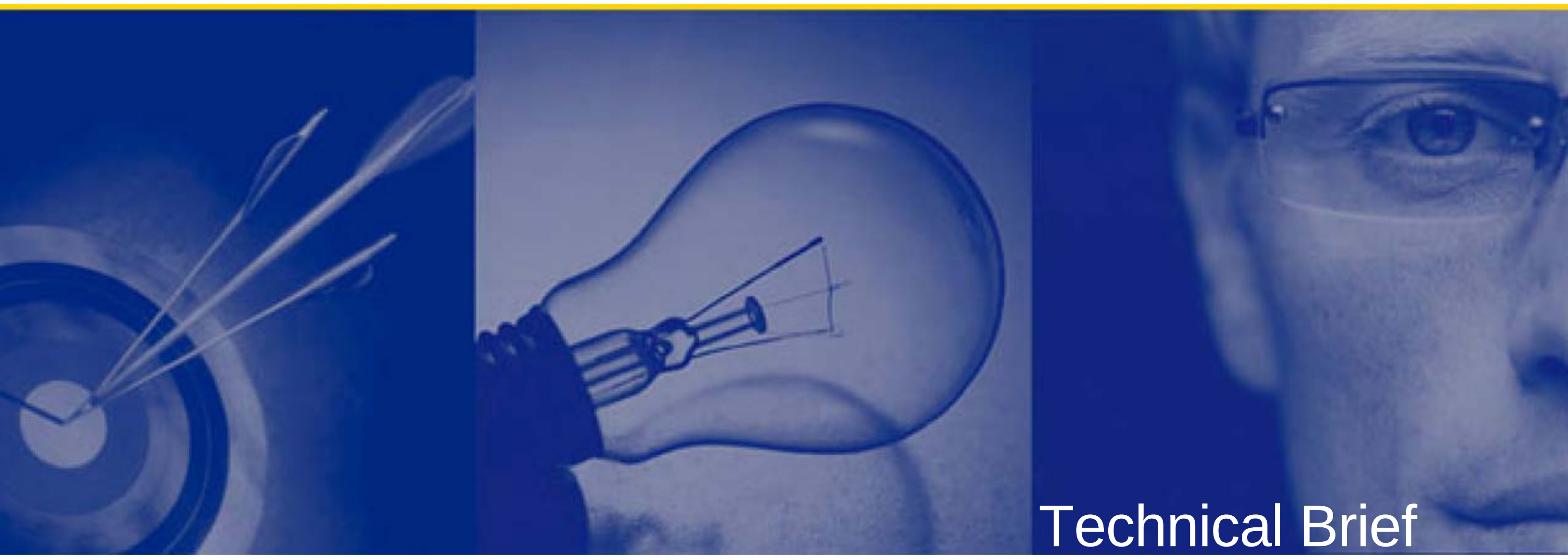
# Code Testing

The Foundation of Successful Applications

*written by*
*Steven Feuerstein,*
*PL/SQL Evangelist,*
*Quest Software, Inc*



World Headquarters
5 Polaris Way
Aliso Viejo, CA 92656
www.quest.com
e-mail: info@quest.com
U.S. and Canada: 949.754.8000

Please refer to our Web site for regional and international office information.

Updated—July 2007

# CONTENTS

## EXECUTIVE SUMMARY

The objective of every software development organization is to build and deploy successful applications. To be successful, an application must meet user functionality requirements, work efficiently to prevent user frustration, contain as few bugs as possible, and be easy to maintain. Many factors contribute to successful applications, yet few are as fundamental and important as comprehensive code testing.

Code testing is the process of examining the individual and usually hidden programs that are the building blocks of an application. It is all too often an ad-hoc and manual process cut short by deadline pressures. The impact of inadequate code testing is far-reaching: acceptance testing by the quality assurance (QA) team and users is much harder and takes much longer; the number of bugs in production increases dramatically, sharply degrading the user experience; and the lack of solid regression testing makes application maintenance a very expensive and slow process over time.

Not only is comprehensive code testing very important, but it is also very challenging. This paper examines the testing challenges faced by Oracle development shops and discusses how Quest Code Tester for Oracle can help overcome those obstacles.

## WHAT IS CODE TESTING?

You should perform a variety of tests on applications, including:

- System or functional tests—performed for an entire system or application, usually by end users and QA teams.
- Stress tests—apply stress in various ways, such as numbers of simultaneous users and volume of data; usually performed by database and system administrators.
- "Monkey" tests—pound away on the keyboard; click the mouse anywhere. What happens? How robust is the software? You can run these tests automatically or by using relatively untrained staff.

Code tests, also known as *unit tests* and *programmer tests*, are processes that test individual units of code (procedures and functions). These tests are the responsibility of the programmers who write the program or other developers on the team.

The following sections focus on the importance and challenges of effective code by programmers.

## Why Is Testing Important?

Everyone says that testing is important; how else can you get rid of bugs in software? But just how important is it? To answer this question, it is necessary to remind ourselves of the role that software plays in our society.

Today, software is the glue and the lubrication of human civilization. From the Internet to embedded systems in medical devices, software runs the computers, large and small, that enable commerce. It delivers human services to those in need and entertains billions.

Given the central importance of software, it's obvious that bugs can have substantially adverse and long-term effects on human life, health and happiness. At a minimum, software bugs are very expensive. The National Institute of Standards and Technology (NIST) published a widely quoted study in 2002[i] estimating that bug-ridden software costs the U.S. economy approximately $60 billion each year, with users absorbing 64 percent of the cost.

The NIST study also revealed some shocking news for programmers: Roughly 80 percent of development costs are allocated to identifying and fixing defects in code. Another impetus for testing has to do with compliance and security. Unless code is thoroughly reviewed and tested, it cannot be trusted.

All kinds of tests are important, but some are more critical than others. Code tests are the lowest level and most fundamental kinds of software testing. Applications are made up of hundreds, even thousands, of individual programs, interacting with each other in complex ways. If these distinct programs are not independently and fully tested, then whenever a bug is encountered (whether in a code test or at a higher level, such as a system test), all of the code must be called into question and investigated.

On the other hand, if code tests have been run on all the individual, low-level programs when bugs appear in system tests, we can concentrate our debugging and repair efforts on higher-level programs and on interactions *between* the programs.

For the remainder of this technical brief, please assume that a reference to *testing* means *code testing* unless it is otherwise noted.

## WHY IS SO LITTLE TESTING DONE?

Given the obvious importance and financial drivers related to thorough testing, you would think that corporations large and small would mandate it. While some industries (such as pharmaceuticals) make product testing a fundamental requirement, corporate and IT management generally turns a blind eye to the realities and requirements of testing when software is the product. They seem to be saying: “We hire professionals to write our software. We assume they test their code.” Unfortunately, denial does not fix bugs.

There are many reasons for the lack of testing:

- **Complexity of code:** Software generally attempts to model or facilitate real-world processes that are very complex. Thus, the software itself is very complex. Unless programmers are extremely disciplined about the way they design and write their code, the programs they produce are very difficult to test.
- **Deadline pressures:** Programmers routinely complain that they are not given enough time to build their software. Compounding this problem is the reality that most developers put off testing as long as possible. After all, who likes to get bad news?
- **Manual test code construction:** Testing intimidates programmers since the volume of code required to test a program can be much larger than the program itself. Who has time to write all that code?
- **Test code maintenance:** Assuming that programmers write comprehensive tests today, they must also *maintain* that test code as the program itself evolves. This requires an enormous amount of discipline and even more time.

## Some Challenges Unique to the PL/SQL World

Beyond the common obstacles to testing, developers in the Oracle PL/SQL world face additional challenges. The logic inside a program can be very complex. That complexity increases by an order of magnitude when the program interacts with an underlying database. Oracle PL/SQL is the quintessential database programming language.

Almost every PL/SQL program reads from and/or writes to database tables. Every table I/O is a “side effect,”—i.e., a dependency or impact—that is not evident through the parameter list of the program. Side effects generally increase testing difficulty. When they involve tables, the situation worsens.

PL/SQL programmers must be able to configure a stable, predictable control set of tables for their tests (both as inputs and as expected outcomes). These tables are often very large, and building them requires a great deal of time and other resources. Testing the content in these tables is also very difficult. The SQL language used to query a table’s content is relatively easy to work with, but the data changed by a program can be hard to analyze.

Compared to the Java and .Net communities, the PL/SQL developer community is relatively small. While many vendors—including, of course Quest Software and Oracle Corporation—provide very useful tools, there is less investment and activity in development of programmer tools than there is for other languages.

The bottom line is that it's not hard at all to understand why PL/SQL developers don't test enough. The obstacles are clear, but so is the cost of not testing. What can be done to help these developers test their code much more thoroughly?

## WHAT DOES THE PL/SQL COMMUNITY NEED?

I have been working with the PL/SQL language since the early 1990s, when it was first released by Oracle. I have published 10 books on it, trained thousands of developers, and written tens of thousands of lines of PL/SQL code. It’s fair to say I have as much familiarity with this language and its adherents as any other human being outside of Redwood Shores, California (the location of Oracle’s headquarters).

My experience leads me to the following conclusions on the state of testing in the PL/SQL world:

- We in the PL/SQL world should be more open to the ideas, methodologies and lessons learned in the other language communities. From extreme programming to test-driven development, there are many exciting developments out there. But for the most part, they pass by the PL/SQL community.
- Developers need tools that will *standardize and automate* the process of unit testing. If all programmers must decide for themselves how to build and run tests, they simply will not test their code.
- In particular, such testing tools must shift most of the burden of writing test code *away* from the programmer. The best standards and guidelines in the world won't help very much if programmers still must write 1,000 lines of test code for a 25-line program.

# CRITICAL FEATURES OF A PL/SQL TESTING TOOL

For a code testing tool to be successful in the Oracle PL/SQL environment (and likely for *any* language), it must:

- Provide a repository for test definitions
- Generate test code
- Automatically verify test results

## Test Definition Repository

One of the biggest problems with ad-hoc, script-based testing is that there is no clear documentation of the tests and how they relate to and cover application requirements.

It simply isn't enough for developers to run a battery of tests and announce to the world that their programs work. The tests need to be repeatable and verifiable. And in this context, the verification question is not “Did my program work?” but rather “Are my test cases comprehensive and responsive to my requirements?”

A critical feature of any robust testing tool is its ability to define tests in a repository which serves as both the launching point for executing your tests as well as a documentation source for your tests.

A test definition repository generally transforms both test data and the test definitions into a valuable corporate asset for the following outcomes:

- Developers and their team leads can see that test cases are complete or at least sufficient.
- Team leads can communicate test status and compliance to higher levels of management.

## Generated Test Code

The testing tool should generate most, if not all, test code, thereby relieving the developer of the burden of writing and maintaining a large volume of code.

It is generally acknowledged within testing circles that you can expect the volume of test code (numbers of lines written or needed) to be significantly larger than that of the unit being tested. Even relatively trivial programs will have many test-case variations.

This reality is one of the most intimidating barriers to writing and running tests. Programmers usually don't have enough time to finish coding their programs, much less constructing even larger scripts to test those programs.

Thus, it is critical to have a testing tool to assist programmers by generating as much of the test code as possible. This generated code might be visible to the programmers—it might even be something they can customize (preferably *before* generation)—but the lines of test code the programmers write should be kept to a minimum and needed only on an exception basis.

Here are some features related to code generation that should be considered for a unit testing tool:

- Template-based generation: The tool should use templates to hold the structure of the test code to be generated. The alternative is to embed that structure within compiled programs, which makes it much harder to debug and change the test code that is generated. Use of templates also means that programmers could be allowed to create or substitute their own templates, thereby making the tool extremely flexible.
- Language consistency: Generated test code should be in the same language as the code being tested. I believe that many developers will not trust test code they cannot see, review and understand. Black-box unit testing does not result in programmer or management confidence.
- Pre-generation customization: It is likely impossible for a tool to handle 100 percent of programmers' needs through predefined templates and generated code. There are simply too many variations of application requirements and coding approaches. Therefore, a robust code testing tool needs to allow programmers to customize the code so that, when generated, it can work precisely as needed.

There are two basic ways to achieve pre-generation customization: insertion of stub calls, and code fragments specified within the tool. With stub calls, the tool generates code that calls programs defined outside the test code itself. These programs are by default empty and do nothing. Programmers can then selectively place testing logic in those stubs and add custom rules and activities without affecting the generation process.

With in-tool code fragments, the testing tool provides opportunities in the interface itself for programmers to deliver code that will then be inserted into the generated test code.

In-tool code fragments have one key advantage over stubs: All the test logic is collected and maintained in one place. These approaches are not, however, incompatible. For maximum flexibility, a testing tool can support both.

## Automatic Verification of Results

Manual verification—i.e., reviewing the outputs from the program and deciding whether or not the program worked—is too slow and error-prone. The tool should figure out automatically whether the test succeeded or failed and where any failures took place.

A tool that is supposed to help you test your code should not leave test verification up to the user. Rather, it should allow users to run tests easily — and quickly. The tool should also provide the user with the overall status of the test (for example, red light meaning failure and green light indicating success). In addition, it should allow the user to drill down to the individual test cases and outcomes within each one to determine exactly where a failure occurred.

## Moving from Criteria to Solutions

Clearly, we face serious challenges in the PL/SQL world when it comes to effectively testing code.

For more than a decade, Quest Software has been deeply involved with the Oracle development process through its many tools—most notably Toad and SQL Navigator—and its array of in-house Oracle domain experts. It was only natural that Quest would also tackle the challenge of building a code testing tool.

So let's now take a look at Quest Code Tester for Oracle and how it helps Oracle development groups comprehensively test their code.

## QUEST CODE TESTER FOR ORACLE

Here's how Quest Code Tester works:

- Rather than writing test code, you can use the Test Builder to simply describe the behavior that you expect to see from your program. In some cases, you will provide unique customization logic, but for the most part, you simply point and click.
- Quest Code Tester for Oracle then stores your test definitions in a repository (Oracle backend) so that you can easily manage them over time. The Test Dashboard function offers a high-level visual perspective on all your test definitions.

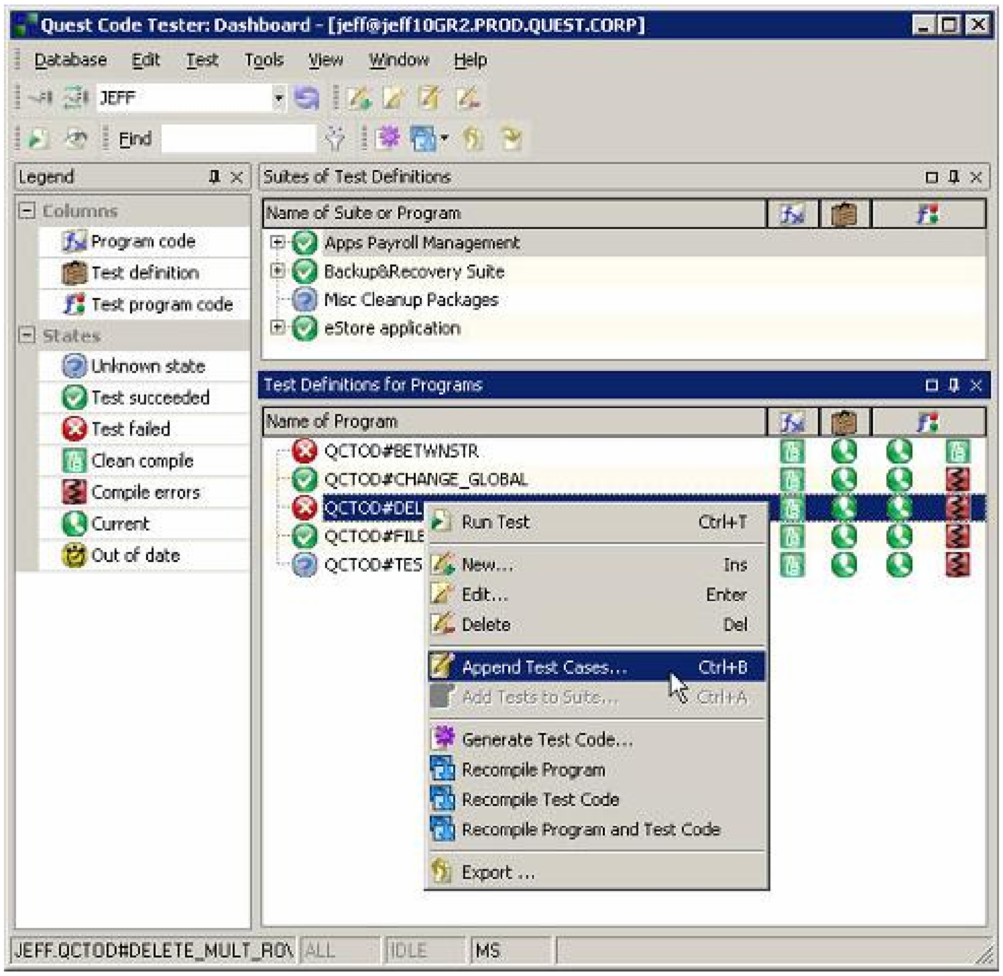
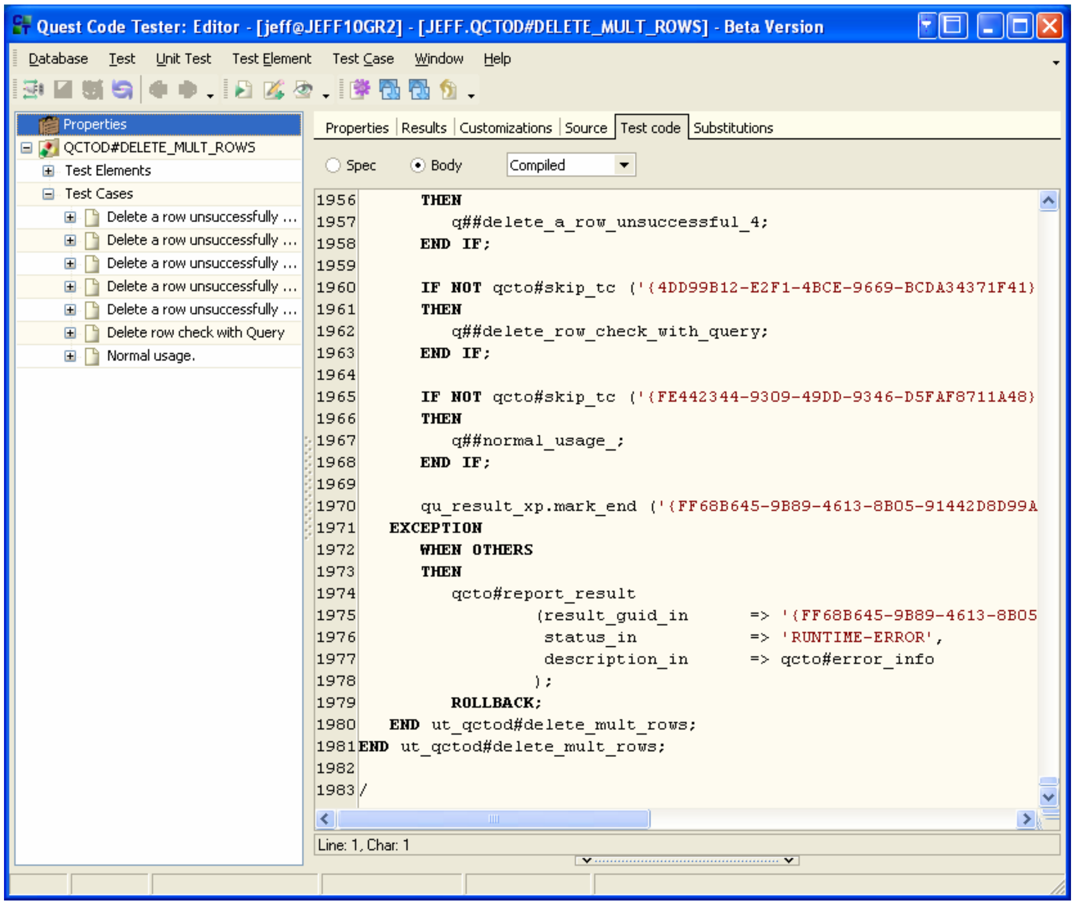


**Figure 1.**

- Quest Code Tester generates a PL/SQL package that exercises your programs according to your definitions—automatically incorporating any customization logic you have provided.

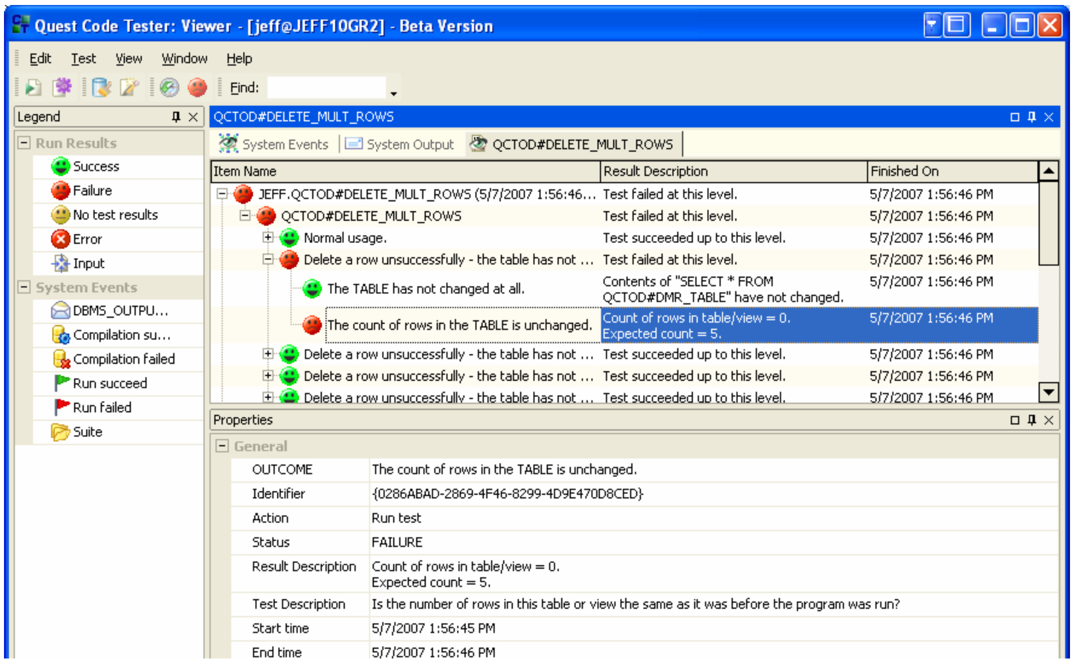

Quest Code Tester: Editor - [jeff@JEFF10GR2] - [JEFF.QCTOD#DELETE_MULT_ROWS] - Beta Version
Database Test Unit Test Test Element Test Case Window Help
Properties
QCTOD#DELETE_MULT_ROWS
Test Elements
Test Cases
Delete a row unsuccessfully ...
Delete a row unsuccessfully ...
Delete a row unsuccessfully ...
Delete a row unsuccessfully ...
Delete a row unsuccessfully ...
Delete row check with Query
Normal usage.
Properties Results Customizations Source Test code Substitutions
Spec Body Compiled
Line: 1, Char: 1

```
        THEN
           q##delete_a_row_unsuccessful_4;
        END IF;

        IF NOT qcto#skip_tc ('{4DD99B12-E2F1-4BCE-9669-BCDA34371F41}
        THEN
           q##delete_row_check_with_query;
        END IF;

        IF NOT qcto#skip_tc ('{FE442344-9309-49DD-9346-D5FAF8711A48}
        THEN
           q##normal_usage_;
        END IF;

        qu_result_xp.mark_end ('{FF68B645-9B89-4613-8B05-91442D8D99A
     EXCEPTION
        WHEN OTHERS
        THEN
           qcto#report_result
                   (result_guid_in      => '{FF68B645-9B89-4613-8B05
                    status_in           => 'RUNTIME-ERROR',
                    description_in      => qcto#error_info
                   );
           ROLLBACK;
     END ut_qctod#delete_mult_rows;
END ut_qctod#delete_mult_rows;

/
```

**Figure 2.**

- You run your test as needed (usually after each change you make to the program and in overnight builds) with the press of a button. Quest Code Tester performs all required initialization and clean up.

- Quest Code Tester automatically verifies the results and displays them in an intuitive “red-light, green-light” Results Viewer.

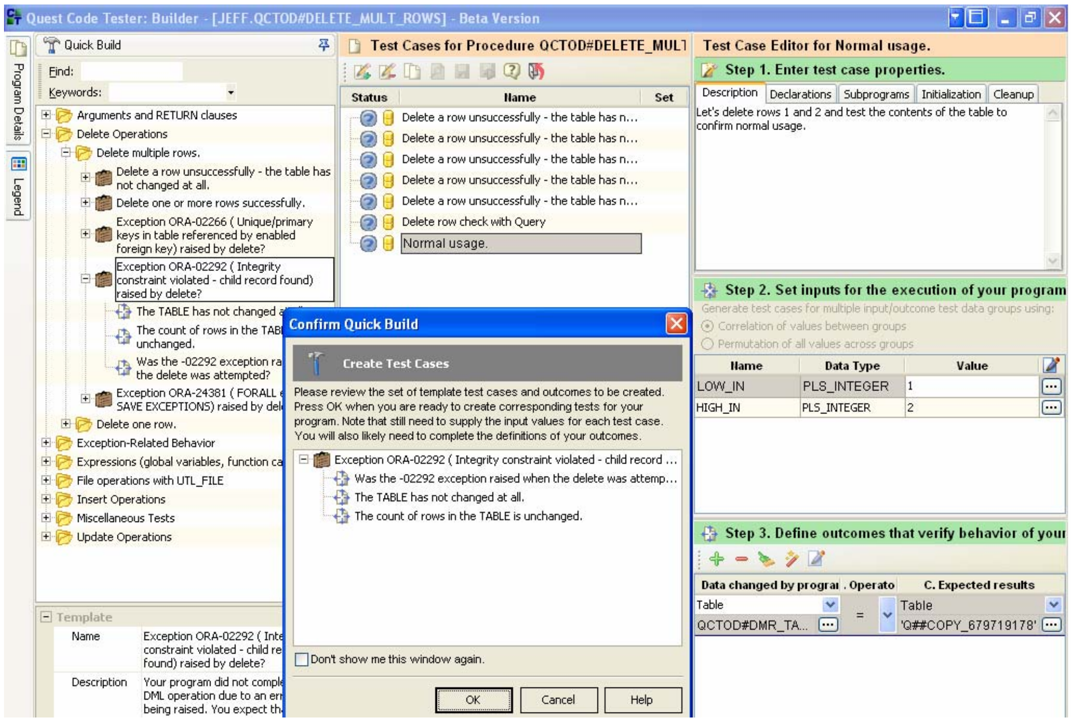


| Item Name | Result Description | Finished On |
|---|---|---|
| JEFF.QCTOD#DELETE_MULT_ROWS (5/7/2007 1:56:46... | Test failed at this level. | 5/7/2007 1:56:46 PM |
| QCTOD#DELETE_MULT_ROWS | Test failed at this level. | 5/7/2007 1:56:46 PM |
| Normal usage. | Test succeeded up to this level. | 5/7/2007 1:56:46 PM |
| Delete a row unsuccessfully - the table has not ... | Test failed at this level. | 5/7/2007 1:56:46 PM |
| The TABLE has not changed at all. | Contents of "SELECT * FROM QCTOD#DMR_TABLE" have not changed. | 5/7/2007 1:56:46 PM |
| The count of rows in the TABLE is unchanged. | Count of rows in table/view = 0. Expected count = 5. | 5/7/2007 1:56:46 PM |
| Delete a row unsuccessfully - the table has not ... | Test succeeded up to this level. | 5/7/2007 1:56:46 PM |
| Delete a row unsuccessfully - the table has not ... | Test succeeded up to this level. | 5/7/2007 1:56:46 PM |
| Delete a row unsuccessfully - the table has not ... | Test succeeded up to this level. | 5/7/2007 1:56:46 PM |

| | |
|---|---|
| OUTCOME | The count of rows in the TABLE is unchanged. |
| Identifier | {0286ABAD-2869-4F46-8299-4D9E470D8CED} |
| Action | Run test |
| Status | FAILURE |
| Result Description | Count of rows in table/view = 0. Expected count = 5. |
| Test Description | Is the number of rows in this table or view the same as it was before the program was run? |
| Start time | 5/7/2007 1:56:45 PM |
| End time | 5/7/2007 1:56:46 PM |

**Figure 3.**

- To accelerate the process of defining your tests, Quest Code Tester offers Quick Build (predefined test templates). With Quick Build, you don’t have to think through, for example, the various tests needed to verify correct inserts. It builds them for you.


Quest Code Tester: Builder - [JEFF.QCTOD#DELETE_MULT_ROWS] - Beta Version
Program Details
Legend
Quick Build
Find:
Keywords:
Template
Test Cases for Procedure QCTOD#DELETE_MULT
Test Case Editor for Normal usage.
Step 1. Enter test case properties.
Description
Declarations
Subprograms
Initialization
Cleanup
Step 2. Set inputs for the execution of your program
Step 3. Define outcomes that verify behavior of your
Confirm Quick Build
Create Test Cases
Don't show me this window again.
OK
Cancel
Help

Arguments and RETURN clauses
Delete Operations
Delete multiple rows.
Delete a row unsuccessfully - the table has not changed at all.
Delete one or more rows successfully.
Exception ORA-02266 ( Unique/primary keys in table referenced by enabled foreign key) raised by delete?
Exception ORA-02292 ( Integrity constraint violated - child record found) raised by delete?
The TABLE has not changed a
The count of rows in the TABL unchanged.
Was the -02292 exception ra the delete was attempted?
Exception ORA-24381 ( FORALL SAVE EXCEPTIONS) raised by del
Delete one row.
Exception-Related Behavior
Expressions (global variables, function ca
File operations with UTL_FILE
Insert Operations
Miscellaneous Tests
Update Operations

| | |
|---|---|
| Name | Exception ORA-02292 ( Inte constraint violated - child re found) raised by delete? |
| Description | Your program did not comple DML operation due to an err being raised. You expect th |

| Status | Name | Set |
|---|---|---|
| | Delete a row unsuccessfully - the table has n... | |
| | Delete a row unsuccessfully - the table has n... | |
| | Delete a row unsuccessfully - the table has n... | |
| | Delete a row unsuccessfully - the table has n... | |
| | Delete a row unsuccessfully - the table has n... | |
| | Delete row check with Query | |
| | Normal usage. | |

Let's delete rows 1 and 2 and test the contents of the table to confirm normal usage.

Generate test cases for multiple input/outcome test data groups using:
( ) Correlation of values between groups
( ) Permutation of all values across groups

| Name | Data Type | Value |
|---|---|---|
| LOW_IN | PLS_INTEGER | 1 |
| HIGH_IN | PLS_INTEGER | 2 |

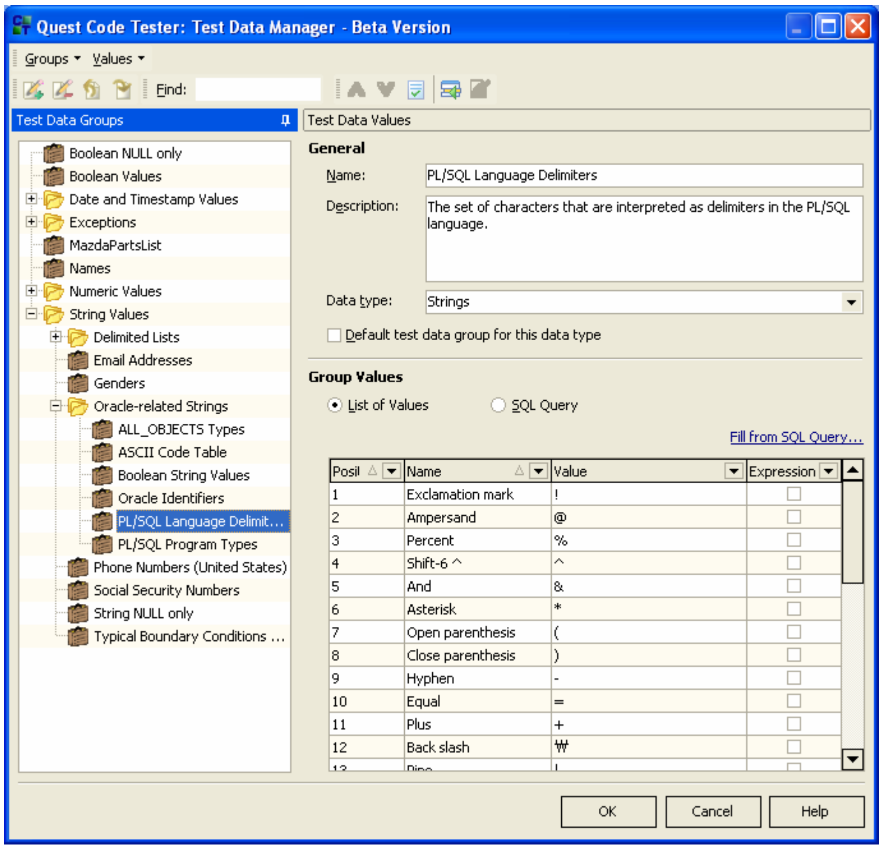
| Data changed by prograı | . Operato | C. Expected results |
|---|---|---|
| Table | | Table |
| QCTOD#DMR_TA... | = | 'Q##COPY_679719178' |

Please review the set of template test cases and outcomes to be created. Press OK when you are ready to create corresponding tests for your program. Note that still need to supply the input values for each test case. You will also likely need to complete the definitions of your outcomes.

Exception ORA-02292 ( Integrity constraint violated - child record ...
Was the -02292 exception raised when the delete was attemp...
The TABLE has not changed at all.
The count of rows in the TABLE is unchanged.

**Figure 4.**

- Quest Code Tester supports efficient definition of boundary condition tests with test data groups (for example, “If I pass null for my first parameter, and anything else for the others, I should always get NULL back”). You can create your own test data groups with often-needed values to further ease your testing efforts.



- Boolean NULL only
- Boolean Values
- Date and Timestamp Values
- Exceptions
- MazdaPartsList
- Names
- Numeric Values
- String Values
  - Delimited Lists
  - Email Addresses
  - Genders
  - Oracle-related Strings
    - ALL_OBJECTS Types
    - ASCII Code Table
    - Boolean String Values
    - Oracle Identifiers
    - PL/SQL Language Delimit...
    - PL/SQL Program Types
  - Phone Numbers (United States)
  - Social Security Numbers
  - String NULL only
  - Typical Boundary Conditions ...

**General**

Name: PL/SQL Language Delimiters

Description: The set of characters that are interpreted as delimiters in the PL/SQL language.

Data type: Strings

☐ Default test data group for this data type

**Group Values**

● List of Values ○ SQL Query

Fill from SQL Query...

| Posi | Name | Value | Expression |
|---|---|---|---|
| 1 | Exclamation mark | ! | ☐ |
| 2 | Ampersand | @ | ☐ |
| 3 | Percent | % | ☐ |
| 4 | Shift-6 ^ | ^ | ☐ |
| 5 | And | & | ☐ |
| 6 | Asterisk | * | ☐ |
| 7 | Open parenthesis | ( | ☐ |
| 8 | Close parenthesis | ) | ☐ |
| 9 | Hyphen | - | ☐ |
| 10 | Equal | = | ☐ |
| 11 | Plus | + | ☐ |
| 12 | Back slash | ₩ | ☐ |
| 13 | Pipe | \| | ☐ |

**Figure 5.**

- You can create back ups of your test definitions and share them among developers and application groups through the import/export facility.

## CONCLUSIONS

Creating Oracle PL/SQL solutions is a challenging process. Faced with tight deadlines, developers are pressured to write code as fast as possible. However, we all know that testing is a vital part of ensuring that projects are delivered on time and with the fewest possible problems.

Until now, PL/SQL technologists have lacked the testing tools required to integrate code testing into their existing development cycles—But with Quest Code Tester for Oracle, the PL/SQL community now has a tool that empowers developers to properly test their PL/SQL code by simply describing their tests. By writing as much as 95 percent of the test code for them, Quest Code Tester for Oracle allows developers to continue concentrating on writing application code while ensuring that their efforts are properly tested.

If you are concerned about the quality of PL/SQL code in your organization and would like additional information, visit Quest's Testing community or download an evaluation copy of Quest Code Tester for Oracle today. Links are located in the *Quest Resources* section of this technical brief.

## QUEST RESOURCES

For more information or to purchase Quest Code Tester for Oracle, visit:

http://www.quest.com/code-tester-for-oracle/

For an evaluation copy, visit:

http://www.quest.com/2_0/registration.aspx?requestdefid=12476

A limited freeware edition is available:

http://www.toadworld.com/Downloads/tabid/60/Default.aspx

Quest Code Tester for Oracle community:

http://unittest.inside.quest.com/

Quest Code Tester for Oracle is a critical component of the Toad Development Suite for Oracle:

http://www.quest.com/Toad-Development-Suite-for-Oracle/

## ABOUT THE AUTHOR

**Steven Feuerstein** is considered one of the world's leading experts on the Oracle PL/SQL language, having written 10 books on PL/SQL, including *Oracle PL/SQL Programming* and *Oracle PL/SQL Best Practices* (all published by O'Reilly Media). Steven has been developing software since 1980, spent five years with Oracle (1987–1992) and serves as PL/SQL Evangelist for Quest Software. In both 2002 and 2006, *Oracle Magazine* named Steven the PL/SQL Developer of the Year. Steven believes that code testing is one of the most critical challenges facing PL/SQL developers and has been leading the development of Quest Code Tester for Oracle to meet that challenge (http://www.quest.com/code-tester-for-oracle). He can be reached at steven.feuerstein@quest.com.

# ABOUT QUEST SOFTWARE, INC.

Quest Software, Inc. delivers innovative products that help organizations get more performance and productivity from their applications, databases and Windows infrastructure. Through a deep expertise in IT operations and a continued focus on what works best, Quest helps more than 50,000 customers worldwide meet higher expectations for enterprise IT. Quest Software is a leading provider of heterogeneous database management solutions for distributed systems, delivering award-winning products to simplify and automate the management of Oracle, SQL Server, DB2, Sybase and MySQL database platforms. Quest Software can be found in offices around the globe and at www.quest.com.

## Contacting Quest Software

| | |
|---|---|
| Phone: | 949.754.8000 (United States and Canada) |
| Email: | info@quest.com |
| Mail: | Quest Software, Inc.<br>World Headquarters<br>5 Polaris Way<br>Aliso Viejo, CA 92656<br>USA |
| Web site | www.quest.com |

Please refer to our Web site for regional and international office information.

## Contacting Quest Support

Quest Support is available to customers who have a trial version of a Quest product or who have purchased a commercial version and have a valid maintenance contract. Quest Support provides around the clock coverage with SupportLink, our web self-service. Visit SupportLink at http://support.quest.com

From SupportLink, you can do the following:

- Quickly find thousands of solutions (Knowledgebase articles/documents).
- Download patches and upgrades.
- Seek help from a Support engineer.
- Log and update your case, and check its status.

View the ***Global Support Guide*** for a detailed explanation of support programs, online services, contact information, and policy and procedures. The guide is available at: http://support.quest.com/pdfs/Global Support Guide.pdf

## NOTE

---

[i] Research Triangle Institute, *The Economic Impacts of Inadequate Infrastructure for Software Testing* (Gaithersburg, MD: National Institute of Standards and Technology, 2002), http://www.nist.gov/director/prog-ofc/report02-3.pdf.