Cheatography

utPLSQL 3.1.10 Cheat Sheet

by Samuel Nitsche (pesse) via cheatography.com/125418/cs/24133/

Define Test Suites with Annotations

Annotations are single-line comments and belong in the package header. Put a newline between package and procedure annotations.

Package Annotations

--%suite(description)	Package is a test-suite (*optional* description). *Mandatory*
--%suitepath(org.utplsql)	Groups suites in hierarchical namespaces
--%rollback(auto|manual)	Automatic (*default*) / manual transaction control
--%context(description) --%endcontext	Starts/ends sub-suite in suite. Can be nested.
--%disabled	Tests of suite/context won't be executed
--%beforeall(procedure [, ...]) --%afterall...	Procedure(s) to run before/after all tests in the suite/context
--%beforeeach(procedure [, ...]) --%aftereach...	Procedure(s) to run before/after each test in the suite/context
--%tags(myTag[, ...)	Label a test/context/suite

Procedure Annotations

--%test(description)	Procedure is a test (with optional description)
--%disabled	Test won't be executed
--%throws(exception [, ...])	Test expects exception(s) to be thrown
--%beforeall, --%afterall --%beforeeach, --%aftereach	Procedure to run before/after all/each tests in the suite/context
--%beforetest(procedure [, ...]) --%aftertest(procedure [, ...])	Procedure(s) to be run before/after annotated test
--%tags(myTag[, ...)	Label test

Run Tests

`exec ut.run();`	Run all tests to the default reporter (DBMS_OUTPUT)
`select * from table( ut.run());`	Run all tests and display output as result set

Consider using utplsql-cli to easily run tests

Run specific tests

`ut.run('my_suite_pckg')`	Run only the specified suite by name (current schema)
`ut.run('HR')`	Run all tests in specific schema
`ut.run('my_suite.test1, my_suite.test2')`	Run list of items
`ut.run(':my.suite.path')`	Run specific suite path

Run Parameters

`a_force_manual_rollback`	True/False. Run specified test and don't rollback changes (leaves open transaction)
`a_random_test_order`	True/False. Run tests in random order
`a_color_console`	True/False. Color-code output (for Windows see ANSICON)
`a_client_character_set`	Define report's character-set
`a_tags`	Run tests with any specified tags. Use `a_tags=>'-tag1'` to exclude a tag

Reporting: Run alternative Reporter

```
exec ut.run(ut_junit_reporter());
```

Reporters

`UT_DOCUMENTATION_REPORTER`	Textual pretty-print of test results. *Default*
`UT_COVERAGE_HTML_REPORTER`	HTML Coverage Report, includes SourceCode
`UT_JUNIT_REPORTER`	Test results conforming to JUnit 4 and above
`UT_REALTIME_REPORTER`	Test results and additional information as XML. Allows to show progress information
`UT_SONAR_TEST_REPORTER`	JSON Test results designed for SonarQube

See full list of built-in reporters. See documentation for coverage and reporters.

Cheatography

utPLSQL 3.1.10 Cheat Sheet

by Samuel Nitsche (pesse) via cheatography.com/125418/cs/24133/

Expectation Syntax

Base expectation

```
ut.expect( actual_value ).to_( matcher );
```

Negated expectation

```
ut.expect( actual_value ).not_to_( matcher );
```

Shortcut syntax (recommended)

```
ut.expect( actual_value ).to_matcher;
ut.expect( actual_value ).not_to_matcher;
```

Provide additional expectation description

```
ut.expect(1, 'Additional text').to_equal(2);
```

Simple Matchers

```
ut.expect(1=1).to_be_true();
ut.expect(1 is null).to_be_false();
ut.expect(null).to_be_null();
ut.expect(to_clob('ABC')).to_be_not_null();
ut.expect( 3 ).to_be_between( 1, 3 );
ut.expect( 3 ).to_be_greater_or_equal( 2 );
ut.expect( 2 ).to_be_greater_than( 1 );
ut.expect( 3 ).to_be_less_or_equal( 3 );
ut.expect( 3 ).to_be_less_than( 4 );
```

Equality Matchers

```
ut.expect( 'a dog' ).to_equal('a dog',
  a_nulls_are_equal => false );
```

a_nulls_are_equal is **true** by default

equal on objects

```
ut.expect(anydata.convertObject(l_expect))
  .to_equal(anydata.convertObject(l_actual));
```

equal on collections

```
ut.expect(anydata.convertCollection(l_expect))
  .to_equal(anydata.convertCollection(l_actual));
```

It's not possible to compare rowtypes and records directly. You have to wrap them in a `table of record` , select from them and use cursor comparsion

Advanced Matchers

be_like	`ut.expect( 'Lorem_impsum' ) .to_be_like( '%re%su' );`
be_like with escape	`ut.expect( 'Lorem_impsum' ) .to_be_like( a_mask => '%rem\_%', a_escape_char => '\' );` see Oracle like operator
match	`ut.expect('some value') .to_match('^some.*');`
match with options	`ut.expect( '123-456-ABcd' ) .to_match( a_pattern=>'\d{3}-\d{3}-[a-z]', a_modifiers=>'i');` See regexp_like function

Cursor Comparison

Basic example

```
open l_expected for select * from all_objects;
open l_actual for select * from all_objects;
ut.expect(l_actual)
  .to_equal(l_expected)
  .join_by('COLUMN_NAME')
  .exclude(ut_varchar2_list('ID','SOMECOLUMN'))
```

Special Matchers

```
ut.expect(l_actual).to_have_count(5);
ut.expect(l_actual).to_be_empty();
```

Helpers

Type: List of strings

```
ut_varchar2_list('String 1', 'String 2', ...)
```

Type: List of reporters

```
ut_reporters(ut_documentation_reporter(), ut_jun-
it_reporter())
```

Select all tests

```
select * from table(
  ut_runner.get_suites_info(USER, 'PACKAGE_NAME'))
  where item_type = 'UT_TEST';
```