
ORACLE®

How to mark up PL/SQL source text after a code coverage run

Bryn Llewellyn
Distinguished Product Manager
Database Division
Oracle HQ
twitter: @BrynLite

Winter 2018

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Wikipedia on “basic block”

The code in a basic block has:

- One entry point, meaning no code within it is the destination of a jump instruction.
- One exit point, meaning only the last instruction can cause the program to begin executing code in a different basic block.

Under these circumstances, whenever the first instruction in a basic block is executed, the rest of the instructions are necessarily executed exactly once, in order.

The code may be source code, assembly code or some other sequence of instructions.

PL/SQL's code coverage tool

- Start coverage, run the tests, stop coverage
- Blocks
  - A block is a run of one or more consecutive characters in the source text of a PL/SQL unit. Every character of the text is in exactly one block.
  - If, during a coverage run, the point of execution enters a particular unit, then information about each of its blocks is recorded in a table
- Each block is labeled with the unit it's in, the line/column where it starts, the covered yes/no status, and the "feasibility" yes/no status
- Can calculate percentage of [feasible] blocks covered
- Can mark up source code with coverage coloring

```
function Three_Condition_If(
  i in pls_integer)
  return pls_integer
  authid Definer
is
begin
  if i < 1 or i > 10 or i is null then
    return 0;
  else
    return 1;
  end if;
end Three_Condition_If;
```

```
function Recip(i in number) return number authid Definer is
begin
  return 1.0/i;
exception
  when Zero_Divide then
    return 42;
  when others then
    ...
    raise;
end Recip;
```

```
declare
  Run_ID pls_integer not null := -1;
begin
  Run_ID := Sys.DBMS_Plsql_Code_Coverage.Start_Coverage(Run_Comment=>'Run_001');

  DBMS_Output.Put_Line(Recip(17));


  Sys.DBMS_Plsql_Code_Coverage.Stop_Coverage();
end;
```

```
function Recip(i in number) return number authid Definer is
begin
  return 1.0/i;
exception
  when Zero_Divide then
    return 42;
  when others then
    ...
    raise;
end Recip;
```

```
declare
  Run_ID pls_integer not null := -1;
begin
  Run_ID := Sys.DBMS_Plsql_Code_Coverage.Start_Coverage(Run_Comment=>'Run_001');

  DBMS_Output.Put_Line(Recip(17));
  DBMS_Output.Put_Line(Recip(0));

  Sys.DBMS_Plsql_Code_Coverage.Stop_Coverage();
end;
```

```
function Recip(i in number) return number authid Definer is
begin
  return 1.0/i;
exception
  when Zero_Divide then
    return 42;
  when others then
    ...
    raise;
end Recip;
```

```
function Recip(i in number) return number authid Definer is
begin
  return 1.0/i;
exception
  when Zero_Divide then
    return 42;
  when others then
    pragma Coverage('NOT_FEASIBLE');
    raise;
end Recip;
```

```
declare
  Run_ID pls_integer not null := -1;
begin
  Run_ID := Sys.DBMS_Plsql_Code_Coverage.Start_Coverage(Run_Comment=>'Run_001');

  DBMS_Output.Put_Line(Recip(17));
  DBMS_Output.Put_Line(Recip(0));

  Sys.DBMS_Plsql_Code_Coverage.Stop_Coverage();
end;
```

```
function Recip(i in number) return number authid Definer is
begin
  return 1.0/i;
exception
  when Zero_Divide then
    return 42;
  when others then
    pragma Coverage('NOT_FEASIBLE');
    raise;
end Recip;
```

```
procedure Case_Stmt_1(i in boolean := true) authid Definer is
begin
  case i
    when true then
      DBMS_Output.Put_Line('i is true');

    when false then
      DBMS_Output.Put_Line('i is false');

    else
      DBMS_Output.Put_Line('i is null');
  end case;
end Case_Stmt_1;
```

```
procedure Case_Stmt_2(i in boolean := true) authid Definer is
begin
  case i
    when true then
      DBMS_Output.Put_Line('i is true');

    when false then
      DBMS_Output.Put_Line('i is false');

    -- Notice that there's no "else" leg.

  end case;
exception when Case_Not_Found then
  DBMS_Output.Put_Line('Case_Not_Found');
end Case_Stmt_2;
```

What comes out of the box?

```
DBMS_Plsql_Code_Coverage.Create_Coverage_Tables(
  Force_It=>false);
```

Creates these tables

DBMSPCC_Runs
DBMSPCC_Units
DBMSPCC_Blocks

DBMSPCC_Runs

Run_ID	not null	number(38)
Run_Comment		varchar2(4000)
Run_Owner	not null	varchar2(128)
Run_Timestamp	NOT NULL	date

DBMSPCC_Units

Run_ID	not null number(38)
Object_ID	not null number(38)
Owner	not null varchar2(128)
Name	not null varchar2(128)
Type	NOT NULL varchar2(12)
Last_DDL_Time	NOT NULL date

DBMSPCC_Blocks

```
Run_ID                not null number(38)
Object_ID             not null number(38)
Line                  not null number(38)
Col                   not null number(38)
Covered               not null number(1)
Not_Feasible          not null number(1)
```

What do you have to do yourself?

View

DBMSPCC_All_Source_With_Blocks

Packages

DBMSPCC_Markup
DBMSPCC_Visualization

Table

DBMSPCC_Marked_Up_Source

DBMSPCC_All_Source_With_Blocks

```
Object_ID          not null number
...
Line               not null number
Text                        varchar2(4000)
Block                       number(38)
Col                         number(38)
Covered                     number
Not_Feasible                number
```

```
function Three_Condition_If(                    1    1
  i in pls_integer)
  return pls_integer
  authid Definer
is
begin
  if i < 1 or i > 10 or i is null then          0   17
  if i < 1 or i > 10 or i is null then          0   25
    return 0;                                   1    5
  else
    return 1;                                   0    5
  end if;
end Three_Condition_If;
```

package DBMSPCC_Markup

```
package DBMSPCC_Markup authid Definer is
  procedure Do;
end DBMSPCC_Markup;
```

```
{1}function Three_Condition_If(
   i in pls_integer)
   return pls_integer
   authid Definer
is
begin
   if i < 1 or i {3}> 10 or {3}i is null
then

     {1}return 0;
   else
     {3}return 1;
   end if;
end Three_Condition_If;
```

package DBMSPCC_Visualization

```
package DBMSPCC_Visualization authid Definer is
  procedure Do;
end DBMSPCC_Visualization;
```

```
function Three_Condition_If(
  i in pls_integer)
  return pls_integer
  authid Definer
is
begin
  if i < 1 or i > 10 or i is null then
    return 0;
  else
    return 1;
  end if;
end Three_Condition_If;
```

Demo time...

Integrated Cloud

Applications & Platform Services