Dependencies and Invalidations

by Arup Nanda

Dependencies between PL/SQL packages can be a perplexing source of application errors. Database administrators and developers unacquainted with how dependencies work can find themselves scratching their heads over sporadic and unrepeatable errors that are seemingly without cause. For example, while executing a procedure in a package you are responsible for, an application throws the following error:

ORA-04068: existing state of package has been discarded

This particular procedure has been executed a million times before. You can swear on your great grandmother’s grave that you haven’t changed the package in a long time, definitely not in the last few seconds and yet the application gets this error, the customer orders fail, and the organization loses money.

Not finding an obvious explanation, you blame a coworker for messing with this package, resulting in a heated argument which you regret instantly. Your manager intervenes and calmly asks you to show her the error. You manually re-execute the procedure in the package and (surprise, surprise) it executes without any error! Amidst that glaring from your accused coworker and the oh-you-need-help look from your manager, you are now thoroughly perplexed as to what happened. Sabotage, Oracle bug, bad karma, or none of the above?

What is this error? And why did it get resolved immediately afterwards without your intervention? The problem is not voodoo, but how PL/SQL code dependencies work. Some object referenced in the package was altered, which caused the package to be invalidated. Then the package revalidated (or, recompiled) later and had to be reinitialized.

Understanding the dependency chain and causes of invalidation is very important when you develop an application. The preceding scenario is an example of how poorly designed application code can cause an outage. This chapter explains dependencies and how you can code to reduce invalidations, thus reducing and possibly eliminating interruptions to the application during runtime.

Dependency Chains

First, you must understand dependency chains and their impact upon PL/SQL code. Suppose you have a table ORDERS and a procedure UPD_QTY that updates the column QUANTITY. Their definitions are as follows (note the dependency of the procedure upon the table; the two objects form a dependency chain):

J. Beresniewicz et al., Expert PL/SQL Practices
© John Beresniewicz et al. 2011
```sql
create or replace procedure upd_qty (  
    p_order_id in orders.order_id%type,  
    p_quantity in orders.quantity%type  
) as  
begin  
    update orders  
    set quantity = p_quantity  
    where order_id = p_order_id;
end;
```

What happens if you drop the column QUANTITY? The procedure will become meaningless, of course, so Oracle rightfully makes it invalid. Later you make another modification, either by adding the column back to the table or by removing that column from the procedure and replacing it with another in such a way that the procedure is now syntactically valid. If you added the column back into the table, you can recompile the procedure by executing:

```sql
SQL> alter procedure upd_qty compile reuse settings;
```

---

**Tip** The RESUSE SETTINGS clause in the recompilation is a handy and powerful tool. Any special compile time parameter such as plsql_optimize_level or warning_level set earlier for the procedure will remain the same after the recompilation. If you didn’t use that clause, the settings for the current sessions would have taken effect. So, unless you want to change those settings, it’s a good idea to use that clause every time.

---

The compilation will check the existence of all the components referenced in the procedure. This establishment of validity occurs only when the procedure is compiled—not at runtime. Why? Imagine the contents of a typical stored code base: thousands of lines long with hundreds of tables, columns, and synonyms, views, sequences, and other stored code trying to validate every object at every execution. The performance will be terrible, making the process infeasible. Therefore, the validation is done at compilation time, which is likely a one-time act. When an object (such as this table) changes, Oracle can’t immediately determine whether the changes will affect the dependent objects (such as this procedure), so it reacts by marking it as invalid. The dependency among objects is clearly shown in the data dictionary view DBA_DEPENDENCIES (or, its counterpart—USER_DEPENDENCIES).

```sql
SQL> desc dba_dependencies  
Name                                      Null?  Type
----------------------------------------------------------------------
OWNER                                    NOT NULL VARCHAR2(30)  
NAME                                      NOT NULL VARCHAR2(30)  
TYPE                                      NOT NULL VARCHAR2(18)  
REFERENCED_OWNER                          NOT NULL VARCHAR2(30)  
```