

















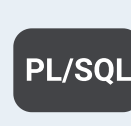



























-  Secrets
-  ABAP
-  Apex
-  C
-  C++
-  CloudFormation
-  COBOL
-  C#
-  CSS
-  Flex
-  Go
-  HTML
-  Java
-  JavaScript
-  Kotlin
-  Kubernetes
-  Objective C
-  PHP
-  PL/I
-  **PL/SQL**
-  Python
-  RPG
-  Ruby
-  Scala
-  Swift
-  Terraform
-  Text
-  TypeScript
-  T-SQL
-  VB.NET
-  VB6
-  XML

PL/SQL

PL/SQL static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your PL/SQL code

- All rules 188
-  Vulnerability 4
-  Bug 45
-  Security Hotspot 2
-  Code Smell 137




Identifiers should be written in lower case
 Code Smell
"PLS_INTEGER" types should be used
 Code Smell
Reserved words should be written in upper case
 Code Smell
Parameter "IN" mode should be specified explicitly
 Code Smell
Lines in a multiline comment should start with "x"
 Code Smell
CASE should be used for sequences of simple tests
 Code Smell
SQL tables should be joined with the "JOIN" keyword
 Code Smell
Constraint names should comply with a naming convention
 Code Smell
Reserved words should be written in lower case
 Code Smell
Comments should not be located at the end of lines of code
 Code Smell
Lines should not end with trailing whitespaces
 Code Smell
The "RELIES_ON" clause should not be used
 Code Smell

Tags ▾

Search by name... 🔍

"PLS_INTEGER" types should be used

Analyze your code

 Code Smell  Minor ?  performance

Using a `NUMBER` to store an integer is less performant than using a `PLS_INTEGER`. `PLS_INTEGER`s require less storage than `NUMBERS`, and benefit from the use of hardware math, as opposed to the library math required for `NUMBERS`. Even more performant is the `SIMPLE_INTEGER` subtype of `PLS_INTEGER`. However, changing to either of these types is only appropriate under certain circumstances.

PLS_INTEGER is only a candidate for `NUMBER` with a scale of up to 9.

SIMPLE_INTEGER has the same size limitation, in addition to it's `NOT NULL` constraint and lack of overflow checking.

This rule raises an issue when a `NUMBER` is declared with a scale of 9 or less.




Noncompliant Code Example

```
DECLARE
    son NUMBER(1);           -- Noncompliant
    rumbo NUMBER(9);         -- Noncompliant
    conga Number(10);        -- Ignored; falls outside the PLS_INTEGER range
    compalsa PLS_INTEGER;
```

Compliant Solution

```
DECLARE
    son SIMPLE_INTEGER;
    rumbo PLS_INTEGER;
    conga Number(10);        -- Ignored; falls outside the PLS_INTEGER range
    compalsa PLS_INTEGER;
```

Available In:

 |  |  Developer Edition