Craig S. Mullins & Associates, Inc.

Database Performance Management

Return to Home Page

June 1998

Computing : News&Review

Wanted: New Type of DBA

By Craig S. Mullins

Until recently, the domain of a database management system was, appropriately enough, to store, manage, and access data. Although these core capabilities are still required of modern DBMS products, all of the popular RDBMSs are adding complex features to integrate procedural logic. Triggers, stored procedures, and user-defined functions are example of logic that is tightly coupled to the DBMS. Typically, as these newer features are exploited, the administration, design, and management of these features is assigned to the database administrator (DBA) 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

The DBMS spans the enterprise, effectively placing the DBA on call for the applications of the entire organization. And the role of the DBA has expanded over the years. Of course, DBAs still design databases, but instead of merely performing physical implementation and administration, DBAs are more intimately involved with data access.

The nature of the RDBMS 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 access paths to the data. The optimization choices must be reviewed by the DBA. Program and SQL design reviews are a vital component of the DBA's job. Furthermore, DBAs perform most monitoring and tuning responsibilities. Backup, recover, and reorganization are just a start. DBAs must use tuning 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.

The Trend of Storing Process With Data
As RDBMS products mature, more functionality is added. The clear trend is to enable procedural logic to

be stored in the database. The most popular and robust RDBMSs support three primary forms of database-administered procedural logic: stored procedures, triggers, and user-defined functions (UDFs).

Stored procedures are procedural logic that is maintained, administered, and executed through the RDBMS. Stored procedures move application code off of a client workstation and on to the database server. This typically results in less overhead because one client can invoke a stored procedure and cause the procedure to invoke multiple SQL statements. This is preferable to the client executing multiple SQL statements directly because it minimizes network traffic which can enhance 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.).

A UDF, or user-defined function, provides a result based upon a set of input values. UDFs are programs that can be executed in place of standard, built-in SQL scalar or column functions. A scalar function transforms data for each row of a result set; a column function evaluates each value for a particular column in each row of the results set and returns a single value. Once written, and defined to the RDBMS, a UDF becomes available just like any others built-in function.

Stored procedures, triggers, and UDFs are just like other database objects such as tables, views, and indexes, in that they are controlled by the DBMS. These objects are often collectively referred to as server code objects, or SCOs, because they are program code that is stored and maintained by a database server as a database object. Depending upon the particular RDBMS implementation, these object may or may not "physically" reside in the RDBMS. They are, however, always registered to, and maintained in conjunction with, the RDBMS.

Why Are Server Code Objects So Popular?
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. 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.

Another common reason to employ SCOs to enhance performance. A stored procedure, for example, 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.

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 the DBMS, 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 must be-a DBA. But testing and debugging of code is not a typical role for DBAs.

With the advent of SCOs, 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 DBA's job into two separate parts based upon the database object to be 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 functions written in C, COBOL, or even procedural SQL. Furthermore, even though many organizations rely upon DBAs to be the SQL experts in the company, often times they are not experts-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.

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.

Other procedural administrative functions that can be allocated to the Procedural DBA include application code reviews, access path review and analysis (from EXPLAIN or show plan), 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.

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.

Synopsis
As businesses implement server code objects to support business rules in their applications, 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 be explicitly defined into manageable components, starting with the Procedural DBA.

From *Computing News and Review*, June 1998.