Copyright © 2010

Revision	Description
4/6/2010	Original

SQL-Hero Unit and Performance Testing

Introduction

Why do we need unit testing in the SQL world? The short answer is "for all the same reasons unit testing is beneficial in the non-SQL world." These benefits include but are not limited to: better adaptation to change (catch issues introduced by change early), detection of the unexpected (e.g. someone adds or drops some indexes and thus changes performance characteristics for some object), and catching errors much earlier in the process – errors that might normally be found at run-time instead. These are just a few examples we've observed where SQL-Hero has been brought in and adapted to solve specific needs.

How does SQL-Hero's support for unit and performance testing differ from some competitors? Here's a summary of what you can do with SQL-Hero's implementation:

- Run tests on a scheduled basis or on demand, and have a complete history of test results
- Generate notifications from the test results – notifications that can be delivered by a number of means including e-mail
- Attempt to automatically generate tests against SQL objects that are missing them, using a shared testing policy; optionally try to use previously captured trace information to build tests. This is an important feature since it's not surprising that many developers are time-pressed and allocating dedicated time for building unit tests can be a challenge. Luckily SQL-Hero is very configurable in this department, and the global testing policy configuration file helps define rules for building tests.
- Opt out of testing certain objects, as needed
- Exercise tests when you commit object changes using the SQL-Hero editor tool, finding some kinds of problems immediately
- Define optional performance targets by object
- Compare test results against different types of "expected results": row counts, data checksum, etc. The most basic test is simply to assert that the object does not produce an error when run. Since this can be a common problem during the application development life-cycle, the concept of automatically constructing tests gains traction: so long as we ensure most of the object's contents "run," the results are secondary in many ways.
- View different kinds of reports that relate to testing, such as code coverage reports

Copyright © 2010

And of course, the integration of the various tools is another benefit that's hard to ignore. This paper will cover examples such as being able to set expected test results right from the result set produced by a procedure.

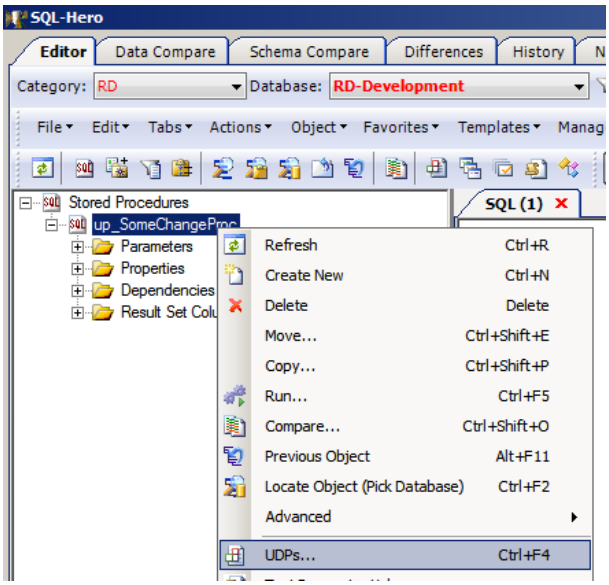
A prerequisite for using unit testing with SQL-Hero is that you have access to an installed instance of the SQL-Hero server components. This is because both test meta-data and test results are stored in the repository. More details on installing SQL-Hero server components can be found in the whitepaper "Installing SQL-Hero."

Defining Tests

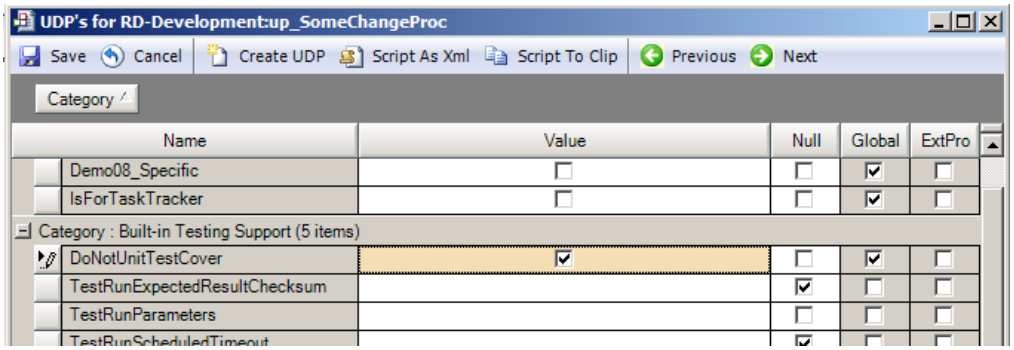
What kinds of objects support testing using SQL-Hero? Stored procedures can be "exercised" by calling them in a transaction that will be rolled back. Tables and views can be "selected from" to ensure they're functional (for a table, this can be useful if it uses computed columns, for example). A future release of SQL-Hero will support testing UDF's and also test via SQL scripts. In all cases, it's important to test objects that are "safe," meaning they can be wrapped in a transaction and have the transaction rolled back either inside or outside of the object, without producing an unexpected data change.

Right away this requirement may disqualify some kinds of objects. For example a procedure that does a complex data load using a cursor loop which performs BEGIN TRAN / COMMIT or ROLLBACK TRAN on its own, may not be appropriate for unit testing.

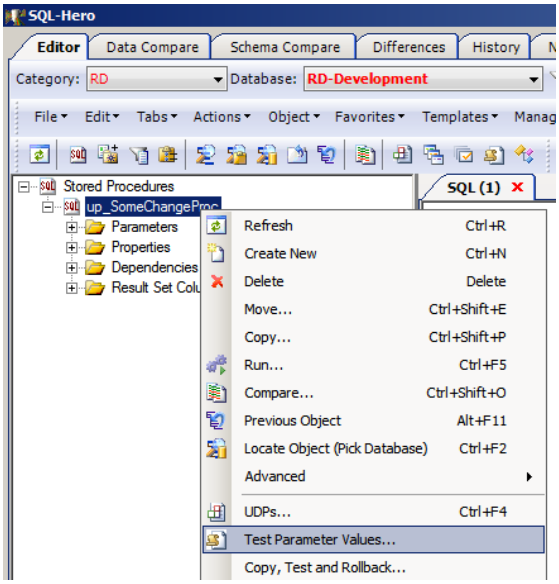
To disable unit testing on an object, one simply needs to flag the object by setting the "DoNotUnitTestCover" user-defined property to True (checked):

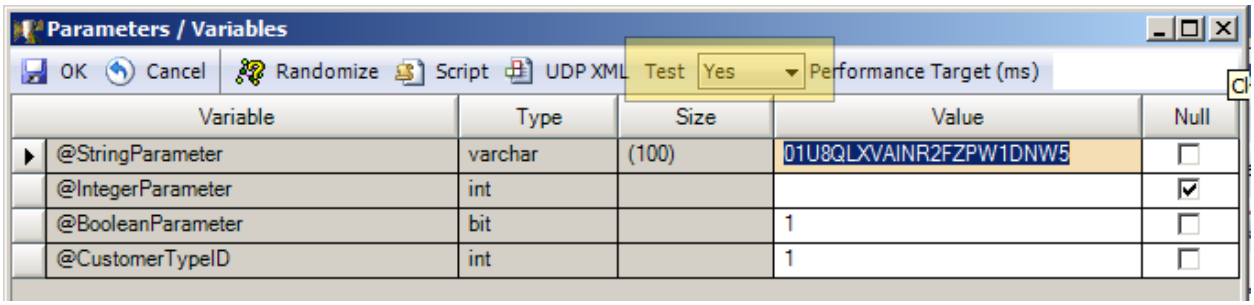


Copyright © 2010



This UDP is "global" and as such applies in all databases, by object name. Another way you can set this property is from the "Test Parameter Values" screen:

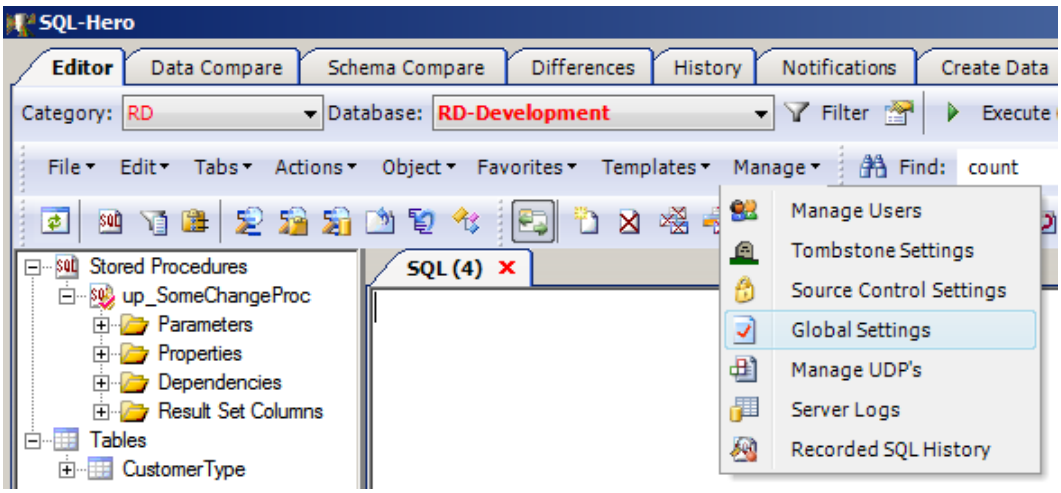


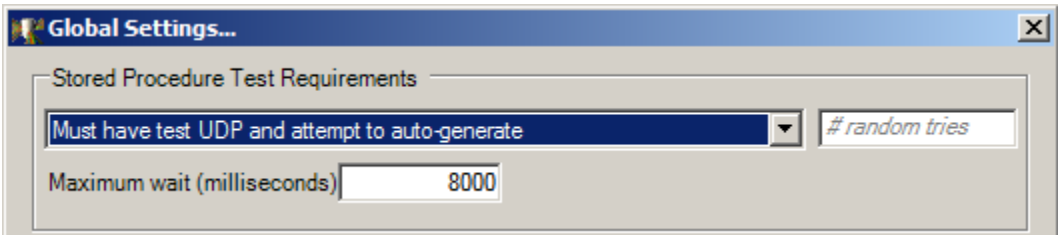


Notice this screen is already showing some parameter values, related to testing. Did someone provide these? No: when the procedure was created by executing it in the SQL-Hero editor, test parameters were arrived at automatically.

One can control this behavior of trying to automatically create tests using the Manage -> Global Settings screen, available from the Editor toolbar menu:

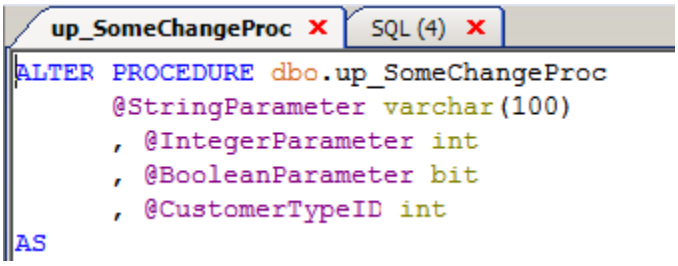
Copyright © 2010





Options here let you opt out entirely from auto-generation of tests, force developers to provide test parameters manually, or attempt to auto-create tests. The default number of random combinations tried is "12" unless specified. The maximum wait is for *all* test creation attempts and prevents you from having to wait more than a fixed timeframe when your object changes are committed in the SQL-Hero editor.

Also, if you follow the standard that foreign keys either end in "ID" or "Key", SQL-Hero is smart enough to properly pick foreign key values that are known to exist. For example, in the procedure below:



If the table CustomerType exists, test values for the @CustomerTypeID parameter are determined based on what exists in the CustomerType table's CustomerTypeID column.

Finer grained control is also possible by modifying the TestingPolicies.xml file, located in your SQL-Hero application directory. If you're using SQL-Hero server components, the server that your client is referencing is used to pull the TestingPolicies.xml file, enabling you to make changes in one single location (i.e. the server copy) and have it apply, enterprise-wide.

Copyright © 2010

The default TestingPolicies.xml file contains some commented out examples of how it can be used to tailor testing behavior.

```
<?xml version="1.0" encoding="utf-8" ?>
<ROOT>
          <!--
                    Insert your customized testing policy configuration here...
                    The version of this file maintained on your SQL-Hero applcation server is taken as the preferred source, else
your local copy
          -->

          <!-- A plug-in gives you maximum control over the test generation process - you can preview, set values based on your
own rules, etc. -->
          <!--
          <PLUGIN>
                    <ASSEMBLY_FILE></ASSEMBLY_FILE>
                    <TYPENAME></TYPENAME>
          </PLUGIN>
          -->

          <!-- Deals with specific objects (procedures)
          <OBJECT>
                    <NAME></NAME>
                    // OR - use a name regular expression to match against
                    <PATTERN></PATTERN>

                    // For the given name/pattern, can be excluded entirely from testing (will be marked as do not test)
                    <IS_EXCLUDED>True</IS_EXCLUDED>
          </OBJECT>
          -->

          <!-- Deals with specific parameter names
          <PARAMETER>
                    <NAME></NAME>
                    // OR - use a name regular expression to match against
                    <PATTERN></PATTERN>
                    // OR - match a specific UDP value
                    <UDP><NAME>UDPName</NAME><VALUE>True</VALUE></UDP>

                    // Can optionally associate only to a specific parent object as well
                    <PARENT_NAME></PARENT_NAME>
                    // OR - use a name regular expression to match parent object name against
                    <PARENT_NAME_PATTERN></PARENT_NAME_PATTERN>

                    // Can optionally set the maximum probability of using Null
                    <NULL_PERCENTAGE></NULL_PERCENTAGE>

                    // Use a constant value always
                    <VALUE></VALUE>
                    // OR - use a random value within a range
                    <VALUE_START></VALUE_START>
                    <VALUE_END></VALUE_END>
                    // OR - use a SQL query to get a range of values from which one is randomly picked
                    <SQL_LOOKUP></SQL_LOOKUP>
          </PARAMETER>
          -->

          <!-- Deals with all parameters of a particular data type (parameter name matching performed first)
          <DATATYPE>
                    <TYPENAME></TYPENAME>

                    // Can optionally set the maximum probability of using Null
                    <NULL_PERCENTAGE></NULL_PERCENTAGE>

                    // Can optionally associate only to a specific parent object as well
                    <PARENT_NAME></PARENT_NAME>
                    // OR - use a name regular expression to match parent object name against
                    <PARENT_NAME_PATTERN></PARENT_NAME_PATTERN>

                    // Use a constant value always
                    <VALUE></VALUE>
                    // OR - use a random value within a range
                    <VALUE_START></VALUE_START>
                    <VALUE_END></VALUE_END>
                    // OR - use a SQL query to get a range of values from which one is randomly picked
                    <SQL_LOOKUP></SQL_LOOKUP>
          </DATATYPE>
          -->

          <!--
          Example:  a proc parameter that ends in areaid, populate using a select from area table that is filtered
          <PARAMETER>
                    <PATTERN>AreaID$</PATTERN>
                    <SQL_LOOKUP>SELECT AreaID FROM Area a WHERE a.AreaAbbr LIKE 'A%'</SQL_LOOKUP>
          </PARAMETER>

          Example:  a sample plug-in, put in the program files\sqlhero\plugins directory
          <PLUGIN>
                    <ASSEMBLY_FILE>SampleTestingPlugIn.dll</ASSEMBLY_FILE>
                    <TYPENAME>SampleTestingPlugIn.MyTestPolicy</TYPENAME>
          </PLUGIN>
```

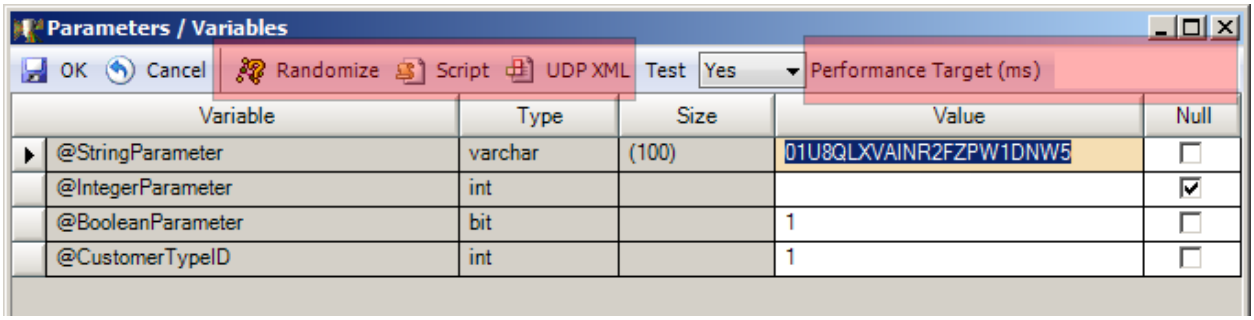
```
        Example:  all objects in the Debug schema would be excluded from testing
        <OBJECT>
                <PATTERN>^\[?Debug\]?\.</PATTERN>
                <IS_EXCLUDED>True</IS_EXCLUDED>
        </OBJECT>
        -->

</ROOT>
```

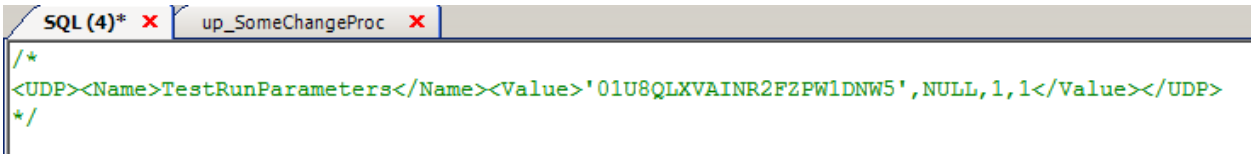
As you can see, you can control what objects will be tested, and even what values can be used for certain parameters.

You may be wondering what's the difference between the "TestRunParameters" and "NullRunParameters" user-defined properties. NullRunParameters is used mainly to support certain code generation tasks, but by having it, it too can serve as a unit test as well. NullRunParameters are generally intended to run a procedure and produce its expected result set (empty or not), without creating side-effects. This setting is global in the sense it will apply to all databases in which an object of that name appears. TestRunParameters on the other hand are database-specific, since the expected results may vary depending on the database.

Another way that you can set test parameters for an object is using the "Test Parameter Values" screen, as illustrated here:



This screen lets you provide explicit values for the parameters, or attempt to arrive at values using a randomization strategy ( ). You can also script out the SQL that would run the current unit test ( ). You can also script out some XML that, if executed in the SQL-Hero editor, would set the testing user-defined properties ( ). From the example above, this would produce:



You could execute this comment in a different database, for example, to set the same TestRunParameters UDP in that database also.

The "Script" command ( ) is useful for a number of reasons, one of which being that you can get the result set that the procedure would produce from the test and then define that as the "expected result" in terms of data, when the test is exercised. Doing this is rather easy: click the "Script" button, resulting in text such as this appearing:

Copyright © 2010

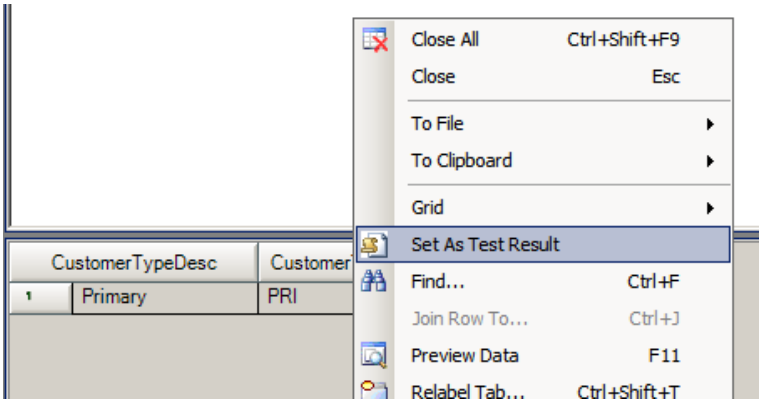
RD-Development:up_SomeChangeProc | RD-Development:SQL (9)* | RD-QA:SQL

```
BEGIN TRAN; SET ARITHABORT ON;
EXECUTE [dbo].[up_SomeChangeProc] '0HTGE14QWE2',154119276,NULL,1;
IF @@TRANCOUNT > 0 ROLLBACK TRAN;
```

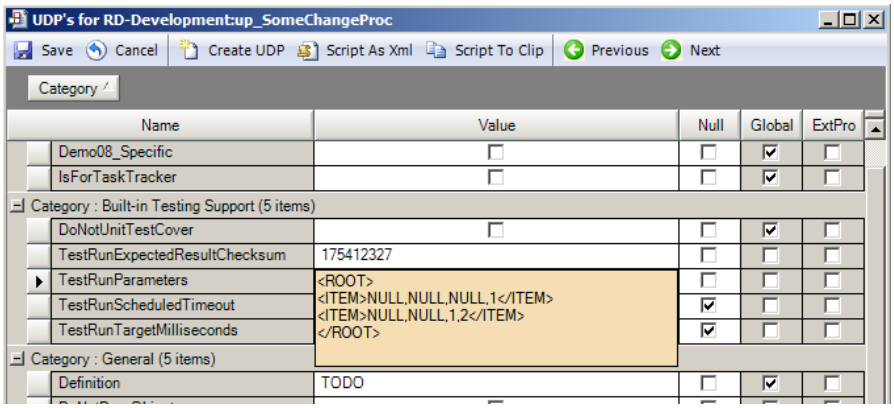
Executing this script may result in a grid such as this:

	CustomerTypeDesc	CustomerTypeCode
1	Primary	PRI

One of the context menu (right-click) options available on the result set is to set it as the expected result for the current test:



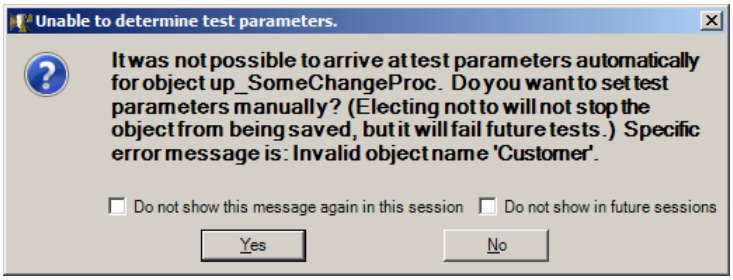
Note that it is possible to have more than one test specified for an object, but to do this, you cannot use the Parameters / Variables screen. Instead you need to modify the TestRunParameter UDP directly, as illustrated here:



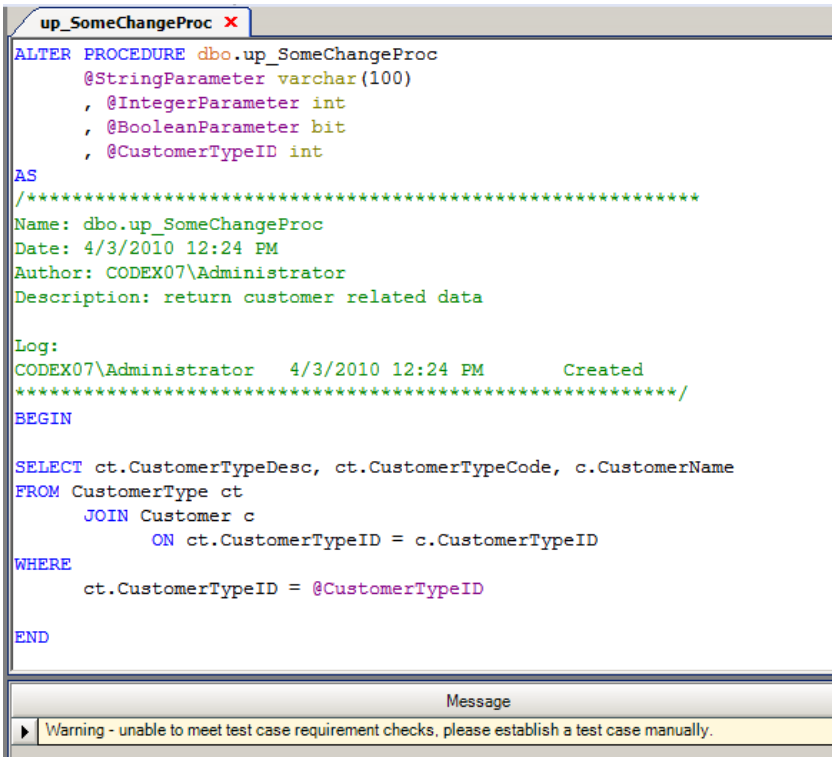
Copyright © 2010

Running Tests

There are a number of ways that tests become a part of different processes in SQL-Hero. For example, if a test is present on a stored procedure, it is exercised whenever you try to commit a change for it when using the SQL-Hero editor. If you've introduced a problem, for example, you can receive a message like this:



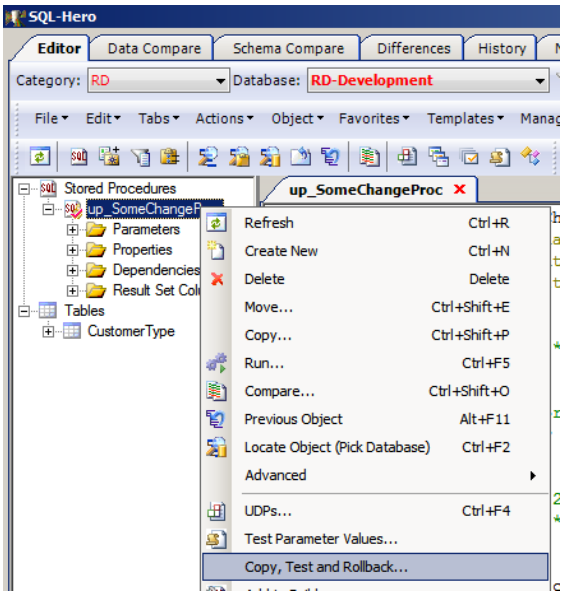
This is what we received when trying to commit the change shown below, in a database that does not yet have the Customer table:



Note the message shown at the bottom: this is what we received when we said "No" to the above dialog box when asked about setting up a test parameter manually.

Another place where tests are evaluated is when you use the "Test, Copy and Rollback" feature in the SQL-Hero editor:

Copyright © 2010



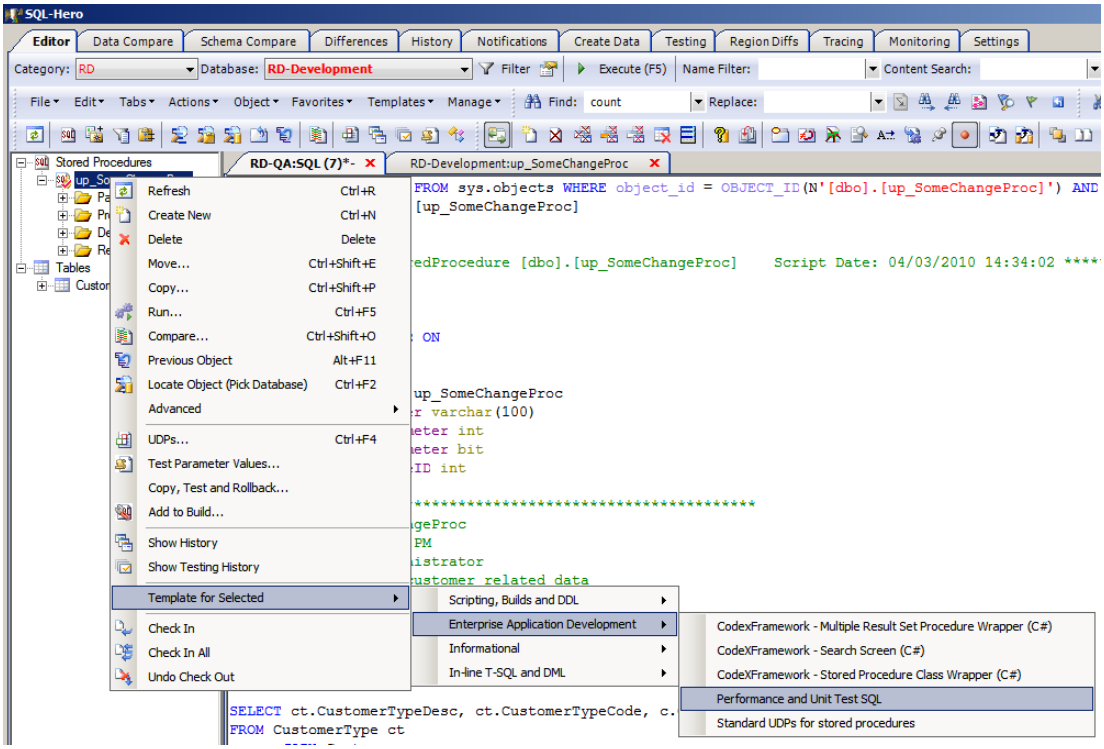
Picking this, you're prompted for a database in which to copy the object, run its unit test, and roll back the schema change which copied it. This can yield useful information such as seen here:

	ObjectName	ObjType	Duration	TestType	PercentOfTarget	ErrMessage	ErrNumber	SQLParms
1	[dbo].[up_SomeChangeProc]	P	26	2	*NULL*	Invalid object name 'CustomerType'.	208	'0HTGE14Q\WE2',154119276,NULL,1

In this case we tried to copy up_SomeChangeProc to a database that did not have the CustomerType table. After this has been run, up_SomeChangeProc was not moved or altered in the target database.

Testing is built to leverage SQL-Hero's powerful template engine. As such, you can run tests directly against one or more objects, using the "Performance and Unit Test SQL" template, as seen here:

Copyright © 2010



Selecting this prompts you with template parameters:

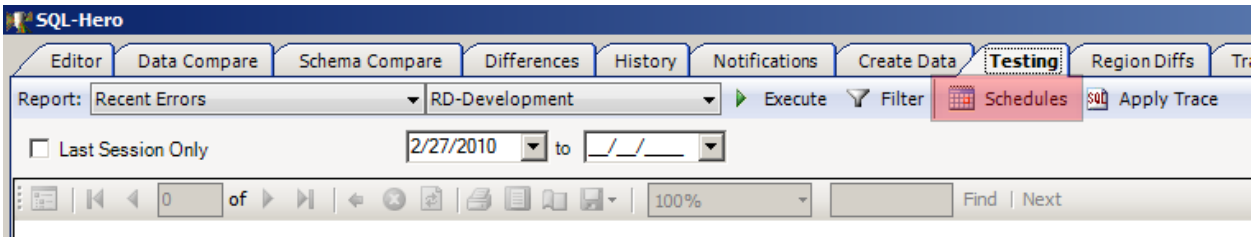
Using the defaults (except for checking the “Select Results” checkbox) yields a SQL script that when run, actually invokes the test or tests. Running the script could result in output that looks like this:

	ObjectName	ObjType	Duration	TestType	PercentOfTarget	ErrMessage	ErrNumber	SQLParms	EventDate	RowNumber
1	[dbo].[up_SomeChangeProc]	P	0	2	*NULL*	*NULL*	*NULL*	'0HTGE14QWE2',154119276,NULL,1	4/3/2010 21:49:03.010	1

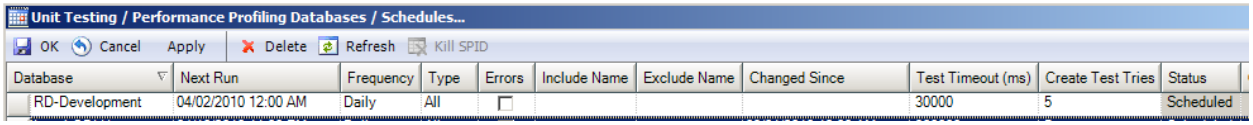
In this example, we successfully executed the test and the test took less than 1 millisecond (seen in the “Duration” column).

Another common way that tests can be run is on a schedule. Testing schedules can be configured on the Testing tool:

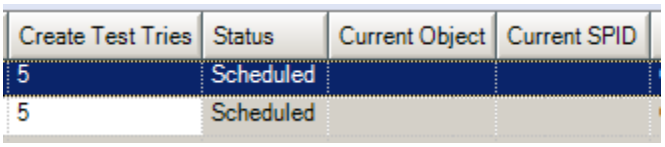
Copyright © 2010



In the example below, we're going to run all types of testing on the RD-Development database, starting next at midnight on 4/2/2010. It will run daily, and will include results on both successful completion and errors (if "Errors" was checked, only errors would be recorded). Optional name filters can be applied, and an optional filter can be used which causes only objects changed since a certain date to be included in the test. A global timeout can also be specified which is applied to every test case that does not have an overridden timeout period (this can be set using the "TestRunScheduledTimeout" user-defined property). In addition, this scheduled test will include a "creation phase" where objects that do not already have tests will go through the process of having tests automatically generated, where possible – here, at most 5 parameter combinations will be tried.



The "Kill SPID" button is only enabled when the scheduled test is actually running and has reached the stage where tests are being run. The object name currently being tested is shown:

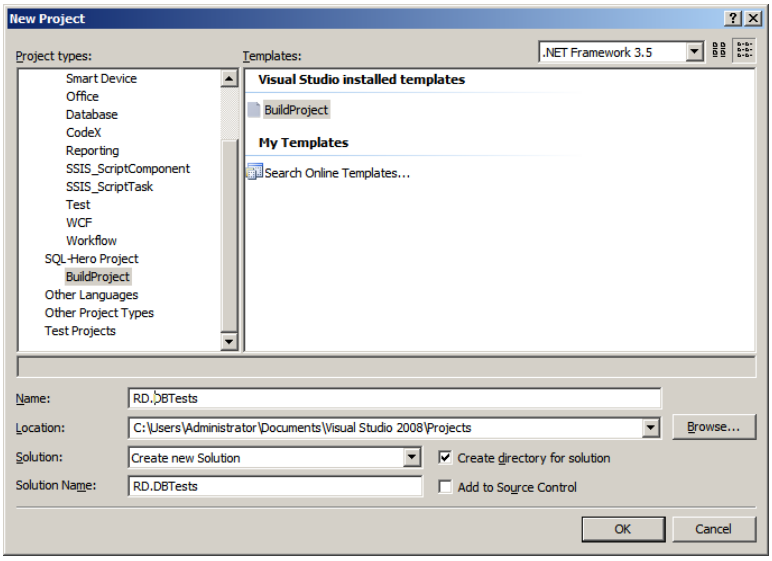


"Kill SPID" becomes useful if you believe the test is hung, taking too long, or otherwise causing a problem.

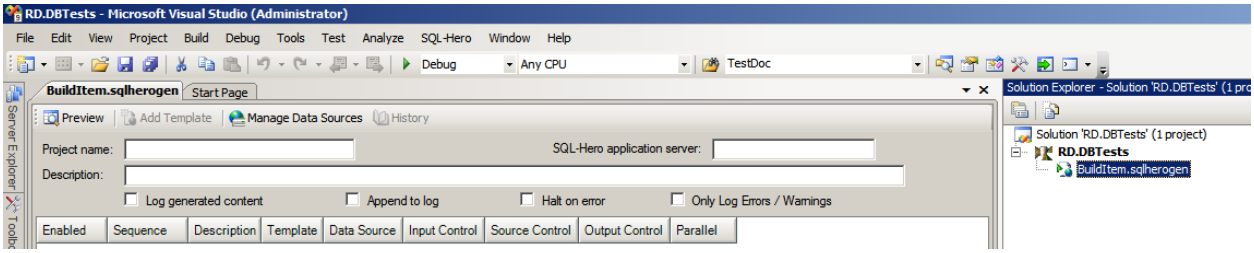
The results of scheduled testing are often effectively reported using SQL-Hero Notifications. Details on setting up notifications are covered in a different whitepaper. Another way to get at the results is using the reporting capabilities of the Testing tool, covered later.

Yet another way in which you can exercise unit tests is using the sqlheroproj project type, under Visual Studio (requires Developer edition or higher of SQL-Hero). In the example below, we're creating a new SQL-Hero project in Visual Studio 2008 that will be specifically for unit testing.

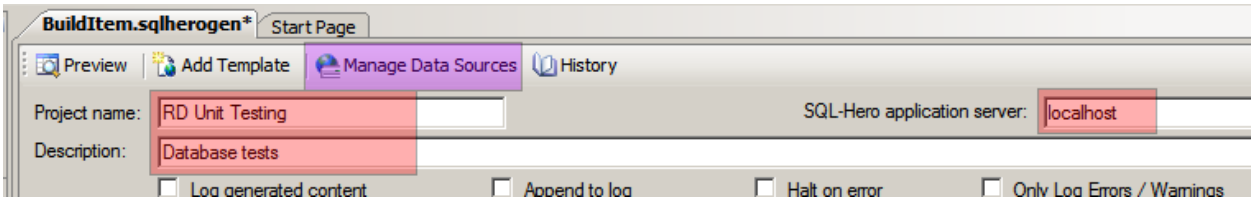
Copyright © 2010



The next step is to open the designer for the BuildItem.sqlherogen file:

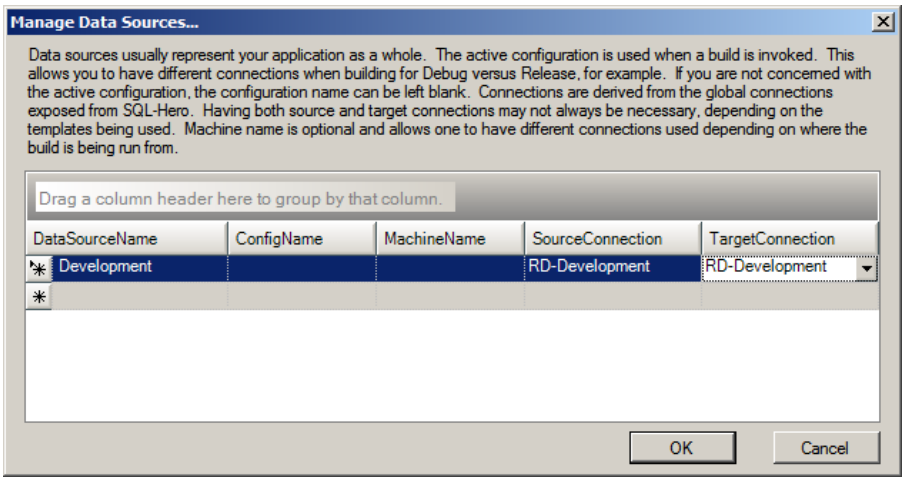


Next, fill in the fields as appropriate. Here we're using "localhost" which assumes we're running this from the same box on which the SQL-Hero server components are installed. After that, click on "Manage Data Sources" to add a new connection for this project.



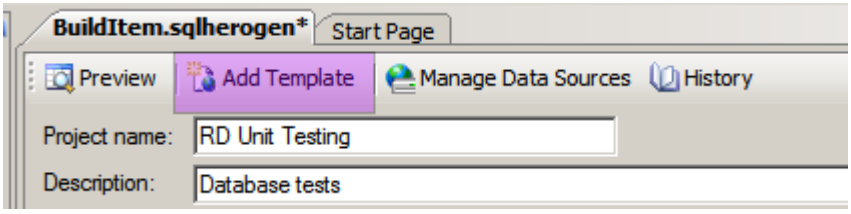
We've added a single connection which will show up as "Development" for this project:

Copyright © 2010

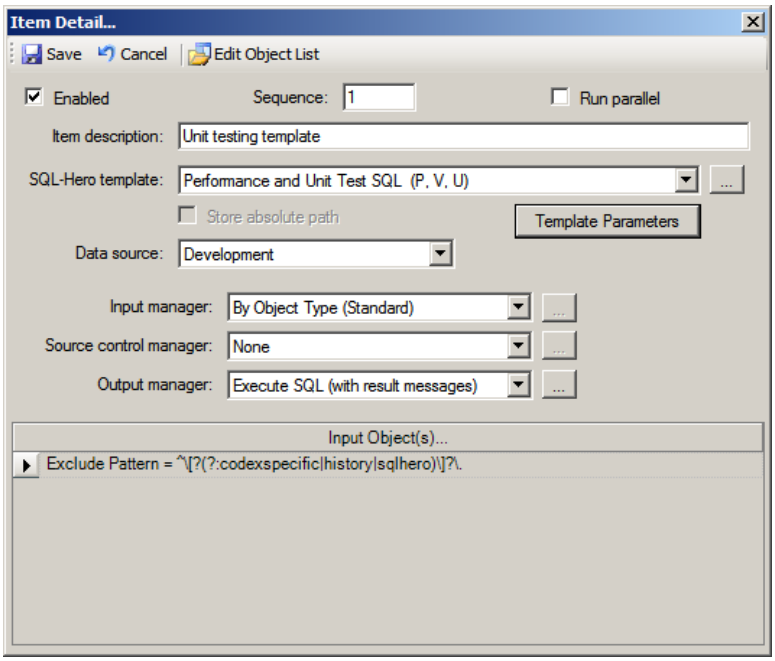


The SourceConnection and TargetConnection must be global connections published in SQL-Hero. You can optionally fine-tune connections based on the current Visual Studio configuration and machine on which the project is being run.

Next, we're going to add a new template to this project:



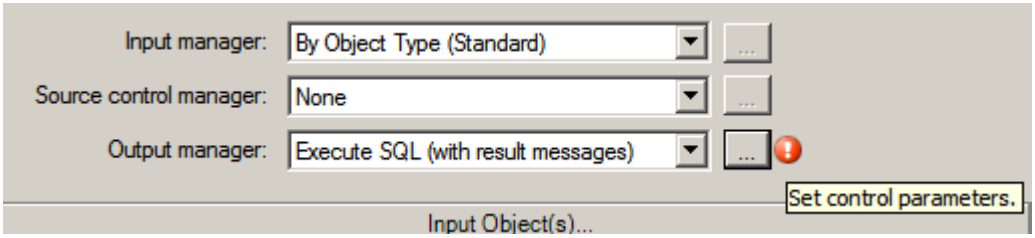
We can configure this template as seen here:



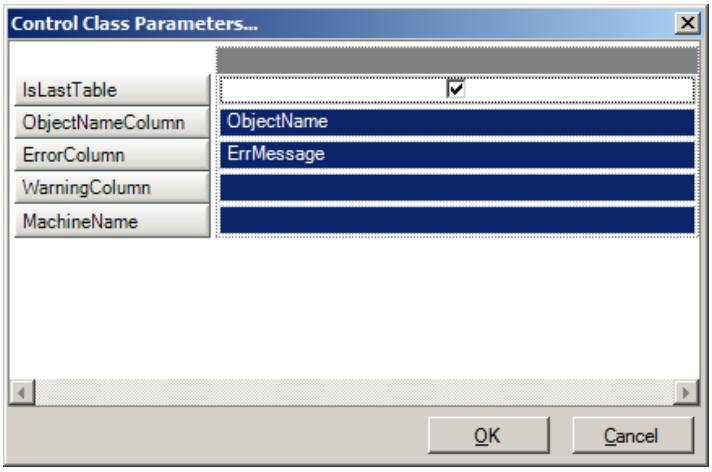
Copyright © 2010

You must also click on the Template Parameters button since we need to be sure the testing script in this case returns a result set:

If you click on Save too soon, you'll see something like this:

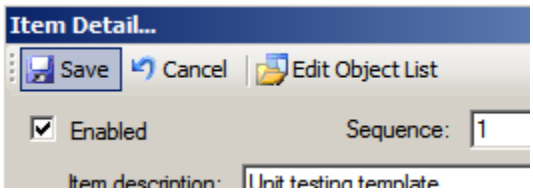


This is telling us we need to set additional output manager parameters as well, by clicking on the "…" button:



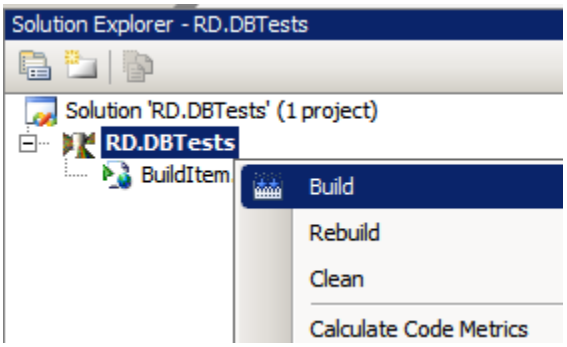
Toggling the IsLastTable checkbox to "checked" also sets the ObjectNameColumn and ErrorColumn to defaults which are applicable here. Continue by clicking OK on this screen.

Now we're done with setting up this template, so click Save:

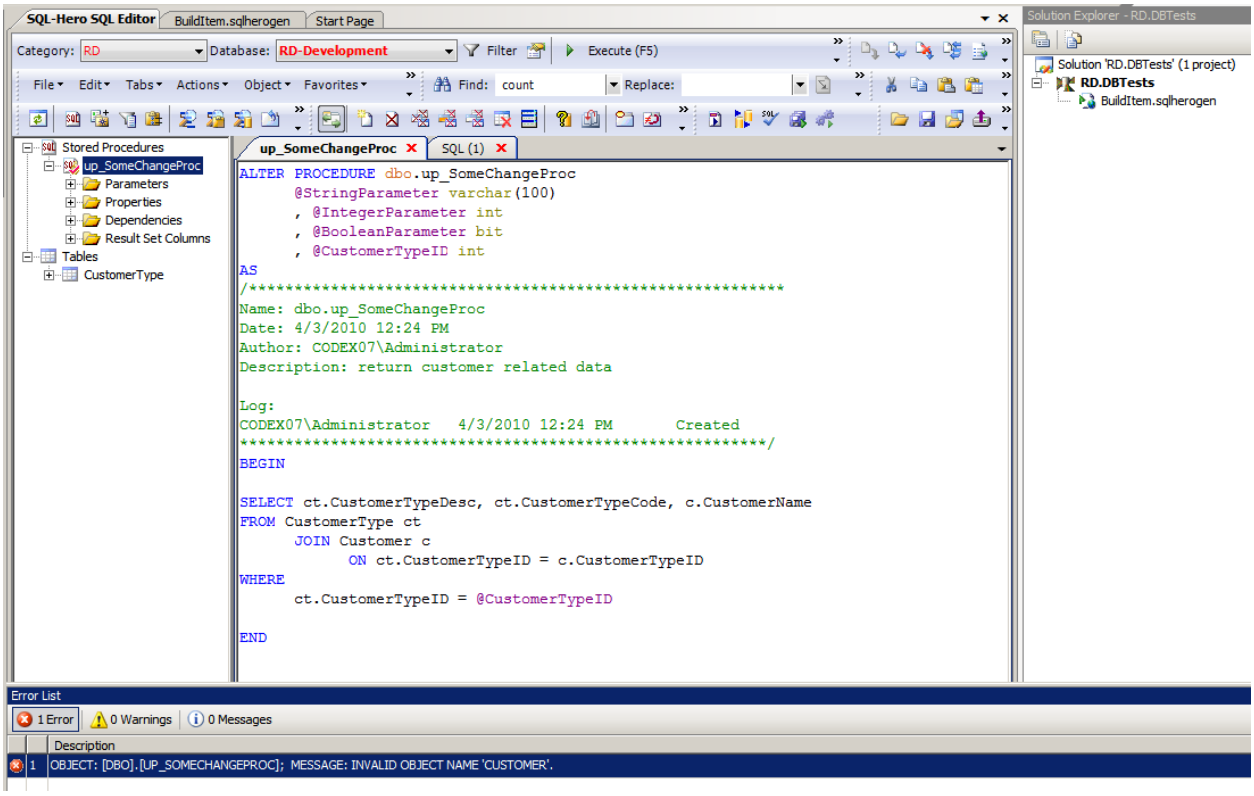


Copyright © 2010

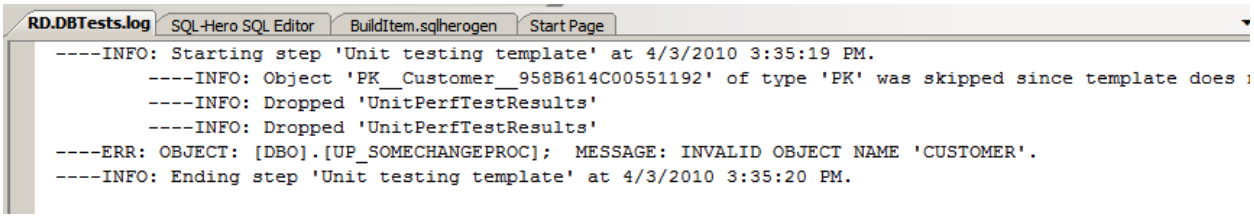
We're now ready to try running the test project, by building it:



Notice that in this case, we ended with a build failure and an error is showing, right within Visual Studio:

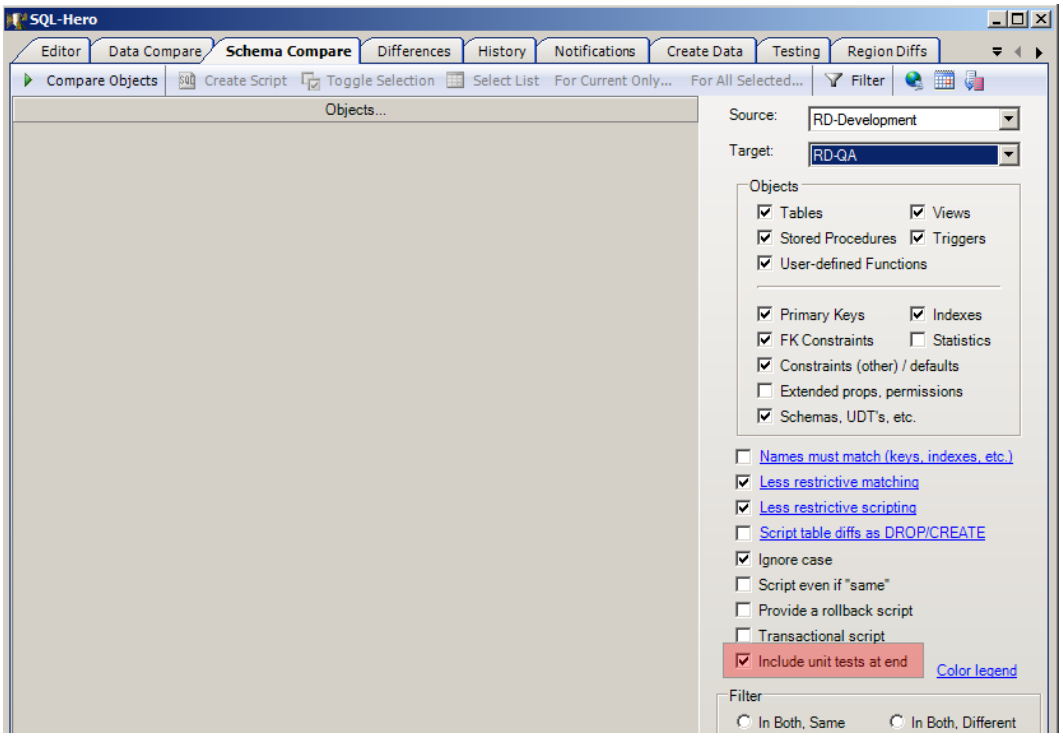


The error in this case is from a failed unit test. Here it makes mention of an invalid object "Customer" and as you can see, there is no Customer table listed in the development database yet. The test has successfully alerted us to a problem. Double-clicking on the error row brings up the detailed error log from the unit test:



Copyright © 2010

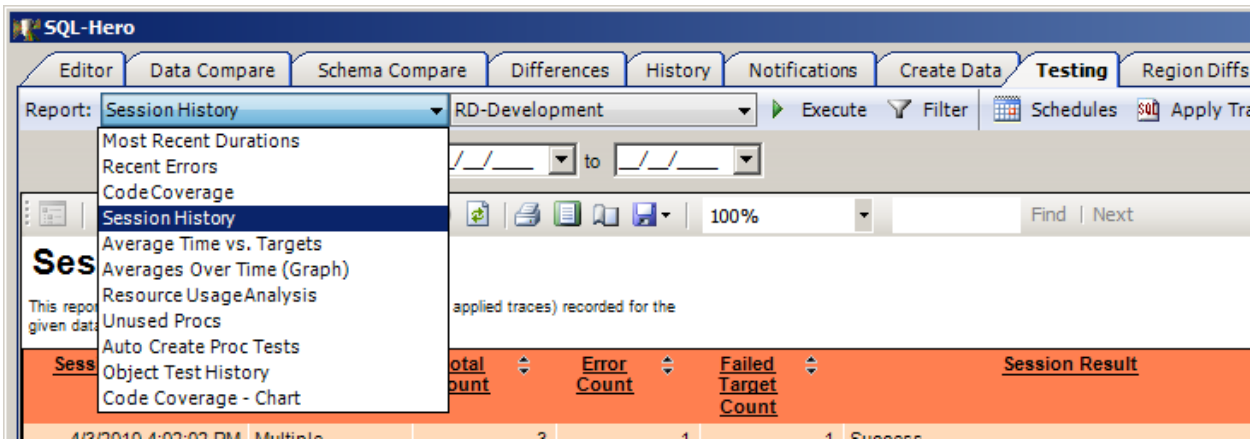
Yet another place that unit testing can be applied is during the scripting of schema changes from one database to another, using the Schema Compare tool:



Here we produce unit testing script (either in the main script or in a different script if transactional scripting is selected) that could be used to test the scripted objects in the target environment. This assumes the test parameters will be valid there as well, so you do need to consider test results here, case by case. However, it is generally a great way to validate a successful migration of changes.

Test Reports

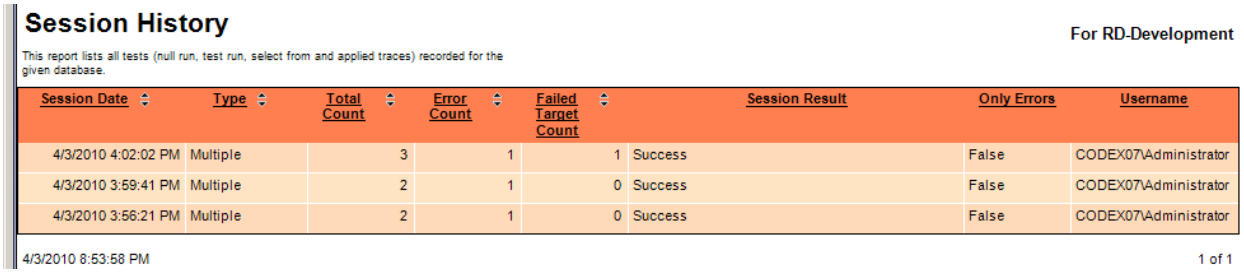
The Testing tool offers a number of different reports that relate to unit testing.



Copyright © 2010

Most reports filter to the database listed next to the report type and parameters specific to the report are sometimes available.

The first report we'll examine is "Session History." This report simply lists all occurrences of scheduled testing that have been recorded in the repository. In the example below, we see that the last testing run tested 3 objects, 1 object failed testing, and 1 failed to meet its performance target. (The "session result" is for the session as a whole and is intended to report problems that might have prevented the testing from completing properly.)



Session History

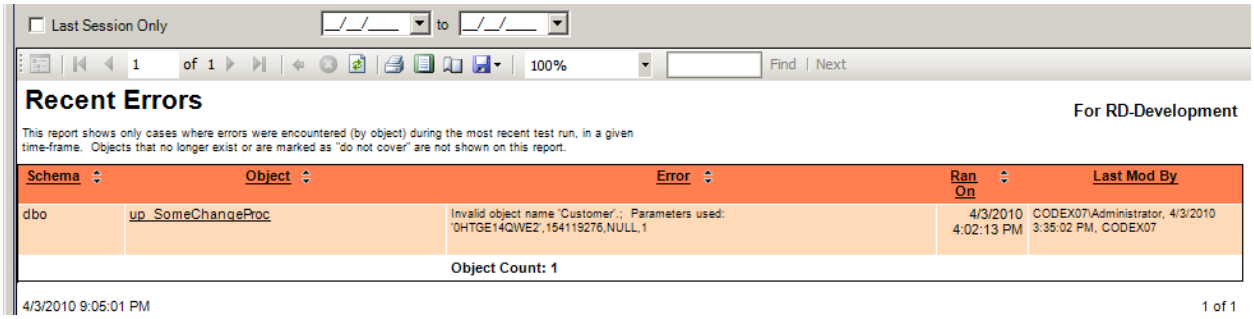
For RD-Development

This report lists all tests (null run, test run, select from and applied traces) recorded for the given database.

Session Date	Type	Total Count	Error Count	Failed Target Count	Session Result	Only Errors	Username
4/3/2010 4:02:02 PM	Multiple	3	1	1	Success	False	CODEX07\Administrator
4/3/2010 3:59:41 PM	Multiple	2	1	0	Success	False	CODEX07\Administrator
4/3/2010 3:56:21 PM	Multiple	2	1	0	Success	False	CODEX07\Administrator

4/3/2010 8:53:58 PM 1 of 1

The "Recent Errors" report shows test failures for objects. If "Last Session Only" is checked, it will only consider the very last testing session. If unchecked, an object that failed its test in an older session could end up being shown if it did not later pass its test. In the example below, we see the problem we'd introduced earlier: the Customer table is not present in the development database currently. The object name is hyperlinked such that clicking on it will navigate you to the current version of it in the SQL-Hero editor, using the RD-Development database.



Last Session Only __/__/____ to __/__/____

1 of 1 100% Find | Next

Recent Errors

For RD-Development

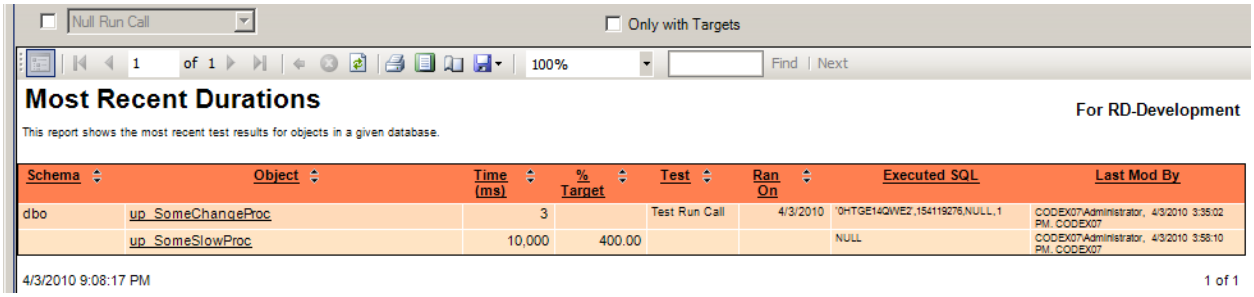
This report shows only cases where errors were encountered (by object) during the most recent test run, in a given time-frame. Objects that no longer exist or are marked as "do not cover" are not shown on this report.

Schema	Object	Error	Ran On	Last Mod By
dbo	up_SomeChangeProc	Invalid object name 'Customer'.; Parameters used: '0HTGE14QWE2',154119276,NULL,1	4/3/2010 4:02:13 PM	CODEX07\Administrator, 4/3/2010 3:35:02 PM, CODEX07
		Object Count: 1		

4/3/2010 9:05:01 PM 1 of 1

The "Most Recent Durations" report shows how long tests took for objects. Note that up_SomeSlowProc had a target time of 2,500 milliseconds, so this last execution that took 10 seconds is showing as being 400% of the target. Repeating values are suppressed in the "Test" and "Ran On" columns.

Copyright © 2010



Null Run Call | Only with Targets | 1 of 1 | 100% | Find | Next

Most Recent Durations

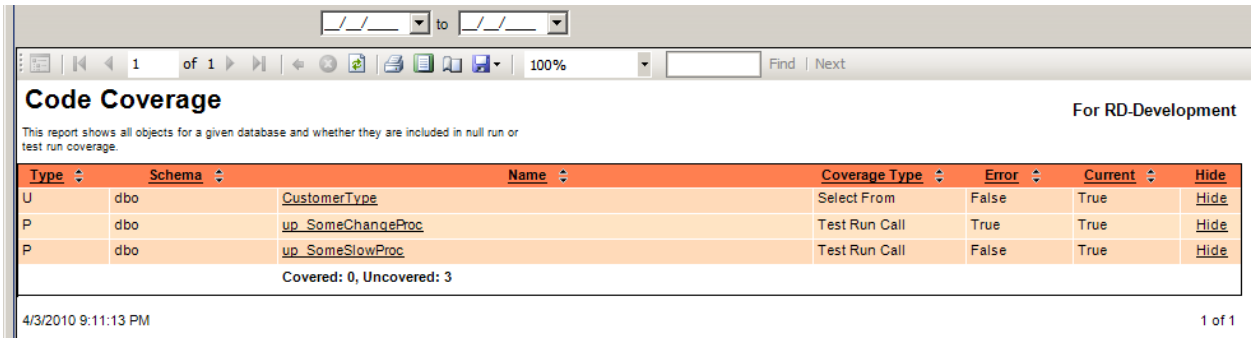
For RD-Development

This report shows the most recent test results for objects in a given database.

Schema	Object	Time (ms)	% Target	Test	Ran On	Executed SQL	Last Mod By
dbo	up_SomeChangeProc	3		Test Run Call	4/3/2010	0HTGE14QWE2:154119276:NULL:1	CODEX07\Administrator, 4/3/2010 3:35:02 PM, CODEX07
	up_SomeSlowProc	10,000	400.00			NULL	CODEX07\Administrator, 4/3/2010 3:58:10 PM, CODEX07

4/3/2010 9:08:17 PM 1 of 1

The "Code Coverage" report lists all testable objects in a database and describes what kind of testing is currently available for each object. Object already marked as "do not test" are not shown here. A coverage type of "None" implies the object is not covered by any test type. "Error" indicates the most recent test success/failure status for the object. Clicking on the "Hide" link allows you to mark an object as "do not test" directly from this report.



__/__/____ to __/__/____ | 1 of 1 | 100% | Find | Next

Code Coverage

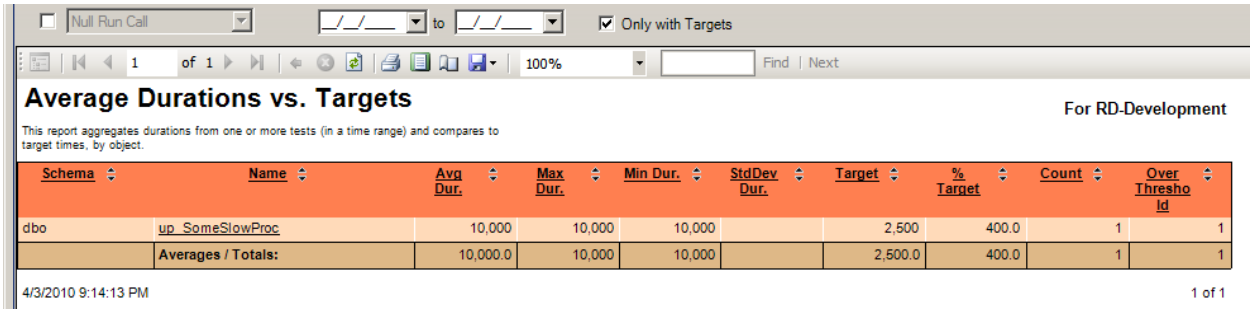
For RD-Development

This report shows all objects for a given database and whether they are included in null run or test run coverage.

Type	Schema	Name	Coverage Type	Error	Current	Hide
U	dbo	CustomerType	Select From	False	True	Hide
P	dbo	up_SomeChangeProc	Test Run Call	True	True	Hide
P	dbo	up_SomeSlowProc	Test Run Call	False	True	Hide
		Covered: 0, Uncovered: 3				

4/3/2010 9:11:13 PM 1 of 1

"Average Durations vs. Targets" shows you an aggregated total of historical test execution times, by object. The report can be filtered to only show objects that have targets (as illustrated below). "Average Over Time (Graph)" is similar in concept, but portrays the average duration on a line graph.



Null Run Call | __/__/____ to __/__/____ | Only with Targets | 1 of 1 | 100% | Find | Next

Average Durations vs. Targets

For RD-Development

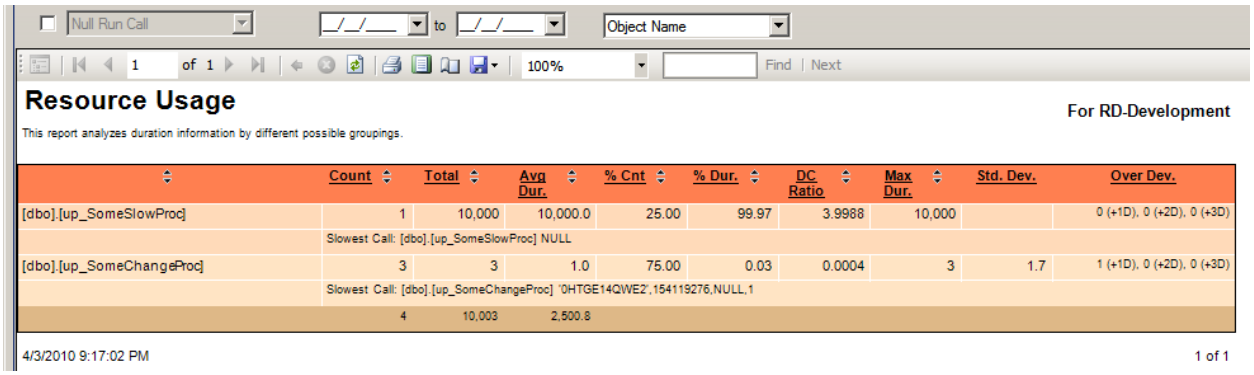
This report aggregates durations from one or more tests (in a time range) and compares to target times, by object.

Schema	Name	Avg Dur.	Max Dur.	Min Dur.	StdDev Dur.	Target	% Target	Count	Over Threshold
dbo	up_SomeSlowProc	10,000	10,000	10,000		2,500	400.0	1	1
	Averages / Totals:	10,000.0	10,000	10,000		2,500.0	400.0	1	1

4/3/2010 9:14:13 PM 1 of 1

The "Resource Usage Analysis" report attempts to group by a selected facet (in the example below, it's object name) and summarize comparatively within that facet. This could help pinpoint members of a given facet that are consistently out of alignment with the performance of other members. Below, the first procedure is taking up 99.97% of the cumulative duration during testing – if this were part of a more realistic data set, it would be an obvious concern!

Copyright © 2010



Null Run Call | __/__/____ to __/__/____ | Object Name | 1 of 1 | 100% | Find | Next

Resource Usage

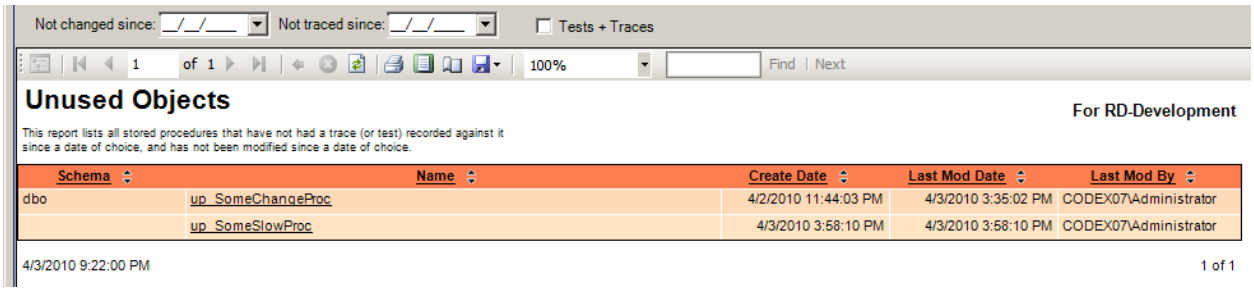
For RD-Development

This report analyzes duration information by different possible groupings.

	Count	Total	Avg Dur.	% Cnt	% Dur.	DC Ratio	Max Dur.	Std. Dev.	Over Dev.
[dbo].[up_SomeSlowProc]	1	10,000	10,000.0	25.00	99.97	3.9988	10,000		0 (+1D), 0 (+2D), 0 (+3D)
Slowest Call: [dbo].[up_SomeSlowProc] NULL									
[dbo].[up_SomeChangeProc]	3	3	1.0	75.00	0.03	0.0004	3	1.7	1 (+1D), 0 (+2D), 0 (+3D)
Slowest Call: [dbo].[up_SomeChangeProc] '0HTGE14QWE2',154119276,NULL,1									
	4	10,003	2,500.8						

4/3/2010 9:17:02 PM 1 of 1

"Unused Procs" is an interesting report because it can look at test data, trace data, and change tracking data. It can be used to help identify procedures that are not being used. In the example below, we see two objects – these have had no trace data collected and stored in the repository for them. If we checked the "Tests + Traces" checkbox and re-ran the report, no data would come back because we do have tests for them. This report should only be considered an aid in determining what object may no longer be used by an application.



Not changed since: __/__/____ | Not traced since: __/__/____ | Tests + Traces | 1 of 1 | 100% | Find | Next

Unused Objects

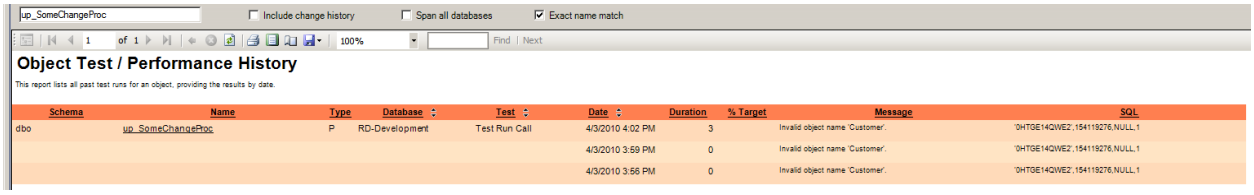
For RD-Development

This report lists all stored procedures that have not had a trace (or test) recorded against it since a date of choice, and has not been modified since a date of choice.

Schema	Name	Create Date	Last Mod Date	Last Mod By
dbo	up_SomeChangeProc	4/2/2010 11:44:03 PM	4/3/2010 3:35:02 PM	CODEX07\Administrator
	up_SomeSlowProc	4/3/2010 3:58:10 PM	4/3/2010 3:58:10 PM	CODEX07\Administrator

4/3/2010 9:22:00 PM 1 of 1

"Object Test History" reviews the complete testing history for a specific object (or objects matching a portion of a name). One can optionally interleave the change history for the object as stored in the repository. Here we observe the three tests run that included up_SomeChangeProc and the fact that all three tests failed with "Invalid object name 'Customer'".



up_SomeChangeProc | Include change history | Span all databases | Exact name match | 1 of 1 | 100% | Find | Next

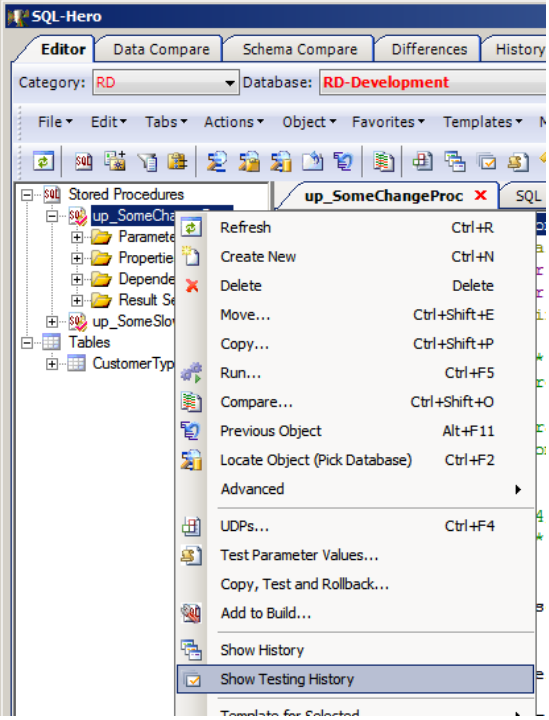
Object Test / Performance History

This report lists all past test runs for an object, providing the results by date.

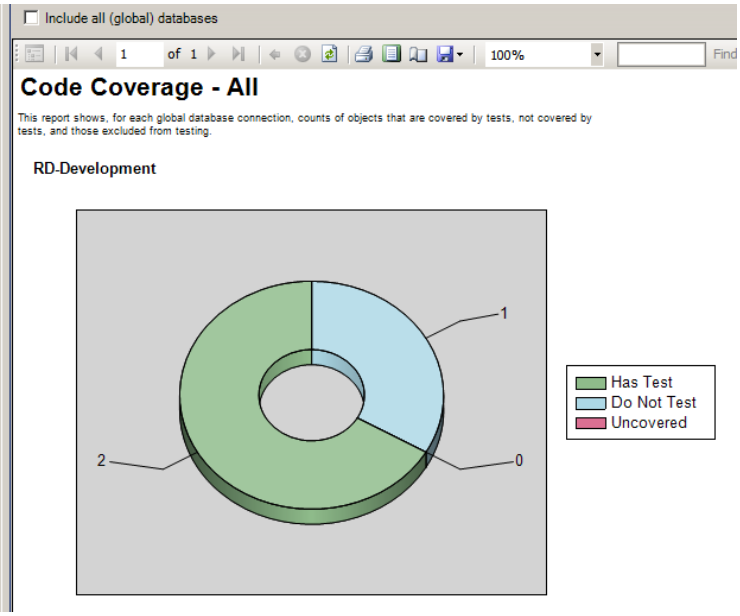
Schema	Name	Type	Database	Test	Date	Duration	% Target	Message	SQL
dbo	up_SomeChangeProc	P	RD-Development	Test Run Call	4/3/2010 4:02 PM	3		Invalid object name 'Customer'.	'0HTGE14QWE2',154119276,NULL,1
					4/3/2010 3:59 PM	0		Invalid object name 'Customer'.	'0HTGE14QWE2',154119276,NULL,1
					4/3/2010 3:56 PM	0		Invalid object name 'Customer'.	'0HTGE14QWE2',154119276,NULL,1

An alternate way to run this report is from the SQL-Hero editor. There is the "Show Testing History" context menu option available on the object tree:

Copyright © 2010



The "Code Coverage – Chart" shows a pie chart representation of how many objects have tests, do not have tests yet, or are marked as "do not test." The report can be run for the named database or for *all* global connections available.



Finally, the "Auto Create Proc Tests" report is available to not just provide output, but actually attempt to create unit tests for objects that are missing them. (There is an option to ignore existing tests and try to create all new tests.) In the example below, we've stipulated "5" parameter combinations to try. As seen in the report, we managed to successfully create a new test for up_SomeChangeProc. On the

Copyright © 2010

other hand we could not create a test of up_SomeSlowProc – the error message that prevented creating a test is shown (if there are different messages involved with different test parameters, the first five messages are included). We're also given the option here to "Hide" the object in the future, which means we are marking it as "do not test." Note that while the test creation is running, a button becomes available listing the object being processed – clicking the button effectively terminates the attempt at creating a test. This can be useful if the process becomes "stuck" on a long-running object.