

DATABASE

TRENDS AND APPLICATIONS

Solutions for the Information Project Team • www.dbta.com

June 2008

THE PHILOSOPHY OF PL/SQL



STEVEN FEUERSTEIN

SPODification: A Fitness Regime for Your Code

I often give training and seminars on best practices for PL/SQL development. Of course, there are many guidelines I could (and do) cover in my courses. Some of these recommendations, such as "Use FORALL for high speed DML operations," are specific to PL/SQL. Most, however, could be applied to any programming language.

And of all the best practices I present, I tell my students that the most fundamental and important of them all is simply, "Don't repeat anything."

Repetition produces code that is hard to debug, fix, maintain and optimize. For example, I uncover a bug in the parsing logic I use to generate test code, and I fix it.

A week later, the user reports the same bug. How is this possible? I fixed it! Ah, but it turns out that I pasted a copy of this logic into another program, and forgot all about it. I tell myself that I surely had good reason for doing this, but the bottom line is that I have a mess on my hands.

Avoiding repetition sounds great, if rather

Hard-coding	SPODification
Literal value	<p>Example: IF l_salary > 1500000 THEN</p> <p>Hide the value behind a packaged function; that way, you change the value without any need to recompile dependent code. Otherwise, a constant is a good SPOD for this hard-coding.</p>
Business rule or formula	<p>Example: IF l_employee.date_of_birth > ADD_MONTHS (SYSDATE, 18 * 12 * -1) THEN</p> <p>In other words, an employee must be at least 18 years old.</p> <p>Rules and formulas always get more complicated over time. No matter how simple your rule is today, you should hide it behind a packaged function and call that function as needed.</p>
SQL statement	<p>Surprise! I bet you don't think about SQL statements as being examples of hard-coding, but consider this: every SQL statement is a snapshot at this moment of time of a portion of an application's data model: "This is the six-way join needed today retrieve the required data." Soon, however, that six-way join will become a seven-way join.</p> <p>The solution? Use a data access layer that hides SQL statements behind procedures that change the underlying tables and functions that retrieve data. Generate as much of this code as you can.</p>
Constrained datatype in declaration	<p>Example: l_last_name VARCHAR2(100);</p> <p>Almost every variable we declare is intended to hold a value retrieved from the database. The above line of code freezes the maximum length of the last name variable to 100. Yet the DBA can always extend the size of the column.</p> <p>Use %TYPE and %ROWTYPE attributes to "anchor" your variables back to the database element they are intended to hold. Use SUBTYPES to create new application-specific datatypes that hide the constraint and serve as a SPOD.</p>

Table 1

dull. Where's the excitement in avoiding repetition? Clearly, what we need is an acronym that is catchy and cool, and will serve as motivation for programmers to get with the "program."

And so I am dedicating this column to the noble purpose of coining a new acronym, one that will single-acronym-edly inspire legions of developers to improve the quality of their code:

SPOD

SPOD is an acronym for Single Point of Definition, and when you apply the principles of SPOD to your code, that code undergoes a process of SPODification. There, that sounds much more interesting than "avoid repetition," correct?

The best way to understand the value of a SPOD is to take a close look at that classic boogey-man of programming: hard-coding.

Most programmers believe they know what hard-coding is all about, and how to avoid it. "Hard coding," you will hear, "is

when I use a literal value, like 'OPEN,' in multiple places in my code. Then, if (when) the value changes, I have to find all the places this value appears and change it."

Nasty! No self-respecting programmer would do such a thing, right? Correct. So instead, we create a constant and assign it the value "OPEN," and then reference the constant by name throughout our code. Now, when the value changes, I only have to change the value assigned to the constant.

The use of that constant is one example of a Single Point of Definition. The value is defined in one place, so it only needs to be changed in one place. Yet this is only the simplest and most obvious application of a SPOD, despite what many developers believe.

The most important aspect of SPOD-aware programming is to be aware of the many forms of hard-coding that can pop up in your code. Tabel 1 covers four examples of hard-coding and how to SPODify them.

There are yet other examples of hard-cod-

ing and related SPODifications, but this should show you that hard-coding is more widespread than one might at first think. It is quite impossible to write high quality PL/SQL (and any other type of) code if you allow repetitions to creep into your code. If, on the other hand, you heighten your awareness of all the ways that hard-coding can appear, and for each of those identify a SPOD that will get rid of the hard-coding, you will soon be writing the most excellent code.

Steven Feuerstein is an expert on the Oracle PL/SQL language, having written 10 books on PL/SQL, including Oracle PL/SQL Programming and Oracle PL/SQL Best Practices. Feuerstein has been developing software since 1980 and spent five years with Oracle. Today, he serves as PL/SQL Evangelist for Quest Software and has spent the past two years on his latest creation, Quest Code Tester for Oracle. For information about Quest Software, go to www.quest.com.