CHAPTER 4

Test-Driven MySQL Development

Systems integrators must overcome the limitations of the systems they are integrating, but sometimes a system lacks certain functions or commands that are needed for the integration. Oracle recognizes this and includes flexible options in the MySQL server that add new functions and commands. This chapter introduces a key element in generating high-quality extensions to the MySQL system. I discuss software testing and explain some common practices for testing large systems, using specific examples to illustrate the accepted practices of testing the MySQL system.

Background

Why include a chapter about testing so early in the book? You need to learn about the testing capabilities available so that you can plan your own modifications by first planning how to test them. This is the premise of test-driven development: to develop and implement the tests from the requirements, write the code, and then immediately execute the tests. This may sound a tad counterintuitive to someone not familiar with this concept—after all, how do you write tests for code that hasn’t been written?

In the following sections, I’ll clarify by providing some background information about this increasingly popular concept.

Why Test?

When I lecture about software quality issues, I often get asked, “Why test?” Some students want to know how much testing is enough. I offer students who feel testing is largely a waste of time, or that it is highly overrated, the opportunity to complete their class projects using a minimal- or no-testing strategy. The results are often interesting and enlightening.

These same students often discuss how well they code their modules and classes and how careful they are to use good modeling practices. Many use Unified Modeling Language (UML) diagrams to assist their software development. While these are good practices, testing involves a lot more than making sure your source code matches your model. Students who insist that their highly honed coding skills are sufficient often produce project deliverables that have feature and functionality issues.

Although most of these projects do not suffer from fatal errors or crashes (which are often found during development), they often have issues with integration and how the software works. That is, the students fail to ensure that their software works the way the customer expects it to.

---

1Which normally include large group projects beginning with requirements elicitation.
2Especially when I announce the next project, in which teams pass their projects to other teams for software testing. It is amazing how many defects they find in other students’ code while insisting their own coding is superior.
If this has happened to you, you now know the value of software testing. Choosing *which* technique to use *when* is the real nature of the science of software testing.

**Tip**  Professional software testers (sometimes called quality control engineers or quality assurance engineers) have a very different view of software. If you ever work with a professional software tester, spend time learning to understand how they approach software testing. They often have incredible insight—which few developers ever hone successfully—into how software works. Don’t be embarrassed if they break your code—that’s their job, and most are very good at it!

### Testing vs. Debugging

Although they often have the same goal—identifying defects, debugging and testing are not the same. Debugging is an interactive process designed to locate defects in the logic of the source code by exposing the internal workings of the source code. Testing, on the other hand, identifies defects in the execution of the source code without examining its inner workings.

### Test-Driven Development

Test-driven development is often associated with agile programming and often used by organizations that adopt extreme programming (XP) methods. That may sound scary, but here’s a secret about XP: you don’t have to adopt XP to use agile practices!

I often encounter developers who are deeply concerned about adopting agile practices because of all the negative hype tossed about by uninformed people. Those who view traditional software engineering processes as cast in stone think that agile practices are designed to do more with less, and that, therefore, they are inferior. Others believe that agile practices “cut out the clutter” of requirements analysis and design to “focus on the code.” None of this is true.

Agile practices are designed to streamline software development, to re-engage the customer, to produce only *what is needed when* it is needed, and to focus on the job at hand (what the customer wants). The customer, not the process, is the focus of agile methods. Clearly, the emphasis is on analysis and design.

Furthermore, agile practices are designed to be used either as a set or selectively in an application. That is, organizations are encouraged to adopt agile practices as they see fit rather than jumping in with both feet and turning their engineers’ world upside down. That is one reason behind the negative hype—that and the resulting failures reported by organizations that tried to do too much too soon. If you would like to learn more about the debate about agile versus traditional methods, direct your browser to the Agile Alliance web site, [http://www.agilealliance.org](http://www.agilealliance.org).

One profoundly useful agile practice is test-driven development. The philosophy of test-driven development is simple: start with a basic model of the solution, write the test, run the test (which will fail), code the solution, and validate it with the test (when the unaltered test passes). While that sounds really intuitive, it is amazing how complicated it can become. Creating the test before the code sounds backward. How can you test something that doesn’t exist? How can that help?

Developing the test first allows you to focus on the design of your software rather than on the code. I’ll explain a typical test-driven agile development process so that you can see how test-driven development complements the design and actually drives the source code. That sounds weird, but give it a chance and it will make sense.

---

1Yes, this is a bit of a dichotomy, considering agile practices are designed to reduce unnecessary work.