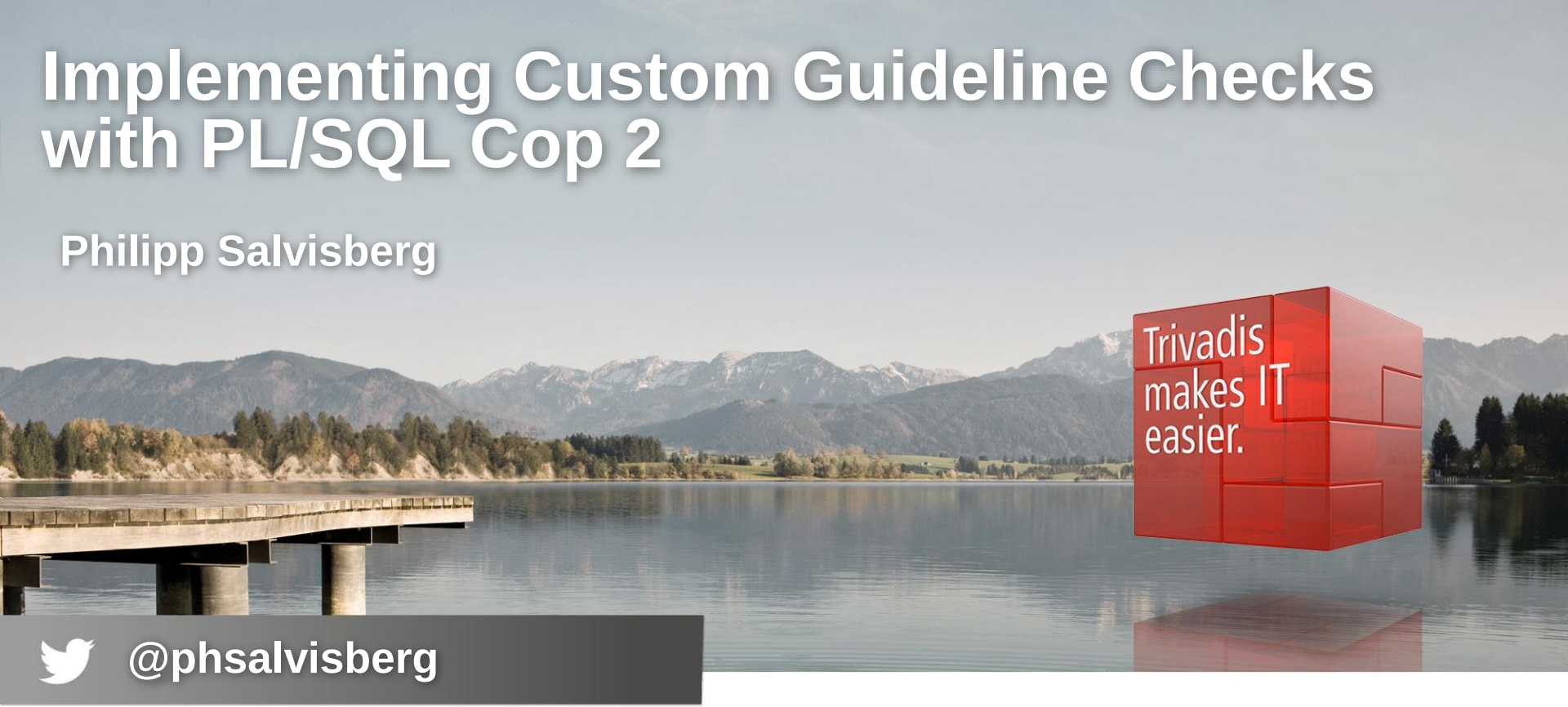
Implementing Custom Guideline Checks with PL/SQL Cop 2

Philipp Salvisberg



@phsalvisberg

About Me

- Trivadian since April 2000
  - Senior Principal Consultant, Partner
  - Member of the Board of Directors
  - www.salvis.com/blog
  - @phsalvisberg
- Database centric development with Oracle database
- Fond of DSLs to build full stack solutions efficiently and keep them manageable
- Author of free SQL Developer Extensions PL/SQL Cop, PL/SQL Unwrapper, oddgen and Bitemp Remodeler

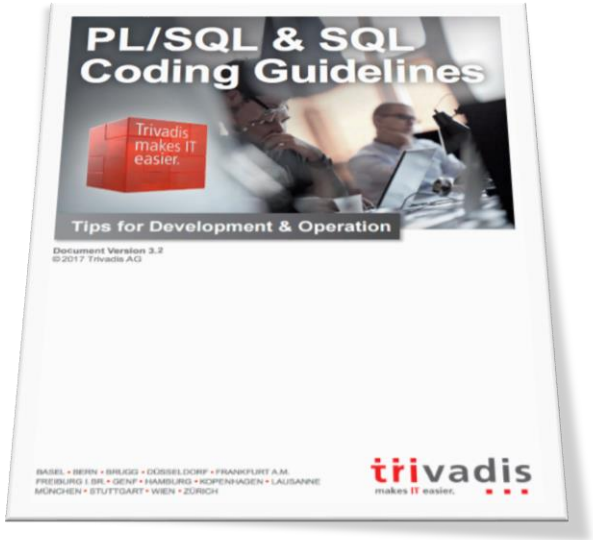
trivadis makes IT easier.

Agenda

1. What Is PL/SQL Cop?
2. Building a Validator
3. Testing a Validator
4. Using a Validator
5. Core Messages

What Is PL/SQL Cop?

Trivadis PL/SQL & SQL Coding Guidelines



Coding Guidelines are a crucial part of software development. It is a matter of fact, that code is more often read than written – therefore we should take efforts to ease the work of the reader, which is not necessarily the author.
I am convinced that this standard may be a good starting point for your own guidelines.

"Roger and his team have done an excellent job of providing a comprehensive set of clear standards that will undoubtedly improve the quality of your code. If you do not yet have standards in place, you should give strong consideration to using these as a starting point."

- Openly available since August 2009
- Download for free from www.trivadis.com

trivadis
makes IT easier.

PL/SQL Cop – Tool Suite

keeps IT fit.

Checks code against Trivadis PL/SQL & SQL Guidelines. Calculates various metrics.

Command-Line Utility

- Code folder
- Snapshot reports
- Since 2013

SonarQube Plugin

- Code folder
- Snapshot/delta reports
- Metric repository
  - Thresholds
  - Evolvement
- Continuous Integration
- Since 2015

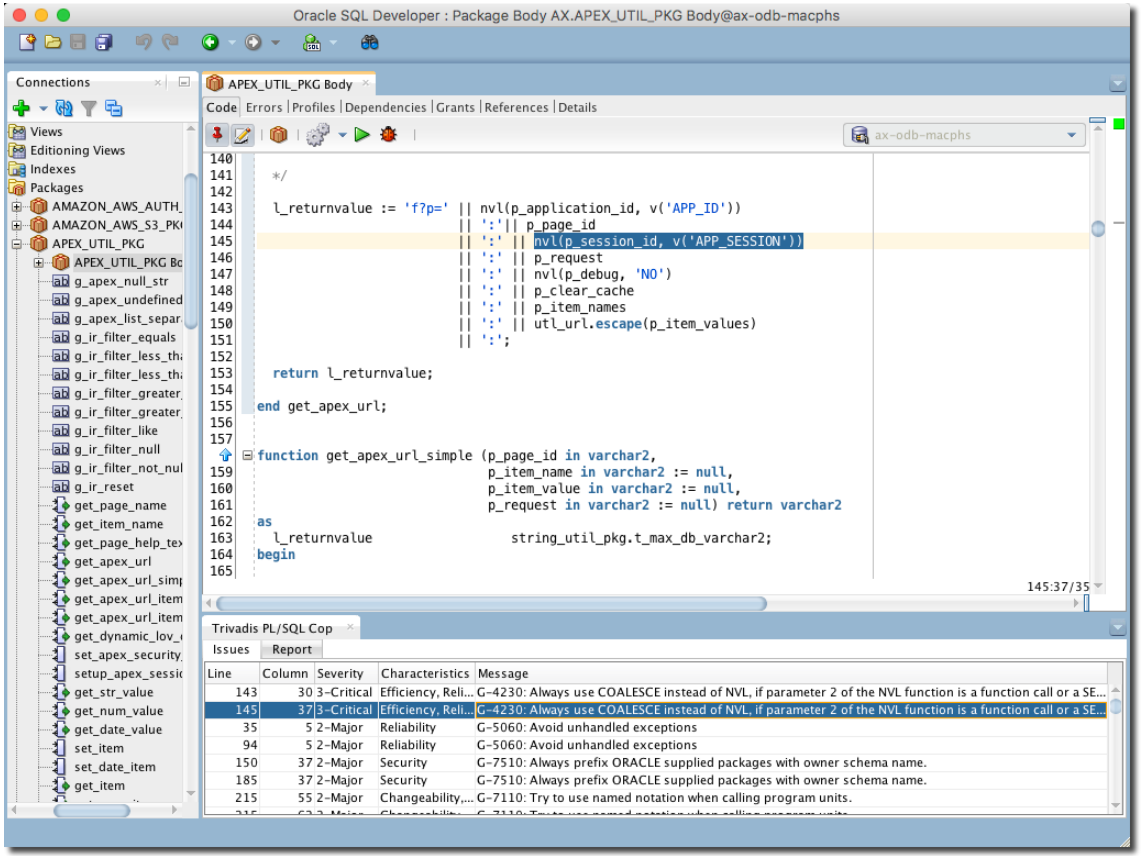
SQL Developer Extension

- Editor content
- Snapshot reports
- Since 2014 (free)

Download from: https://www.salvis.com/blog/download/

trivadis makes IT easier.

SQL Developer Extension – Issues



- Navigate through issues using the cursor keys
- Related code section is highlighted in the linked editor.

trivadis makes IT easier.

Building a Validator

Prerequisites



Maven™

- JDK 8u121
- Maven 3.3.9
- PL/SQL Cop 2.0.3

PL/SQL Cop

keeps IT fit.

trivadis

makes IT easier.

Recommended Options

- Eclipse IDE for Java and DSL Developers 4.6.2
- SQL Developer 4.1.5
- SonarQube 4.5.0 .. 5.1.2
- Jenkins 2.46.2

Validator to Check Simple Naming Conventions

- Global variables should start with 'g_'.
- Local variables should start with 'l_'.'
- Parameters should start with 'p_'.

Structure of a Validator

```
class GLP extends PLSQLJavaValidator implements PLSQLCopValidator {
    private HashMap<Integer, PLSQLCopGuideline> guidelines

    override getGuidelines() {...}

    def checkVariableName(VariableDeclaration v) {...}

    def checkParameterName(ParameterDeclaration p) {...}
}
```

PLSQLCopGuideline – Attributes

Attribute	Description
Id	Guideline identifier
Message	Guideline message
Severity	Info, Minor, Major, Critical, Blocker
Subcharacteristic	SonarQube SQALE subcharacteristic, 31 values which map to exactly one SQALE characteristic
Remediation	SonarQube remediation function: ConstantPerIssue, Linear, LinearWithOffset
Characteristics	List of SQALE characteristics (beside the one derived from the assigned subcharacteristic)

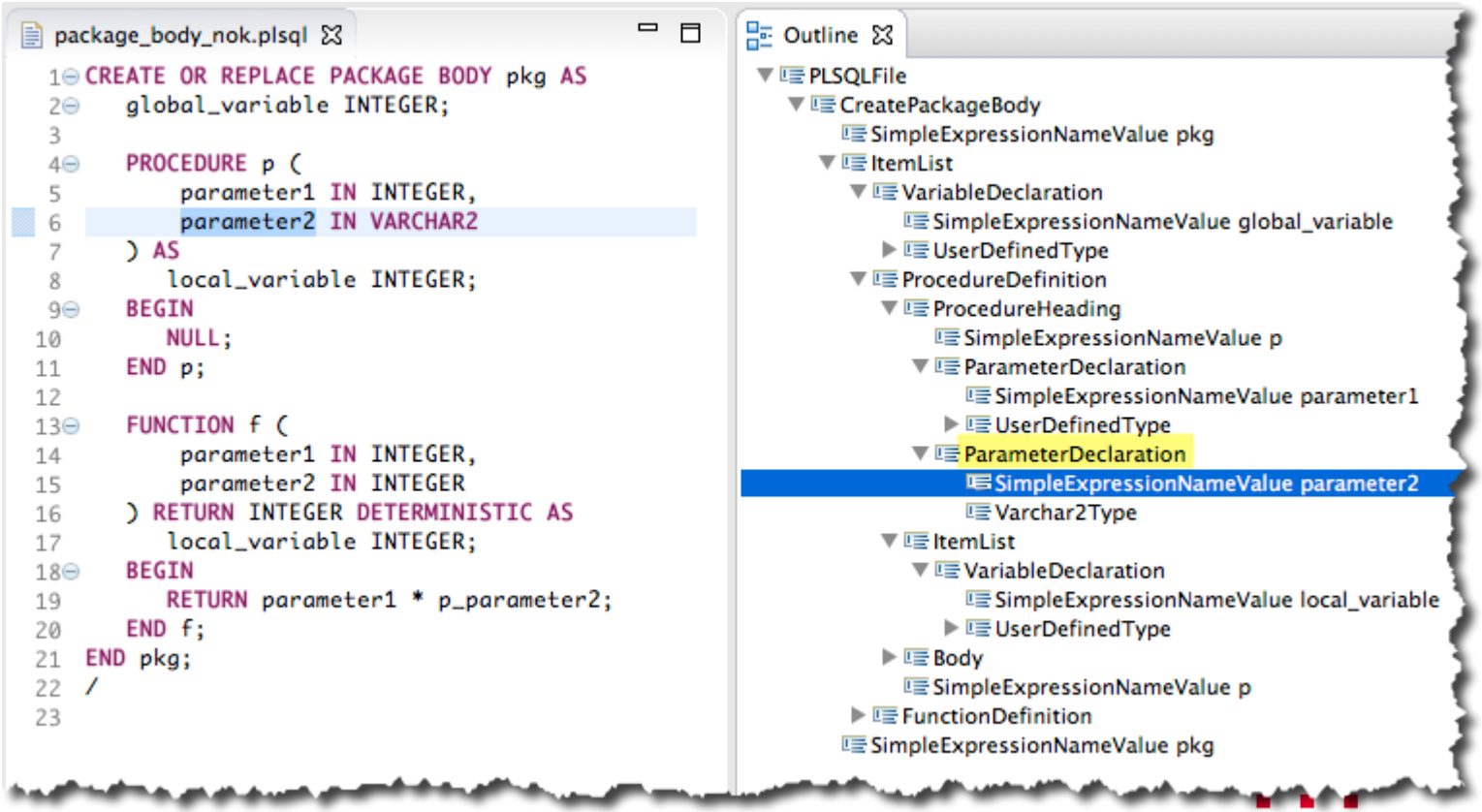
SQALE Characteristics & Subcharacteristics

Characteristic	Subcharacteristics
Changeability	Architecture_changeability, Data_changeability, Logic_changeability
Efficiency	Cpu_efficiency, Memory_efficiency, Network_use
Maintainability	Readability, Understandability
Portability	Compiler_related_portability, Hardware_related_portability, Language_related_portability, Os_related_portability, Software_related_portability, Time_zone_related_portability
Reliability	Architecture_reliability, Data_reliability, Exception_handling, Fault_tolerance, Instruction_reliability, Logic_reliability, Resource_reliability, Synchronization_reliability, Unit_tests
Reusability	Modularity, Transportability
Security	Api_abuse, Errors, Input_validation_and_representation, Security_features
Testability	Integration_testability, Unit_testability

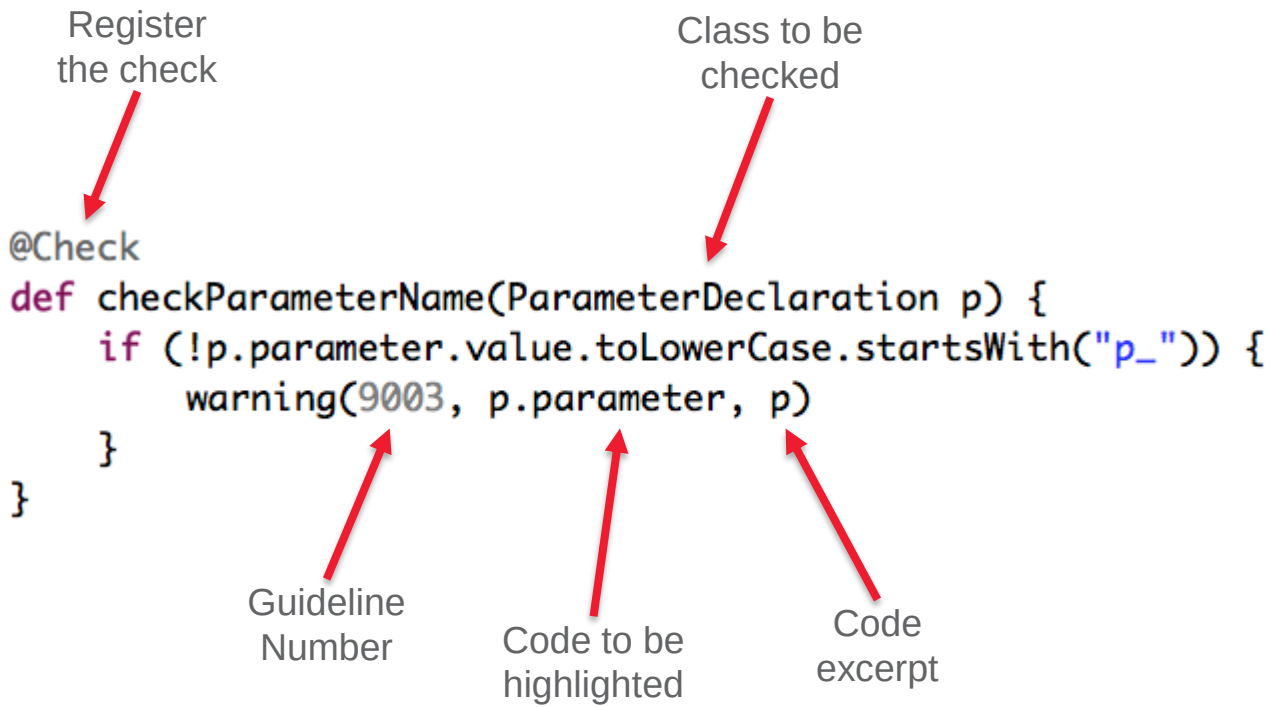
Register Guidelines

```
override getGuidelines() {
    if (guidelines == null) {
        guidelines = new HashMap<Integer, PLSQLCopGuideline>()
        // register parent guidelines
        for (k : super.getGuidelines().keySet) {
            guidelines.put(k, super.getGuidelines().get(k))
        }
        // register guidelines
        guidelines.put(9001,
            new PLSQLCopGuideline(9001, '''Global variables should start with 'g_'.''', MAJOR, UNDERSTANDABILITY,
                Remediation.createConstantPerIssue(1)))
        guidelines.put(9002,
            new PLSQLCopGuideline(9002, '''Local variables should start with 'l_'.''', MAJOR, UNDERSTANDABILITY,
                Remediation.createConstantPerIssue(1)))
        guidelines.put(9003,
            new PLSQLCopGuideline(9003, '''Parameters should start with 'p_'.''', MAJOR, UNDERSTANDABILITY,
                Remediation.createConstantPerIssue(1)))
    }
    return guidelines
}
```

Finding the Class to Be Checked in the PL/SQL Model



Implement a Check



Building the Validator

```
cd /usr/local/bin/tvdcc/sample/plugin
mvn clean package

[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building trivadis.tvdcc.validators 1.0.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
…
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ trivadis.tvdcc.validators ---
[INFO] Building jar: /usr/local/bin/tvdcc/sample/plugin/target/validators.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
…
```

Testing a Validator

Junit Test


```
class GLPDemoTest extends AbstractValidatorTest {
    @BeforeClass
    static def setupValidator() {
        PLSQLValidatorPreferences.INSTANCE.validatorClass = GLP
    }

    @Test
    def void procedureNok() {
        val stmt = '''
            CREATE OR REPLACE PROCEDURE p (a IN INTEGER, b IN VARCHAR2) AS
               c INTEGER;
            BEGIN
               NULL;
            END p;
        '''
        val issues = stmt.issues.filter[it.code.startsWith("G-9")]
        Assert.assertEquals(3, issues.size)
        // a
        val issue1 = issues.get(0)
        Assert.assertEquals(1, issue1.lineNumber)
        Assert.assertEquals(32, issue1.column)
        Assert.assertEquals(1, issue1.length)
        Assert.assertEquals("G-9003", issue1.code)
        Assert.assertEquals("G-9003: Parameters should start with 'p_'.", issue1.message)
        Assert.assertEquals("a IN INTEGER", issue1.data.get(0))
    }
}
```

Configure Validator

Parse stmt and get a list of issues

Using a Validator

PL/SQL Cop – Setup & Usage

Copy the validator into the plugin directory

```
cd /usr/local/bin/tvdcc
cp ./sample/plugin/target/validators.jar ./plugin
```

Use validator parameter when calling the command line utility

```
tvdcc.sh path=. validator=com.trivadis.tvdcc.validators.GLP

Trivadis PL/SQL Cop Version 2.0.3 (2017-02-12 20:32:44 +0100 [2382])
…
transforming tvdcc_report.xml into tvdcc_report.html... done.
transforming tvdcc_report.xml into tvdcc_report.xlsx... done.
cleanup completed.
```

PL/SQL Cop – Result

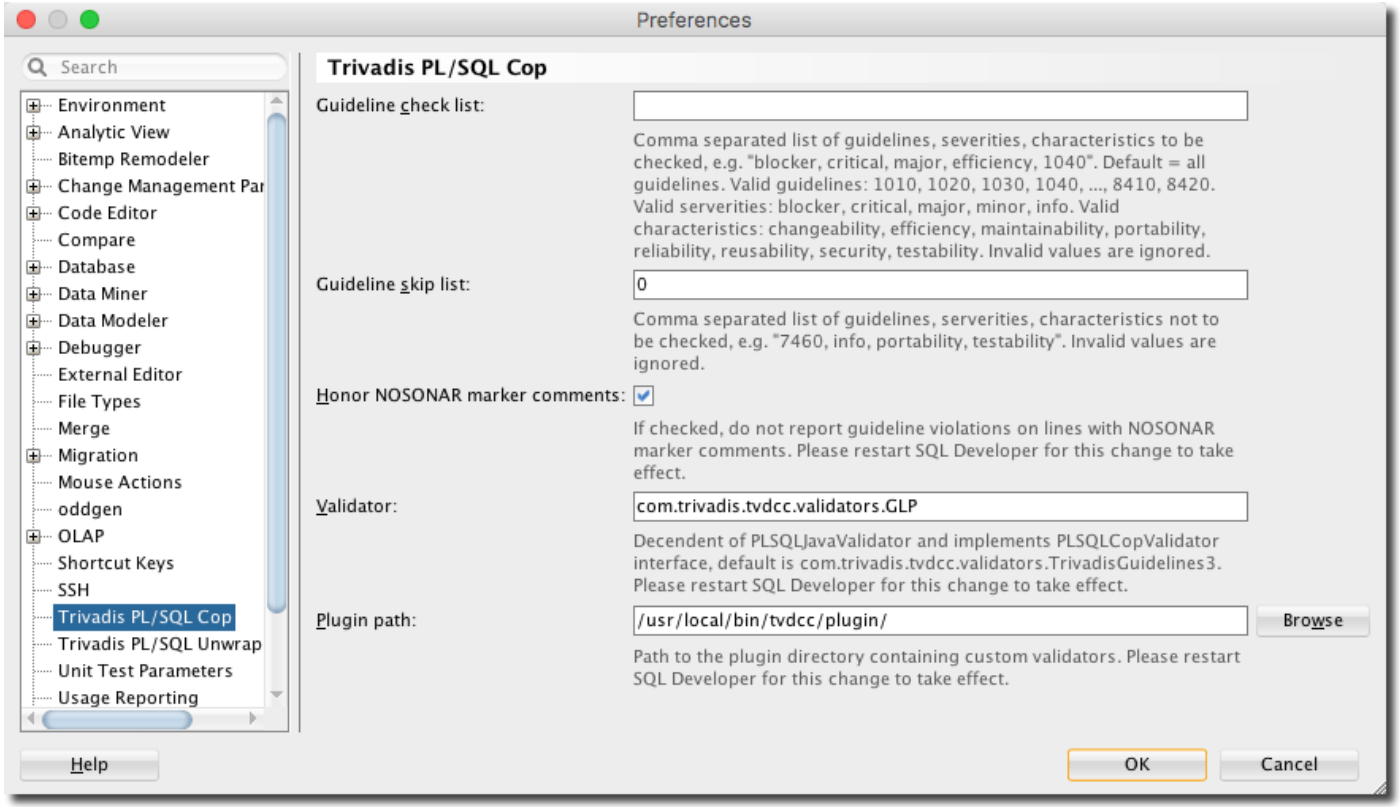
sample/guidelines/guideline_6020_59.sql overview

Issue#	Line	Severity	Message	Code Excerpt
1	5	Major	G-9003: Parameters should start with 'p_'.	in_employee_id IN employees.employee_id%TYPE
2	6	Major	G-9003: Parameters should start with 'p_'.	in_increase_pct IN types_up.percentage
3	7	Major	G-9003: Parameters should start with 'p_'.	out_new_salary OUT employees.salary%TYPE
4	22	Major	G-9003: Parameters should start with 'p_'.	in_employee_id IN employees.employee_id%TYPE
5	23	Major	G-9003: Parameters should start with 'p_'.	in_increase_pct IN types_up.percentage
6	24	Major	G-9003: Parameters should start with 'p_'.	out_new_salary OUT employees.salary%TYPE

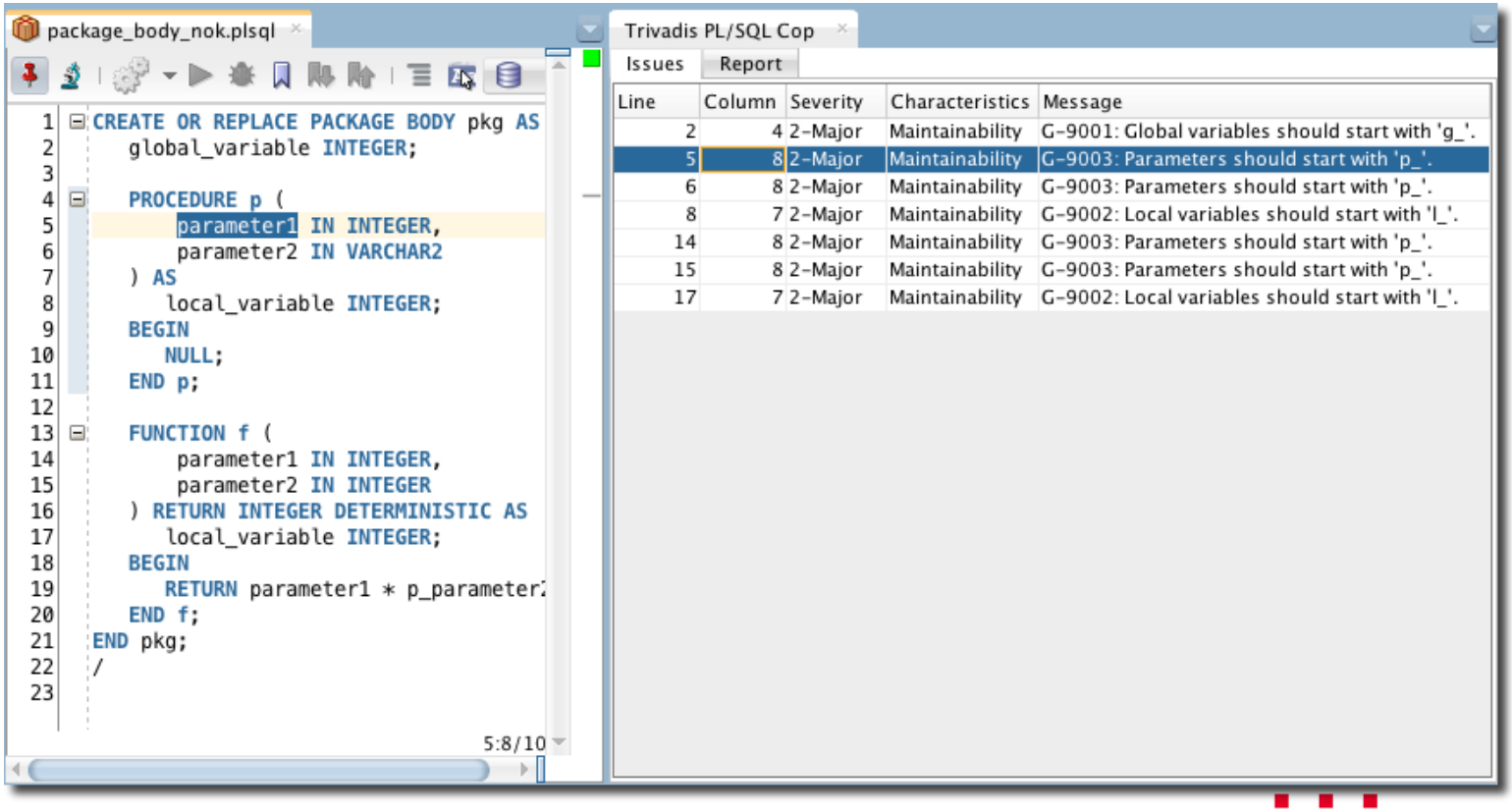
sample/guidelines/guideline_7130_62.sql overview

Issue#	Line	Severity	Message	Code Excerpt
1	6	Major	G-9002: Local variables should start with 'l_'.	r_emp employees%rowtype;
2	37	Major	G-9002: Local variables should start with 'l_'.	r_emp employees%rowtype;
3	5	Major	G-9003: Parameters should start with 'p_'.	in_employee_id IN employees.employee_id%TYPE
4	36	Major	G-9003: Parameters should start with 'p_'.	in_employee_id IN employees.employee_id%TYPE
5	39	Major	G-9003: Parameters should start with 'p_'.	in_salary IN employees.salary%TYPE
6	40	Major	G-9003: Parameters should start with 'p_'.	in_comm_pct IN employees.commission_pct%TYPE

PL/SQL Cop for SQL Developer – Setup



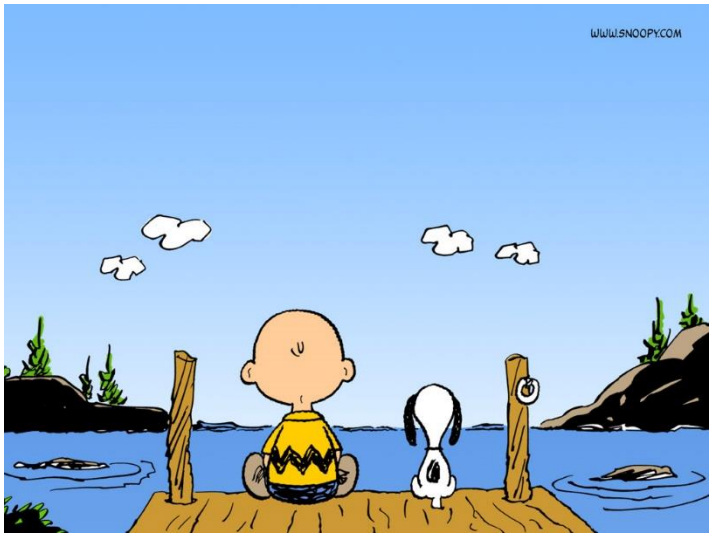
PL/SQL Cop for SQL Developer – Result



Core Messages

The Sky Is the Limit

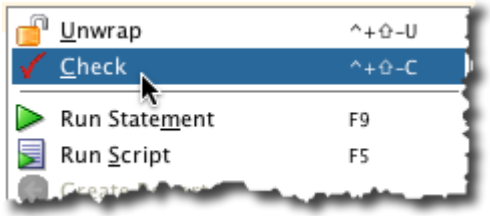
- Building a validator from scratch is challenging…
  …but adapting an existing one is much easier!
- The capabilities of a validator increase
  if you examine more than just the current file
- Use a CI environment to improve your
  code quality continuously



Get PL/SQL Cop – Now!

keeps IT fit.

- The PL/SQL Developer extension is free and has no limitations
- Drop me an e-mail if you need an unlimited license key for the command line utility



Download from: https://www.salvis.com/blog/download/

trivadis makes IT easier.

Questions and Answers

Philipp Salvisberg
Senior Principal Consultant

Tel. +41 58 459 52 31
philipp.salvisberg@trivadis.com

trivadis
makes IT easier.