Any developer accessing the Oracle Database uses SQL whether running SQL reports and queries developed personally or provided by another developer. While it may not be true to say that every developer runs and uses PL/SQL when working with the Oracle Database, the number is no doubt very high as any procedural access to the Oracle Database is best done using PL/SQL. As a developer, if you create and work with PL/SQL, you should be debugging and testing the PL/SQL to ensure accurate and efficient code.

In this chapter, you’ll look at PL/SQL unit testing: what is meant by unit testing, why it’s important in the development lifecycle, and how to build tests. There are a few tools and utilities available to help you build tests. It doesn’t matter which you use. What does matter is that you get into the habit of building and running tests against your code. I use Oracle SQL Developer to do any testing and so that is the tool I’ll use to illustrate an approach to building and running PL/SQL unit tests.

Why Test Your Code?

Software and application developers all write code; some write better code than others, but everyone tries to write code that does what it is meant to do and hopes it does that well. That is hardly a vote of confidence for writing good code! Not only is it essential your code is accurate and performs well, it is also important to know that it is robust and able to stand the test of time. This means that the code should perform in the same way, producing the same results over time regardless of changes made to the environment around it.

How can you guarantee this? Perhaps you can’t because you can’t be 100 percent certain of all the changes that may happen in the future, but if you write regression tests that can be run whenever you want or need to, then you can determine whether a program unit performs as expected and continues to do so. The more tests you write, with more variations, the closer you get to ensuring that the program units are robust and still working as intended.

It is crucial that not only do you know that your code is working today, but also that you can prove that it works—and that at any time in the future you can prove the same results. Depending on the requirements of the program unit, your code may have to support a range of scenarios or use cases, and you should be able to test a wide range of use cases. If you can show numerically that you have tested a broad spectrum of these use cases and rerun these test cases, you come closer to empirically proving that the code is robust. If you are able to rerun tests at any time and once again verify the results, you can quickly determine the points of failure and address them.
Without regression tests, you can’t know if your code is working or when it stopped working. Even the best code may be unreliable, but you can’t prove it one way or the other.

What Is Unit Testing?

Often used by the Java and Agile development communities, unit testing is a generic term for writing small repeatable tests where you are testing a “unit of work” or an area of responsibility. Writing unit tests is common in the Java development community where the developers write tests as they write their code. Many in the Java community use JUnit, a development framework for writing unit tests. Most serious Java developers will write unit tests alongside the code they’re developing. Some even write tests before they start to write the code. They know the input types the code is expecting and they know the expected outcomes or expected results, so they can build up the tests before they begin. The general thinking in the Java world is “write a little, test a little, write a little, test a little...” and it is wise advice.

While PL/SQL developers have been developing PL/SQL program units for years, creating paper-based tests or using SQL*Plus, there are very few unit test frameworks available that support building up suites of tests to support and test the code. Some developers do build regression tests but without a framework, building unit tests costs time and it’s hard to justify the extra time to management, so most have not ventured into this territory. When presenting on this topic at conferences and training events, the norm is that one or two people in the audience are writing unit tests for their PL/SQL regression testing.

As PL/SQL developers, you should see unit tests as part of the development cycle, part of the set of deliverables. Write your unit tests while writing the PL/SQL code and store the tests with the program units. You should be able to run and rerun tests at any point to verify the code still works as required and desired.

Debugging or Testing?

What’s the difference between debugging a piece of code and building a unit test for it? This is not an either/or situation: you do need to debug your code to make sure it is doing what it needs to do or to find out why it is not doing what you thought it should do. However, if you want to consider differences, then one difference is the repeatability. If you have written a PL/SQL program unit and want to verify that it’s doing what you want it to do, once it has compiled successfully, you can then instrument the code to verify the status at various points in the code, what the variable values are, which procedures call which procedure, and so on. You can pass in various sets of parameters and verify that the outcome is correct. However, if in a month’s time you want to verify the same, you have to run through the same process. Martin Fowler, author of many books on agile development, refactoring, and extreme programming sums it up perfectly:

> Whenever you are tempted to type something into a print statement or a debugger expression, write it as a test instead.

When Should You Build Tests?

Consider the lifecycle of your PL/SQL code. You have a problem to solve and you know what the code needs to do. You either write the code using paper and pen, text editor, or graphical SQL development tool with a more sophisticated editor. Once you have written the PL/SQL, you compile it to verify that the syntax is correct. If the code compiles successfully, you can then execute it against the schema in the