Functions are an integral part of any well-designed PL/SQL application. They are an embodiment of programming best practices, such as code modularization, reuse, and the encapsulation of business or application logic. When used as simple building-blocks for larger programs, they can be an elegant and simple way to extend functionality while reducing code-complexity at minimal cost.

Conversely, when PL/SQL functions are heavily used, particularly in SQL statements, there can be a range of associated costs, most notably for performance. Depending on the nature of the function, simply calling PL/SQL from SQL and/or excessive I/O can degrade performance of even the most trivial of queries.

Given this, I will describe a variety of costs associated with calling PL/SQL functions from SQL and, more importantly, demonstrate a range of techniques that you can use to reduce their impact. By the end of this chapter, you will have a greater awareness of the costs of using PL/SQL functions in your applications. You will therefore be better placed to make good design decisions for your new or refactored applications and improve your performance tuning efforts on critical parts of your system.

---

**Note**  
The examples in this chapter have all been run on an 11.2.0.2 Oracle Enterprise Linux database. The code listings are available for download in their entirety from the Apress web site. In many cases I have reduced the output to the minimum required to convey my message and maintain the flow of the chapter. All examples can be executed in SQL*Plus and the download pack includes any utilities that I’ve used in addition to those provided in the Oracle Database software.

---

**The Cost of Using PL/SQL Functions in SQL**

The SQL and PL/SQL languages are seamlessly integrated. Since Oracle Database 9i, the two languages have even shared the same parser. We use this inter-operability to our advantage in most of the code that we write, such as SQL cursors in PL/SQL programs or PL/SQL functions that are called from SQL. We often do this with little concern for what Oracle Database has to do under the covers to run our programs.
Despite this integration, there is a runtime cost to combining them, in particular when calling PL/SQL from SQL (the focus of this chapter, of course). I’m going to spend the next few pages exploring some of these costs. What is interesting is that Oracle has been working hard to reduce the impact of combining SQL and PL/SQL code over several versions, so the issues I describe are not necessarily in their final state.

My investigation of the costs of using PL/SQL functions in SQL can be broken down into three main areas:

- Performance
- Predictability (which is related to performance)
- Side effects

---

**Note** I do not cover syntactic or operational restrictions (such as DML being prohibited in PL/SQL functions called from SQL). These are not runtime costs as such and are well-documented elsewhere.

---

**Context-Switching**

Context-switching is an almost-irreducible cost of computing that occurs in CPUs, operating systems, and software. The mechanics differ depending on where it is occurring, but it is often a CPU-intensive operation that designers attempt to tune as much as is possible.

When we consider “context switching” in relation to Oracle Database, we specifically refer to the exchange of processing control between the SQL and PL/SQL engines (without necessarily understanding what occurs in such a process). These two engines are separate and distinct but we use them interchangeably. This means that when we call SQL from PL/SQL or vice versa, the calling context needs to store its process state and hand over control and data to its counterpart engine (which may or may not be picking up from an earlier switch). This switching cycle is computationally intensive and can typically be repeated so many times that its effects on response times can become quite noticeable.

For many, the first encounter with context-switching in Oracle Database would have been the introduction of the BULK COLLECT and FORALL PL/SQL features in Oracle 8i, designed specifically to reduce context-switching. These performance features made it possible to bind arrays of data between the SQL and PL/SQL engines in a single context-switch, dramatically reducing the number of context-switches that were generated by the row-by-row PL/SQL techniques that were prevalent at the time. For several major releases, almost all of the performance benefits of these language features could be attributed to the reduction in context-switching they facilitated. (In more recent releases, some further benefits such as redo optimization for bulk INSERT operations can also be realized when using PL/SQL array processing.)

---

**Note** The reduction of context-switching is such an effective performance technique that the Oracle designers added an implicit BULK COLLECT for cursor-for-loops to the PL/SQL optimizing compiler introduced in Oracle Database 10g.