The Trivadis PL/SQL & SQL Coding Guidelines Are Dead – What Now?

Philipp Salvisberg
14th May 2025

GRISSELBAV
simple & accurate IT solutions.

Welcome

Philipp Salvisberg
Founder, Owner and CEO of Grisselbav GmbH

- Database-centric Development
- Model-driven Software Development
- Open-Source Development

philipp.salvisberg@grisselbav.com
https://www.salvis.com/blog/

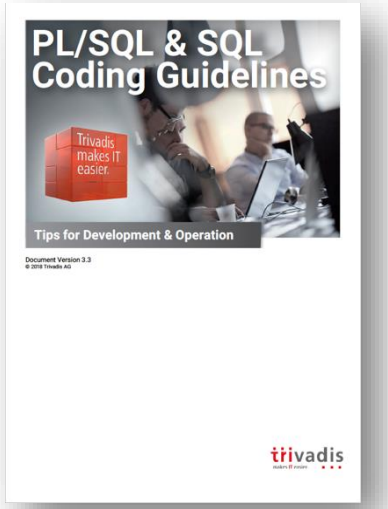
Introduction (History)

Trivadis PL/SQL & SQL Coding Guidelines

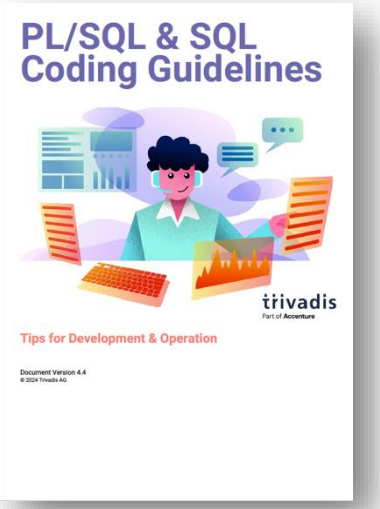
v1.2
August 2009
52 pages
Word

v2.0
August 2011
57 pages
Word

v3.0
January 2016
149 pages
Word

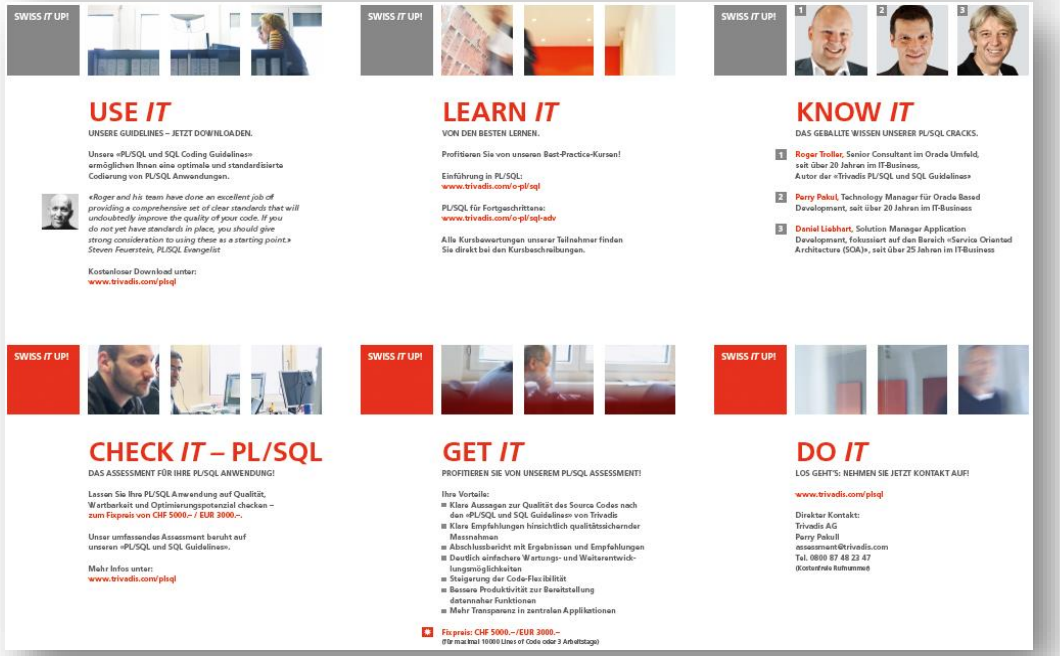


v3.3
July 2018
141 pages
MkDocs



v4.4
March 2024
208 pages
MkDocs

Tooling in 2009/2010?



- Cook-Book
  - Quest CodeXpert
  - SQL scripts with/without PL/Scope
  - Manual code reviews
  - Interviews

No custom rules
Sonar used also
CodeXpert back then

renamed to
PL/SQL Cop,
db* CODECOP

Code Checker Prototype
April 2010
G-3110, G-5020, G-6010

Requirements (My Whish List)

Guidelines

Similar rules as in the Trivadis Guidelines

Configurable

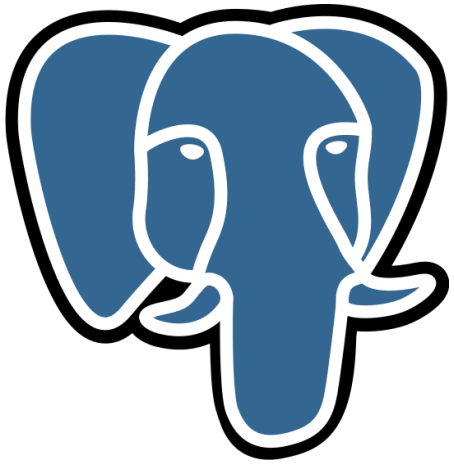
Extensible

Regular updates

Grammar for Static Code Analysis

- Oracle Database
  - SQL*Plus
  - SQLcl
  - SQL
  - PL/SQL

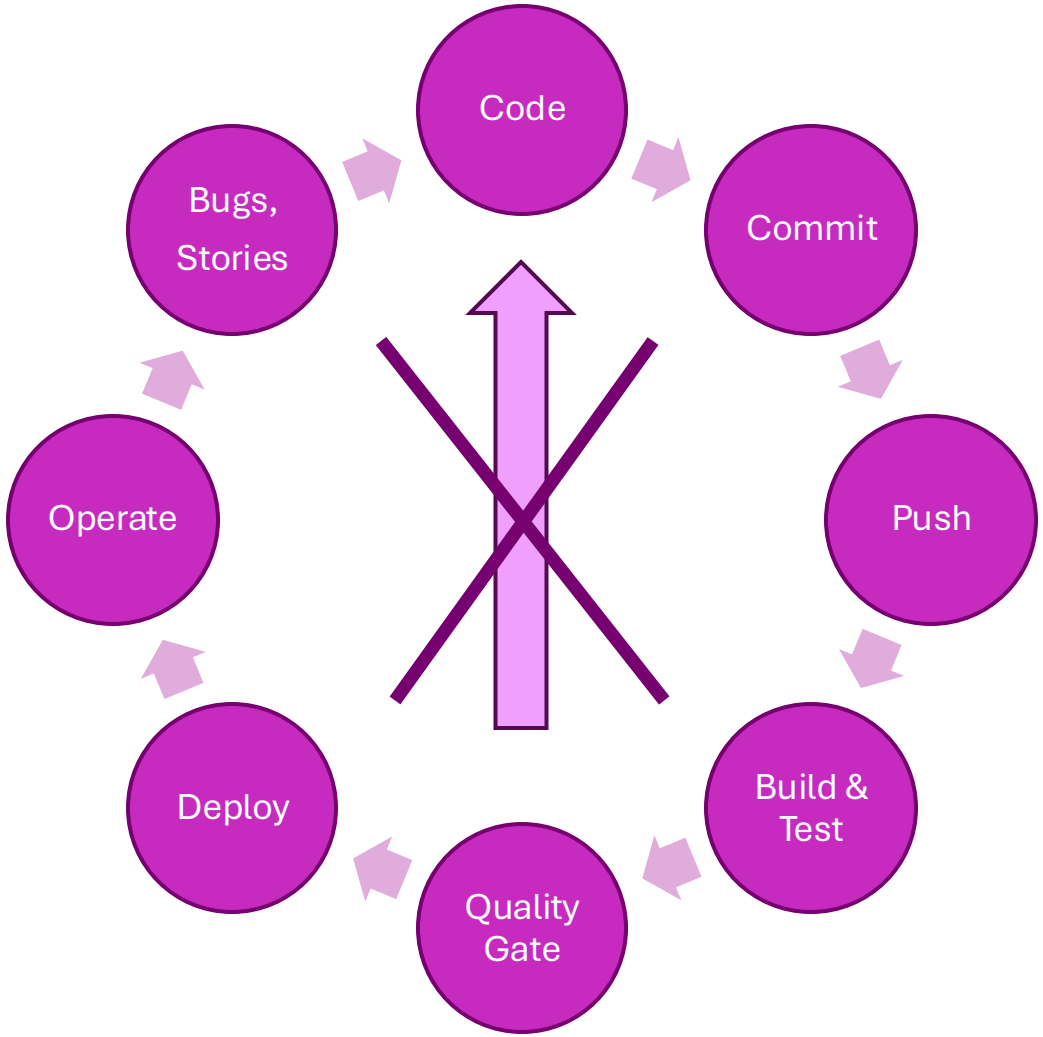
- PostgreSQL
  - psql
  - SQL
  - PL/pgSQL



PostgreSQL 17

- Regular updates

DevOps Lifecyle – Focus Development



Short feedback loop with IDE support

IDE integration is mandatory

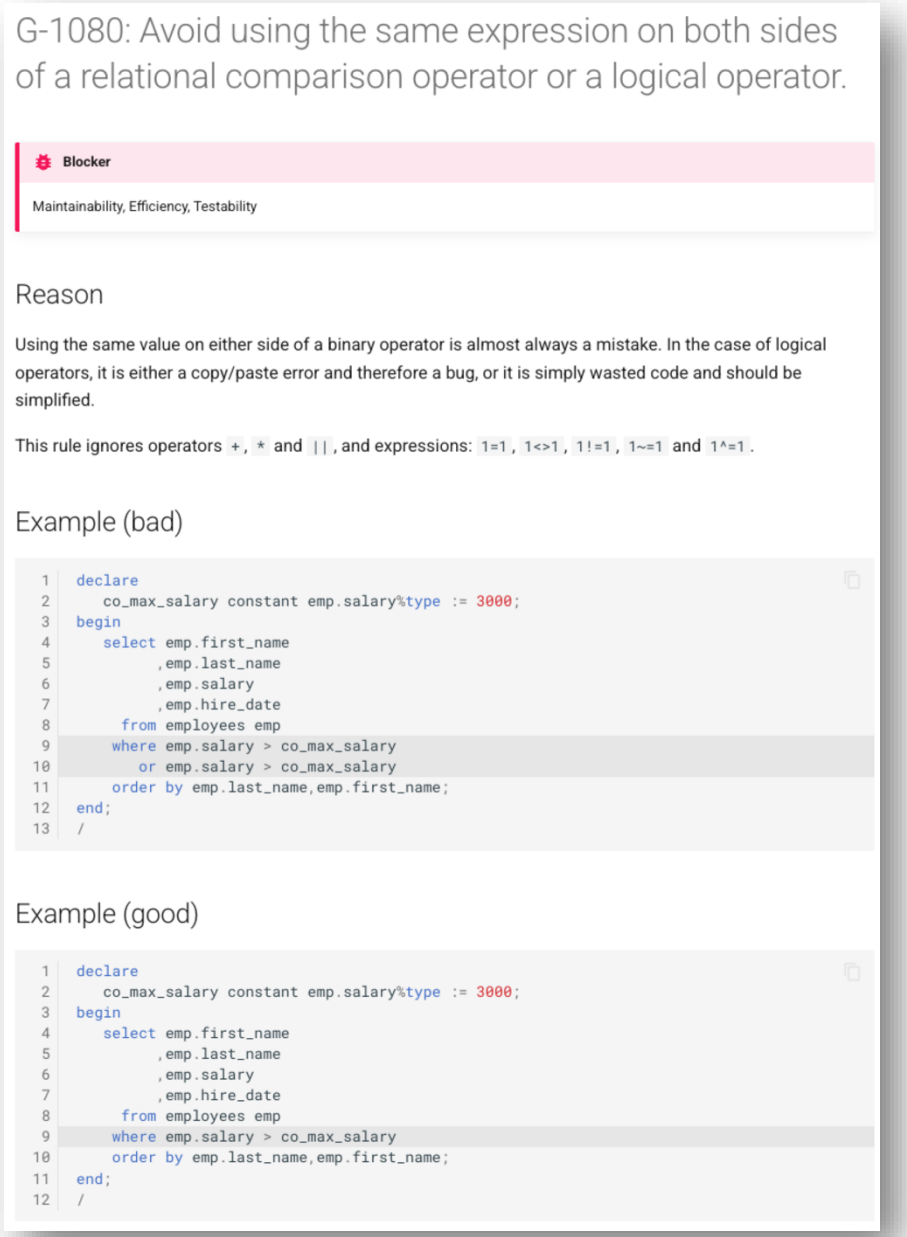
Tooling

- Integration
  - IDEs (VS Code, SQL Developer, PL/SQL Developer, TOAD, Data Grip, DBeaver, …)
  - Build Pipelines (Jenkins, Bamboo, GitHub, GitLab, Bitbucket, OCI, Azure, AWS, Google Cloud, …)
  - Code Quality Repositories (SonarQube Community Build/Server/Cloud, Codacy, …)
- Static code analysis
  - Report issues with clear messages and quick fixes, if applicable
  - Configurable rules (parameters and SQL lookups)
  - Custom rules
- SQL-based Test
  - Implement rules via SQL queries returning issues and migration scripts if applicable
- Shared configuration

Guidelines (for Static Code Analysis)

Trivadis PL/SQL & SQL Coding Guidelines

- 124 rules
  - PL/SQL
  - Oracle SQL
- Apache-2.0 license
- Fork on GitHub
- SQLDev extension; CLI; SonarQube plugin



G-1080: Avoid using the same expression on both sides of a relational comparison operator or a logical operator.

Blocker

Maintainability, Efficiency, Testability

Reason

Using the same value on either side of a binary operator is almost always a mistake. In the case of logical operators, it is either a copy/paste error and therefore a bug, or it is simply wasted code and should be simplified.

This rule ignores operators `+`, `*` and `||`, and expressions: `1=1`, `1<>1`, `1!=1`, `1~=1` and `1^=1`.

Example (bad)

```
1  declare
2     co_max_salary constant emp.salary%type := 3000;
3  begin
4     select emp.first_name
5           ,emp.last_name
6           ,emp.salary
7           ,emp.hire_date
8       from employees emp
9      where emp.salary > co_max_salary
10       or emp.salary > co_max_salary
11     order by emp.last_name,emp.first_name;
12 end;
13 /
```

Example (good)

```
1  declare
2     co_max_salary constant emp.salary%type := 3000;
3  begin
4     select emp.first_name
5           ,emp.last_name
6           ,emp.salary
7           ,emp.hire_date
8       from employees emp
9      where emp.salary > co_max_salary
10     order by emp.last_name,emp.first_name;
11 end;
12 /
```

Source: https://trivadis.github.io/plsql-and-sql-coding-guidelines/v4.4/4-language-usage/1-general/g-1080/

SQLcl Codescan

- 106 Rules
  - 94 Trivadis Coding Guidelines
  - 12 PSR rules
  - Implemented as Arbori Queries
  - No list of supported rules
- NFTC-license
- CLI

```
sql -nolog <<EOF
codescan -path d2-1080-nc.sql -format json -ignore G-1030,G-5010
exit
EOF

SQLcl: Release 25.1 Production on Sun May 04 19:42:42 2025

Copyright (c) 1982, 2025, Oracle.  All rights reserved.

1 files, 1 total distinct warnings
[
{"file": "d2-1080-nc.sql", "issues": [
    { "line": 7, "col": 7, "ruleNo": "G-1080", "msg": "Avoid using the same expression on both sides of a relational comparison operator or a logical operator"}
]}
]
```

Source: https://docs.oracle.com/en/database/oracle/sql-developer-command-line/25.1/sqcug/codescan.html

PMD

- 21 rules
  - PL/SQL
- BSD-style license
- Fork on GitHub
- CLI

TomKytesDespair

Since: PMD 5.1

Priority: Medium (3)

"WHEN OTHERS THEN NULL" hides all errors - (Re)RAISE an exception or call RAISE_APPLICATION_ERROR

This rule is defined by the following XPath expression:

```
//ExceptionHandler[QualifiedName/@Image='OTHERS']
                  [count(Statement)=1]
                  [Statement/UnlabelledStatement/Expression/PrimaryPrefix/Literal/NullLiteral]
```

Example(s):

```
CREATE OR REPLACE PACKAGE BODY update_planned_hrs
IS

PROCEDURE set_new_planned (p_emp_id IN NUMBER, p_project_id IN NUMBER, p_hours IN NUMBER)
IS
BEGIN
   UPDATE employee_on_activity ea
   SET ea.ea_planned_hours = p_hours
   WHERE
            ea.ea_emp_id = p_emp_id
            AND ea.ea_proj_id = p_project_id;

EXCEPTION
          WHEN NO_DATA_FOUND THEN
          RAISE_APPLICATION_ERROR (-20100, 'No such employee or project');

END set_new_planned;

FUNCTION existing_planned (p_emp_id IN NUMBER, p_project_id IN NUMBER) RETURN NUMBER

IS

existing_hours NUMBER(4);

BEGIN
   SELECT ea.ea_planned_hours INTO existing_hours
   FROM employee_on_activity ea
   WHERE
            ea.ea_emp_id = p_emp_id
            AND ea.ea_proj_id = p_project_id;

   RETURN (existing_hours);

   EXCEPTION
          WHEN OTHERS THEN NULL;

   END existing_planned;

END update_planned_hrs;
/
```

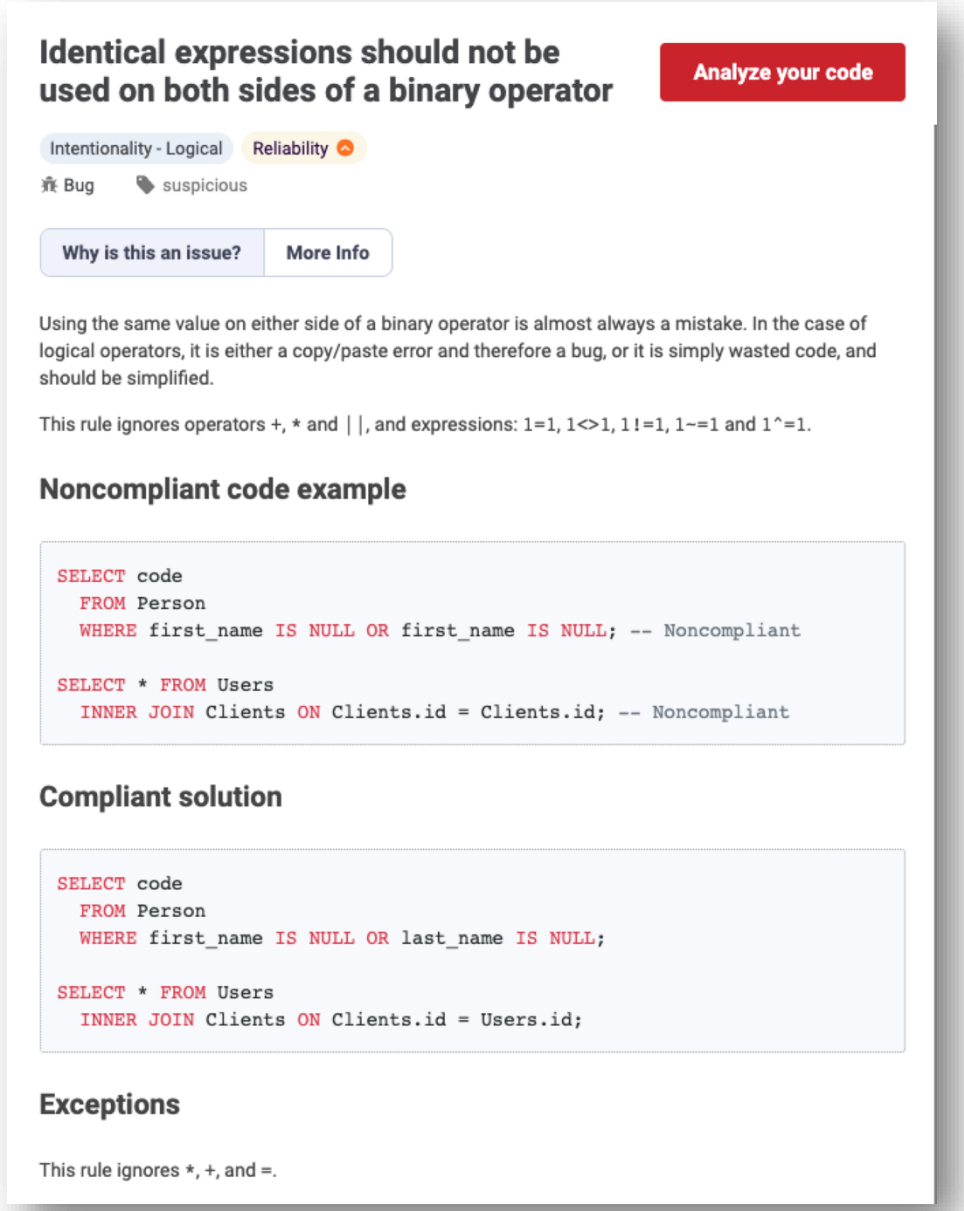
Use this rule by referencing it:

```
<rule ref="category/plsql/bestpractices.xml/TomKytesDespair" />
```

Source: https://docs.pmd-code.org/latest/pmd_rules_plsql_bestpractices.html#tomkytesdespair

SonarQube

- 189 rules (≥ Developer Edition)
  - PL/SQL
  - Oracle SQL
- 89 rules (≥ Developer Edition)
  - T-SQL
  - SQL Sever SQL
- LGPL-3.0 license
  (only for parts of Community Build)
- XPath 1.0 based custom rules
- VS Code extension, DataGrip plugin;
  CLI; SonarQube plugin



Identical expressions should not be used on both sides of a binary operator

Analyze your code

Intentionality - Logical | Reliability

Bug | suspicious

Why is this an issue? | More Info

Using the same value on either side of a binary operator is almost always a mistake. In the case of logical operators, it is either a copy/paste error and therefore a bug, or it is simply wasted code, and should be simplified.

This rule ignores operators +, * and | |, and expressions: 1=1, 1<>1, 1!=1, 1~=1 and 1^=1.

Noncompliant code example

```
SELECT code
  FROM Person
  WHERE first_name IS NULL OR first_name IS NULL; -- Noncompliant

SELECT * FROM Users
  INNER JOIN Clients ON Clients.id = Clients.id; -- Noncompliant
```

Compliant solution

```
SELECT code
  FROM Person
  WHERE first_name IS NULL OR last_name IS NULL;

SELECT * FROM Users
  INNER JOIN Clients ON Clients.id = Users.id;
```

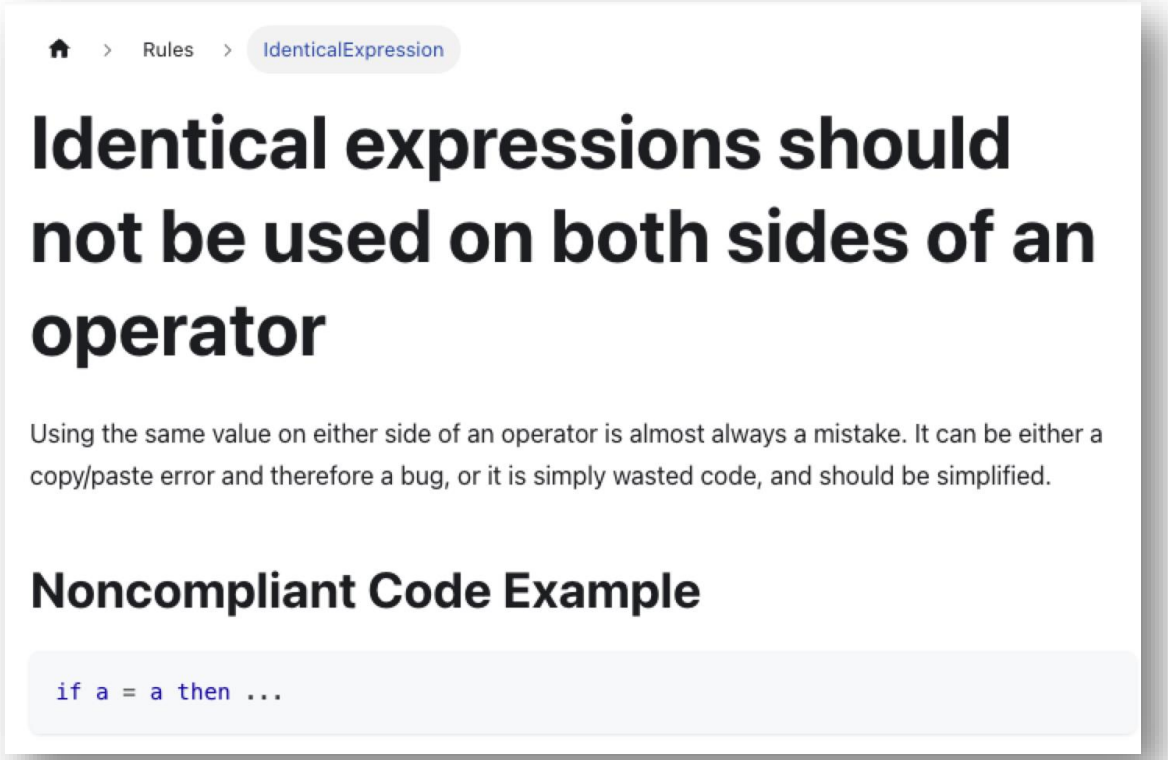
Exceptions

This rule ignores *, +, and =.

Source: https://rules.sonarsource.com/plsql/RSPEC-1764/

ZPA

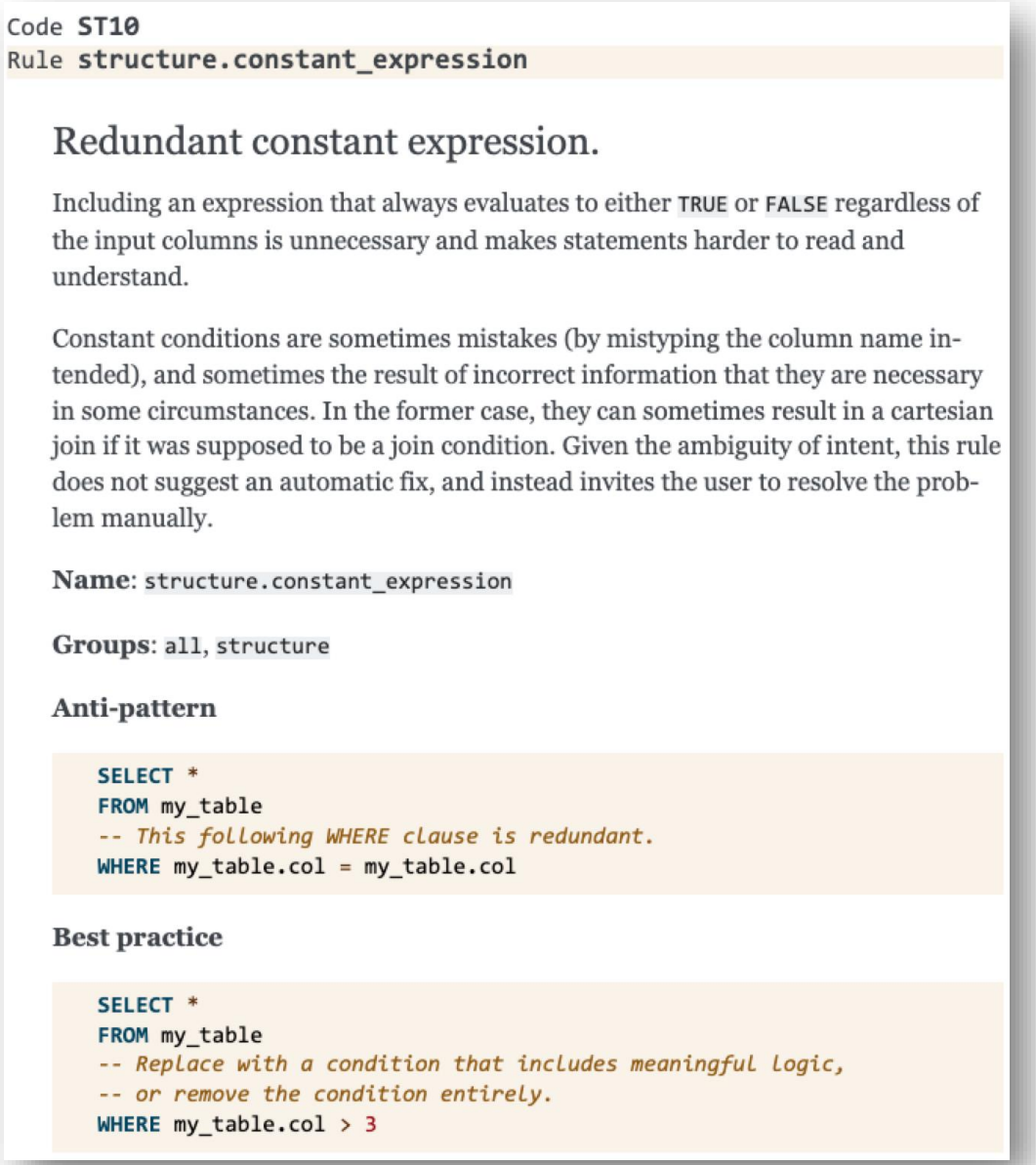
- 54 rules (all versions of SonarQube Server)
  - PL/SQL
  - Oracle SQL
- LGPL-3.0 license
- Fork on GitHub
- XPath 1.0 based custom rules
- CLI; SonarQube plugin



Source: https://zpa.felipebz.com/docs/rules/IdenticalExpression/

SQLFluff

- 67 rules
  - SQL (ANSI, Athena, BigQuery, Clickhouse Databricks, Db2, DuckDB, Exasol, Greenplum, Hive, Impala, MariaDB, Materialize, MySQL, Oracle, PostgreSQL, Redshift, Snowflake, SOQL, Spark SQL, SQLite, StarRocks, Teradata, Trino, T-SQL, Vertica)
- MIT license
- Fork on GitHub
- VS Code extension; CLI



Code **ST10**
Rule **structure.constant_expression**

Redundant constant expression.

Including an expression that always evaluates to either `TRUE` or `FALSE` regardless of the input columns is unnecessary and makes statements harder to read and understand.

Constant conditions are sometimes mistakes (by mistyping the column name intended), and sometimes the result of incorrect information that they are necessary in some circumstances. In the former case, they can sometimes result in a cartesian join if it was supposed to be a join condition. Given the ambiguity of intent, this rule does not suggest an automatic fix, and instead invites the user to resolve the problem manually.

Name: `structure.constant_expression`

Groups: `all`, `structure`

Anti-pattern

```
SELECT *
FROM my_table
-- This following WHERE clause is redundant.
WHERE my_table.col = my_table.col
```

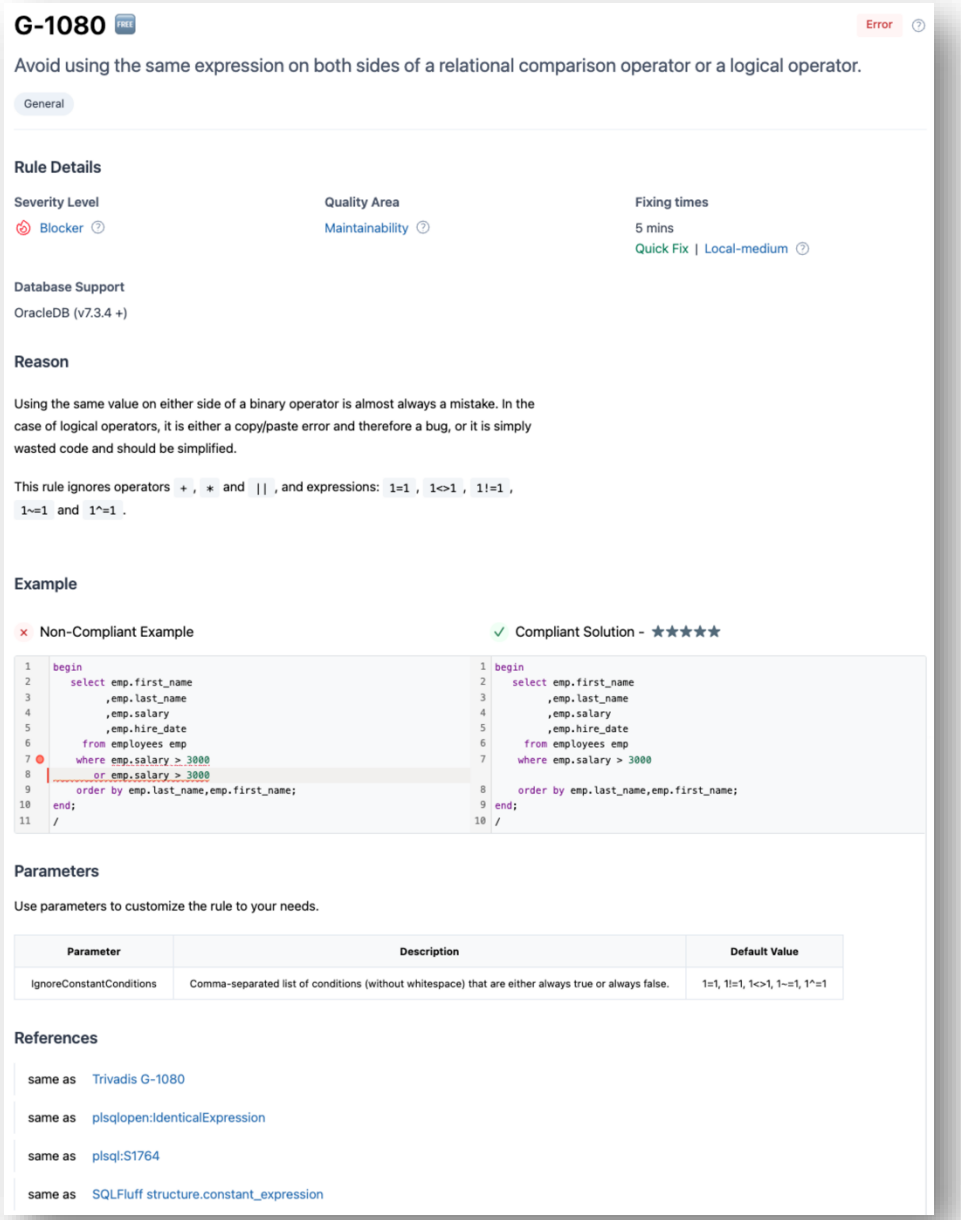
Best practice

```
SELECT *
FROM my_table
-- Replace with a condition that includes meaningful logic,
-- or remove the condition entirely.
WHERE my_table.col > 3
```

Source: https://docs.sqlfluff.com/en/stable/reference/rules.html#rule-structure.constant_expression

dbLinter

- 183 rules
  - Oracle Database (≥ 7.3.4): SQL, PL/SQL, SQL*Plus, SQLcl
  - PostgreSQL (≥ 7.0.3): SQL, PL/pgSQL, psql
- IslandSQL parser (Apache-2.0 license)
- EULA
- Joint Venture between Grisselbav & United Codes
- Free plans
- Custom rules with professional subscription
- VS Code extension, CLI, SonarQube plugin



G-1080 FREE — Error

Avoid using the same expression on both sides of a relational comparison operator or a logical operator.

General

Rule Details

Severity Level	Quality Area	Fixing times
Blocker	Maintainability	5 mins Quick Fix | Local-medium

Database Support

OracleDB (v7.3.4 +)

Reason

Using the same value on either side of a binary operator is almost always a mistake. In the case of logical operators, it is either a copy/paste error and therefore a bug, or it is simply wasted code and should be simplified.

This rule ignores operators `+`, `*` and `||`, and expressions: `1=1`, `1<>1`, `1!=1`, `1~=1` and `1^=1`.

Example

Non-Compliant Example

```
begin
   select emp.first_name
         ,emp.last_name
         ,emp.salary
         ,emp.hire_date
     from employees emp
    where emp.salary > 3000
       or emp.salary > 3000
    order by emp.last_name,emp.first_name;
end;
/
```

Compliant Solution - ★★★★★

```
begin
   select emp.first_name
         ,emp.last_name
         ,emp.salary
         ,emp.hire_date
     from employees emp
    where emp.salary > 3000

    order by emp.last_name,emp.first_name;
end;
/
```

Parameters

Use parameters to customize the rule to your needs.

Parameter	Description	Default Value
IgnoreConstantConditions	Comma-separated list of conditions (without whitespace) that are either always true or always false.	1=1, 1!=1, 1<>1, 1~=1, 1^=1

References

same as Trivadis G-1080

same as plsqlopen:IdenticalExpression

same as plsql:S1764

same as SQLFluff structure.constant_expression

Source: https://dblinter-rules.united-codes.com/rules/G-1080/

dbLinter Rules RoadMap

Status	Trivadis	PMD	SonarQube	ZPA	SQLFluff
Included	124	4	88	24	8
To include	-	1	27	17	16
Do not include	-	16	74	13	43
Total	**124**	**21**	**189**	**54**	**67**

Examples of rules not considered:

- PMD: AvoidTabCharacter (code style)
- SonarQube: Constant declarations should contain initialization assignments (copying the compiler)

Examples of rules unlikely to be considered:

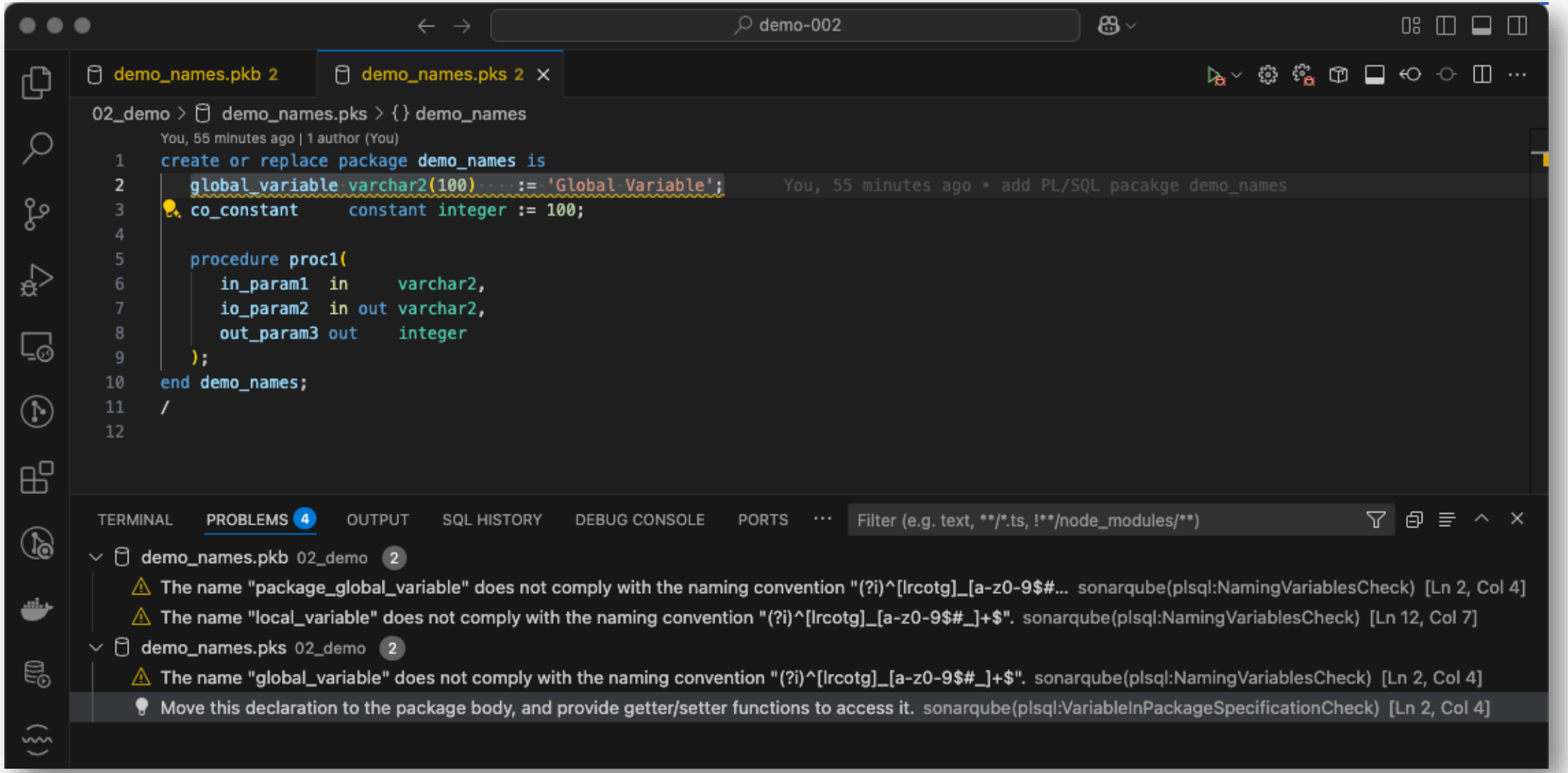
- SonarQube: An "ORDER BY" direction should be specified explicitly (ASC or DESC)
- ZPA: Add parentheses in nested expression (regardless of default precedence)
- SQLFluff: Implicit/explicit aliasing of columns (AS keyword)

Static Code Analysis Demo

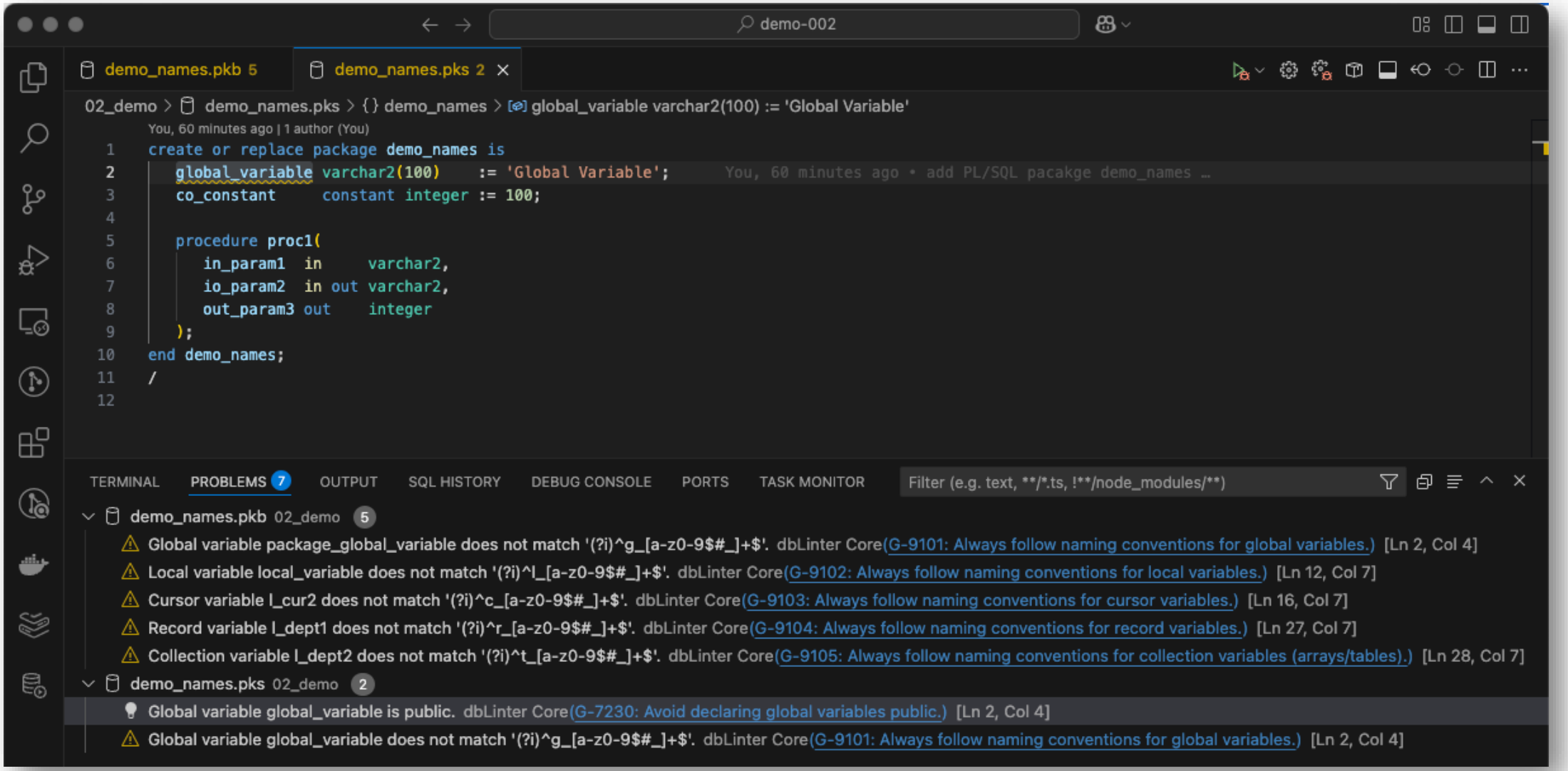
PL/SQL Naming Conventions

Identifier Type	dbLinter Rule	dbLinter Regex	SonarQube Rule	SonarQube Regex
Global Variable	G-9101	(?i)^g_[a-z0-9$#_]+$	plsql:NamingVariablesCheck	(?i)^[lrcotg]_[a-z0-9$#_]+$
Local Variable	G-9102	(?i)^l_[a-z0-9$#_]+$	plsql:NamingVariablesCheck	(?i)^[lrcotg]_[a-z0-9$#_]+$
Cursor Variable	G-9103	(?i)^c_[a-z0-9$#_]+$	plsql:NamingCursorsCheck	(?i)^c_[a-z0-9$#_]+$
Record Variable	G-9104	(?i)^r_[a-z0-9$#_]+$	plsql:NamingVariablesCheck	(?i)^[lrcotg]_[a-z0-9$#_]+$
Collection Variable	G-9105	(?i)^t_[a-z0-9$#_]+$	plsql:NamingVariablesCheck	(?i)^[lrcotg]_[a-z0-9$#_]+$
Object Variable	G-9106	(?i)^o_[a-z0-9$#_]+$	plsql:NamingVariablesCheck	(?i)^[lrcotg]_[a-z0-9$#_]+$
Cursor Parameter	G-9107	(?i)^p_[a-z0-9$#_]+$	plsql:NamingCursorParametersCheck	(?i)^p_[a-z0-9$#_]+$
IN Parameter	G-9108	(?i)^in_[a-z0-9$#_]+$	plsql:NamingFunctionAndProcedureParametersCheck	(?i)^(in|out|io)_[a-z0-9$#_]+$
OUT Parameter	G-9109	(?i)^out_[a-z0-9$#_]+$	plsql:NamingFunctionAndProcedureParametersCheck	(?i)^(in|out|io)_[a-z0-9$#_]+$
IN/OUT Parameter	G-9110	(?i)^io_[a-z0-9$#_]+$	plsql:NamingFunctionAndProcedureParametersCheck	(?i)^(in|out|io)_[a-z0-9$#_]+$
Record Type Definition	G-9111	(?i)^r_[a-z0-9$#_]+_type$	plsql:NamingTypesCheck	(?i)^([rt]_)?[a-z0-9$#_]+_(ot\|ct\|type)$
Collection Type Definition	G-9112	(?i)^t_[a-z0-9$#_]+_type$	plsql:NamingTypesCheck	(?i)^([rt]_)?[a-z0-9$#_]+_(ot\|ct\|type)$
Exception	G-9113	(?i)^e_[a-z0-9$#_]+$	plsql:NamingExceptionsCheck	(?i)^e_[a-z0-9$#_]+$
Constant	G-9114	(?i)^co_[a-z0-9$#_]+$	plsql:NamingConstantsCheck	(?i)^co_[a-z0-9$#_]+$
Subtype	G-9115	(?i)^[a-z][a-z0-9$#_]*_type$	plsql:NamingTypesCheck	(?i)^([rt]_)?[a-z0-9$#_]+_(ot\|ct\|type)$
Record Field	G-9116	(?i)^[a-z0-9$#_]+$	plsql:NamingRecordField	(?i)^[a-z0-9$#_]+$

SonarQube for IDE (VS Code)



dbLinter for VS Code

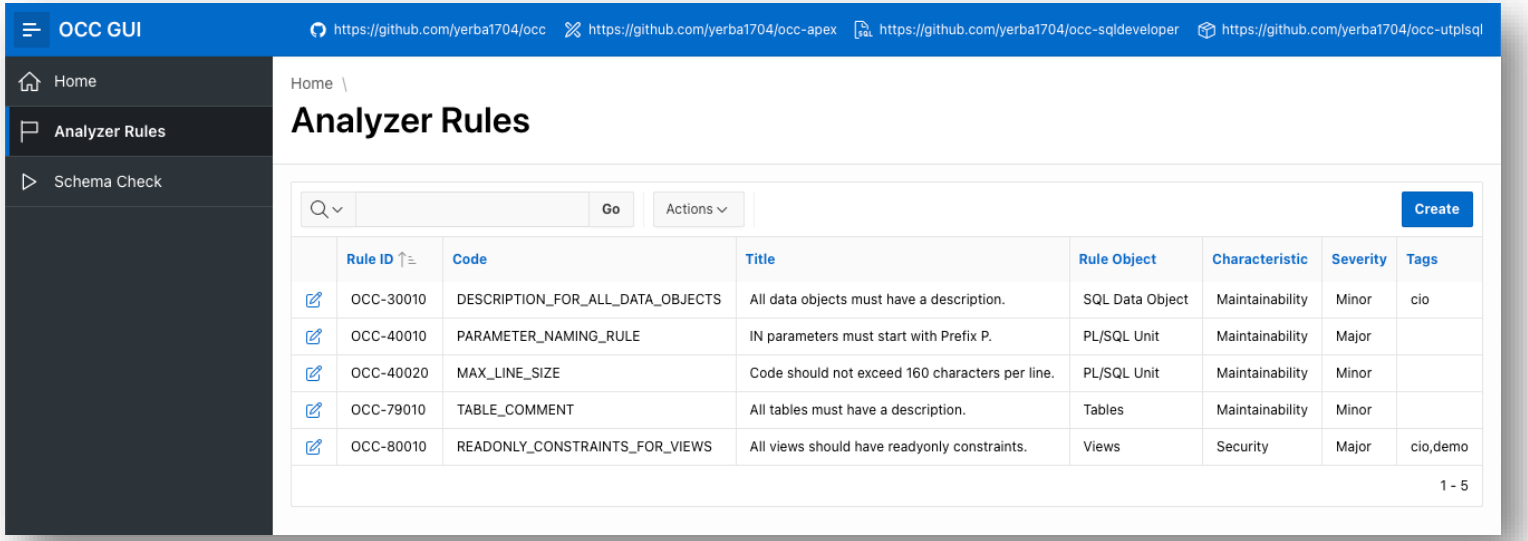


Tools for SQL-based Tests

utPLSQL

- Compare actual and expected SQL cursors
- Perfect for hard rules without exceptions
- Not suited for warnings
- No concept to share common tests across projects
- Output in UI or in JUnit XML format from CLI
- Apache-2.0 license
- Source of Core, CLI, IDE plugins on GitHub
- See documentation

ora* CODECOP

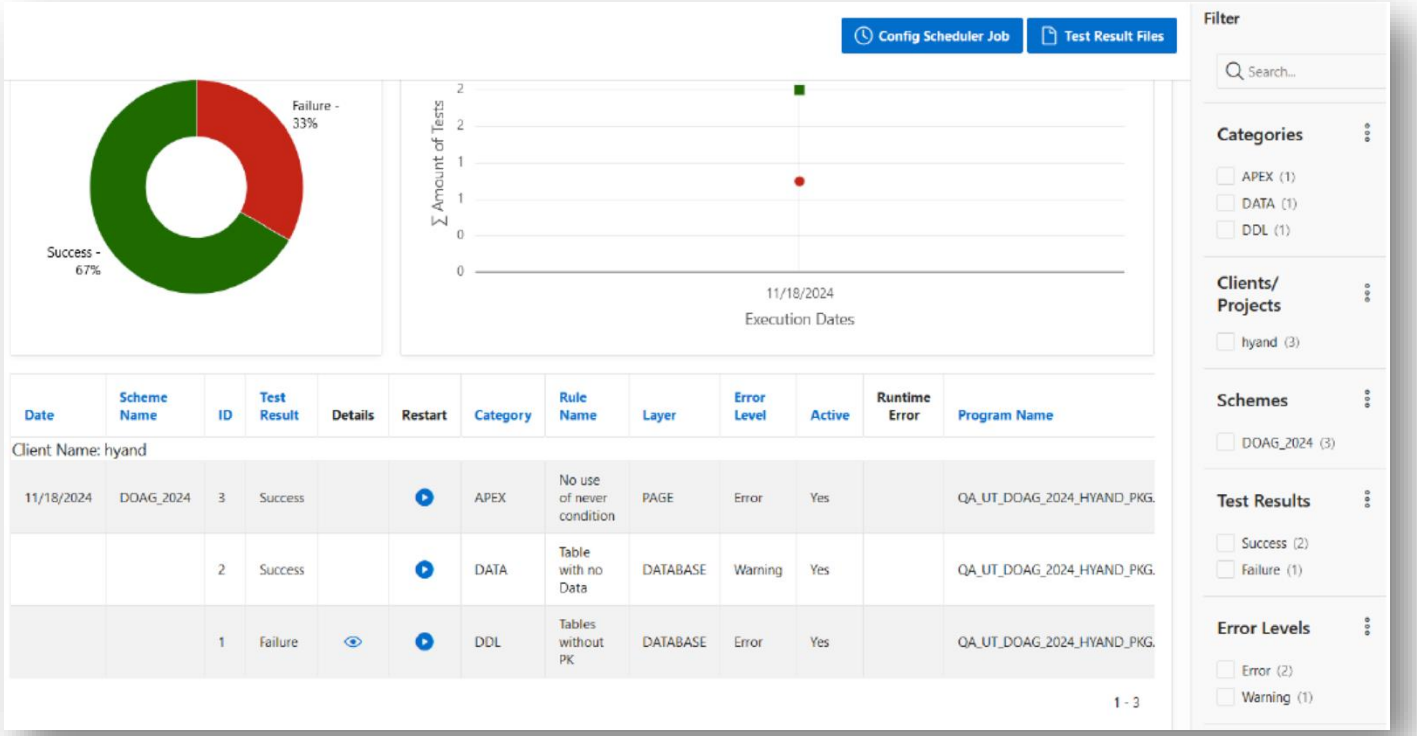


- Implements rules with SQL
- Rules in DB
- Results in DB
- MIT license
- Fork on GitHub
- Contributed by Markus Schulze, Helaba Invest Kapitalanlagegesellschaft mbH
- Presented at DOAG Conference + Exhibition 2023

Source: https://github.com/yerba1704/occ-apex

QUASTO

- Implements rules with SQL
- Rules in DB
- Results in DB of multiple schemas
- utPLSQL integration
- MIT license
- Fork on GitHub
- Product Lead is Oliver Lemm, Hyand Solutions GmbH
- Presented at DOAG Conference + Exhibition 2024



Source: https://apex.hyand.com/ords/r/portal/apex/quasto

dbLinter

- Implements rules with SQL or static code analysis
- Runs tests from client
- Ignore chosen findings
- Apply migration scripts (fixes)
- Output in UI or in JUnit XML format from CLI
- See rules with tests in dblinter.app

G-1240 FREE

Warning

Try to index foreign key columns.

Modeling Performance

▼ Test SQL query

```
with
   function create_fk_index_stmt (
       in_owner           in varchar2,
       in_table_name      in varchar2,
       in_constraint_name in varchar2
   ) return clob is
      co_templ constant varchar2(200 char) :=  'create index #IF_NOT_EXISTS# #OWNER#.#INDEX_NAME# on #OWNER#.#TABLE_NAME#(#COLS#);';
      l_stmt varchar2(4000 byte) := co_templ;
      l_cols varchar2(4000 byte);
   begin
      if dbms_db_version.version >= 23 then
         l_stmt := replace(l_stmt, '#IF_NOT_EXISTS#', 'if not exists');
      else
         l_stmt := replace(l_stmt, '#IF_NOT_EXISTS#', null);
      end if;
      l_stmt := replace(l_stmt, '#OWNER#', in_owner);
      l_stmt := replace(l_stmt, '#INDEX_NAME#', in_constraint_name || '_I');
      l_stmt := replace(l_stmt, '#TABLE_NAME#', in_table_name);
      select listagg(column_name, ', ') within group (order by position)
        into l_cols
        from dba_cons_columns
       where owner = in_owner
         and table_name = in_table_name
         and constraint_name = in_constraint_name;
      l_stmt := replace(l_stmt, '#COLS#', l_cols);
      return l_stmt;
   end create_fk_index_stmt;
select c.owner || '.' || c.constraint_name || '.' || cc.column_name as identifier,
       'Column ' || cc.column_name || ' in the foreign key constraint ' || c.constraint_name
          || ' of table ' || c.owner || '.' || c.table_name || ' is not indexed in the order of the constraint.' as message,
       create_fk_index_stmt(c.owner, c.table_name, c.constraint_name) as migration
  from dba_constraints c
  join dba_cons_columns cc
    on cc.owner = c.owner
       and cc.constraint_name = c.constraint_name
       and cc.table_name = c.table_name
  left join dba_ind_columns ic
       on ic.table_owner = c.owner
       and ic.table_name = c.table_name
       and ic.column_name = cc.column_name
       and ic.column_position = coalesce(cc.position, 1)
 where c.constraint_type = 'R'
   and c.status = 'ENABLED'
   and ic.column_name is null
   and c.owner in (#SchemaNames#)
 order by c.owner, c.constraint_name, c.table_name, cc.position, cc.column_name
```

Test results

Identifier	Message	Migration
DBL_OWNER.CHILD_PARENT_FK.PARENT_ID	Column PARENT_ID in the foreign key constraint CHILD_PARENT_FK of table DBL_OWNER.CHILD is not indexed in the order of the constraint.	create index if not exists DBL_OWNER.CHILD_PARENT_FK_I on DBL_OWNER.CHILD(PARENT_ID);

Parameters

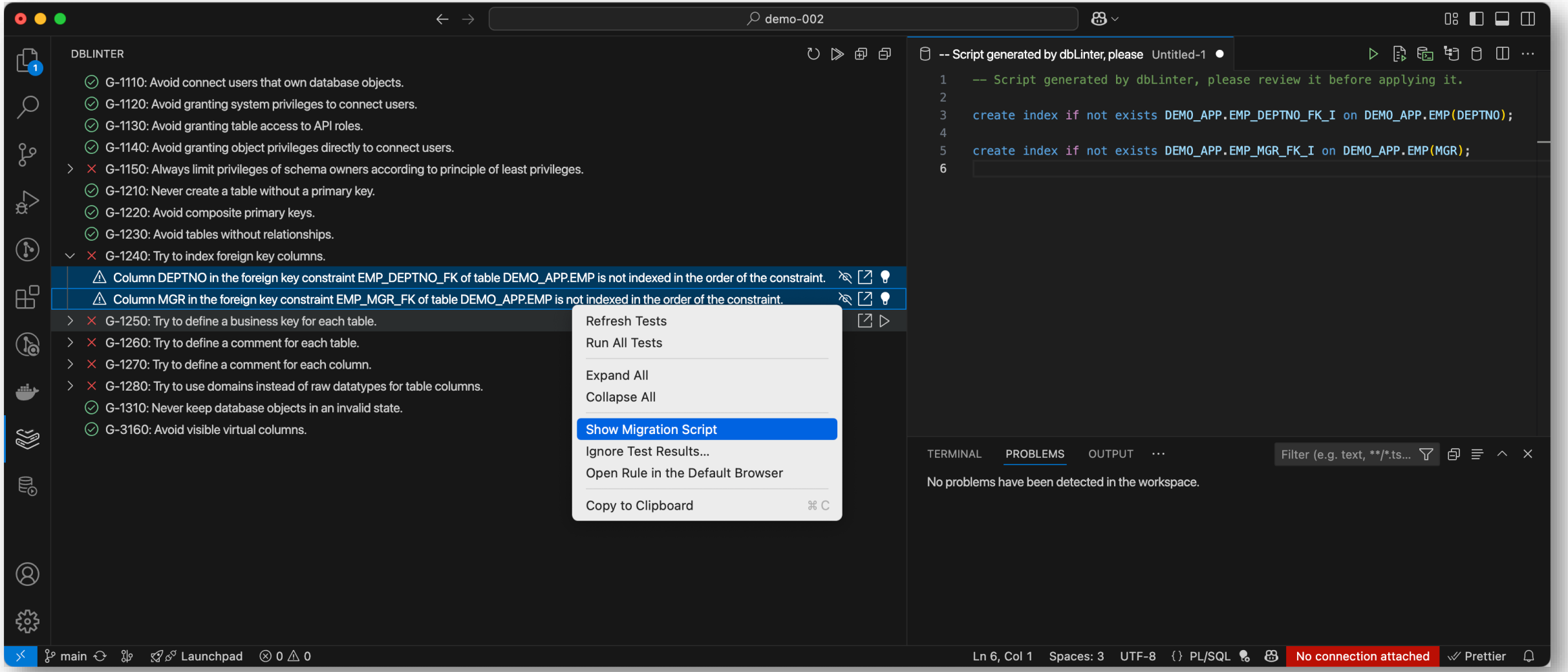
Use parameters to customize the rule to your needs.

Parameter	Description	Default Value
SchemaNames	Comma-separated list of database schemas owning the database objects of an application.	dbl_owner

Source: https://dblinter-rules.united-codes.com/rules/G-1240/

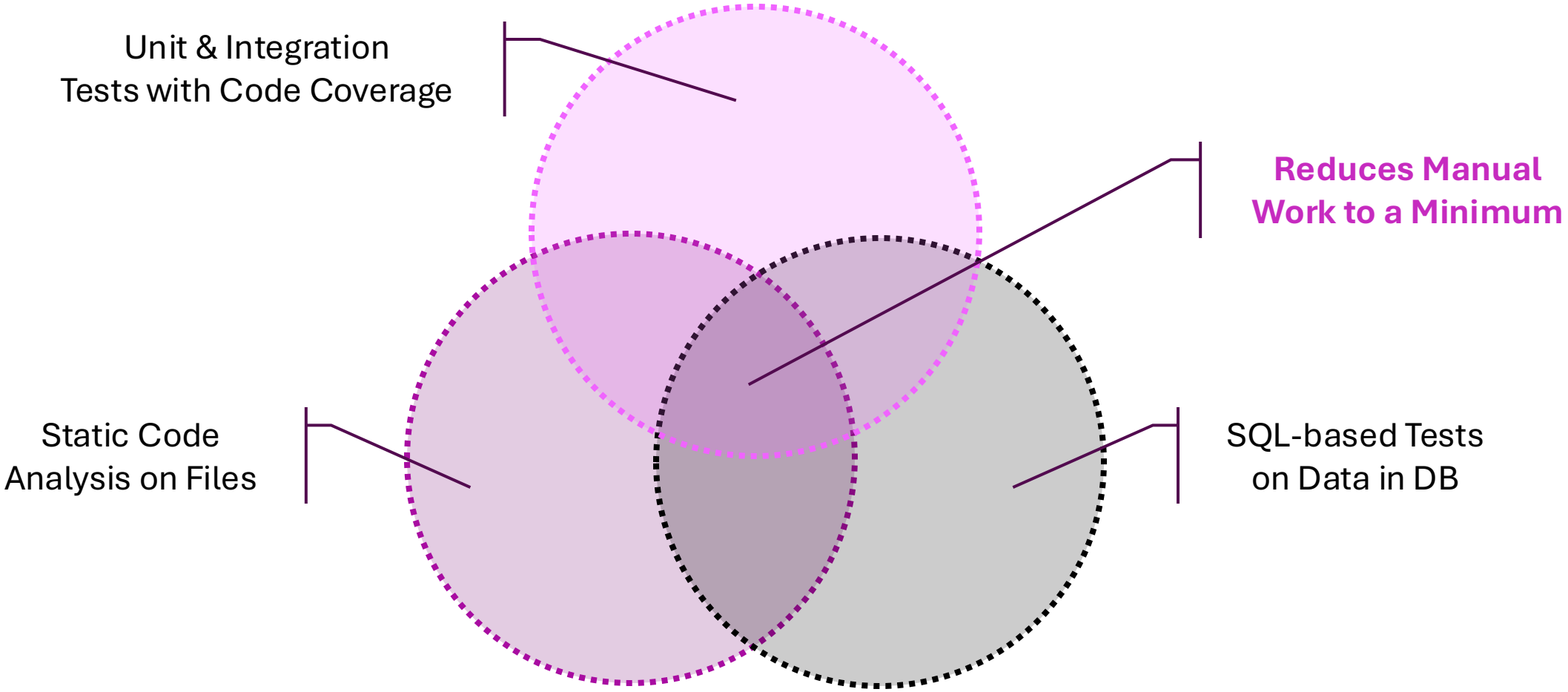
SQL-based Tests Demo

dbLinter for VS Code

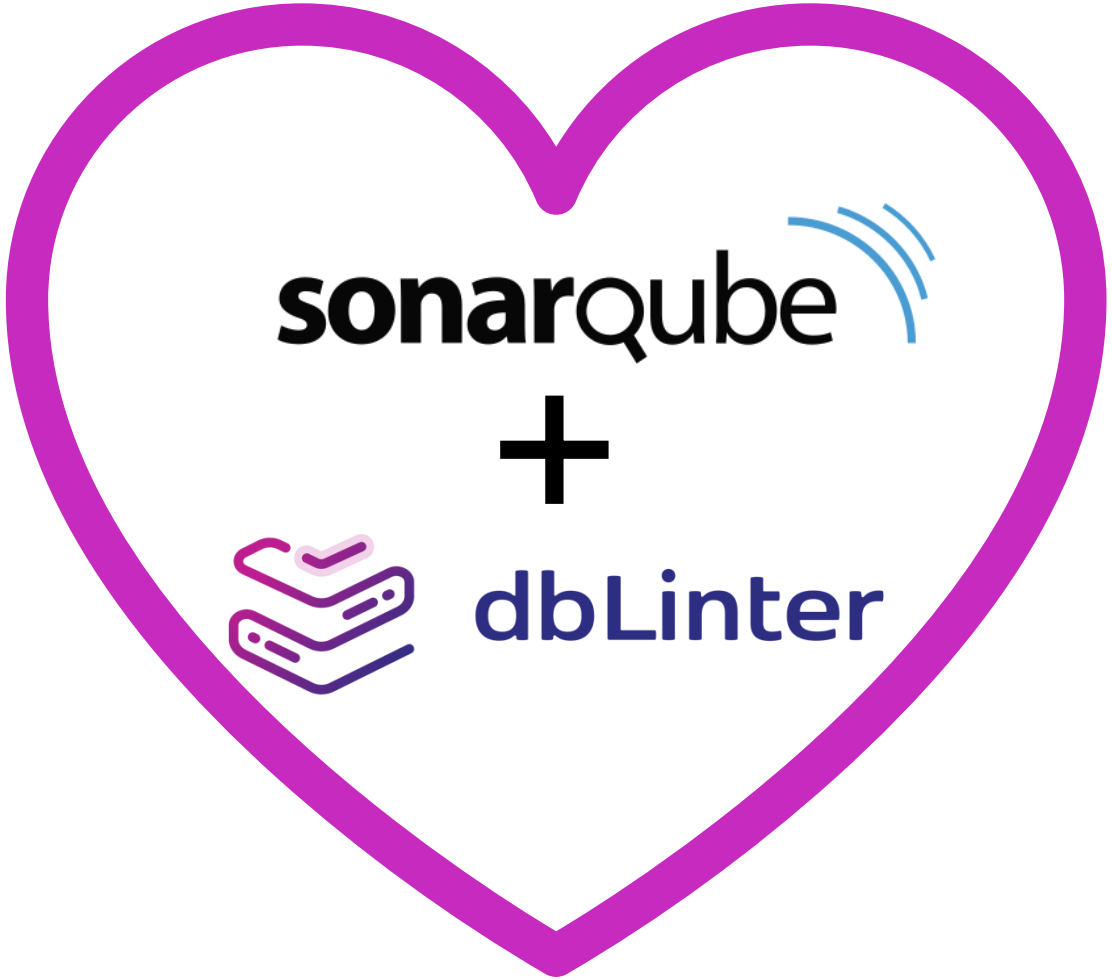


Core Messages

Quality Assessment of Code in the DB



Good Fit in DevOps Lifecycle



IDE Integration for PL/SQL & SQL

SonarQube for IDE

- ❌ Undefined grammar scope
- 🤔 Own set of rules
- 🤔 Disable check as quick fix
- ❌ No SQL-based Tests
- 🤔 Custom rules with Xpath
- ❌ No custom quick fixes
- ✅ Mature solution

dbLinter for VS Code

- ✅ Recent grammar versions
- ✅ Rules based on Trivadis Guidelines
- ✅ Quick fixes
- ✅ SQL-based Tests with migrations
- ✅ Custom rules with SQL and Java
- ✅ Custom quick fixes
- 🤔 Preview, GA in 2025

Thank you!

GRISSELBAV

simple & accurate IT solutions.