Choosing the Right Cursor

by Melanie Caffrey

Anyone who has ever written a PL/SQL function or procedure that performs any looping logic knows the pain of choosing just the right type of cursor—or the pain of choosing the wrong type of cursor. Choosing the right type of cursor for the right programmatic situation is what this chapter strives to teach you. Choosing the wrong type of cursor may result in your users, your peers, or your managers (or all of them) losing faith in your ability to serve the technical needs of the business requirements.

Choosing the wrong type of cursor may also lead to a great amount of time debugging system slowdowns in production and, as what you may deem a worst-case scenario, a diminished paycheck. Given the potential pitfalls and consequences, every PL/SQL programmer should strive to choose a type of cursor that works best for each individual technical problem she must solve.

This chapter highlights four types of cursors—not because they are the only four types of cursors available to you, but because they are the most common types of cursors most PL/SQL programmers typically implement. Implementing them correctly, given a particular set of business requirements, is key to having PL/SQL code that is performant and scalable. The four types of cursors discussed in this chapter are:

- Explicit
- Implicit
- Static REF Cursors
- Dynamic REF Cursors

Your goal in writing any PL/SQL program that obtains sets of records for processing programmatically (either individually or in bulk) is to choose a type of cursor that allows you and the Oracle database to obtain the correct answer with the least amount of work. It is really just that simple.

Of course, there are many types of cursors you can use in many situations. But should you? This is the question you need to ask yourself each time you write a PL/SQL program. Choose knowledgeably. Know first what business question you are trying to answer, and then choose the best programmatic tool to answer it quickly and correctly given each individual situation.
Explicit Cursors

The most common type of cursor used in any PL/SQL program, hands down, is the explicit cursor. Everyone learns the explicit cursor upon first learning the PL/SQL language, and most PL/SQL programmers feel instantly drawn to it because they are under the impression that it gives them more programmatic control over their processing. Programmers (at least the newly minted ones) are all about control; their impression is often that if they do not have control over every aspect of their program, then it will not execute correctly.

Explicit cursors are often referred to as open, fetch, close cursors due to their required keywords: OPEN, FETCH, and CLOSE. If you write an explicit cursor, you are required to explicitly open the cursor, fetch from the cursor, and close the cursor. So far, that doesn’t sound too bad and may even give an all-about-control programmer peace of mind. However, it is important to find out not only in which cases such types of cursors may reasonably be used, but also in which cases they may actually be doing more harm than good. Consider the following explicit cursor example in Listing 10-1:

Listing 10-1. An Explicit Cursor Used to Fetch Just One Value

```sql
CREATE FUNCTION f_get_name (ip_emp_id in number ) RETURN VARCHAR2
AS
CURSOR c IS SELECT ename FROM emp WHERE emp_id = f_get_name.ip_emp_id;
lv_ename emp.ename%TYPE;
BEGIN
OPEN c;
FETCH c INTO lv_ename;
CLOSE c;
RETURN lv_ename;
END;
```

Note The naming conventions for parameter names and variable names in this chapter are as follows: ip_ for input parameters, op_ for output parameters, lv_ for local variables, and gv_ for global variables. When input and output parameters are referenced in a function or procedure, they are prepended with the function or procedure name in order to avoid confusion and to illustrate scope. In the previous example, in Listing 10-1, ip_emp_id is referenced as f_get_name.ip_emp_id.

At first glance, this function probably looks like any typical example of a get function (a function whose sole purpose is to obtain, or get, one row or even one value). The business requirement is obviously to grab at least one (and at most one) employee name, ename, from the emp table given an entered employee ID, emp_id. The cursor is opened; the single value is fetched from the open cursor and placed into the variable, lv_ename; the cursor is closed; and finally the function returns the value stored in the lv_ename variable to the calling program. So, why might this type of cursor be an inappropriate choice for this type of business requirement? Because the function, f_get_name, is a bug waiting to happen.