From the Editor

Welcome to the new issue of our online magazine, The Oracle Magician! This quarterly newsletter focuses on various “tricks of the trade” in the Oracle world—from DBAs, architects, developers, designers, and report writers.

Thank you to the many notes and comments from readers of my book, The Art & Science of Oracle Performance Tuning. Your notes are most appreciated!

In this issue we discuss the performance ramifications of including function calls in your SQL. When do they hurt performance?

Also in this issue we present an easy way to incorporate bind variables in your SQL*Plus test cases.

As always, we accept ideas or articles from readers that have interesting performance ideas.

Please send all ideas to Editor@OracleMagician.com

Chris Lawson

Editor

How Functions Can Wreck Performance

By Chris Lawson

Overview

Sometimes performance issues come down to one simple thing. Have you ever added a PL/SQL function, only to find that your performance is instantly wrecked?

The Scenario

In our discussion, we’ll focus on the impact of calling functions repeatedly. We’ll build test functions that run quickly for 1 pass, and then see what happens when we include them in certain SQL.

For instance, consider a query that sums up a firm’s sales for the current quarter. If you include a function in the SELECT clause, or as part of some set operation, you can easily end up calling the function millions of times. Even a slight delay for each call can translate into a massive delay.

In this issue, we examine the consequences of adding simple functions to your SQL code.

Table of Contents

From the Editor ................................................................. Page 1
How Functions Can Wreck Performance ......................... Page 1
Update and Notes ............................................................. Page 3
Using Bind Variables in SQL*Plus ................................. Page 4

Continued on page 2
A Simple Function

Let’s first consider a function that just does a simple mathematical calculation. As shown in Figure 1, our function simple does a simple calculation, then exits.

To check our function, let’s try a simple select from Dual:

```
SELECT Simple (100) RESULT from Dual;
```

<table>
<thead>
<tr>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
</tr>
</tbody>
</table>

Naturally, the response time is nearly instantaneous for a single call.

To better estimate the performance impact, let’s see what happens when we call the same function a huge number of times. To do this, we’ll build a PL/SQL procedure.

As shown in Figure 2, we call our procedure with an argument specify-

```
CREATE OR REPLACE FUNCTION SIMPLE (VARX IN NUMBER) RETURN VARCHAR2 IS
  VARY NUMBER;
  BEGIN
    VARY := VARX + 1;
    RETURN VARY;
  END SIMPLE;
/
```

```
CREATE OR REPLACE PROCEDURE CHRIS_TEST (ITER IN NUMBER) AS
  DUMMY VARCHAR2(60);
  BEGIN
    FOR I IN 1 .. ITER LOOP
      SELECT SIMPLE (I) INTO DUMMY FROM DUAL;
    END LOOP;
  END CHRIS_TEST;
/
```

Figure 1. Simple function

Figure 2. PL/SQL procedure to invoke function
Continued from page 2

We've seen that the act of calling a function even a million times doesn't necessarily degrade performance. The real problem is when the function itself contains a SQL statement. It's easy to see why: calling this sort of function 1 million times results in 1 million extra SQL executions!

Setup a new function

Figure 3 shows a new function. This function is also very simple, but this one

Continued on page 4

Figure 3. Revised Function

CREATE OR REPLACE FUNCTION SIMPLE (VARX IN NUMBER)
RETURN CHAR IS
DUM VARCHAR2 (40); BEGIN
SELECT MAX (OBJECT_NAME) INTO DUM FROM CHRIS_TEST2 WHERE OBJECT_ID < VARX;
RETURN DUM;
END SIMPLE;
/

Updates and Notes

If you find these tuning issues interesting, I discuss performance tuning in greater detail in my book, The Art and Science of Oracle Performance Tuning. It is available at most large bookstores, or online at Amazon.com.
performs a short SQL query. The execution plan is very simple—an index read, followed by a table access via `ROWID`. For this case, I use a simple table called `CHRIS_TEST2`, which has an index on the column `OBJECT_ID`. This setup ensures that the query runs very quickly, and only consumes a few logical reads.

Let’s test our new function:

```sql
Select Simple (100) RESULT from Dual;
```

Sql*Plus Autotrace confirms that we only consume 5 logical reads (consistent gets) for a one-time execution of our new function. So our SQL is about as efficient as it can be.

**Performance for many executions**

Let’s now invoke our procedure to call our revised function many times. Since the function must execute a SQL query each time, we would expect much slower response time. Let’s try 100,000 iterations:

```sql
Execute CHRIS_TEST (100000);
```

This time, the response is not nearly so good. The elapsed time is about 2:30 — far worse than the original function. By extrapolation, we can see that 1 million calls would be about 22 minutes. This is about 20x worse than the function without any SQL. Remember also that the SQL statement we used in the function was extremely simple. Just imagine the performance problems if we used a more complicated SQL query.

**Summary**

Adding a PL/SQL function to your code doesn’t always mean that performance will suffer. You can head-off performance issues by asking these questions:

- Given the SQL, will a function be called a huge number of times?
- Does the function itself contain a SQL statement that will end up being executed many times?

By keeping these points in mind, you won’t have to answer the embarrassing question, “How did the performance get wrecked?”

---

**Using Bind Variables In Sql*Plus**

If you want to exactly duplicate a SQL statement, you might need to run the bind-variable version. Here’s how you do it:

1. Define a variable In Sql*Plus
2. Run the query using an anonymous Pl/Sql Block

Let’s see how it works. In this example, we define a variable “a” then run an anonymous PL/SQL block with a query referencing this variable.

```sql
SQL > var a number
SQL >
Begin
:a := 100;
Update TST set CO1 = 1 where CO1 = :a;
end;
/
```