Mullins Consulting, Inc.
Database Performance Management

Return to Home Page

February 2000

DB2 update

DB2 and the Procedural DBA

By Craig S. Mullins

In 1995 I coined the term *Procedural DBA* to describe a new type of database administration required by modern database management. The concept is a simple one: a new type of DBA is required to manage the procedural logic that is increasingly being stored in relational database management systems.

Initially, the sole purpose of a DBMS was to store, manage, and access data. Over time, database management systems have evolved by integrating procedural logic in the form of complex triggers, stored procedures, and user-defined functions. This code is tightly coupled to

the DBMS. As these features are exploited, management tasks such as administration, design, and tuning typically are assigned to the current DBA staff by default. But this is not always the best approach. What is required is an expansion of the role of database administration.

The Classic Role of the DBA

When a DBMS is implemented appropriately, its use spans the enterprise. Multiple applications, consisting of multiple programs, access and manipulate data stored in databases that are managed by the DBMS. A scenario such as this is good because it reduces data redundancy and increases data integrity. However, this same situation also effectively places the DBA on call for all of the applications of the organization. If the database portion of any application fails, the DBA must be able to fix the problem bringing the database back on-line so the application can execute.

To make matters more difficult, the role of the DBA has expanded over the years. In the pre-relational days, both database design and data access was complex. Programmers were

required to explicitly code program logic to navigate through the database structure to access data. Usually the pre-relational DBA was assigned the task of designing the hierarchic or network database design. Almost always, this process consisted of both logical and physical database design, although it was not always recognized as such at the time. Once the database was planned, designed, and generated, and the DBA created backup and recovery jobs, little more than space management and reorganizations were required. Of course, this sounds easier than it actually was. Pre-relational DBMS products (such as IMS) require a complex series of utility programs to be run in order to perform backup, recovery, and reorganization, consuming a large amount of time and effort.

Today, of course, DBAs still design databases, and perform tasks such as backup, recovery, and reorganization. But increasingly databases are generated from logical data models created by data administration staffs using data modeling and database design tools. Additionally, the utilities for performing backup, recovery, and reorganization are simpler to build in the relational world.

Although the up-front effort required to design a relational database is reduced, it is not eliminated. Relational design still requires physical implementation decisions such as table design, partitioning, indexing, normalization and denormalization. But instead of just performing physical implementation and administration, DBAs are more intimately involved with procedural data access too. The nature of relational technology requires additional involvement during the design of data access routines. This is true because relational optimizer technology embedded into the RDBMS is used to choose the best access paths to the data. The optimization choices must be reviewed by the DBA. Therefore, application program and SQL design reviews are a vital component of the DBA's job.

Furthermore, DBAs perform most monitoring and tuning responsibilities. DBAs use tools like EXPLAIN, performance monitors, and SQL analysis tools to proactively administer RDBMS applications.

Oftentimes, DBAs are not adequately trained in these areas. It is a distinctly different skill to program than it is to create well-designed

relational databases. Yet, DBAs quickly learn that they have to be able to understand application programming techniques to succeed.

DBMS-Coupled Application Logic

Although DB2 was one of the last major RDBMS products to gain a full complement of tools for storing procedural logic in the database, its current support as of Version 6 is very robust. DB2 provides support for stored procedures, triggers, user-defined functions, a procedural version of SQL based on SQL/PSM, and user-defined data types.

A procedural SQL language adds features such as looping, branching, and flow of control statements to make SQL a more functionally complete and useful programming language. Using DB2's version of SQL/PSM, developers can create complex functional stored procedures and triggers without the need to code a 3GL program.

Let's define the different types of logic that can be stored, accessed, and managed in DB2 databases.

Stored procedures are procedural logic that is maintained, administered, and executed through the RDBMS. The primary reason for using stored procedures is to move application code off of the client and on to the database server. This can result in reduced overhead because one client can invoke a stored procedure consisting of multiple SQL statements. Invoking one procedure to execute multiple SQL statements is preferable to the client executing multiple SQL statements directly because it minimizes network traffic thereby enhancing overall application performance. 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.

Triggers are event-driven specialized procedures that are stored in, and executed by, the RDBMS. Each trigger is attached to a single, specified table. Triggers can be thought of as an advanced form of "rule" or "constraint" written using procedural logic. A trigger can not be directly called or executed; it is automatically executed (or "fired") by the RDBMS as the result of an action—usually a data modification to the

associated table. Once a trigger is created it is always executed when its "firing" event occurs (update, insert, delete, etc.).

User-defined functions, or **UDFs**, provide developers with the ability to extend the SQL language. Once coded, a UDF can be specified wherever a built-in SQL function can be specified. In general, DB2 functions (both built-in and user-defined) can be used any place an expression can be used (with some exceptions). Functions are called by specifying the function name and any required operands.

You can think of stored procedures and triggers and user-defined functions like other database objects such as tables, views, and indexes, because are controlled by and managed within DB2. These objects are often collectively referred to as *server code objects, or SCOs,* because they are actually program code that is managed by a database server as a database object.

Why Use Server Code Objects?

The predominant reason for using SCOs is to promote code reusability. Instead of replicating code on multiple servers or within multiple

application programs, SCOs enable code to reside in a single place: the database server. SCOs can be automatically executed based upon context and activity or can be called from multiple client programs as required. This is preferable to cannibalizing sections of program code for each new application that must be developed. SCOs enable logic to be invoked from multiple processes instead of being re-coded into each new process every time the code is required.

An additional benefit of SCOs is increased consistency. If every user and every database activity (with the same requirements) is assured of using the SCO instead of multiple, replicated code segments, then the organization can be assured that everyone is running the same, consistent code. If each individual user deployed his or her own individual and separate code, no assurance could be given that the same business logic was being used by everyone. In fact, it is almost a certainty that inconsistencies would occur.

Additionally, SCOs are useful for reducing the overall code maintenance effort. Because SCOs exist in a single place (the RDBMS), changes

can be made quickly without requiring propagation of the change to multiple workstations.

Finally, SCOs can be coded to support database integrity constraints, implement security requirements, reduce code maintenance efforts, support remote data access, and, as mentioned earlier, enhance performance. Of course, in order to achieve these gains SCOs need to be effectively managed and administered. Hence the need for a Procedural DBA.

The Procedural DBA

Once server code objects are coded and made available to the RDBMS, applications and developers will begin to rely upon them. Although the functionality provided by SCOs is unquestionably useful and desirable, DBAs are presented with a major dilemma. Now that procedural logic is being stored in DB2, DBAs must grapple with the issues of quality, maintainability, and availability. How and when will these objects be tested? The impact of a failure is enterprise-wide, not relegated to a single application. This increases the visibility and criticality of these objects. Who is

responsible if they fail? The answer should be—a DBA. But testing and debugging of code is not a typical role for DBAs.

With the advent of server code objects, the role of the DBA is expanding to encompass too many responsibilities for a single person to perform the job capably. The solution is to split the job of DBA into two separate parts based upon the type of database object being supported: data objects or server code objects.

Administering and managing data objects is more in line with the traditional role of the DBA, and is well-defined. But DDL and database utility experts can not be expected to debug procedures and triggers written in sometimes very complex SQL and application code. Debugging a procedure is a very different task than creating a database schema and ensuring that there are no syntax errors. Furthermore, even though many organizations rely upon DBAs to be the SQL experts in the company, often times they are not--at least not DML experts. Simply because the DBA knows the best way to create a physical database design and DDL, does not mean he will know the best way to access that data. It is not uncommon for

a DBA to come from the ranks of network and/or system administration, and SQL is foreign to many system administrators.

The role of administering the procedural logic in a RDBMS should fall upon someone skilled in that discipline. A new type of DBA must be defined to accommodate server code object and procedural logic administration. This new role can be defined as a procedural DBA.

The Procedural DBA should be responsible for database management activities that require programming and similar activities. This includes primary responsibility for server code objects. Whether SCOs are actually programmed by the Procedural DBA may differ from shop-to-shop. This will depend on the size of the shop, the number of DBAs available, and the scope of server code object implementation. Minimally, the Procedural DBA should lead SCO code reviews and perform SCO administration.

Additionally, the Procedural DBA must be on-call for SCO failures. Consider the ramifications if this were not the case. As part of a new project, Team A implements a new stored procedure to

obtain customer information. The project is completed and moved to production. Team B, working in a different department, decides to re-use the customer information stored procedure in their new application. Team B completes its project and moves it into production. Three weeks later, at 2:00 A.M., Team B's application fails because of a failure in the shared stored procedure. Who comes in to fix it? Team A, the team that coded the stored procedure but whose application is working? Or Team B, whose application is down but has no one that understands the stored procedure?

Of course, the correct answer is the Procedural DBA.

Additional issues arise as SCOs are implemented. One such example is the proper maintenance of DB2 triggers once they have been defined. If multiple triggers are defined on the same table for the same action, DB2 will fire the triggers in the order they were created. This can have an impact on the end result of a transaction.

Consider, for example, the following admittedly contrived scenario. Two update triggers exist on

TABLE1, namely TRIGGER1 and TRIGGER2. TRIGGER1 adds the value 5 to COLX in TABLE2; TRIGGER2 multiple the value of COLX in TABLE2 by the value 3. If COLX is equal to 3, after the two triggers fire, COLX would be equal to (3 + 5) * 3, or 24.

Now, consider what would happen if TRIGGER1 is modified for some reason. The trigger is dropped and recreated. Now, TRIGGER2 would fire first, followed by TRIGGER1. The following example would have a different result. If COLX is equal to 3, after the two triggers fire, COLX would be equal to (3 * 3) + 5, or 14. The order in which triggers fire is important, but the only way to control it is based on which trigger was created first. This process must be controlled by someone who understands these nuances, ideally a Procedural DBA.

Other procedural administrative functions that can be allocated to the Procedural DBA include application code reviews, access path review and analysis, SQL debugging, complex SQL analysis, and re-writing queries for optimal execution. Off-loading these tasks to the Procedural DBA will enable the traditional, data-oriented DBAs to concentrate on the actual

physical design and implementation of databases. This should result in much better designed databases… and much better performing SQL.

The Procedural DBA should still report through the same management unit as the traditional DBA and not through the application programming staff. This enables better skills sharing between the two distinct DBA types. There must be a high degree of synergy between the Procedural DBA and the application programmer. The typical job path for the Procedural DBA should grow from the programming ranks because this is where the coding skill-base exists.

The procedural DBA should still report through the same management unit as the traditional DBA and not through the application programming staff. This enables better skills sharing between the two distinct DBA types. Of course, there will need to be a greater synergy between the procedural DBA and the application programmer/analyst. In fact, the typical job path for the procedural DBA should come from the application programming ranks because this is where the coding skill-base exists.

Synopsis

As you implement triggers, stored procedures, and user-defined functions to support business rules in your applications, be aware that database administration becomes more complex. The role of the DBA is rapidly expanding to the point where no single professional can be reasonably expected to be an expert in all facets of the job. It is high time that the job is explicitly defined into manageable components, starting with the Procedural DBA.

From DB2 Update (Xephon), February 2000.

Home.