How to write a safe result-cached PL/SQL function

Happy families are all alike; every unhappy family is unhappy in its own way.
Leo Tolstoy, *Anna Karenina*

Leo would probably be surprised to learn that his novel had established a principle:

The Anna Karenina principle describes an endeavor in which a deficiency in any one of a number of factors dooms it to failure. Consequently, a successful endeavor (subject to this principle) is one where every possible deficiency has been avoided.[1]

The author of a result-cached PL/SQL function faces several dooming temptations. A blacklist of safety violations to avoid is no help; it would be infinite. But a compact whitelist of mandates that guarantee safety does the job nicely.

A result-cached PL/SQL function is safe if it always produces the same output for any input that it would produce were it not marked with the *result_cache* keyword. This safety is guaranteed when, and only when, these conditions are met:

- all tables that the function accesses are ordinary, non-*Sys*-owned permanent tables in the same database as the function
- the function's result must be determined only by the vector of input actuals together with the committed content, at the current SCN, of the tables that it references
- when the function is run, it must have no side-effects.

1. The Internet abounds with the references to the Anna Karenina principle and it is often referred to simply as AKP. Apparently it was named after Jared Diamond's book *Guns, Germs & Steel* used it to explain why so few animal species of all those that exist are domesticated.

The safety of caches

It goes without saying that all caching schemes are intended to improve performance. Critically, a cache must have no semantic consequence. It's possible that performance might be harmed, rather than improved, if items are frequently cached on first use but are never again used. But this must be the worst that can happen.

Oracle Database has many caches: the block buffer cache, the library cache, PL/SQL's cache of semantically closed, but actually still open, session cursors, and many more you may never have heard of. The essential cache property is that any item used from some cache must work exactly the same has if it had been used without the cache. We never hear the word *safety* when these caches are discussed. This is because Oracle Database has complete control over every aspect of their use so that the requirement that using the cache has *no semantic consequence* is always met. An item is automatically removed from the cache when Oracle Database determines that the cached value differs from the directly accessed value.

What, then, is special about the PL/SQL function result cache? The specialness is a direct consequence of the flexibility that is the *sine qua non* of procedural programming.

Deterministic functions

An index is just another form of cache. The *no semantic consequence* principle requires that, when an index is used, the outcome must be identical to the outcome that would result if the index were not used. The ability to involve a PL/SQL function in the calculation of values that are stored in an index (brought by Oracle 8*i* Database) is the event that first focused the attention of PL/SQL programmers on the fact that they might break rules – and that should they do this, safety is compromised.

The rules are simply stated:

- The function's return value must be entirely specified by the vector of input actuals – at least for the function's lifetime until recompilation. (Recompilation mandates rebuilding the dependent index.)

- The function must have no side-effects. A side-effect of a function that may or may not be run might or might not happen; if it doesn't always happen, it cannot be relied upon; and if it cannot be relied upon, it's valueless to program it in the first place[2].

These rules imply that all sorts of things that a function might ordinarily do must not be done. Here, too, a blacklist is no help because it would be infinite.

---

2. There's one notable exception to this logic: execution tracing. This is typically done during the development phase of a software system. The simplest example is temporarily to include a "got here" print statement in each subprogram. It's useful for the developer to be able to see that the result of a function was delivered without actually running the function, that is that caching really did happen.

Computer science tells us that a compiler cannot prove that a function meets these rules. Rather, the programmer marks a PL/SQL function with the *deterministic* keyword to promise that the rules are met. Oracle Database enforces the rule that a PL/SQL function that is used in the expression upon which an index is based must be marked with the *deterministic* keyword.

Result-cached PL/SQL functions

How are result-cached PL/SQL functions implemented?

A PL/SQL function invocation is uniquely identified by its owner, its name[3], and the vector of input actuals that are used to invoke it.

If a PL/SQL function is marked with the *result_cache* keyword, then each time it is invoked, an instance-wide cache is interrogated using a key formed from the function's owner and name and the present vector of input actuals. If the entry is found, then the value[4] is returned from the cache; and if it is not, then the function is executed, its return value is stored in the cache using the present key, and then the value is returned.

To make this work, a particular function must always return the same value for the same vector of input actuals. If something happens to change this relationship, then the cached value must be immediately invalidated so that it can be repopulated afresh.

How to ensure that marking a PL/SQL function with the *result_cache* keyword is safe?

The discussion of how to ensure that the *no semantic consequence* requirement is met when a PL/SQL function is marked with the *result_cache* keyword is a simple extension of the corresponding discussion for the use of the *deterministic* keyword.

- Self-evidently, a PL/SQL function that can honestly be marked with the *deterministic* keyword can safely be marked with the *result_cache* keyword.

- More interestingly, a PL/SQL function can safely be marked with the *result_cache* keyword even if its result is determined not only by the vector of input actuals but also by the results of a SQL statement[5]. Notice that this statement makes just this one single relaxation to the "determined only by the vector of input actuals" requirement by allowing SQL into the picture.

How can return values from a function whose results depend on the SQL that it does be safely cached? Oracle Database notices tables that are used during the computation of the result for a particular vector of input actuals and labels the

---

3. When edition-based redefinition is in use, an editioned PL/SQL function is identified by its owner, its name, and the edition in which it exists. The PL/SQL function result cache accommodates editioned functions. But the purpose of this essay is adequately served without referring again to this point.

4. Notice that the term *value* does not imply a scalar. It could, for example, be a collection.

5. While it isn't required that a result-cached PL/SQL function do SQL, this is the overwhelmingly most common reason to use the feature. It's fairly uncommon (but not at all impossible) to find functions that stay entirely within PL/SQL and that take enough time to execute that caching would be beneficial.

corresponding cache entry with the table list. In the parlance of the documentation, this is the list of tables that the result *relies on*. Then, when a change is committed in a table, all cached results that rely on that table are invalidated.

It's obvious that Oracle Database can notice changes only in ordinary tables within the same database as the result-cached PL/SQL function. It cannot, for example, notice changes in a table accessed over a database link or in an external table. Nor, it turns out, does it allow itself to notice changes in tables owned by *Sys*.[6]

Using a result-cached PL/SQL function that *relies on* a table with a VPD policy

It's worth spelling out a consequence of the rule that the results of a function that is to be safely marked with the *result_cache* keyword must be determined only by the vector of input actuals together with the committed content of any tables that it references (assuming that these tables are of the allowed kind). The point is made compellingly by this contrived counter-example:

```
function u1.f return x authid Definer is
  v x;
begin
  select a.c1
  bulk collect into v
  from u1.t a
  where a.Username = Sys_Context('Userenv', 'Session_User')
  order by a.c1;
  return v;
end f;
```

Assume that the column *t.Username* holds only values found in *DBA_Users.Username* and that type *x* is a nested table of the datatype of *t.c1*. Clearly, the function *u1.f* will return a value that depends on the value of the *Session_User* property of the present session – and this is not specified in the vector of input actuals. Therefore, should *u1.f* be marked with the *result_cache* keyword, the results would be nonsense. Safety can be brought, in general in this kind of situation, by an obvious device: add formal parameters to the function that correspond to any session-specific properties that are used, explicitly or implicitly, by the function's implementation. It's sufficient just to change the function's list of formals, thus:

```
function u1.f(u in varchar2) return x result_cache authid Definer is
  ...
```

and then to ensure that whenever it is invoked, the appropriate actual argument is used.

The safe approach, when the table that a result-cached PL/SQL function *relies on* has a VPD policy, is now clear. The author must study the implementation of the table's VPD policy function and ensure that the result-cached PL/SQL function has a formal parameter for each property of the session that the policy uses. These formals will not be referenced in its implementation; but at every call site where the

---

6. At the risk of laboring this point, it's also obvious that Oracle Database cannot notice any property of the function that, in the absence of its doing SQL, would disqualify the honest use of the *deterministic* keyword. If this were possible, then Oracle Database would have been doing it over the years, and the notion that it is the programmer that promises that the marking is honest would never have been be needed.

result-cached PL/SQL function is used, the same expressions must be used for the actual arguments as are used in the VPD policy function's implementation.

Similar reasoning applies when other features that are governed by a policy that is attached to a table are in use. One example is redaction (brought by Oracle Database 12*c*).

Correctness here must be policed by the application's overall architect.

Some uncomfortable emergent behavior

When a PL/SQL function is marked with the *result_cache* keyword, its results will most likely be cached[7] so that, later, a cached value, when it is available, will be used instead of executing the function. In other words, while the *result_cache* keyword isn't a mandate, it does tend to have its intended effect. This means that, sooner or later, you *will* get what you ask for – so you had better ask for it only when you know that the *no semantic consequence* requirement will be met.

Case 1: the SQL done by the result-cached PL/SQL function uses a global temporary table

We are concerned, in this section, with the *"on commit preserve rows"* flavor of a global temporary table (hereinafter GTT). A GTT has database-wide metadata, just like a permanent table, but its rows are private to the session that inserted them. The PL/SQL function result cache mechanisms don't treat a GTT in a special way. You can therefore easily imagine the following scenario. And, if you want, you can demonstrate it for yourself with a simple test that simulates concurrent use, from different sessions, of the same result-cached PL/SQL function.

When a result-cached PL/SQL function is invoked, and the computation of its result causes a GTT to be queried, that result will be cached and the GTT will be added to the *relies on* list for that cached result. The cache that stores the result is instance wide; but the cached result reflects the present session's private content in the GTT. Therefore, all sessions that use the GTT might see nominally session-private data from other concurrent sessions. The invalidation mechanism for the PL/SQL result cache sees a GTT by its database-wide metadata, and sees it, therefore, as an ordinary table. This means that when any session issues *commit* while it has its own unfinalized changes in the GTT, the session-wide cache is invalidated so that every session will see a change in what the function reports about its session-private data. In consequence, a single session has no idea what the result-cached PL/SQL function in question might return.

This behavior was not implemented deliberately to meet a specified functional requirement. Rather, it emerged from the way the mechanism works because defensive code was not written to prevent it.[8] The behavior is clearly of no practical

---

7. The use of the cache is not guaranteed for every invocation of the function because various factors might temporarily bypass its use. Another possibility is that a value that once was cached is aged out because of pressure to cache more recently used values.

8. This outcome is highly undesirable. I therefore filed bug 21907155 against Oracle Database version 12.1.0.2 to request that it somehow be prevented.

use; on the contrary, it can only cause problems. Therefore, the conclusion is obvious: never reference a GTT in the SQL that implements a result-cached PL/SQL function.

You can see this as a special case of the general rule for a function that is to be marked with the *deterministic* keyword: the function's return value must not be affected by anything that is session-specific and cannot be expressed as an input actual to the function.

Case 2: the SQL issued by a result-cached PL/SQL function executes at a historical SCN

Here are two ways to make the SQL issued by a session execute at an earlier SCN than the current one:

- Set *"transaction read only"* before issuing the SQL.
- Execute the *Enable_At_System_Change_Number* procedure in the *DBMS_Flashback* package before issuing the SQL.

There are other ways. In the regime where many concurrent sessions invoke the same result-cached PL/SQL function (and this is the regime that motivated the feature), and one session arranges that the SQL it issues will execute at an earlier SCN than the current one, then the values returned by the function cannot be relied upon. Here too, if you want, you can demonstrate it for yourself with a simple test that uses a couple of concurrent SQL*Plus invocations.

You can see this, too, as a special case of the general rule for a function that is to be marked with the *deterministic* keyword: the function's return value must not by affected by anything that is session-specific.

Nothing that might be on the call stack above the result-cached PL/SQL function when it is invoked must ever cause SQL to execute at an earlier SCN than the current one. This rule must be policed by the application architect.[9]

9. I filed bugs 21905695 and 21907155 against Oracle Database version 12.1.0.2 to track this general issue and suggested that the cache should be bypassed in these cases. This is the present behavior with the SQL result cache when *"transaction read only"* is set. Other solutions might be possible.

Summary

A result-cached PL/SQL function is safe if it always produces the same output for any input that it would produce were it not marked with the *result_cache* keyword. This safety is guaranteed when, and only when, these conditions are met:

- all tables that the function accesses are ordinary, non-*Sys*-owned permanent tables in the same database as the function
- the function's result must be determined only by the vector of input actuals together with the committed content, at the current SCN, of the tables that it references
- when the function is run, it must have no side-effects.[10]

10. The Oracle Database PL/SQL Language Reference, through version 12.1 of this book, does not state these rules loud and clear. I filed documentation bug 21885173 asking that this omission be remedied.