

# Testing with utPLSQL

Made Easy with SQL Developer

Philipp

 @phsalvisberg

 <https://www.salvis.com/blog>

23.02.2021

BASEL | BERN | BRUGG | BUCHAREST | DÜSSELDORF | FRANKFURT A.M. | FREIBURG I. BR.  
GENEVA | HAMBURG | LAUSANNE | MANNHEIM | MUNICH | STUTTGART | VIENNA | ZURICH



# Introduction

# Test Automation

"The use of software separate from the software being tested to **control the execution of tests** and the comparison of actual outcomes with predicted outcomes."

Source: [https://en.wikipedia.org/wiki/Test\\_automation](https://en.wikipedia.org/wiki/Test_automation)

# Testing Scope in Database Development

Every component of an application  
that is deployed in the database.



For example:  
Types, packages, procedures, functions,  
views, triggers, constraints.



## Core Testing Framework

- Schema in the database
- No repository
- Annotation based tests



## Development

- Realtime Reporter
- Code Coverage, Code Templates, etc.



## Test Automation

- Command Line Client
- Maven Plugin **Maven™**
- Various Reporters



# Test Declaration

```
CREATE OR REPLACE PACKAGE test_package_name AS
    --%suite
    --%test
    PROCEDURE procedure_name;
END;
```

--%displayname(<description>)  
--%test(<description>)  
--%tags(<tag>[,...])  
--%throws(<exception>[,...])  
--%beforeall  
--%afterall  
--%beforeeach  
--%aftereach  
--%beforetest([...])  
--%aftertest([...])  
--%rollback(manual)  
--%disabled

--%suite(<description>)  
--%suitepath(<path>)  
--%tags(<tag>[,...])  
--%displayame(<description>)  
--%beforeall([...])  
--%afterall([...])  
--%beforeeach([...])  
--%aftereach([...])  
--%rollback(manual)  
--%disabled  
--%context  
--%endcontext

# Test Implementation

```
CREATE OR REPLACE PACKAGE BODY test_package_name AS
    PROCEDURE procedure_name IS
        l_actual    SYS_REFCURSOR;
        l_expected SYS_REFCURSOR;
    BEGIN
        open l_actual for select * from dept where deptno != 30;
        open l_expected for select * from dept;
        ut.expect(l_actual).to_equal(l_expected).join_by('DEPTNO');
    END procedure_name;
END;
```

Matcher:

be\_between, be\_empty, be\_false, be\_greater\_than,  
be\_greater\_or\_equal, be\_less\_or\_equal, be\_less\_than,  
be\_like, be\_not\_null, be\_null, be\_true, contain, **equal**,  
have\_count, match

Extended options for refcursor, object type, JSON, nested table and varray:

- include(<items>)
- exclude(<items>)
- unordered
- **join\_by(<items>)**

# Test Run

```
SET SERVEROUTPUT ON SIZE UNLIMITED
EXEC ut.run('test_package_name')
```

List of ...

- [schema.]**package**[.procedure]
- [schema]:suitepath[.context][.procedure]

List of ...

- tag (include)
- -tag (exclude)

```
test_package_name
procedure_name [.03 sec] (FAILED - 1)
```

Failures:

```
1) procedure_name
   Actual: refcursor [ count = 3 ] was expected to equal: refcursor [ count = 4 ]
   Diff:
   Rows: [ 1 differences ]
      PK <DEPTNO>30</DEPTNO> - Missing: <DEPTNO>30</DEPTNO><DNAME>SALES</DNAME><LOC>CHICAGO</LOC>
   at "SCOTT.TEST_PACKAGE_NAME.PROCEDURE_NAME", line 8 ut.expect(l_actual).to_equal(l_expected)
                                              .join_by('DEPTNO');
```

```
Finished in .034792 seconds
1 tests, 1 failed, 0 errored, 0 disabled, 0 warning(s)
```

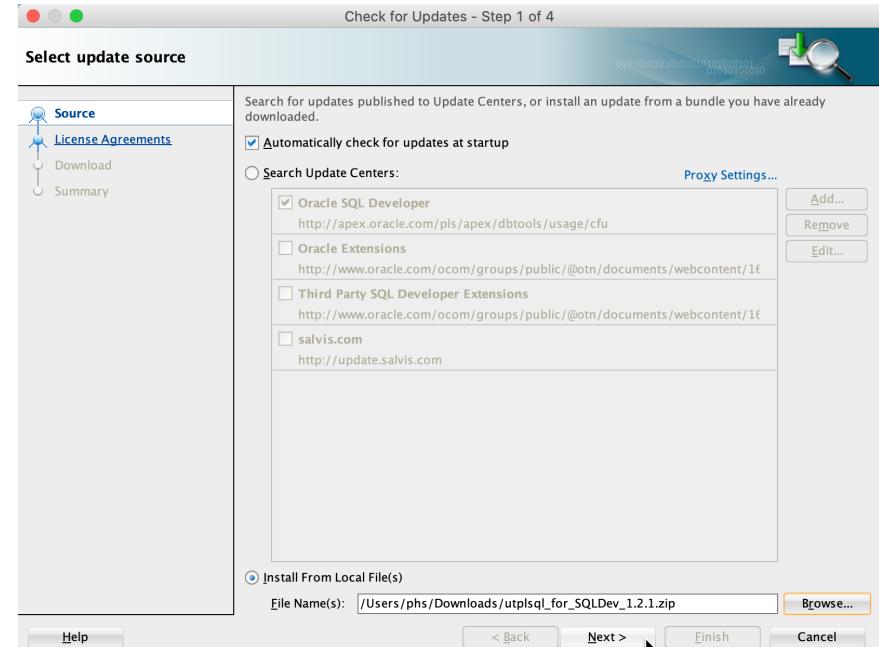
# Installation

# Install utPLSQL Core Testing Framework

- Download utPLSQL.zip from <https://github.com/utPLSQL/utPLSQL/releases>
- Unzip utPLSQL.zip
- cd source
- sqlplus / as sysdba @install\_headless.sql
  - User UT3
  - Password XNtxj8eEgA6X6b6f
  - Tablespace USERS

# Install utPLSQL for SQL Developer from File \*)

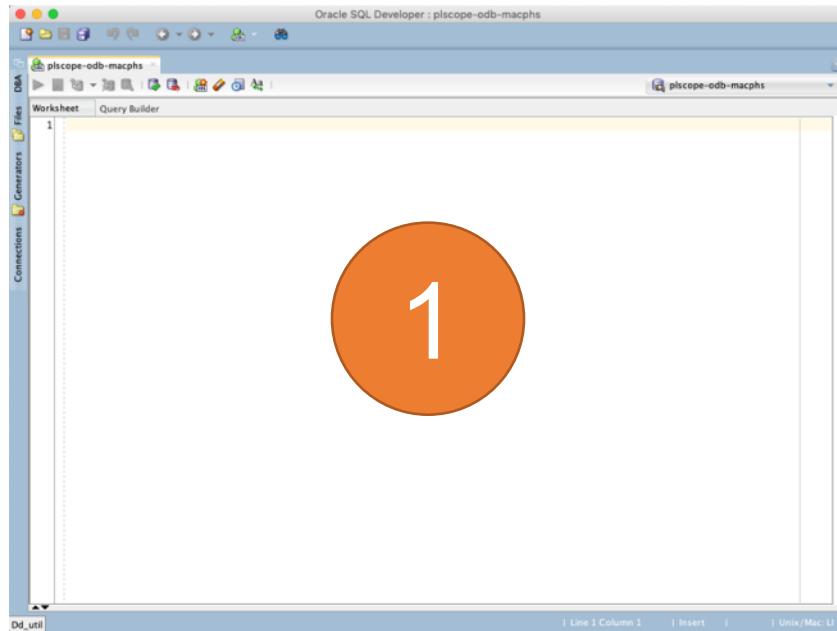
- Download `utplsql_for_SQLDev_*.zip` from <https://github.com/utPLSQL/utPLSQL-SQLDeveloper/releases>
- Start SQL Developer
- Select "Check for Updates..." in the help menu \*)
- Use the "Install From Local File" option to install the previously downloaded "`utplsql_for_SQLDev_*.zip`" file
  - User must have read/write access to SQL Developer installation directory
  - Run as Administrator, if required
- Restart SQL Developer



\*) You can also configure an Update Center, see <https://github.com/PhilippSalvisberg/sqldev-update>

# Build & Run Tests in SQL Developer

# Starting Point?



A screenshot of the Oracle SQL Developer interface showing a package body named "PLSCOPE.ETL Body@plscope-odb-macphs". The code is as follows:

```
1 create or replace PACKAGE BODY etl AS
2   g_unused_column v$statistic.sid%TYPE;
3
4   PROCEDURE clear_deptsal IS
5   BEGIN
6     DELETE FROM deptsal;
7     DELETE FROM deptsal_err;
8     sys.dbms_output.put_line('deptsal an deptsal_err deleted.');// use synonym
9   END clear_deptsal;
10
11  PROCEDURE load_from_tab IS
12  BEGIN
13    clear_deptsal;
14    INSERT INTO deptsal (dept_no, dept_name,
15      SELECT /*+ordered */ d.deptno, d.dname,
16        FROM dept d
17        LEFT JOIN (SELECT * FROM emp WHERE hiredate >= trunc(sysdate) - 1)
18          ON e.deptno = d.deptno
19        GROUP BY d.deptno, d.dname;
20    COMMIT;
21    sys.dbms_output.put_line('deptsal loaded and committed (from tab).');
22  END load_from_tab;
23
24  PROCEDURE load_from_view IS
25  BEGIN
26    clear_deptsal;
27    INSERT INTO deptsal (dept_no, dept_name, salary)
28      SELECT dept_no, dept_name, salary
29        FROM source_view;
30    COMMIT;
31    sys.dbms_output.put_line('deptsal loaded and committed (from view).');
32  END load_from_view;
33
34  PROCEDURE load_from_syn IS
35  BEGIN
36    clear_deptsal;
37    INSERT INTO deptsal (dept_no, dept_name, salary)
38      SELECT dept_no, dept_name, salary
```

# Run Code Coverage Reports in SQL Developer

# Code Coverage – Definition

"A measure used to describe the degree to which the source code of a program is executed when a particular test suite runs."

Source: [https://en.wikipedia.org/wiki/Code\\_coverage](https://en.wikipedia.org/wiki/Code_coverage)

# Line Coverage

```
CREATE OR REPLACE FUNCTION f(a IN INTEGER) RETURN INTEGER IS
BEGIN
    IF a IS NULL THEN
        RETURN 0;
    ELSE
        RETURN a*a;
    END IF;
END f;
/
```



Two test cases for  
100% coverage

# Code Block Coverage (12.2 and higher)

```
CREATE OR REPLACE FUNCTION f(a IN INTEGER) RETURN INTEGER IS
BEGIN
    IF a IS NULL THEN RETURN 0; ELSE RETURN a*a; END IF;
END f;
/
```

Two test cases for  
100% coverage

```
CREATE OR REPLACE FUNCTION f(a IN INTEGER) RETURN INTEGER IS
BEGIN
    RETURN coalesce(a*a, 0);
END f;
/
```

One test case for  
100% coverage  
when passing NULL

# utPLSQL – Line & Code Block Coverage

test\_f.sql

```
1 CREATE OR REPLACE PACKAGE test_f IS
2   --%suite
3
4   --%test
5   PROCEDURE f1;
6
7 END test_f;
8 /
9
10 CREATE OR REPLACE PACKAGE BODY test_f IS
11
12   PROCEDURE f1 IS
13     l_actual INTEGER := f(2);
14     l_expected INTEGER := 4;
15   BEGIN
16     ut.expect(l_actual).to_equal(l_expected);
17   END f1;
18
19 END test_f;
20 /
```

Passing a  
not null  
value

Code coverage

PLSCOPE.F

100 % lines covered

1 relevant lines. 1 lines covered (including 1 lines partially covered ) and 0 lines missed

```
1. FUNCTION f(a IN INTEGER) RETURN INTEGER IS
2. BEGIN
3.   RETURN coalesce(a*a, 0);
4. END f;
```

1 of 2  
code blocks  
covered

Is this good?  
helpful?

# Core Messages

# The First Step Is the Hardest

- **Set up a test-friendly environment**
  - Install utPLSQL core testing framework
  - Install SQL Developer for utPLSQL
- **Start with tests**
  - to reproduce bugs
  - for new requirements
- **utPLSQL will change how you code**
  - Write smaller units
  - Isolate code that is difficult to test





Making a **WORLD** possible  
in which **intelligent IT**  
facilitates **LIFE** and **WORK** as a  
**matter of course.**