Generic

Feature	utPLSQL	ruby-plsql
Installation	per DB instance	per client (developer/CI server)
Cross sessions testing	NO	YES
Cross database testing	NO	YES
Can be used for privileges testing	NO	YES
Can be used for VPD/RLS testing	NO	YES
Exception handling	Poor	Full stack trace
Test - tested code isolation	Low	High
Runs with invalid DB dependencies	NO (disappearing tests)	YES
Runs without tested DB objects / code	NO (disappearing tests)	YES
Migration across databases	Needs installation of framework and tests	Trivial. Change of connect string
Test language maturity	Low	Industry standard
Performance	Excellent (100% in database)	Sufficient Suffers from network overhead Suffers from Ruby startup (2-5 secs)
Cucumber support	NO	YES
Suitable for integration testing	NO	YES
Community Activity	Low	Low for ruby-plsql High for RSpec

Assertions

Feature	utPLSQL	ruby-plsql
Assertion types	Two One for equality based matchers One to check if expression evaluates to TRUE	Multiple assertions(matchers) <,>,=,!=,inclusion,regexp,datatype(class),...
Assertion definition	Defined per datatype	Defined per operator
Assertions are used the same way	NO Different usage depending on compared type	YES All assertions follow common pattern
Assertion on User Defined Type data	NO	YES
Assertion on Collection Type data	YES (cumbersome and undocumented usage)	YES
Assertion on PL/SQL records data	YES (cumbersome and undocumented usage)	YES

By **Jacek Gebal** (jgebal)
cheatography.com/jgebal/
www.oraclethoughts.com

Published 17th August, 2015.
Last updated 25th April, 2017.

Assertions (cont)		
Assertion on Cursor data	YES (complex usage)	YES
Assertion on complex structured data	NO	YES
Assertion on TIMESTAMP/CLOB/BLOB/RAW	NO	YES
Assertion on success (no exception)	NO	YES
Can tests table/view structure	NO	YES

Test structure		
Feature	**utPLSQL**	**ruby-plsql**
Physical test location	Tests located in database schemes / packages or procedures	Yests organized in project into folders/spec files
Physical test organization	Strict - database oriented Schema/package/procedure	Flexible - project oriented Within a test file, tests organized into nestable example groups
Logical test organization	Limited to package level grouping Packages can be organized into suites Each suite can contain many packages Each package can be placed in many suites	Flexible Each test(example) can be labelled with tags Each example group can be labelled with tags Each tag can be assigned to many tests/example groups
Test execution granularity	All tests in a single test package or All tests in a single suite of test packages	Single test or All tests in a specified example group or All tests in a mask-specified directory/file or All test with a specific tag(s) or All tests except specific tag(s) and more
Identifying and naming tests	Each assertion has a mandatory text description Assertion is a test	Each example group can have a descriptive free text name Each example can have a descriptive free text name Each example can contain many assertions composing the test

Test structure (cont)		
Re-usable tests/-shared examples	NO Separate tests are needed for two functions do the same thing but on different objects(datatypes)	YES Standard of shared examples for testing of identical behavior on different objects(datatypes)
Test suites definition	Defined in database tables	Defined as tags in test definition files or by test file location in directory structure
Suites management	Calls to API prior to test execution, persisted in DB per user	Tags defined beside the test definition in test files
Reporting config-uration	Calls to API prior to test execution, persisted in DB per user	Parameter when executin tests
Customizations within test/suite/project	NO One common library per database	YES Own assertions/configurations can be added to tests or project

Test execution		
Feature	**utPLSQL**	**ruby-plsql**
Needs compilation prior to execution	YES	NO
Test invocation	- connect to DB - call API to execute package or suite	execute "rspec" from command prompt in project root
Default tests execution scope	NONE You need to explicitly state either a suite or a package to be tested'	All test for project By default, calling "rspec" command from project root will exeute all tests for project
Parallel test executiuon	Doable - do it yourself. By splitting test into separate suites and running them from CI in parallel jobs.	YES-automatic with open-source libraries
Transaction management	Manual	Automatic Conforms with RSpec standard for keeping the object(s) unchanged outside of test scope

By **Jacek Gebal** (jgebal)
cheatography.com/jgebal/
www.oraclethoughts.com

Published 17th August, 2015.
Last updated 25th April, 2017.

Test execution (cont)		
Test setup/-cleanup	One mandatory setup and cleanup per test package. (Bolierplate code when not used) Useless when different setup needed for each test.	Optional multiple setups cleanups can be defined on each level of example group. Two triggering modes can be mixed for setup/cleanup: - Before all tests in example group - Before each example group Setups /cleanups available for entire suite (before suite/after suite) Setups /cleanups can be invoked with filtering by tags too

Reporting		
Feature	**utPLSQL**	**ruby-plsql**
Build in report types	3 build-in types: - screen output to client console - file output (needs to write to DB server) - html file output (needs to write to DB server) Outputs incomplete, console output noisy.	4 build-in types - dotted - very dense, useful for developers - documentation - QA text reporting oriented - HTML - like documentation but in publishable form - JSON - for machine processing
Extensibility / third party	Can be extended - do it yourself	Available open-source libraries for other output formatting (like CI JUnit formatters)
Build in code coverage generation	NO	YES
Supplies timing for tests	NO	YES
Supplies count of tests executed	not directly	YES
Full stack trace for exceptions	NO	YES
Self-documenting tests / tests expressivness	NO procedure_name_30_char_limit no place for test description description is placed inside single assertion "som-ewhere inside test code"	YES tests and example blocks have a "full text descriptive names"

API		
Feature	**utPLSQL**	**ruby-plsql**
Complexity	Complex and inconsistent	Consistent and dense
Test coverage for API	NO Tested manually by users and contributors	YES API is tested by unit tests
Learning curve	High Mailnly due to inconsistency and workarounds and tricks used to overcome nature and limitations of PL/SQL	Low/medium Need to learn RSpec and Ruby basics and how to use Array and Hash Objects
IDE support	No IDE support for API itself. Running tests, reporting from IDE is not supported	Highly supported with JetBrains Rubymine (or IntelliJ) Ruby, RSpec, Cucumber, Gherkin syntax highlighting and code completion Test execution, exporting test results Support for GIT/SVN/Mercurial, PLSQL, SQL, Jira, Stash and more (many of free plugins available)
Completeness	Medium	High
Integrates with CI (Jenkins)	not directly Doable through external calls with Java and Maven Integration suffers from API reporting limitations	YES CI-JUnit Reporter plugins available
Documentation	Incomplete online documentation	Everything described by examples Concept of self documenting tests

By **Jacek Gebal** (jgebal)
cheatography.com/jgebal/
www.oraclethoughts.com

Published 17th August, 2015.
Last updated 25th April, 2017.