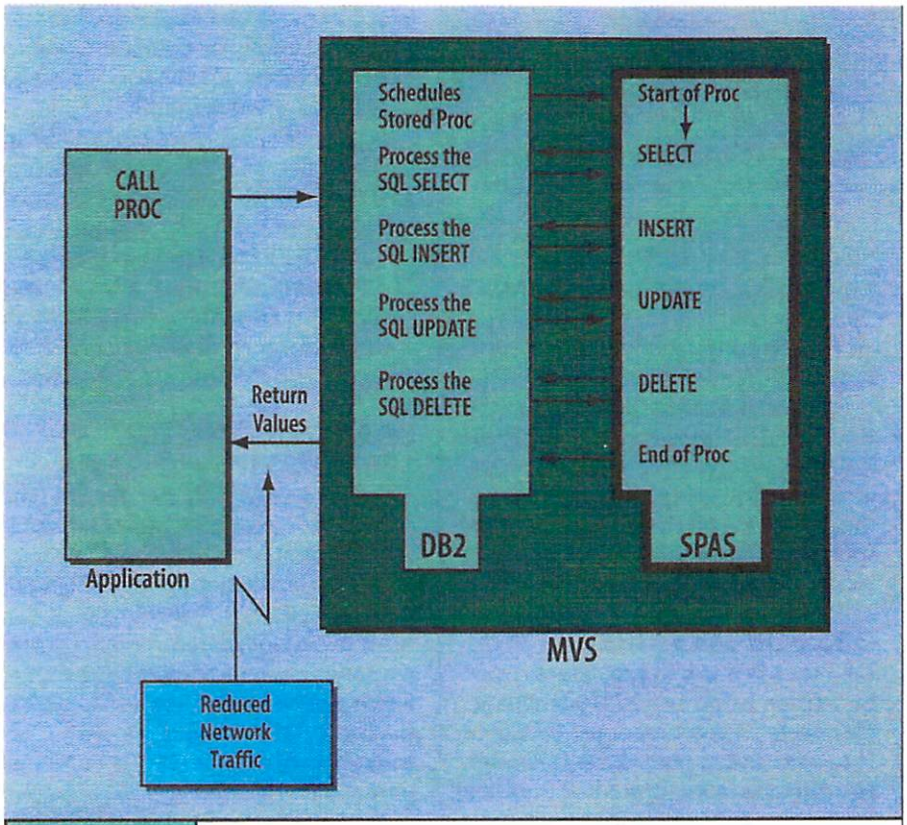


Figure 1 *Calling a stored procedure.*

the stored procedure be run, and one to pass the results back to the calling agent.

**Security.** You can use stored procedures to implement and simplify data security requirements. If a given group of users requires access to specific data items, you can develop a stored procedure that returns only those specific data items. You can then grant access to users to call the stored procedure, without giving them any additional authorization to the underlying objects within the body of the stored procedure.

Stored procedures provide a myriad of other useful benefits, including:

- *Flexibility.* Stored procedures can issue both static and dynamic SQL statements and access DB2 and non-DB2 data.
- *Ease of Training.* DB2 stored procedures are written in traditional programming languages that many application programmers already know.
- *Database Protection.* Stored procedures can be developed (they must be for MVS) to run in a separate address space from the database engine, thereby eliminating the possibility of users corrupting the DBMS installation.

## STORED PROCEDURE TIPS AND TRICKS

On the surface, stored procedures appear to be a simple and highly effective way to improve application performance, enhance database administration, and promote code reusability. However, as with every new database feature, there are good and bad ways to proceed with implementing stored procedures. Consider the following caveats before coding stored procedures at your shop.

*Avoid calling stored procedures from other stored procedures.* When one procedure calls another procedure, the ensuing structure is called a nested procedure. Nested procedures are difficult to test and modify. Furthermore, when one procedure calls another, the likelihood of reuse decreases as the complexity increases, because the stored procedure that results is more difficult to understand and use. Although DB2 does not currently support nested procedures, it is wise to avoid nested procedures throughout RDBMS stored procedure development (in case IBM changes this restriction in a future release).

*Design and implement only useful stored procedures.* By useful, I mean only those stored procedures that support a business rule and are robust enough to perform a complete task but are not small enough to be trivial (a two-line procedure) or too large to be understood (a thousand-line procedure that performs every customer function known to the organization). To be useful, a stored procedure must:

- perform one task and perform it very well
- correspond to a useful business function
- be documented (including a description of the input, output, and process)

*Always specify parameters at an atomic level.* In other words, every stored procedure parameter must be complete and non-divisible. When parameters are coded as non-atomic variable blocks, the stored procedure logic must parse the block. If changes to the data cause changes in length or data type, procedures that use atomic parameters will be easier to modify and test.

*Be sure to read and understand the limitations and requirements of DB2 stored procedures.* For example, certain statements and commands cannot be issued from within a stored procedure. DB2 for MVS stored procedures cannot issue CALL, COMMIT, ROLLBACK, CONNECT, SET CONNECTION, and RELEASE; and DB2 for Common Server stored procedures cannot issue CALL, CONNECT, SET CONNECT, RELEASE, CONNECT RESET, CREATE DATABASE, DROP DATABASE, BACKUP, RESTORE, and FORWARD RECOVERY.

*Ensure that the political aspects of stored procedure creation, usage, and support have been adequately determined and documented prior to implementation.* For example, who will code stored procedures — DBAs or application programmers? This decision can vary from shop to shop based upon the size of the organization, the number of DBAs, and the commitment of the organization to stored procedures. You can make a credible case that the task should be a centralized function in order to promote reusability and documentation.

*Once you decide who develops the stored procedures, you must next decide who supports them.* Stored procedure support must encompass design and code review, quality assurance testing, documentation review, reusability testing, and "on call" support. If a centralized group is not "on call" for stored procedure failures, organizational in-fighting will probably occur.

Consider a stored procedure developed by the marketing application staff that modifies customer information. The stored procedure is developed, tested, documented, and migrated to production. Because proper reusability guidelines were followed, the sales application staff calls the same stored procedure in their code. Once in production, the sales application fails at 2:00 A.M. Who gets called in to fix the problem? The sales

| | | |
|---|---|---|
| C | Cobol | PL/I |
| C++[1] | OO Cobol[2] | Assembler |

[1] Requires APAR PN78797, PTF UN86554, and LE/370 V1.4
[2] Requires APAR PN78797, PTF UN86554, and LE/370 V1.5

Table 1 *LE/370 languages you must use to write stored procedures for DB2 for MVS.*

staff argues that the stored procedure was created by marketing. The marketing staff argues that their application did not bomb — sales' application did. Without a centralized support function, the argument could go on all night.

To solve these problems, the role of supporting stored procedures should fall to a group of professionals skilled in program development and procedural logic, as well as SQL and database administration. A new type of DBA should be defined to support and manage stored procedures (and other server code objects) and other code-related DBA tasks, such as:

- Server Code Object Support: reviewing, supporting, and possibly even coding stored procedures, triggers, and user-defined functions
- Application Program Design Reviews: reviewing every application program completely before migrating the code to a production environment
- Access Path Review and Analysis: using `EXPLAIN` and other tools to determine the type of access chosen by DB2
- SQL Debugging: assisting developers with difficult SQL syntax and structures
- Complex SQL Analysis and Rewrite: tweaking SQL for optimal performance

This new role can be defined as a procedural DBA. (See Figure 2.) When the procedural tasks are offloaded from the traditional, data-oriented DBAs, these DBAs will be free to concentrate on the actual physical design and implementation of databases. This separation of responsibility should result in much better database design and performance.

## DB2 FOR MVS TIPS

DB2 for MVS stored procedures must be written using an LE/370 language. (See Table 1.) You cannot use VS Cobol II to code stored procedures (although you can call a stored procedure from any DB2-compatible host language, even non-LE/370 languages).

If your shop has technical DBAs who like to code their own administration tools and performance monitoring applications, consider using stored procedures to issue DB2 commands and access trace records using the IFI (Instrumentation Facility Interface). You can develop generalized procedures that are maintained by the DBA and accessed by multiple programs to start, stop, and display database objects or analyze IFCIDs and display performance details.

Remember, DB2 stored procedures can access flat files, VSAM files, and other files as well as DB2 tables. In addition, because stored procedures utilize the Call Attach Facility (CAF), they can access resources in CICS, IMS, and other MVS address spaces. This capability makes them an ideal vehicle for applications that require access to both IMS and DB2 databases.

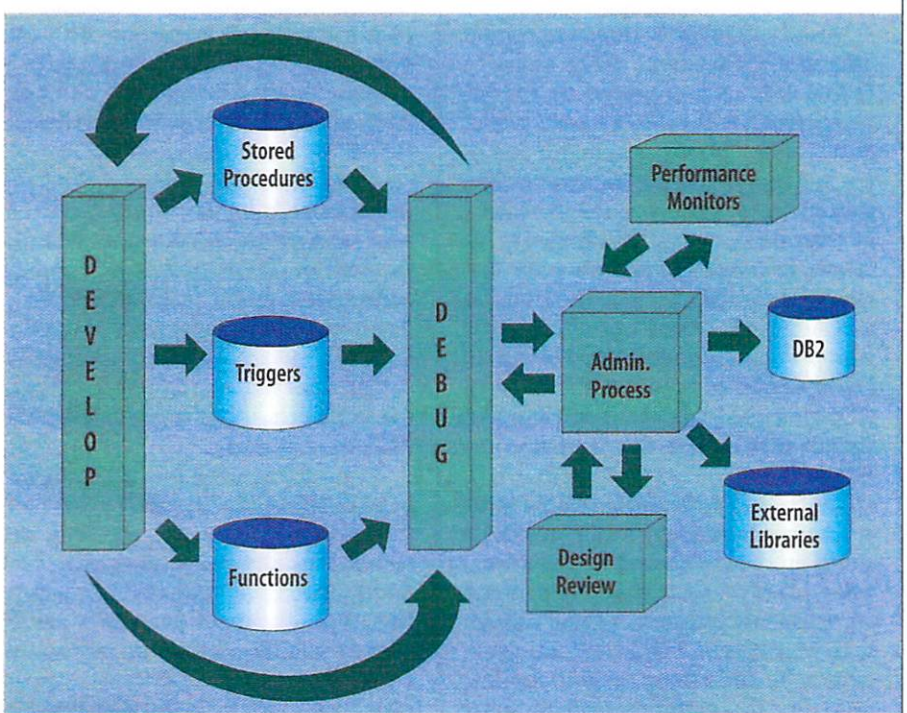


Figure 2 *Procedural DBA tasks.*

## DB2 FOR COMMON SERVERS TIPS

Stored procedures written for DB2 for Common Servers must be written using a traditional programming language. However, the list of supported languages differs by platform, which makes it difficult to develop stored procedures that are portable across multiple DB2s on different platforms. For AIX and OS/2, see the list in Table 2. Note that REXX, although available for AIX, is only supported under OS/2.

Avoid using the Database Application Remote Interface (DARI), which was the stored procedure interface used for DB2 for Common Servers version 1. Although DARI is still available under version 2, the standard SQL `CALL` statement is the preferable method of invoking stored procedures because of its compliance with industry standards.

Always run stored procedures in "fenced mode." A fenced stored procedure runs in a separate process from the database agent processes. In other words, the stored procedure must accrue additional communication overhead. However, fencing protects the database manager's control structure from an errant stored procedure command. You can build stored procedures to run unfenced, but the potential for damage far outweighs any possible performance gains.

## PROCEDURAL SQL?

How does DB2's stored procedure support differ from the other RDBMS vendors? The most significant difference is the manner in which the stored procedure is coded. Oracle, Sybase, Microsoft SQL Server, and Informix let developers write stored procedures using procedural extensions to SQL. Each of these languages is proprietary (the vendor created and owns the code). In addition, these languages cannot interoperate with one another. DB2, on the other hand, requires that stored procedures be written in traditional programming languages.

| Product | Language | Implementation |
|---|---|---|
| DB2 for AIX | C | IBM XL C Compiler 1.2.1 or 1.3 |
| | | IBM C for AIX 3.1 |
| | C++ | IBM C/Set++ for AIX 2.1 or 3.1 |
| | Cobol | IBM Cobol Set for AIX 1.1 |
| | | Micro Focus Cobol 3.1 or later |
| | Fortran | IBM AIX XL Fortran/6000 2.3 |
| | | IBM XL FORTRAN for AIX 3.2 |
| DB2 for OS/2 | C or C++ | IBM C/SET++ for OS/2 2.1 |
| | | IBM VisualAge C++ for OS/2 3 |
| | Cobol | IBM Cobol VisualSet for OS/2 1.1 |
| | | Micro Focus Cobol 3.1 or later |
| | Fortran | Watcom Fortran 77 32 version 9.5 |
| | REXX | IBM Procedures Language 2/REXX |

Table 2 *Languages with which you can write stored procedures for DB2 for Common Servers.*

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 (do...while), branching (goto), conditional processing (if...then...else), and flow of control statements. As the ANSI committee works on creating a standard version of SQL that includes procedural support, look for IBM to relent and supply a procedural version of its SQL dialect.

## DATABASIC

IBM also provides an add-on stored procedure development product named DataBasic. DataBasic provides facilities for developing, testing, and maintaining stored procedures written for DB2 for AIX and DB2 for OS/2 (support for DB2 for MVS is reportedly in development and will be provided in the future). DataBasic provides a Basic interpreter similar in functionality to Microsoft Visual Basic for the development of DB2 stored procedures. Using DataBasic, developers can create portable stored procedures (at least across OS/2 and AIX) in a visual programming environment.

## SYNOPSIS

Although not new to the RDBMS industry, stored procedures are a powerful new feature of DB2. They let you execute multiple data access statements with a single request. In addition, they are controlled and managed by DB2, providing a consistent and reusable point of reference for frequently executed database code. If implemented effectively, they promise to be one of the most exciting and useful features of DB2 development in a client/server environment.

**Craig S. Mullins** is a frequent contributor to computer industry publications. His book, *DB2 Developer's Guide* (Sams Publishing, 1994), contains DB2 development techniques, tips, and guidelines. Craig has over a dozen years experience in all facets of database systems development. You can email Craig at 70410.237@compuserve.com.

*Continued from page 11*

tionality. This amount, however, will depend on how the implementation is phased and on how many users will access the data. Typically, the implementation phase might be based on the data load sequence or on the data subject area (customer, product, risk management, revenue/expenses, and so on). These subject areas are the building blocks of the data warehouse that will have a major impact on the implementation phases. Remember the phrase: "If you build it they will come." Data storage and load cycles are very expensive. If you load too much data into the data warehouse, it may become underutilized or, worse, not used at all.

• *What is the level of data granularity?* This concept identifies whether the data is stored at a detailed level, summarized level, or both. The level will be noted in your architecture document and will drive a number of other issues. It will impact whether derived data is calculated each time for queries or stored in the data warehouse. If it is stored, will it be available in the main warehouse or derived and brought down to the delivery layer each time? Remember to provide flexibility for future use and data integration.

• *How often must the data be refreshed?* The refresh cycle could be daily, weekly, or monthly. It may also be all the above, depending on the type of data. Customer-indicative data might be refreshed monthly, whereas revenue- or credit-related data might be refreshed weekly or daily depending on the OLAP requirements designated by the business. The refresh cycle will also depend on the update schedules of the operational systems from which the data is being acquired.

**Data storage and load cycles are very expensive.**

• *On which platform will the data warehouse be developed and implemented?* You'll usually find the answer to this question in your architecture document, and it depends on many things: the refresh cycle, the data volume, the complexity of the legacy information acquisition process, the access requirements, and the tools available.

## A GOOD FOUNDATION

It should now be clear that you need to develop a clearly defined data warehouse architecture early in your projects. During the development process, the data warehouse development team will constantly refer to your architecture document for direction and guidance — so ensure that they are accurate and complete. With a complete set of blueprints you can successfully build a data warehouse that is complex in nature and that can withstand extreme stress.

**Denis Kosar** is the vice president of Enterprise Data Architecture at Chase Manhattan Bank. He has played a lead role in the two major data warehouses developed in the wholesale bank. He is a contributing author to the upcoming book, *Data Warehousing, Practical Advice from the Experts* (Prentice Hall). Parts of this article have been extracted with the permission of Prentice Hall. You can email Denis at 76345.2142@compuserve.com.