PL/SQL Code Checker – At the Bleeding Edge

Philipp Salvisberg
philipp.salvisberg@trivadis.com

Anatoli Barski
anatoli.barski@trivadis.com

TechEvent
Regensdorf, 15th April 2011

trivadis
makes IT easier.

Agenda

Data are always part of the game.

- Introduction
- Xtext Solution
- Limitations
- ANTLR & XQuery Alternative Solution
- Conclusion

Starting Position (1)

- Trivadis PL/SQL & SQL Coding Guidelines Release 2009
- PL/SQL Assessment Offering using a Cookbook based on
  - Quest SQL Navigator 6.2.1 Code Expert
  - Quest TOAD 10.0 Code Expert
  - TVD Scripts with PL/Scope 11g to check Naming Conventions
  - TVD Scripts for rules not handled by Quest
- Shortcoming of PL/SQL Assessment Offering
  - One snapshot – Assessment of a defined release
  - Repetitive execution is time-consuming, expensive, not feasible
  - Solution is not part of an automated, continuous integration strategy

Starting Position (2)

- PL/SQL Code Checker Prototype based on Xtext
  - Implementation by Itemis AG, sponsored by Technology Division
  - Sample code for guidelines #25, #47, #54 supported only
  - Command-line interface and Eclipse Plug-In
  - Support for multiple error reporting strategies (text, HTML)
  - See also TechEvent_201009_Pakull_PLSQL_Code_Checker.pptx
- TIPP project with Commerzbank
  - Extend prototype
  - Run against customer source code
  - Define and implement customer rules
  - Verify feasibility of this approach
- Additional Technology Projects to complete Parser
  - Full Parse of PL/SQL source embedded in SQL*Plus files
  - Basic Parse of other components in SQL files (using "BaseText")

Intention

- Support Trivadis PL/SQL & SQL Coding Guidelines completely
- Continuous support of new Oracle Releases
- Explore additional, functional areas with further TIPP projects
  - Dependency Analysis
  - Complexity Analysis
  - Externalize Configuration
- Use Code Checker in conjunction with PL/SQL & SQL Coding Guidelines as Marketing Instrument
  - DOAG SIG Development/Oracle Tools, 22nd September 2011, Köln
    Checking compliance with custom guidelines for PL/SQL code
  - *Oracle World 2011, 2nd-6th October 2011, San Francisco*
    Modern PL/SQL Code Checking and Dependency Analysis
  - *DOAG 2011 Conference, 15th-17th November 2011, Nürnberg*
    Modern PL/SQL Code Checking and Dependency Analysis
- Explore further, functional areas
  - Complete Eclipse Plug-In, Syntax Highlighting, Code Formatter, Quick Fixes, Code Completion, Refactoring
  - Plug-In for existing IDEs like SQL Developer, TOAD, …
  - Web Services

Primary Scope of PL/SQL Code Checker

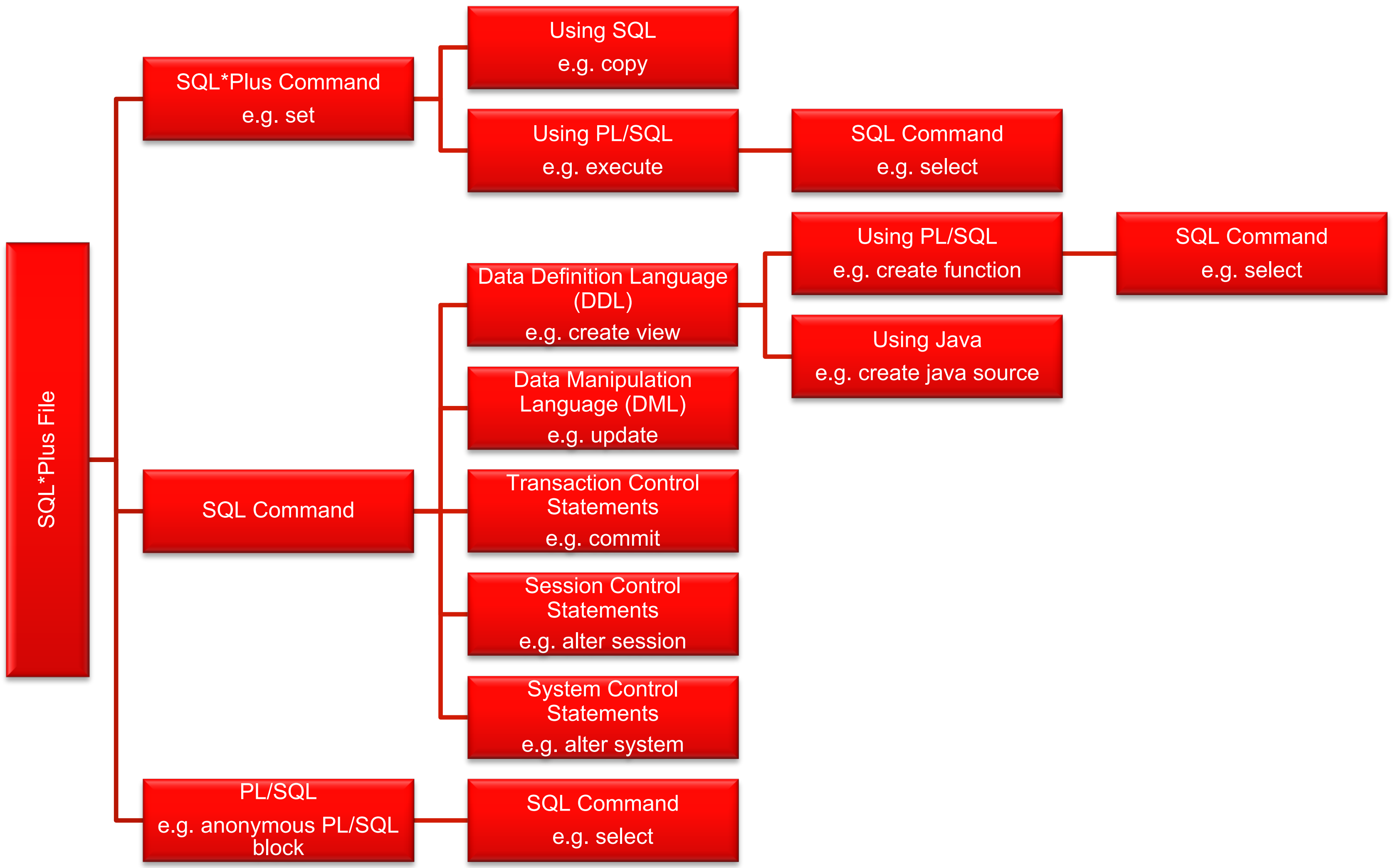
- Process SQL*Plus files within a directory tree using a command line interface
  - Support typical file extensions out of the box:
    - sql, prc, fnc, pks, pkb, trg, vw, tps, tbp, plb, pls, rcv
    - spc, typ
    - aqt, aqp, ctx, dbl, tab, dim, snp, con, collt, seq, syn, grt
    - sp, spb, sps
  - Ignore other file extensions

- Oracle Database Version 9i and later

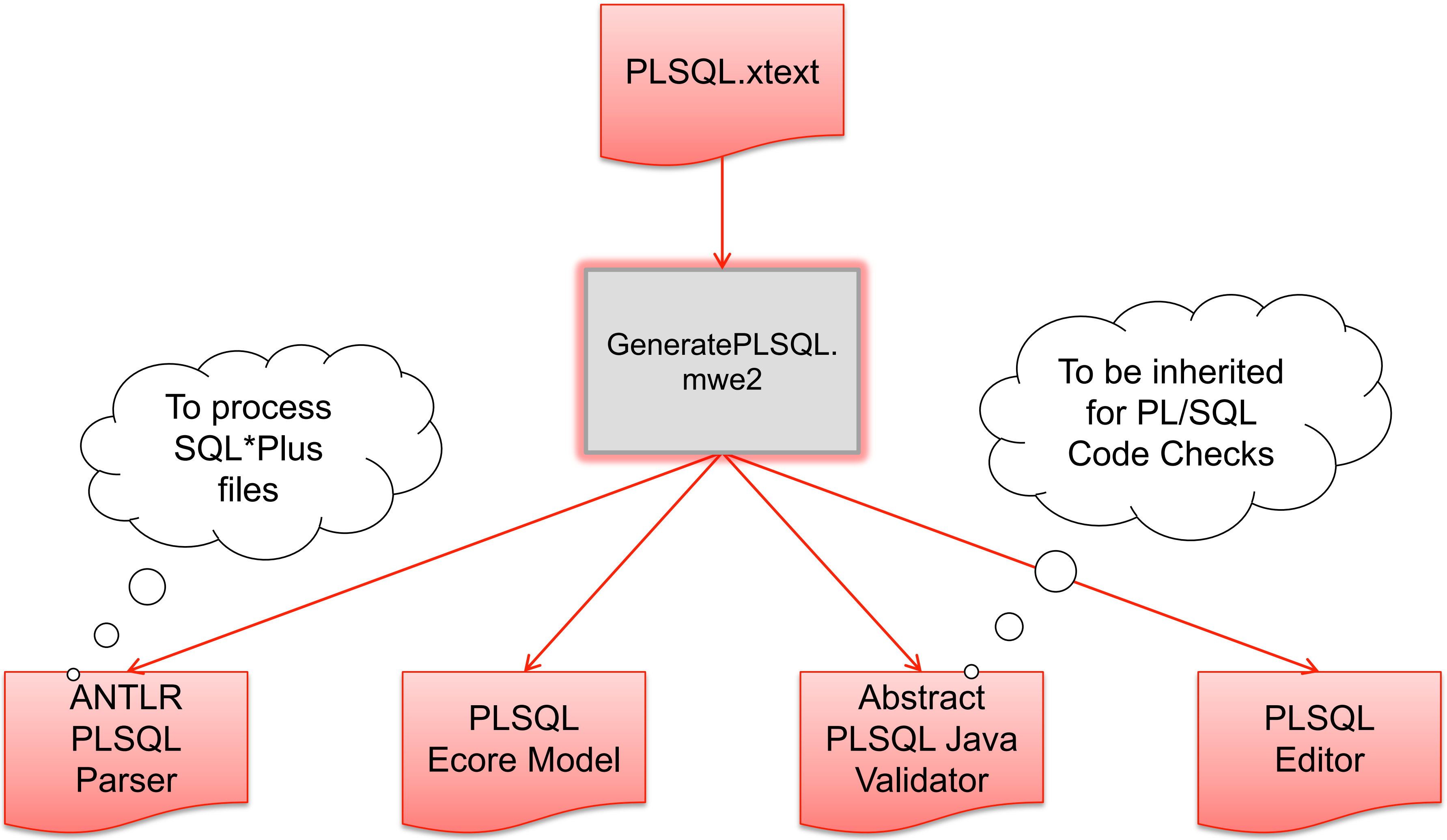
Agenda

- Introduction
- Xtext Solution
- Limitations
- ANTLR & XQuery Alternative Solution
- Conclusion

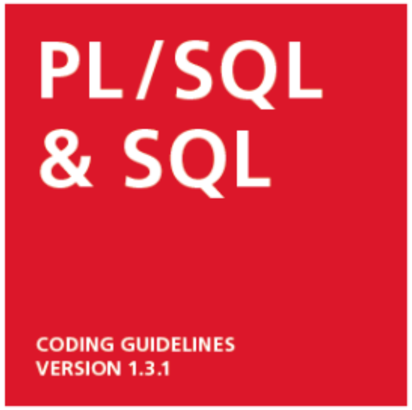
What's in a SQL*Plus File?


SQL*Plus File
SQL*Plus Command e.g. set
Using SQL e.g. copy
Using PL/SQL e.g. execute
SQL Command e.g. select
SQL Command
Data Definition Language (DDL) e.g. create view
Using PL/SQL e.g. create function
SQL Command e.g. select
Using Java e.g. create java source
Data Manipulation Language (DML) e.g. update
Transaction Control Statements e.g. commit
Session Control Statements e.g. alter session
System Control Statements e.g. alter system
PL/SQL e.g. anonymous PL/SQL block
SQL Command e.g. select

Generate PL/SQL Grammar via Xtext


PLSQL.xtext
GeneratePLSQL.mwe2
To process SQL*Plus files
To be inherited for PL/SQL Code Checks
ANTLR PLSQL Parser
PLSQL Ecore Model
Abstract PLSQL Java Validator
PLSQL Editor

Trivadis PL/SQL & SQL Guideline #54


PL/SQL & SQL

CODING GUIDELINES
VERSION 1.3.1

54. Always use a string variable to execute dynamic SQL.

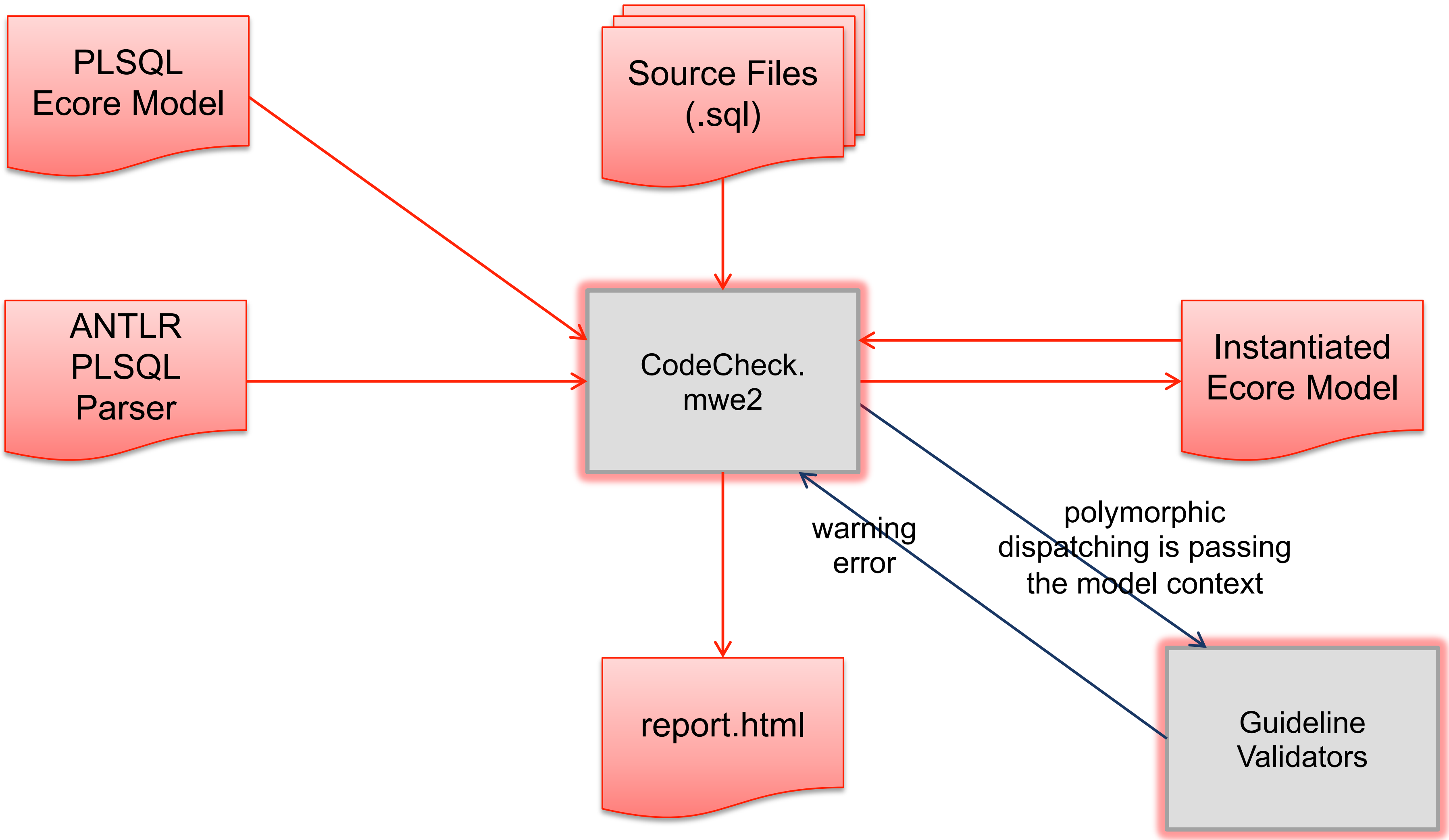
Reason: Having the executed statement in a variable makes it easier to debug your code.

Example:

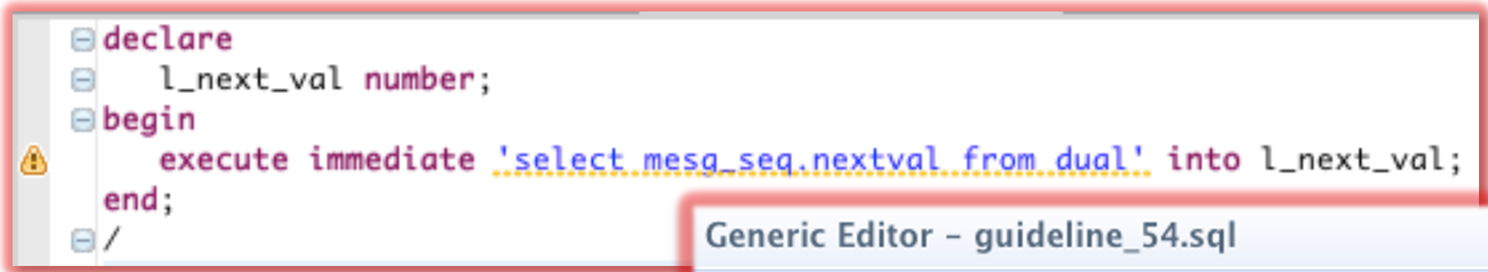
```
-- Bad
DECLARE
   l_empno emp.empno%TYPE := 4711;
BEGIN
   EXECUTE IMMEDIATE 'DELETE FROM emp WHERE epno = :p_empno' USING l_empno;
END;
```

```
-- Good
DECLARE
   l_empno emp.empno%TYPE := 4711;
   l_sql   VARCHAR2(32767);
BEGIN
   l_sql := 'DELETE FROM emp WHERE epno = :p_empno';
   EXECUTE IMMEDIATE l_sql USING l_empno;
EXCEPTION
   WHEN others
   THEN
      DBMS_OUTPUT.PUT_LINE(l_sql);
END;
```

Apply Code Checks

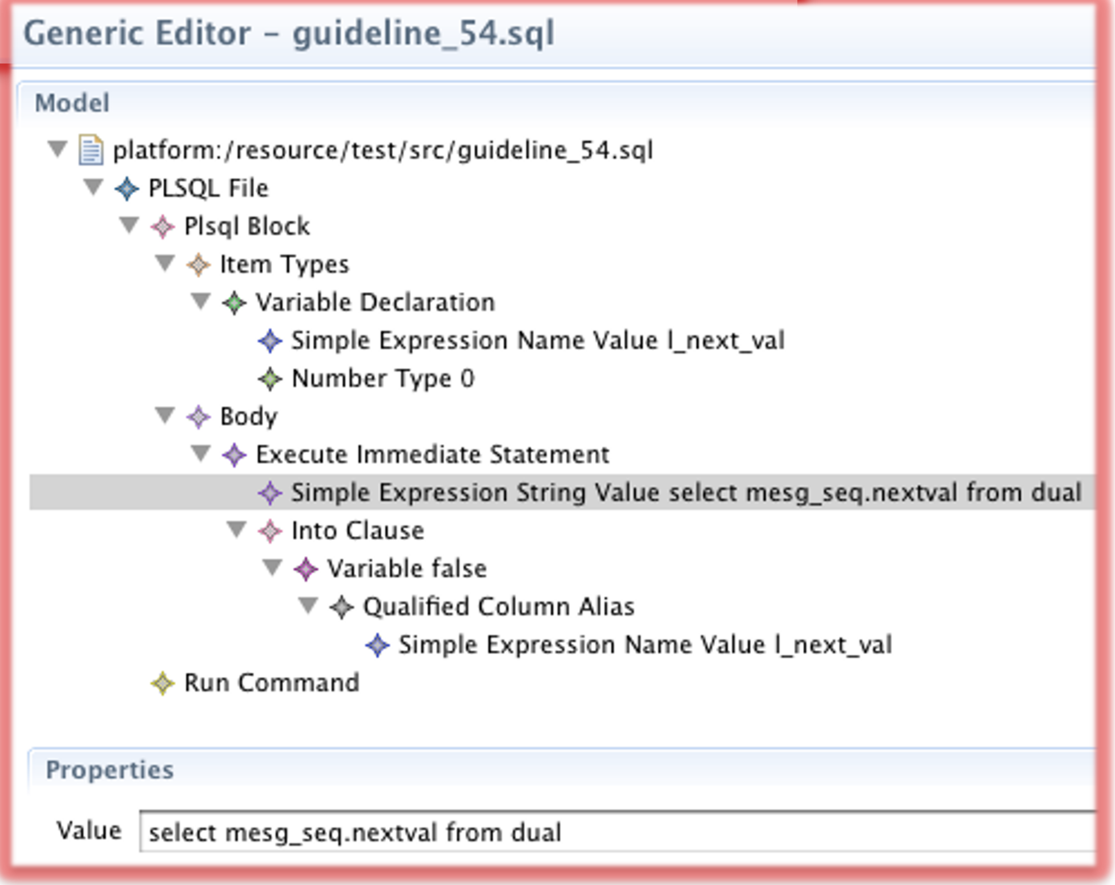


Source, Model & Warning for Guideline #54



```
declare
   l_next_val number;
begin
   execute immediate 'select mesg_seq.nextval from dual' into l_next_val;
end;
/
```

line 4 - Guideline 54 violated: Always use a string variable to execute dynamic SQL.



Generic Editor – guideline_54.sql

Model

```
▼ platform:/resource/test/src/guideline_54.sql
  ▼ PLSQL File
    ▼ Plsql Block
      ▼ Item Types
        ▼ Variable Declaration
            Simple Expression Name Value l_next_val
            Number Type 0
      ▼ Body
        ▼ Execute Immediate Statement
            Simple Expression String Value select mesg_seq.nextval from dual
          ▼ Into Clause
            ▼ Variable false
              ▼ Qualified Column Alias
                  Simple Expression Name Value l_next_val
    Run Command
```

Properties

Value: select mesg_seq.nextval from dual

Validator for Guideline #54

PLSQL.xtext

```
ExecuteImmediateStatement:
  'execute' 'immediate'
  statement=Expression
  (
      (
        into=(IntoClause | BulkCollectIntoClause)
        using=UsingClause?
      )
    | using=UsingClause
    | returning=DynamicOrStaticReturningClause
  )?
;
```

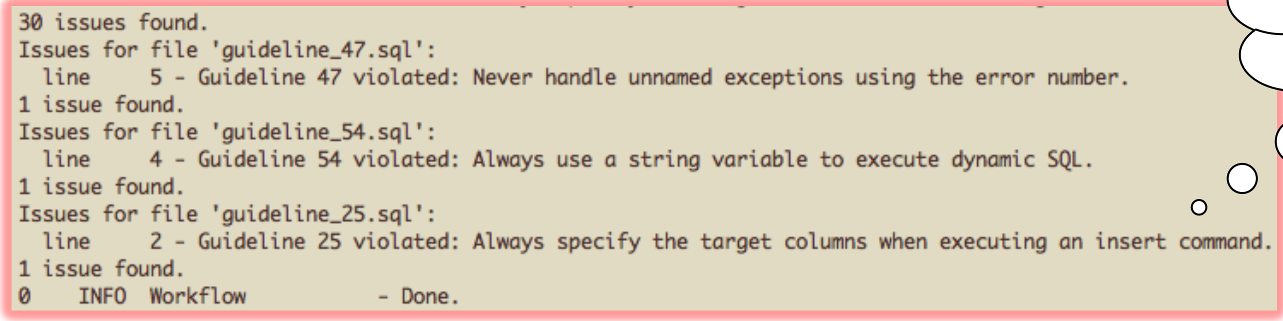
PLSQLJavaValidator.java

```java
@Check
public void checkGuideline54(ExecuteImmediateStatement statement) {
    Expression executedStatement = statement.getStatement();
    // plain string?
    if (executedStatement instanceof SimpleExpressionStringValue) {
        warning("Guideline 54 violated: Always use a string variable to execute dynamic SQL.",
                executedStatement, null, GUIDELINE_54,
                serialize(executedStatement.eContainer()));
    }
    // expression?
    else if (executedStatement != null) {
        List<SimpleExpressionStringValue> stringValues = EcoreUtil2
                .getAllContentsOfType(executedStatement,
                        SimpleExpressionStringValue.class);
        // string values found?
        if (stringValues != null) {
            if (stringValues.size() > 0) {
                warning("Guideline 54 violated: Always use a string variable to execute dynamic SQL.",
                        stringValues.get(0), null, GUIDELINE_54,
                        serialize(executedStatement.eContainer()));
            }
        }
    }
}
```

Command Line Interface

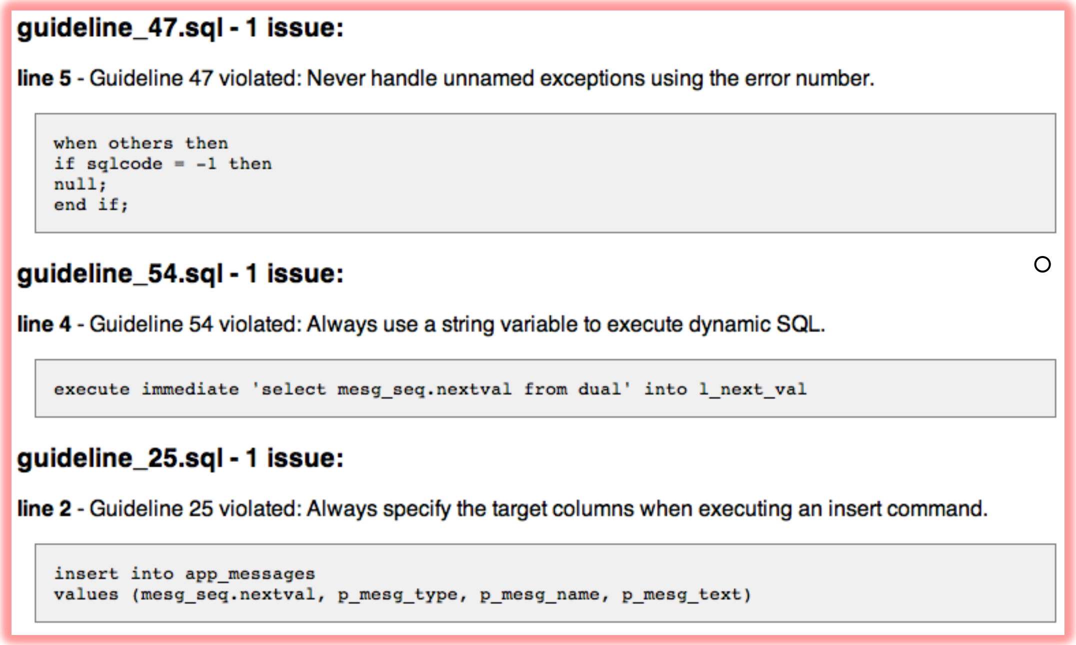
DEMO

- java -jar codecheck.jar .



```
30 issues found.
Issues for file 'guideline_47.sql':
  line     5 - Guideline 47 violated: Never handle unnamed exceptions using the error number.
1 issue found.
Issues for file 'guideline_54.sql':
  line     4 - Guideline 54 violated: Always use a string variable to execute dynamic SQL.
1 issue found.
Issues for file 'guideline_25.sql':
  line     2 - Guideline 25 violated: Always specify the target columns when executing an insert command.
1 issue found.
0    INFO  Workflow           - Done.
```

Console Strategy



guideline_47.sql - 1 issue:

line 5 - Guideline 47 violated: Never handle unnamed exceptions using the error number.

```
when others then
if sqlcode = -1 then
null;
end if;
```

guideline_54.sql - 1 issue:

line 4 - Guideline 54 violated: Always use a string variable to execute dynamic SQL.

```
execute immediate 'select mesg_seq.nextval from dual' into l_next_val
```

guideline_25.sql - 1 issue:

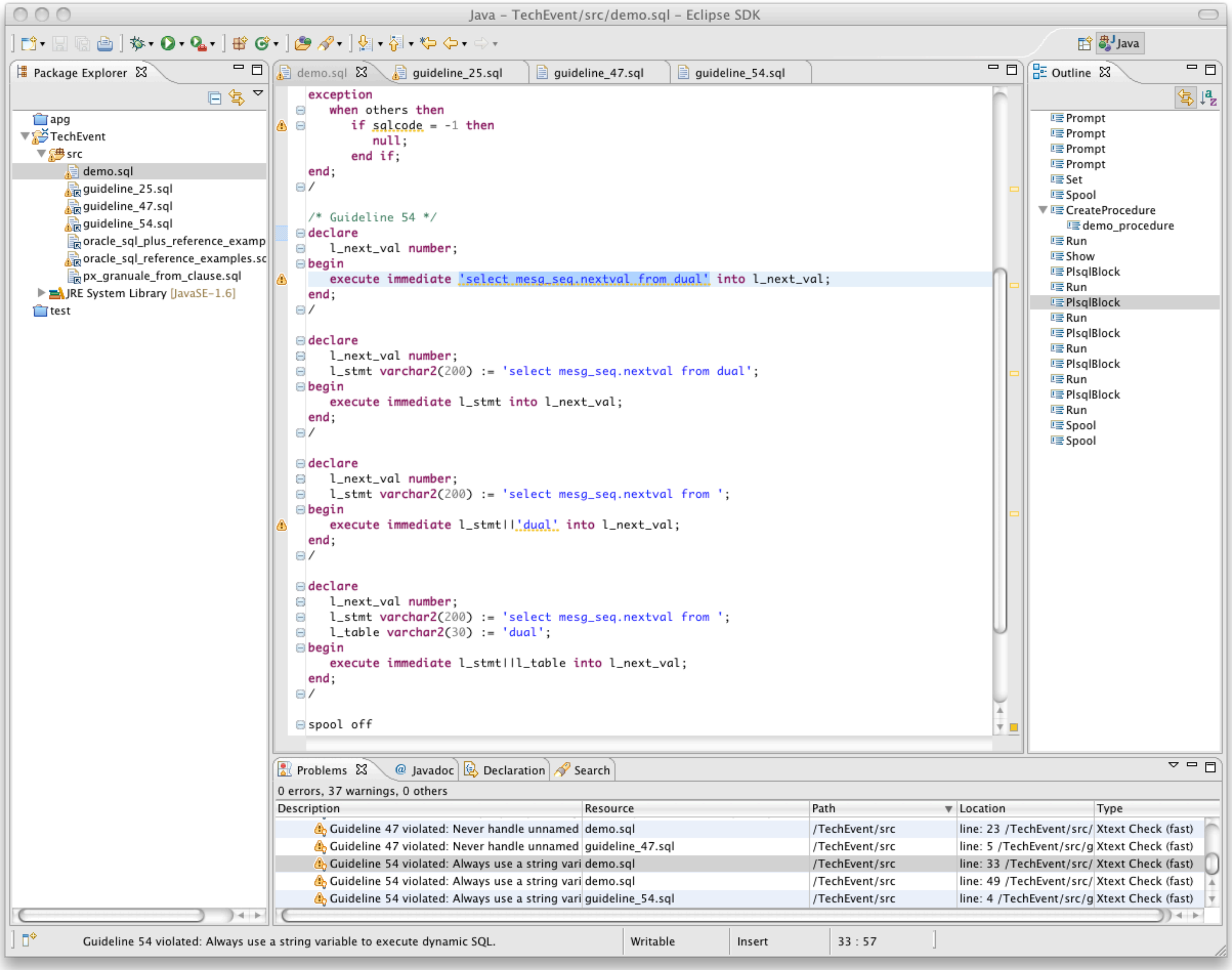
line 2 - Guideline 25 violated: Always specify the target columns when executing an insert command.

```
insert into app_messages
values (mesg_seq.nextval, p_mesg_type, p_mesg_name, p_mesg_text)
```

HTML Strategy

Eclipse Plug-In

DEMO



Java – TechEvent/src/demo.sql – Eclipse SDK

Package Explorer

```
apg
▼ TechEvent
  ▼ src
      demo.sql
      guideline_25.sql
      guideline_47.sql
      guideline_54.sql
      oracle_sql_plus_reference_examp
      oracle_sql_reference_examples.sc
      px_granuale_from_clause.sql
  ▶ JRE System Library [JavaSE-1.6]
test
```

demo.sql | guideline_25.sql | guideline_47.sql | guideline_54.sql

```
exception
   when others then
      if sqlcode = -1 then
         null;
      end if;
end;
/

/* Guideline 54 */
declare
   l_next_val number;
begin
   execute immediate 'select mesg_seq.nextval from dual' into l_next_val;
end;
/

declare
   l_next_val number;
   l_stmt varchar2(200) := 'select mesg_seq.nextval from dual';
begin
   execute immediate l_stmt into l_next_val;
end;
/

declare
   l_next_val number;
   l_stmt varchar2(200) := 'select mesg_seq.nextval from ';
begin
   execute immediate l_stmt||'dual' into l_next_val;
end;
/

declare
   l_next_val number;
   l_stmt varchar2(200) := 'select mesg_seq.nextval from ';
   l_table varchar2(30) := 'dual';
begin
   execute immediate l_stmt||l_table into l_next_val;
end;
/

spool off
```

Outline

```
Prompt
Prompt
Prompt
Prompt
Set
Spool
▼ CreateProcedure
    demo_procedure
Run
Show
PlsqlBlock
Run
PlsqlBlock
Run
PlsqlBlock
Run
PlsqlBlock
Run
PlsqlBlock
Run
Spool
Spool
```

Problems | Javadoc | Declaration | Search

0 errors, 37 warnings, 0 others

Description	Resource	Path	Location	Type
Guideline 47 violated: Never handle unnamed	demo.sql	/TechEvent/src	line: 23 /TechEvent/src/	Xtext Check (fast)
Guideline 47 violated: Never handle unnamed	guideline_47.sql	/TechEvent/src	line: 5 /TechEvent/src/g	Xtext Check (fast)
Guideline 54 violated: Always use a string vari	demo.sql	/TechEvent/src	line: 33 /TechEvent/src/	Xtext Check (fast)
Guideline 54 violated: Always use a string vari	demo.sql	/TechEvent/src	line: 49 /TechEvent/src/	Xtext Check (fast)
Guideline 54 violated: Always use a string vari	guideline_54.sql	/TechEvent/src	line: 4 /TechEvent/src/g	Xtext Check (fast)

Guideline 54 violated: Always use a string variable to execute dynamic SQL. | Writable | Insert | 33 : 57

Validator for Parameter Naming

DEMO

```
ParameterDeclaration:
    parameter=ColumnAlias
        self?='self'? in?='in'? (out?='out' noCopy?='noCopy'?)?
        type=ElementType
        default=DefaultClause?
;
```

```
ColumnAlias:
        SimpleExpressionNameValue
      | ReservedKeywordExpression
;
```

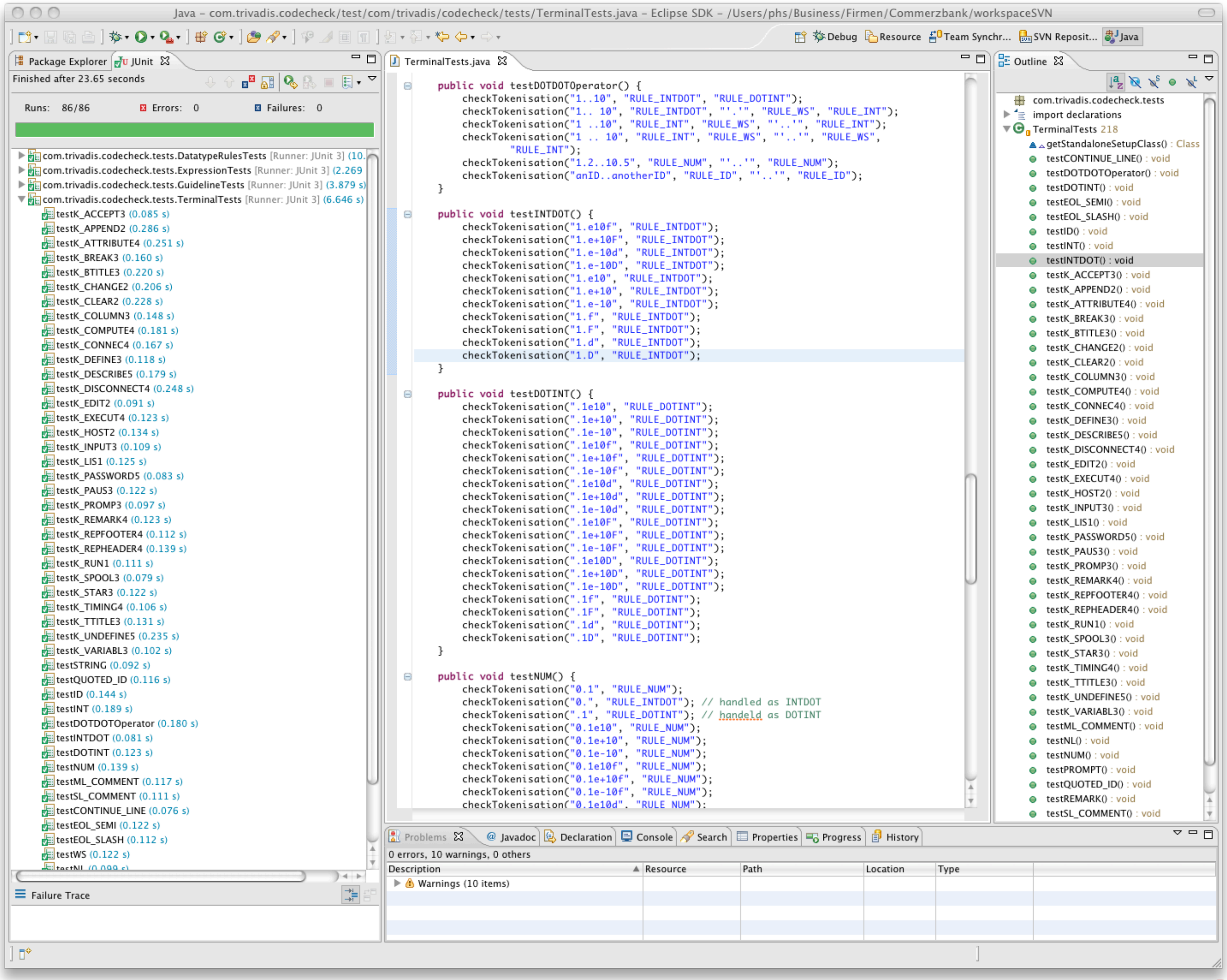
```
SimpleExpressionNameValue:
    columnName=SqlName
;
```

```
public static final String CUSTOM_GUIDELINE_1 = "codecheck.custom.guideline.1";

// @Check
public void checkCustomGuideline1(ParameterDeclaration parameter) {
    ElementType type = parameter.getType();
    ColumnAlias alias = parameter.getParameter();
    String name = "";
    // i_ and s_ prefixed parameters are simple expression name values
    if (alias instanceof SimpleExpressionNameValue) {
        name = ((SimpleExpressionNameValue) alias).getColumnName();
        if (name.length() >= 2) {
            name = name.substring(0, 2).toLowerCase();
        }
    }
    // apply rule for standalone functions/procedures, type (body)
    // functions/procedures, package (body) functions/procedures and
    // cursors
    if (type instanceof NumberType && !name.equals("i_")
            || type instanceof Varchar2Type && !name.equals("s_")) {
        warning("Custom Guideline 1 violated: parameter name must start with 'i_' for number and with 's_' for varchar2 parameters.",
                alias, null, CUSTOM_GUIDELINE_1,
                serialize(parameter.eContainer()));
    }
}
```

Unit Testing

DEMO



Java – com.trivadis.codecheck/test/com/trivadis/codecheck/tests/TerminalTests.java – Eclipse SDK – /Users/phs/Business/Firmen/Commerzbank/workspaceSVN

```
Finished after 23.65 seconds

Runs:  86/86      Errors:  0      Failures:  0

com.trivadis.codecheck.tests.DatatypeRulesTests [Runner: JUnit 3] (10.
com.trivadis.codecheck.tests.ExpressionTests [Runner: JUnit 3] (2.269
com.trivadis.codecheck.tests.GuidelineTests [Runner: JUnit 3] (3.879 s)
com.trivadis.codecheck.tests.TerminalTests [Runner: JUnit 3] (6.646 s)
  testK_ACCEPT3 (0.085 s)
  testK_APPEND2 (0.286 s)
  testK_ATTRIBUTE4 (0.251 s)
  testK_BREAK3 (0.160 s)
  testK_BTITLE3 (0.220 s)
  testK_CHANGE2 (0.206 s)
  testK_CLEAR2 (0.228 s)
  testK_COLUMN3 (0.148 s)
  testK_COMPUTE4 (0.181 s)
  testK_CONNEC4 (0.167 s)
  testK_DEFINE3 (0.118 s)
  testK_DESCRIBE5 (0.179 s)
  testK_DISCONNECT4 (0.248 s)
  testK_EDIT2 (0.091 s)
  testK_EXECUT4 (0.123 s)
  testK_HOST2 (0.134 s)
  testK_INPUT3 (0.109 s)
  testK_LIS1 (0.125 s)
  testK_PASSWORD5 (0.083 s)
  testK_PAUS3 (0.122 s)
  testK_PROMP3 (0.097 s)
  testK_REMARK4 (0.123 s)
  testK_REPFOOTER4 (0.112 s)
  testK_REPHEADER4 (0.139 s)
  testK_RUN1 (0.111 s)
  testK_SPOOL3 (0.079 s)
  testK_STAR3 (0.122 s)
  testK_TIMING4 (0.106 s)
  testK_TTITLE3 (0.131 s)
  testK_UNDEFINE5 (0.235 s)
  testK_VARIABL3 (0.102 s)
  testSTRING (0.092 s)
  testQUOTED_ID (0.116 s)
  testID (0.144 s)
  testINT (0.189 s)
  testDOTDOTOperator (0.180 s)
  testINTDOT (0.081 s)
  testDOTINT (0.123 s)
  testNUM (0.139 s)
  testML_COMMENT (0.117 s)
  testSL_COMMENT (0.111 s)
  testCONTINUE_LINE (0.076 s)
  testEOL_SEMI (0.122 s)
  testEOL_SLASH (0.112 s)
  testWS (0.122 s)
  testNL (0.099 s)

Failure Trace
```

TerminalTests.java

```java
    public void testDOTDOTOperator() {
        checkTokenisation("1..10", "RULE_INTDOT", "RULE_DOTINT");
        checkTokenisation("1.. 10", "RULE_INTDOT", "'.'", "RULE_WS", "RULE_INT");
        checkTokenisation("1 ..10", "RULE_INT", "RULE_WS", "'..'", "RULE_INT");
        checkTokenisation("1 .. 10", "RULE_INT", "RULE_WS", "'..'", "RULE_WS",
                "RULE_INT");
        checkTokenisation("1.2..10.5", "RULE_NUM", "'..'", "RULE_NUM");
        checkTokenisation("anID..anotherID", "RULE_ID", "'..'", "RULE_ID");
    }

    public void testINTDOT() {
        checkTokenisation("1.e10f", "RULE_INTDOT");
        checkTokenisation("1.e+10F", "RULE_INTDOT");
        checkTokenisation("1.e-10d", "RULE_INTDOT");
        checkTokenisation("1.e-10D", "RULE_INTDOT");
        checkTokenisation("1.e10", "RULE_INTDOT");
        checkTokenisation("1.e+10", "RULE_INTDOT");
        checkTokenisation("1.e-10", "RULE_INTDOT");
        checkTokenisation("1.f", "RULE_INTDOT");
        checkTokenisation("1.F", "RULE_INTDOT");
        checkTokenisation("1.d", "RULE_INTDOT");
        checkTokenisation("1.D", "RULE_INTDOT");
    }

    public void testDOTINT() {
        checkTokenisation(".1e10", "RULE_DOTINT");
        checkTokenisation(".1e+10", "RULE_DOTINT");
        checkTokenisation(".1e-10", "RULE_DOTINT");
        checkTokenisation(".1e10f", "RULE_DOTINT");
        checkTokenisation(".1e+10f", "RULE_DOTINT");
        checkTokenisation(".1e-10f", "RULE_DOTINT");
        checkTokenisation(".1e10d", "RULE_DOTINT");
        checkTokenisation(".1e+10d", "RULE_DOTINT");
        checkTokenisation(".1e-10d", "RULE_DOTINT");
        checkTokenisation(".1e10F", "RULE_DOTINT");
        checkTokenisation(".1e+10F", "RULE_DOTINT");
        checkTokenisation(".1e-10F", "RULE_DOTINT");
        checkTokenisation(".1e10D", "RULE_DOTINT");
        checkTokenisation(".1e+10D", "RULE_DOTINT");
        checkTokenisation(".1e-10D", "RULE_DOTINT");
        checkTokenisation(".1f", "RULE_DOTINT");
        checkTokenisation(".1F", "RULE_DOTINT");
        checkTokenisation(".1d", "RULE_DOTINT");
        checkTokenisation(".1D", "RULE_DOTINT");
    }

    public void testNUM() {
        checkTokenisation("0.1", "RULE_NUM");
        checkTokenisation("0.", "RULE_INTDOT"); // handled as INTDOT
        checkTokenisation(".1", "RULE_DOTINT"); // handeld as DOTINT
        checkTokenisation("0.1e10", "RULE_NUM");
        checkTokenisation("0.1e+10", "RULE_NUM");
        checkTokenisation("0.1e-10", "RULE_NUM");
        checkTokenisation("0.1e10f", "RULE_NUM");
        checkTokenisation("0.1e+10f", "RULE_NUM");
        checkTokenisation("0.1e-10f", "RULE_NUM");
        checkTokenisation("0.1e10d", "RULE_NUM");
```

```
Outline
com.trivadis.codecheck.tests
  import declarations
  TerminalTests 218
    getStandaloneSetupClass() : Class
    testCONTINUE_LINE() : void
    testDOTDOTOperator() : void
    testDOTINT() : void
    testEOL_SEMI() : void
    testEOL_SLASH() : void
    testID() : void
    testINT() : void
    testINTDOT() : void
    testK_ACCEPT3() : void
    testK_APPEND2() : void
    testK_ATTRIBUTE4() : void
    testK_BREAK3() : void
    testK_BTITLE3() : void
    testK_CHANGE2() : void
    testK_CLEAR2() : void
    testK_COLUMN3() : void
    testK_COMPUTE4() : void
    testK_CONNEC4() : void
    testK_DEFINE3() : void
    testK_DESCRIBE5() : void
    testK_DISCONNECT4() : void
    testK_EDIT2() : void
    testK_EXECUT4() : void
    testK_HOST2() : void
    testK_INPUT3() : void
    testK_LIS1() : void
    testK_PASSWORD5() : void
    testK_PAUS3() : void
    testK_PROMP3() : void
    testK_REMARK4() : void
    testK_REPFOOTER4() : void
    testK_REPHEADER4() : void
    testK_RUN1() : void
    testK_SPOOL3() : void
    testK_STAR3() : void
    testK_TIMING4() : void
    testK_TTITLE3() : void
    testK_UNDEFINE5() : void
    testK_VARIABL3() : void
    testML_COMMENT() : void
    testNL() : void
    testNUM() : void
    testPROMPT() : void
    testQUOTED_ID() : void
    testREMARK() : void
    testSL_COMMENT() : void
```

```
Problems  Javadoc  Declaration  Console  Search  Properties  Progress  History
0 errors, 10 warnings, 0 others
Description              Resource    Path    Location    Type
  Warnings (10 items)
```

Agenda

- Introduction
- Xtext Solution
- Limitations
- ANTLR & XQuery Alternative Solution
- Conclusion

Xtext 1.0.2 – Most Annoying Limitations

- One Grammar, One Parser
  - The workflow GeneratePLSQL.mwe2 needs 4 minutes to complete
  - ~~Bug 328153~~ - Split grammar definition into multiple Xtext files
  - Bug 256403 - Multiple Grammar Mixin / Grammars as Library

- Maximum Size of 64 KB for Java Classes and Methods
  - Bug 328083 - Configure FieldsPerClass in addition to ClassSplitting
  - Bug 328753 - Too many constants error in generated internalXXXParser.java for huge grammar
  - The class splitting highly depends on specific version of the parser generator used (ANTLR 3.0 for Xtext 1.0, ANTLR 3.2 for Xtext 2.0) since the generated code is amended in way to get it compiled
  - The current PL/SQL grammar needs custom (TVD specific) features to avoid "… is exceeding 65535 bytes…" errors (since this is working only under certain circumstances it's currently not part of the Xtext distribution)
  - The Code Assist (part of the UI project) is currently disabled since this for to Code Assist extended grammar variant is far too large

Known, Major Code Checker Limitations

- Unquoted Identifiers may conflict with keywords of other grammars, e.g. "describe" is a keyword, but not a reserved word in SQL (valid for table, views, etc.)
  - It would be easy to handle all this keywords technically, but this currently leads to methods/classes > 64 KB
- Undocumented and old grammar may break the parser
  - The grammar is continuously extended according real live code
- User defined operators are not supported (a sample operator "contains" is hard-coded)
  - Currently defined grammar is becoming ambiguous
  - This problem may be addressed (probably) by refactoring the Expression and Condition parser rules
  - The workaround is, to simply add the customer's operators when needed

Known, Minor Code Checker Limitations

- The SQL*Plus block terminator character '.' is not supported (nor configurable)
- The SQL*Plus command separator character ';' is not supported (nor configurable)
- The SQL*PLUS SQLTerminator is not configurable, the default ';' is supported
- The SQL*Plus line continuation character '-' does not support tailing whitespaces
- The SQL*Plus run command abbreviation '/' does not support tailing whitespaces
- The SQL*Plus execute command must end on ';' if the last token is an expression (it's working only for syntactically fully defined statements)
- The SQL*Plus SQLTerminator ';' does not support tailing whitespaces (it's working only for syntactically fully defined statements)
- The SQL*Plus Commands REMARK and PROMPT must not contain unterminated single/double quotes, single line or multi line comments (using terminals lead to other conflicts)

Using Xtext – Reasons for Steep Learning Curve

- Output of underlying parser generator is passed 1:1 to the user
  - Fundamentals of ANTLR are mandatory
  - Ability to distinguish between ANTLR and Xtext artifacts necessary
- Convention over configuration
  - The first DSL incl. editors are created very fast using Xtext
  - Typically it's working but you easily do not know why and how
  - Usually things may be amended very elegantly and with just a few lines of code (e.g. outline, validators, formatter)
  - However, to find out what to do could take a serious time for an inexperienced fellow
- Consider your limitations when using Xtext ;-)

Agenda

- Introduction
- Xtext Solution
- Limitations
- ANTLR & XQuery Alternative Solution
- Conclusion

Data are always part of the game.

Choosing the Right Ingredients (1)

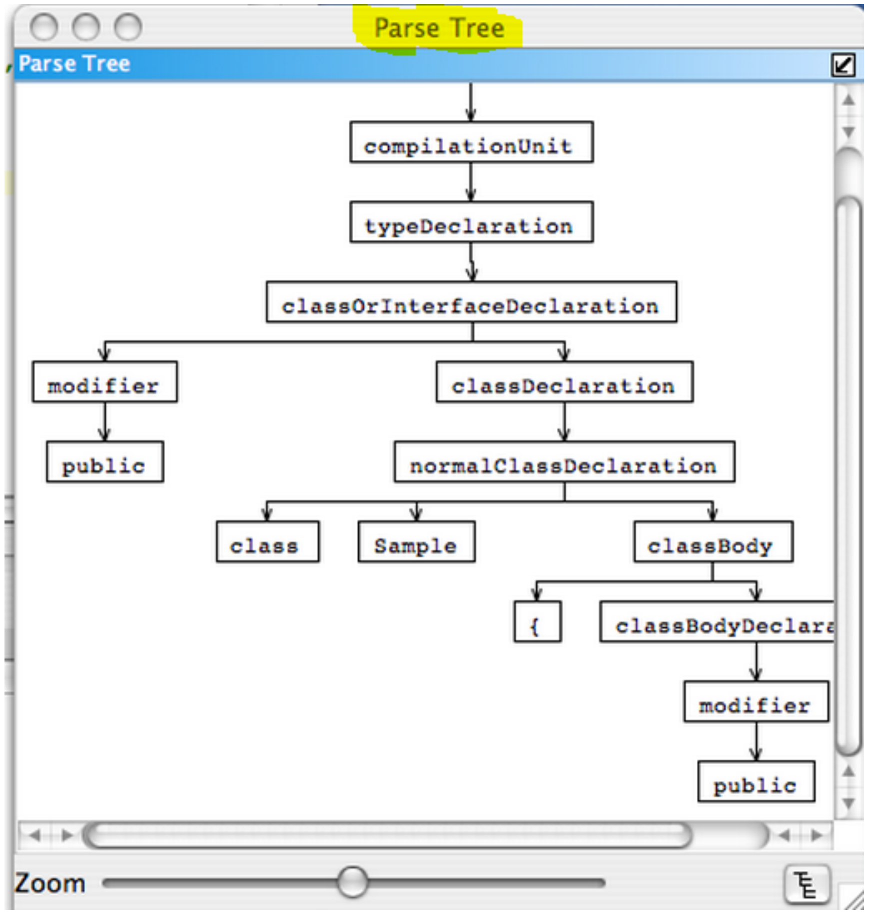
The Pragmatic Programmers
The Definitive ANTLR Reference
Building Domain-Specific Languages
Terence Parr

Choosing the Right Ingredients (2)

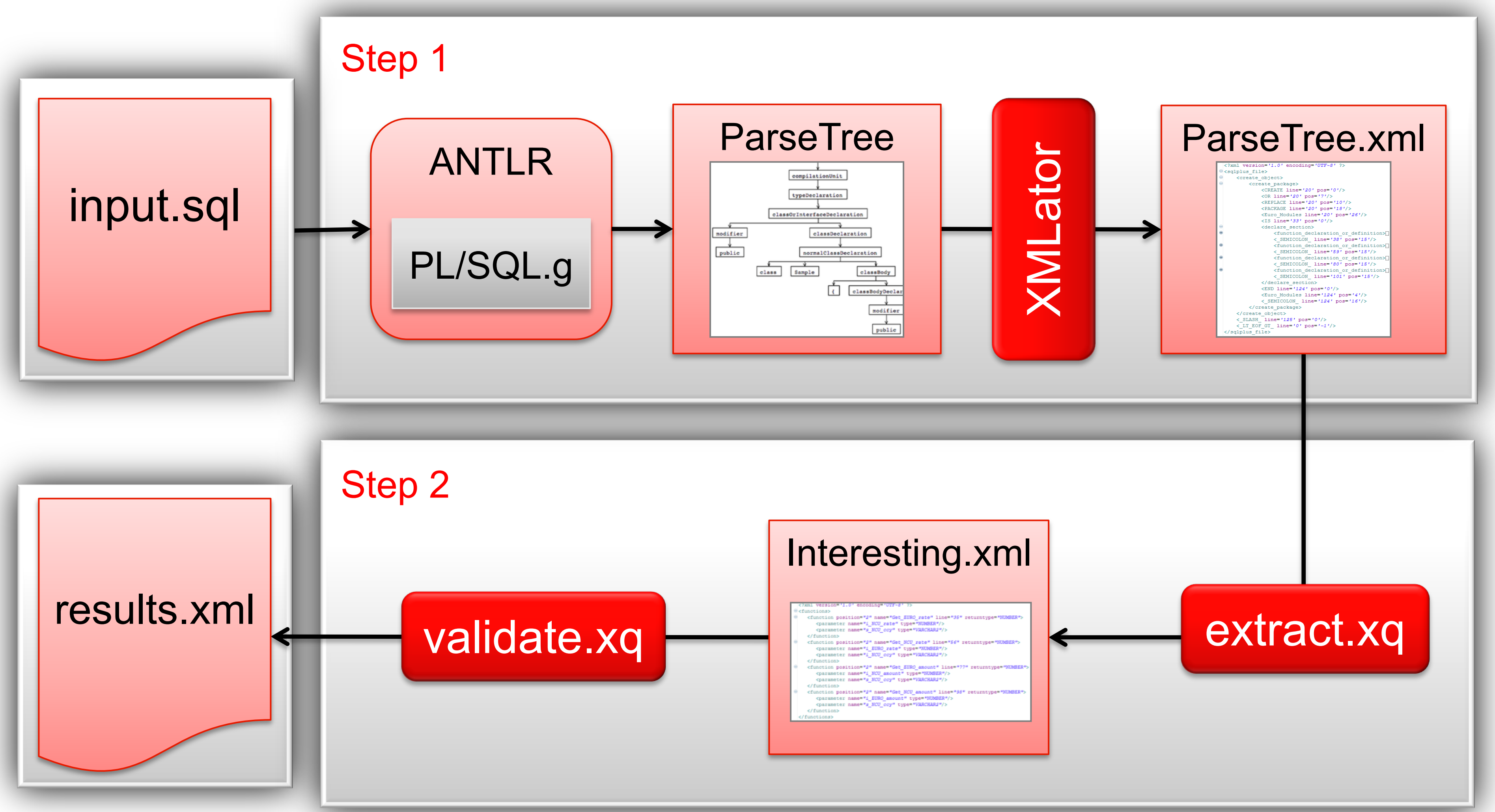
- SQL & PL/SQL grammars available on http://www.antlr.org/grammar/list
  - OracleSQL
    Ivan Brezina Fri Sep 3, 2010 07:19
    Oracle SQL grammar, including 11g features.
  - PL/SQL
    Patrick Higgins Fri Jul 16, 2010 15:20
    Parser for Oracle PL/SQL. Works with 11g. More details can be found in the header of the grammar.
  - Oracle PL/SQL Grammar for ANTLR v3
    Andrey Kharitonkin Sat Apr 26, 2008 08:59
    Based on PL/SQL grammar for ANTLR v2 published here.
  - ORACLE PL/SQL Grammar With Code Counting Hooks
    David Edwards Fri Mar 16, 2007 12:17
    Developed from the PL/SQL Grammar that was already present on the site, this version works better with more recent versions of PL/SQL. Nevertheless, it is far from being complete.

- SQL & PL/SQL grammar as part of Oracle JDeveloper Extensions
  - http://www.oracle.com/technetwork/developer-tools/jdev/index-099997.html
    see Class oracle.javatools.parser.plsql.PlsqlParser
  - Required libraries (e.g. javatools-nodeps.jar) are part of the SQL Developer distribution

What Do We Get Else For Free?



Putting It All Together



```
<?xml version='1.0' encoding='UTF-8' ?>
<sqlplus_file>
    <create_object>
        <create_package>
            <CREATE line='20' pos='0'/>
            <OR line='20' pos='7'/>
            <REPLACE line='20' pos='10'/>
            <PACKAGE line='20' pos='18'/>
            <Euro_Modules line='20' pos='26'/>
            <IS line='33' pos='0'/>
            <declare_section>
                <function_declaration_or_definition>
                <_SEMICOLON_ line='38' pos='15'/>
                <function_declaration_or_definition>
                <_SEMICOLON_ line='59' pos='15'/>
                <function_declaration_or_definition>
                <_SEMICOLON_ line='80' pos='15'/>
                <function_declaration_or_definition>
                <_SEMICOLON_ line='101' pos='15'/>
            </declare_section>
            <END line='124' pos='0'/>
            <Euro_Modules line='124' pos='4'/>
            <_SEMICOLON_ line='124' pos='16'/>
        </create_package>
    </create_object>
    <_SLASH_ line='125' pos='0'/>
    <_LT_EOF_GT_ line='0' pos='-1'/>
</sqlplus_file>
```

```
<?xml version='1.0' encoding='UTF-8' ?>
<functions>
   <function position="2" name="Get_EURO_rate" line="35" returntype="NUMBER">
      <parameter name="i_NCU_rate" type="NUMBER"/>
      <parameter name="s_NCU_ccy" type="VARCHAR2"/>
   </function>
   <function position="2" name="Get_NCU_rate" line="56" returntype="NUMBER">
      <parameter name="i_EURO_rate" type="NUMBER"/>
      <parameter name="i_NCU_ccy" type="VARCHAR2"/>
   </function>
   <function position="2" name="Get_EURO_amount" line="77" returntype="NUMBER">
      <parameter name="i_NCU_amount" type="NUMBER"/>
      <parameter name="s_NCU_ccy" type="VARCHAR2"/>
   </function>
   <function position="2" name="Get_NCU_amount" line="98" returntype="NUMBER">
      <parameter name="i_EURO_amount" type="NUMBER"/>
      <parameter name="s_NCU_ccy" type="VARCHAR2"/>
   </function>
</functions>
```

XMLator, extract.xq, validate.xq

DEMO

Agenda

- Introduction
- Xtext Solution
- Limitations
- ANTLR & XQuery Alternative Solution
- <u>Conclusion</u>

Data are always part of the game.

Conclusion

- The SQL*Plus grammar is huge and a solution to simplify the grammar was necessary to make it work with Xtext - there are still some simplifications possible (e.g. views)
- Coming releases of Xtext will for sure address at least some of the limitations (we may accelerate that if really needed)
- The advantage of the Pure-Parser-And-XQuery-Approach is that it is build on an existing parser, which does not need to be maintained by Trivadis, but this comes with limitations such as
  - No support for SQL*Plus files
  - Rudimentary model without the ability to handle references (which will become very handy for dependency analysis)
  - Validators are not really easier to write and maintain
- Xtext is build on sound concepts, e.g. good separation of parser and validators
- Xtext is a complete DSL framework (more than just a parser generator)
- Even if a significant subset of the SQL*Plus, SQL, PL/SQL grammar needs to be maintained continuously, Xtext is a good choice to implement the future PL/SQL Code Checker and Dependency Analysis requirements

Thank you!

?

www.trivadis.com

trivadis
makes IT easier.

Basel Bern Lausanne Zurich Düsseldorf Frankfurt/M. Freiburg i. Br. Hamburg Munich Stuttgart Vienna