17

# Using PL/SQL Basic Block Coverage to Maintain Quality

The PL/SQL basic block coverage interface helps you ensure some quality, predictability and consistency, by assessing how well your tests exercise your code.

The code coverage measurement tests are typically executed on a test environment, not on a production database. The goal is to maintain or improve the regression tests suite quality over the lifecycle of multiple PL/SQL code releases. PL/SQL code coverage can help you answer questions such as:

- Is your testing suites development keeping up with the development of your new code?
- Do you need more tests?

The PL/SQL basic block coverage interface collects coverage data for PL/SQL units exercised in a test run.

**Topics:**



> **See Also:**
>
> - *Oracle Database PL/SQL Language Reference* for the `COVERAGE PRAGMA` syntax and semantics
> - *Oracle Database PL/SQL Packages and Types Reference* for more information about using the `DBMS_PLSQL_CODE_COVERAGE` package
> - *Oracle Database PL/SQL Language Reference* for more information about the PLSQL_OPTIMIZE_LEVEL compilation parameter

## 17.1 Overview of PL/SQL Basic Block Coverage

The `DBMS_PLSQL_CODE_COVERAGE` package enables you to collect data at the basic block level. PL/SQL developers and testing engineers use code coverage testing results as part of their standard quality assurance metric.

Code coverage is a measure of the percentage of code which is covered by automated tests. A program with high code coverage has less chance of containing bugs than a program with low code coverage. The most important is the percent of basic blocks executed by a test suite. A basic block is a linear segment of code with no branches. A basic block has a single entry point (no code within a basic block is the destination of a jump instruction) and a single exit point (only the last instruction, or an exception, can move the point of execution to a different basic

block). Basic block boundaries cannot be predicted by visual inspection of the code. The compiler generates the blocks that are executed at runtime.

Coverage information at the unit level can be derived accurately by collecting coverage at the basic block level. Utilities can be produced to report and visualize the test coverage results and help identify code that is covered, partially covered, or not covered by tests.

It is not always feasible to write test cases to exercise some basic blocks. It is possible to exclude these blocks from the coverage calculation by marking them using the `COVERAGE` pragma. The source code can be marked as not feasible for coverage either a single basic block, or a range of basic blocks.

# 17.2 Collecting PL/SQL Code Coverage Data

This example shows you the basic steps to collect and analyze PL/SQL basic block code coverage data using the `DBMS_PLSQL_CODE_COVERAGE` package.

PL/SQL basic block coverage data is collected when program units use `INTERPRETED` compilation ( parameter set `PLSQL_CODE_TYPE` = `INTERPRETED`). PL/SQL basic block coverage data is not collected when program units use `NATIVE` compilation. You can disable the `NATIVE` compiler by setting the parameter `PLSQL_OPTIMIZE_LEVEL` <= 1. Regardless of the compilation mode, coverage data for wrapped units is not collected.

Follow these steps to collect and analyze PL/SQL basic block code coverage data using the `DBMS_PLSQL_CODE_COVERAGE` package.

1. Run the procedure `CREATE_COVERAGE_TABLES` to create the tables required by the package to store the coverage data. You only need to run this step once as part of setup.

   ```
   EXECUTE DBMS_PLSQL_CODE_COVERAGE.CREATE_COVERAGE_TABLES;
   ```

2. Start the coverage run.

   ```
   DECLARE    testsuite_run NUMBER;
   BEGIN
      testsuite_run :=  DBMS_PLSQL_CODE_COVERAGE.START_COVERAGE(RUN_COMMENT => 'Test
   Suite ABC');
   END;
   /
   ```

3. Run the tests.
4. Stop the coverage collection and write the data to the collection tables.

   ```
   EXECUTE  DBMS_PLSQL_CODE_COVERAGE.STOP_COVERAGE;
   ```

5. Query the coverage tables to inspect results.

# 17.3 PL/SQL Code Coverage Tables Description

The following tables are created by the `PLSQL_CODE_COVERAGE.CREATE_COVERAGE_TABLES` procedure to collect code coverage data.

The `DBMSPCC_RUNS` table contains one row for each execution of the `DBMS_PLSQL_CODE_COVERAGE.START_COVERAGE` function. The primary key is the `RUN_ID`.

**Table 17-1 DBMSPCC_RUNS Table Columns**

| Column Name | Column Data Type | Description |
|---|---|---|
| RUN_ID | NUMBER(38) | Unique identifier automatically generated for this run of DBMS_PLSQL_CODE_COVERAGE.START_COVERAGE |
| RUN_COMMENT | VARCHAR2(4000) | User comment to identify the run |
| RUN_OWNER | VARCHAR2(128) | User who started the run |
| RUN_TIMESTAMP | DATE | Date timestamp when the run started |

The DBMSPCC_UNITS table contains the PL/SQL units information exercised in a run. The primary key is RUN_ID, and OBJECT_ID. The OBJECT_ID and LAST_DDL_TIME allows you to determine if a unit has been modified since the run started by comparing to the object LAST_DDL_TIME in the static data dictionary view ALL_OBJECTS.

**Table 17-2 DBMSPCC_UNITS Table Columns**

| Column Name | Column Data Type | Description |
|---|---|---|
| RUN_ID | NUMBER | References the RUN_ID column in the DBMSPCC_RUNS table |
| OBJECT_ID | NUMBER | Unique identifier for the unit |
| OWNER | VARCHAR2(128) | Owner of the unit |
| NAME | VARCHAR2(128) | Unit name |
| TYPE | VARCHAR2(12) | Unit type |
| LAST_DDL_TIME | DATE | Date timestamp for the last modification of the unit resulting from a DDL statement captured at run time |

The DBMSPCC_BLOCKS table identifies all the blocks in a unit. The block location is indicated by its starting position in the source code (LINE, COL). The primary key is RUN_ID, OBJECT_ID and BLOCK. It is implicit that one block ends at the character position immediately before that of the start of the next. More than one block can start at the same location. If a unit has not been modified since the run started, the source code lines can be extracted from the static data dictionary view ALL_SOURCE.

**Table 17-3 DBMSPCC_BLOCKS Table Columns**

| Column Name | Column Data Type | Description |
|---|---|---|
| RUN_ID | NUMBER(38) | References the RUN_ID column in the DBMSPCC_UNITS table |
| OBJECT_ID | NUMBER(38) | References the OBJECT_ID in the DBMSPCC_UNITS table |
| BLOCK | NUMBER(38) | Basic block number |
| LINE | NUMBER(38) | Starting line number of the basic block |

**Table 17-3 (Cont.) DBMSPCC_BLOCKS Table Columns**

| Column Name | Column Data Type | Description |
|---|---|---|
| COL | NUMBER(38) | Starting column number of basic block |
| COVERED | NUMBER(1) | Set to 1 if a basic block is covered, or to 0 otherwise |
| NOT_FEASIBLE | NUMBER(1) | Set to 1 if a basic block is marked as NOT_FEASIBLE, or to 0 otherwise |