November 1998

SQL Server update

The Pros and Cons of Procedural SQL

By Craig S. Mullins

One of the biggest benefits derived from moving to a relational database management system is the ability to operate on sets of data with a single line of code. Consider, for example, SQL-the predominant RDBMS query language. 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.

Well, this limitation is quickly becoming a thing of the past. Several major RDBMS vendors support a procedural dialect of SQL that adds looping, branching, and flow of control statements. For example, Microsoft SQL Server provides procedural support in Transact-SQL.

Although a single procedural SQL script can impact hundreds (or even thousands) of database rows, it can do so in an organized and step-by-step fashion. Prior to extended, procedural SQL dialects like Transact-SQL, many complex tasks required embedding SQL into a 3rd or 4th generation language that supporting looping, branching, and flow control statements.

Procedural SQL has major implications on database design. Instead of merely designing a database for data structures, the DBA now has to incorporate analysis, design, and administrative effort for procedural objects such as triggers and stored procedures into the development and maintenance of every database. At times, the administration burden required to support procedural objects can be greater than that required to administer data objects.

Sample Procedural SQL

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.

Consider the example Transact-SQL code shown in Figure 1. This piece of logic will retrieve rows from the titles table, format the output, and write the output. It utilizes looping, flow control, exiting, blocking, and cursors.

Figure 1. Procedural Transact-SQL.

```
create proc book_list
as
declare title_cursor cursor for
        select title_id, title, pub_id
        from titles
declare @tid char(6), @title varchar(80), @pubid char(4), @outline varchar(92)

open title_cursor
fetch title_cursor into @tid, @title, @pubid

while @@sqlstatus !=2
begin
        if @@sqlstatus = 1
        begin
                print "Select failed"
                return
        end
        else
        begin
                select @outline = @tid + " " + @title + " " + @pubid
                print @outline
                fetch title_cursor into @tid, @title, @pubid
```

```
        end
end

close title_cursor
deallocate cursor title_cursor
return
```

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
- 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. Often times, 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 (C, Visual Basic, PowerBuilder, etc.) programming required. This is a major benefit.

Additionally, when triggers and stored procedures can be written using just SQL, more users will be inclined to use these features. Some RDBMS products (such as DB2 on the mainframe) require stored procedures to be written in a host language. This scares off many potential developers. Most DBAs I know avoid programming like the plague. But writing SQL isn't really programming so why not give them a go?

In addition to SQL-only triggers and 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. Transact-SQL (and other procedural SQL dialects) can accomplish this quite handily using cursors and looping. Cursor support

enables developers to code robust applications, using nothing but the procedural SQL.

The Drawbacks of Procedural SQL

The biggest drawback to procedural SQL is that it is not currently in the ANSI SQL standard. This results in each DBMS vendor supporting a different flavor of procedural SQL. Microsoft (and Sybase) Transact-SQL, Oracle PL/SQL, and Informix SPL, though similar, are distinct languages and not interoperable with one another. 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 written using procedural SQL constructs. If an application needs to be scaled to a platform which utilizes a DBMS that does not support procedural SQL, then exhaustive re-coding is exactly what must be done. Consider, for example, the steps necessary to move a SQL Server application, written in Transact-SQL using triggers and stored procedures, to a DB2 subsystem on a mainframe. Stored procedures must be converted from Transact-SQL into a host language such as COBOL or C. Triggers must be coded into the application programs supporting the DB2 database because DB2 does not yet support triggers. This process is a nightmare that will cause any DBA to get a headache just thinking about it.

An additional potential drawback comes in the form of the potential for performance degradation. Consider, for example, a SQL Server stored procedure. The first time this procedure is executed it is optimized. The optimized form of the procedure is stored in the procedure cache and will be re-executed for subsequent users. However, the procedure was optimized for the particular data request that was issued for the first execution. It is very likely that future executions of the procedure are for different amounts and types of data. If the logic were instead embedded into an application program and compiled statically, the performance would be optimized for all ranges of local variables. Of course, SQL Server provides an option to always optimize (with the recompile option), but then dynamic SQL is always used — which can cause different types of performance problems.

The solution would be to provide a form of static SQL for stored procedures that is not optimized for a particular type of request — but, of course, this option is not currently available.

Other performance drawbacks can be realized when using procedural SQL if the developer is not careful. For example, careless cursor coding 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 user's screen and mechanisms for data input/output (other than to relational tables).

Synopsis

Whether or not you are in favor of procedural SQL extensions is actually a moot point. Procedural SQL is here to stay. It simply offers too many benefits to be ignored. And, if you seriously doubt the veracity of this statement, reread this article in a couple of years — by then the ANSI standard should include procedural SQL statements. For SQL Server users, the interesting dilemma will be just how much of the SQL standard will be compatible with Transact-SQL.

From SQL Server Update (Xephon) November 1998.

Home. Phone: 281-494-6153 Fax: 281-491-0637