

Package annotations

--%suite	Package is a test suite
--%suite(description)	A suite with description
--%suitepath(org.my_org.my_project)	Similar to Java package. Groups suites in namespaces

Annotations are single-line comments starting with a % sign.
Needed in package specification only ([documentation](#))

Procedure annotations

--%test	The procedure is a test
--%test(description)	A test with description
--%beforetest(procedure_name)	Runs the procedure before annotated test
--%aftertest(procedure_name)	Runs the procedure after annotated test
--%beforeall	The procedure to run before first test in suite
--%afterall	The procedure to run after last test in suite
--%beforeeach	The procedure to run before each test
--%aftereach	The procedure to after each test

Unary matchers

be_empty

```
open l_cursor for select * from dual where 1 = 0;
ut.expect( l_cursor ).to_( be_empty() );
```

be_false

```
ut.expect( ( 1 = 0 ) ).to_( be_false() );
```

be_not_null

```
ut.expect( to_clob(' ABC') ).to_( be_not_null() );
```

be_null

```
ut.expect( 1 ).to_( be_null() );
```

be_true

```
ut.expect( ( 1 = 1 ) ).to_( be_true() );
```

Common annotations

--%disabled	Suite / test will not execute
--%rollback(auto)	Automatic savepoints/rollbacks - default
--%rollback(manual)	No savepoint/rollback

Expectation structure

Base expectation block

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

Negated expectation block

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

Shortcuts to matchers

```
ut.expect( actual_value ).[not_]to_matcher;
```

Equality matcher

equal

```
ut.expect( 'a dog' ).to_(
    equal( 'a dog' , a_nulls_are_equal => false )
a_nulls_are_equal is true by default
```

equal with cursors

```
open l_expected for select * from dual;
open l_actual for select * from dual where 1 = 0;
ut.expect( l_expected ).to_(
    equal( l_actual, a_exclude => ' column_a, column_b' ) );
```

equal on objects

```
ut.expect( anydat a_convert Object ( l_expected ) ).to_(
    equal( anydat a_convert Object ( l_actual ) ) )
```

equal on collections

```
ut.expect( anydat a_convert Collection( l_expected ) ).to_(
    equal( anydat a_convert Collection( l_actual ) ) );
```

Non-equality matchers

be_like

```
ut.expect( 'Lorem ipsum' ).to_(
    be_like( a_mask => '%rem%', a_escape_char => ' '
ut.expect( 'Lorem ipsum' ).to_( be_like( '%rem%' ) );
a_mask, a_escape_char -> see Oracle like operator
```



Non-equality matchers (cont)

match

```
ut.expect( a_actual => '123-4 56- ABcd' ).to_(
  match( a_pattern => '\d{3} -\d {3} -[a -z]', a_modifiers => 'i'
) );
ut.expect( 'some value' ).to_( match( '^some.*' ) );
a_pattern, a_modifiers -> see regexp\_like function
```

be_between

```
ut.expect( 3 ).to_( be_between( 1, 3 ) );
```

be_greater_or_equal

```
ut.expect( sysdate ).to_( be_greater_or_equal( sysdate - 1 ) );
```

be_greater_than

```
ut.expect( 2 ).to_( be_greater_than( 1 ) );
```

be_less_or_equal

```
ut.expect( 3 ).to_( be_less_or_equal( 3 ) );
```

be_less_than

```
exec ut.expect( 3 ).to_( be_less_than( 2 ) );
```

Reporting

Color output

```
exec ut.run( a_color_console => true );
With sqlcl
or sqlPlus (Mac, Unix, Windows ANSICON)
```

XUnit reporter

```
exec ut.run( ut_xunit_reporter() );
JUnit-compatible XML report for CI servers
```

TeamCity reporter

```
exec ut.run( ut_teamcity_reporter() );
TeamCity-specific report
```

Sonar Test reporter

```
exec ut.run( ut_sonar_test_reporter() );
Sonar-specific XML tests report
```

Coverage html reporter

```
exec ut.run( ut_coverage_html_reporter
() );
Produces HTML coverage report
```

Documentation for [coverage](#) and [reporters](#)

Executing tests

Run all unit tests in my current schema

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

Run all unit tests in current schema after it was changed to HR

```
alter session set current_schema = 'HR';
exec ut.run();
```

Run all unit tests in specific schema

```
exec ut.run( 'HR' );
```

Run all unit tests in specific package of current schema

```
exec ut.run( 'test_be_twn_str' );
```

Run all unit tests in specific schema.package

```
exec ut.run( 'hr.test_be_twn_str' );
```

Run one specific test only

```
exec ut.run( 'hr.test_be_twn_str.big_end_position' );
```

Run several items

```
exec ut.run( ut_varchar2_list(
  'hr.test_award_bonus',
  'hr.test_betwnstr.big_end_position' ) );
```

Run test using suitepath

```
exec ut.run( ':com.my_organization' );
```

Run the tests as a select statement

```
select * from table( ut_run() );
```

Catching exceptions

```
procedure my_code_raises_zero_divisor is
  l_my_number number;
begin
  l_my_number := 1/0;
  ut.fail( 'Expected exception but nothing was raised' );
exception
  when others then
    ut.expect( sqlcode ).to_equal( -1476 );
end;
```

