CREATE STATIC ANALYSIS RULE

STEP BY STEP TUTORIAL

This tutorial will show a step by step example of how to create a custom static analysis rule with SQL Enlight.

We are going to create a rule checking for code, which doesn't have a proper handling of divisions and doesn't guard against divide by zero errors.

GETTING STARTED

SQL Enlight analysis rules are Extensible Stylesheet Language templates, which are applied on XML syntax tree generated from T-SQL code or on XML generated from database schema.

In general, SQL Enlight analysis rules match a specific pattern in the T-SQL script or in the analysis context using XPath and result a specific to the rule message, which is reported to the user.

RULE TYPES

There are two types of rules based on their scope - Context and Batch rules.

CONTEXT RULES

The Context rules can be used to only analyze current database schema. The Analysis Context XML can be referenced in the rule expression using the following XSLT variables:

- $context - references the root Context XML element.
- $server - references the Server XML element.
- $database - references the Database XML element.

Candidates for Context Only rules are all the analysis rules that do not need to check T-SQL code, but only use the current analysis context.

BATCH RULES

The Batch analysis rules also have access to the analysis context, but are focused on the T-SQL code and its generated XML syntax tree. The syntax tree is available to the Batch rule through the $batch variable. The rule can also use the context variables to get some additional schema information form the database.

HOW THE RULES ARE APPLIED

When a script is analyzed, the Batch rules are applied separately for each of the analyzed T-SQL script batches. If there are any active Context rules when the script is analyzed, these rules are applied on schema of the currently connected database.

When a database is analyzed, the Context rules are applied only once for a database while the Batch rules are applied for each of the SQL modules contained in the database.

RULE VARIABLES

The variables $v-rulename, $v-ruledescription, $v-ruleseverity are initialized when the rule is generated and hold respectively the rule name, description and severity.

Several variables are available inside the analysis rule.

Common:

- $parameters – holds node-set of Param nodes for accessing rule parameter values

Context variables available inside both Context and Batch rules:

- $context – holds reference to the analysis context node set
- $server – references the server node in the analysis context
- $server-name – current server name
- $database – references the node of the current database on the current server
- $database-name – the name of the current context database
- $server-case-sensitive –is true when the server collation is case sensitive
- $database-case-sensitive –is true when database has default case sensitive collation

Batch variables available inside Batch rules:

- $batch – contains XML representation of the syntax tree of the current batch
- $batch-sql – contains the text of the currently analyzed SQL batch

ANALYSIS CONTEXT

The analysis context is a XML document that holds information for the current SQL Server and the current database.

STEP BY STEP

Here in several steps we will define the rule, decide the rule type, choose an implementation method, create new rule, set rule properties, add test cases to the test script, implement the rule expression, add suppression support to the rule, add a parameter to the rule and at the end test the result, and save the rule.

STEP 1: DEFINITION

In our example, we will create a rule, which to check for division operations that are not handled according the practice.

These statement could cause ‘Divide by Zero’ error in case column col2 is equal to 0.

```
SELECT col1/col2 FROM Table1
```

The best way to handle this and avoid an error is to use the function NULLIF, which returns NULL if its first argument is equal to the second argument. When dividing anything by NULL will result a NULL.

```
SELECT col1/NULLIF(col2,0) FROM Table1
```

While this executes without error, we still receive a null as a result. If you want to return a different value, you may want to wrap the equation in an ISNULL.

```
SELECT ISNULL(col1/NULLIF(col2,0), 0) FROM Table1
```

STEP 2: RULE TYPE

First we have to decide the type of the rule – Batch or Context rule.

In our case, the rule must be a Batch rule as it needs to check the T-SQL code.

STEP 3: PATTERN MATCHING

As the analysis rules match a specific pattern in the T-SQL script or in the analysis context, it is very important to decide what the rule will verify and what approach we will use to implement it.

Depending on the practice you are enforcing with the rule, the rule can use one of these methods:

A. Match that an unwanted script pattern appears in the code.

For example: To check that table hints are not used, you will have to match where a table hint is used and warn.

B. Match that the script pattern that you want have is missing.

For example: To check that all the statements must be terminated with semicolon, you will have to match and report all statements that are not terminated.

What method you will generally choose depends mainly on the variations, which each method has to cover. If there are too many unwanted script patterns and only a few wanted patterns, B will be the best choice, else the choice will be A.

In our example rule, we will use method B and report if the proper handling of divisions is missing.

STEP 4: CREATE NEW RULE

1. Open SQL Enlight options dialog, select Analysis -> Rules
2. Use the New toolbar button and select New Rule -> Batch Rule
3. The Rule Designer will appear and a new batch rule.

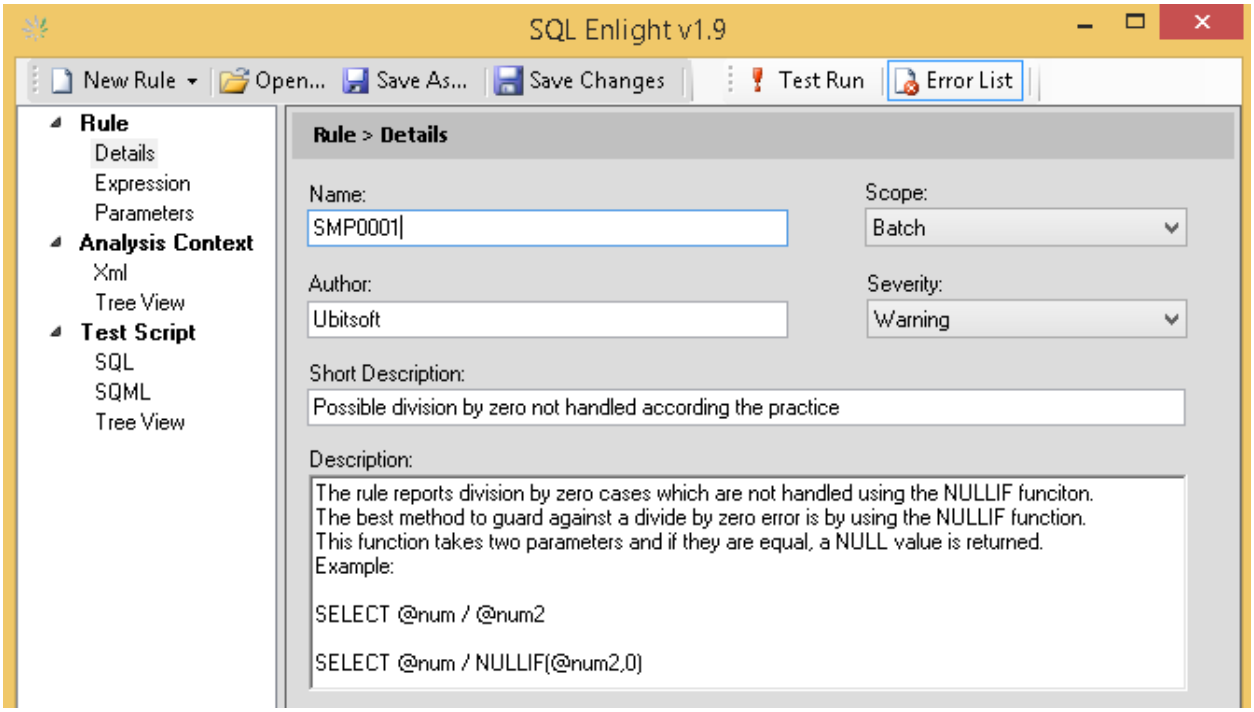
The new batch rule will not be empty, but will contain a sample expression and test script, which to be used as a starting point for writing your own rule. The sample batch rule enumerates the statements in the scripts and outputs their type code.

In our example we won't need the sample code and we will do the rule from scratch.

STEP 5: SET PROPERTIES

The properties that must be set are name, default message, description and severity.

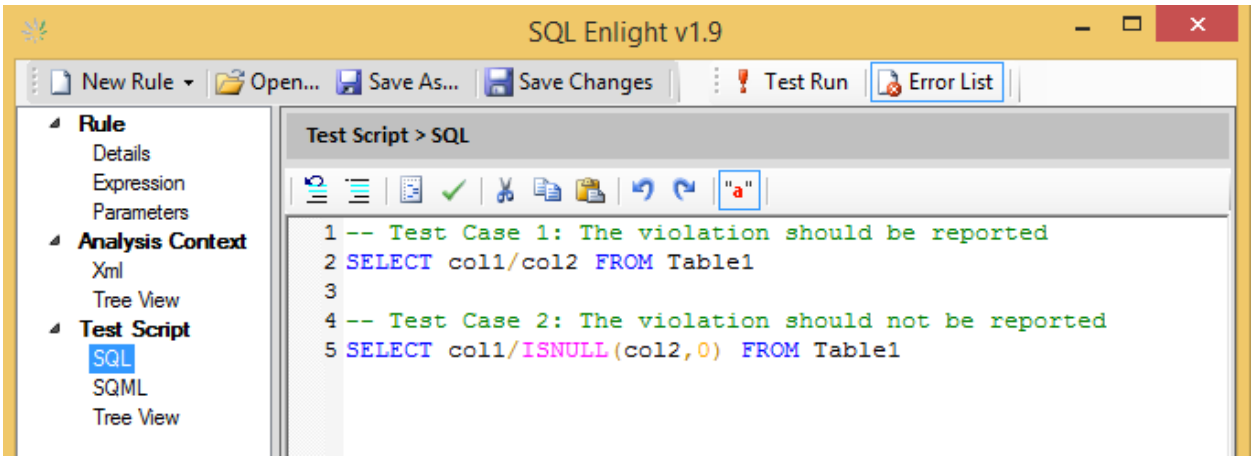
- Rule name must not contain white space
- The default message is the message that will be reported by default in case of rule violation
- The rule description is only informational and can describe the best practice that the rule is enforcing.
- Rule severity – error, warning, information or task.



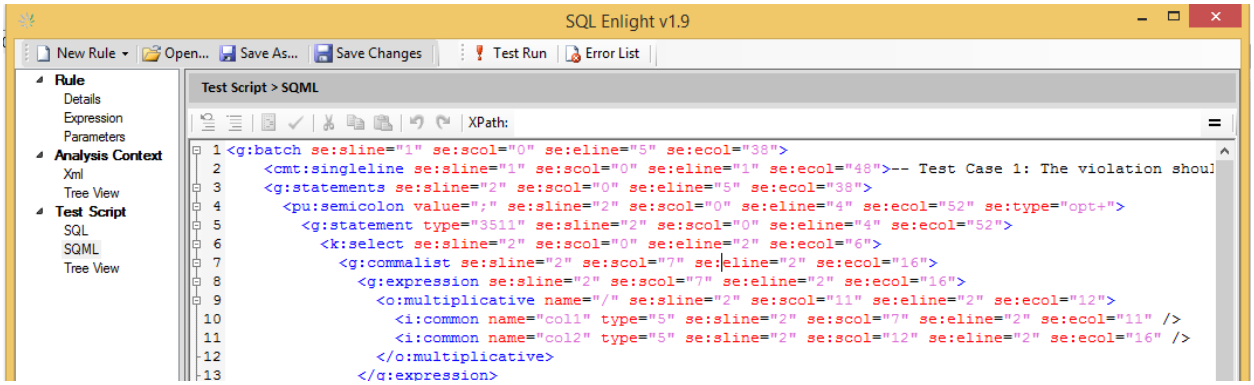
STEP 6: TEST CASES

We will need at least 2 test cases – one case for that the rule will report a violation and one case, which is according the practice and for that the rule will not report violation.

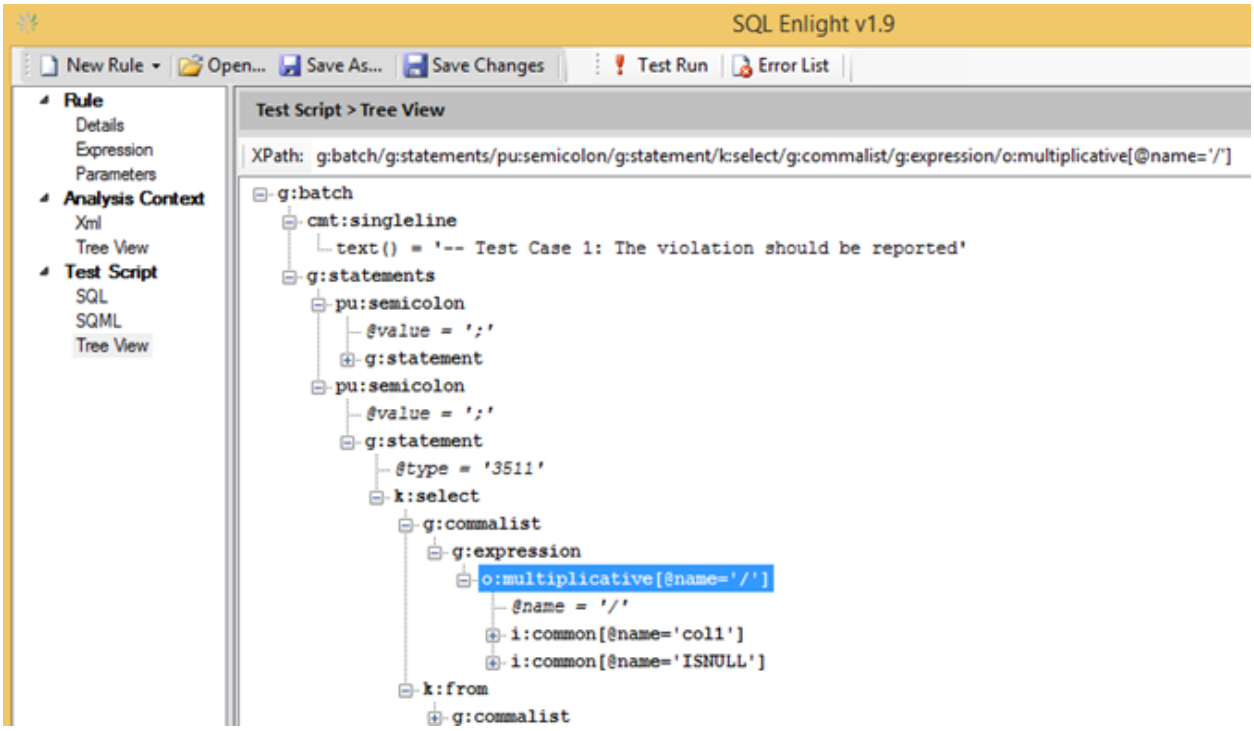
1. Open the Test Script -> SQL tab and replace the default batch rule test script with the scripts for our rule.



2. The SQML view contains the XML representation of the test script's syntax tree.



3. The Tree View tabs also contains the syntax tree, but can be used to quickly get the XPath expression to a selected node.



The SQML view and the Tree view will help us when creating the rule expression to match the nodes that the rule will test.

STEP 7: RULE EXPRESSION

The rule expression is the implementation of the rule and can be entered at the Expression tab.

In the rule expression, we will go through each statement and get all division operators in the statement. For each operator we will get its operands. The binary operator nodes contains their operands as child nodes. We will get

the right operand and will check if it is wrapped in NULLIF function. If the operand is not inside NULLIF function, we will report a rule violation.

1. Iterate each statement in the batch

```
<xsl:for-each select="$batch-statements">
<xsl:variable name="statement" select="." />
<xsl:variable name="statement-generated-id" select="generate-id(.)" />
```

2. Match division operators in the current statement

```
<xsl:for-each select="$statement//o:multiplicative[@name='/']
                      [generate-id(ancestor::g:statement[1]) = $statement-generated-id]">
<xsl:variable name="operator" select="."/>
```

The statement-generated-id variable is used to limit the scope of the division operator matching expression and to select only the operators in from one statement at a time.

3. Get left and right operands

```
<xsl:variable name="left-operand" select="$operator/*[not(self::cmt:*)][1]"/>
<xsl:variable name="right-operand" select="$operator/*[not(self::cmt:*)][2]"/>
```

4. Check if the right operand is not wrapped in NULLIF function. And set the result of the test in a local variable.

```
<xsl:variable name="division-by-zero-handled-with-nullif" select="$right-
operand/self::k:nullif/g:brackets/g:commalist/g:expression[2]/co:exact-
number[text()='0']"/>
```

5. If the variable's value is false, report error by calling the output-message template.

```
<xsl:if test="not($division-by-zero-handled-with-nullif)">
<xsl:call-template name="output-message">
<xsl:with-param name="line" select="$operator/@se:sline" />
<xsl:with-param name="column" select="$operator/@se:scol" />
<xsl:with-param name="msg" select="$v-rulename" />
<xsl:with-param name="desc" select="concat($v-rulename,' : ', $v-ruledescription)" />
<xsl:with-param name="near" select="$operator/@name" />
<xsl:with-param name="type" select="$v-ruleseverity" />
</xsl:call-template>
```

STEP 8: SUPPRESSION

We will add support for the three types of suppression: token, line and statement.

1. Add statement suppression support

```
<xsl:for-each select="$batch-
statements[not(self::g:statement/parent::pu:semicolon//cmt:*[self::cmt:* =
$RuleSuppressMarksStatement])]">
```

2. Add Line suppression support

```
<xsl:for-each select="$statement//o:multiplicative[@name='/']
        [generate-id(ancestor::g:statement[1]) = $statement-generated-id]
        [not(set:has-same-node($RuleSuppressMarksAll,.//cmt:*)]">
```

3. Add operator suppression support

```
<xsl:for-each select="$statement//o:multiplicative[@name='/']
        [generate-id(ancestor::g:statement[1]) = $statement-generated-id]
        [not(set:has-same-node($RuleSuppressMarksAll,.//cmt:*) or
        $RuleSuppressMarksLine/@se:sline = ./@se:sline)]">
```

4. Add test script for the suppression

```
-- Test Case 5: The violation should not be reported, because the rule
is ignored for the operator
    SELECT @num = @num + col1/@i  /*IGNORE:SMP0001*/ FROM Table1


-- Test Case 6: The violation should not be reported, because the rule
is ignored for the line
    SELECT @num = @num + col1/@i FROM Table1 -- IGNORE:SMP0001(LINE)


-- Test Case 7: The violation should not be reported, because the rule
is ignored for the statement
    SELECT @num = @num + col1/@i
    FROM Table1   -- IGNORE:SMP0001(STATEMENT)
```

STEP 9: PARAMETERS

The parameter that we are going to add will specify whether a default value should be required for the division expression and have to be provided using the ISNULL or COALLESCE functions.

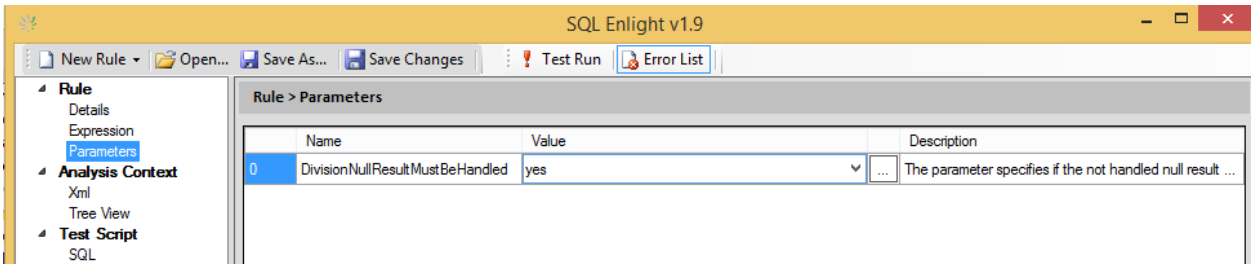
1. Add new test script that to cover the division default value requirement

```
DECLARE @i int = 5;
DECLARE @num float = 10;

WHILE @i > -5
BEGIN
    SELECT ISNULL(@num / NULLIF(@i,0),@num);

    SELECT COALESCE(@num / NULLIF(@i,0),@num);
END
```

2. Remove any existing parameters, which were added by default when the Batch rule was created.
3. Add new parameter DivisionNullResultMustBeHandled at the Parameters tab.



- Set parameter's possible values: 'yes' and 'no'.
- Ensure parameter selected value is 'yes'.
- Enter parameter description
- Add new variable in the rule expression that to contain the parameter's value.

```
<xsl:variable name="DivisionNullResultMustBeHandled"
select="boolean($parameters/Param[@Name='DivisionNullResultMustBeHandled']/text() = 'yes')"/>
```

4. Add new variable that to hold the result from the test if a default value is set for the division and the DivisionNullResultMustBeHandled parameter's value is set to 'yes'.

```
<xsl:variable name="division-null-result-handled"
select="not($DivisionNullResultMustBeHandled and
not($operator/ancestor::g:brackets/parent::*[self::k:coalesce or
str2:compare(self::i:common/@name,'isnull', true()) = 0]))"/>
```

5. Add new test condition next to the check if the division by zero is handled.

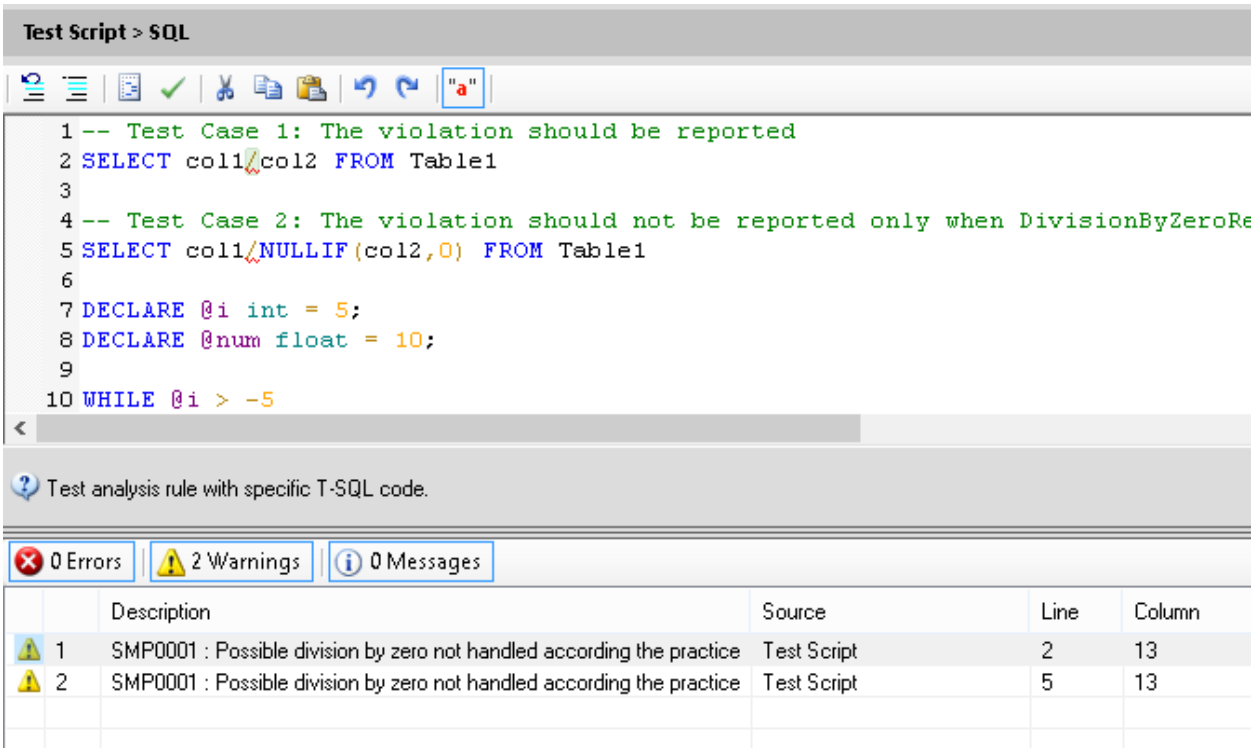
```
<xsl:if test="not($division-by-zero-handled-with-nullif) or not($division-null-result-
handled)">
```

STEP 10: TEST

We have to test the rule with DivisionNullResultMustBeHandled parameter set to its both possible values.

First set the parameter value to 'yes' and use the Test Run button.

You will get 2 rule violations:



Now set the parameter value to 'no' and run the test.

This time there should be only 1 rule violation as the second case is valid according the practice and the parameter value which in this case does not enforce a default value for the division.

STEP 11: DONE

Now when the rule is ready, we can save it in the current template using the Rule Designer's Save or Save Changes button. The rule will be added to the current analysis template and there we can further associate it in a group and enable it.

RESULT

You can download the analysis rule here, and these are the complete rule expression and test script.

RULE EXPRESSION

```
<xsl:variable name="DivisionNullResultMustBeHandled"
select="boolean($parameters/Param[@Name='DivisionNullResultMustBeHandled']/text() = 'yes')"/>


<xsl:for-each select="$batch-
statements[not(self::g:statement/parent::pu:semicolon//cmt:*[self::cmt:* =
$RuleSuppressMarksStatement])]">
        <xsl:variable name="statement" select="." />
        <xsl:variable name="statement-generated-id" select="generate-id(.)" />
        <xsl:for-each select="$statement//o:multiplicative[@name='/']
                        [generate-id(ancestor::g:statement[1]) = $statement-generated-id]
```

```
                        [not(set:has-same-node($RuleSuppressMarksAll,.//cmt:*) or
$RuleSuppressMarksLine/@se:sline = ./@se:sline)]">
                <xsl:variable name="operator" select="."/>
                <xsl:variable name="left-operand" select="$operator/*[not(self::cmt:*)][1]"/>
                <xsl:variable name="right-operand" select="$operator/*[not(self::cmt:*)][2]"/>

                <xsl:variable name="division-by-zero-handled-with-nullif"
                                        select="$right-
operand/self::k:nullif/g:brackets/g:commalist/g:expression[2]/co:exact-number[text()='0']"/>
                <xsl:variable name="division-null-result-handled"
                                        select="not($DivisionNullResultMustBeHandled and
not($operator/ancestor::g:brackets/parent::*[self::k:coalesce or
str2:compare(self::i:common/@name,'isnull', true()) = 0]))"/>

                <xsl:if test="not($division-by-zero-handled-with-nullif) or not($division-null-
result-handled)">

                <xsl:call-template name="output-message">
                        <xsl:with-param name="line" select="$operator/@se:sline" />
                        <xsl:with-param name="column" select="$operator/@se:scol" />
                        <xsl:with-param name="msg" select="$v-rulename" />
                        <xsl:with-param name="desc" select="concat($v-rulename,' : ', $v-
ruledescription)" />
                        <xsl:with-param name="near" select="$operator/@name" />
                        <xsl:with-param name="type" select="$v-ruleseverity" />
                </xsl:call-template>

                </xsl:if>
        </xsl:for-each>
</xsl:for-each>
```

TEST SCRIPT

```
-- Test Case 1: The violation should be reported
SELECT col1/col2 FROM Table1

-- Test Case 2: The violation should not be reported only when
DivisionByZeroRequiresDefaultValue parameter is 'yes'
SELECT col1/NULLIF(col2,0) FROM Table1

DECLARE @i int = 5;
DECLARE @num float = 10;

WHILE @i > -5
BEGIN
```

```
-- Test Case 3: The case is handled correctly
    SELECT ISNULL(@num / NULLIF(@i,0),@num);

-- Test Case 4: The case is handled correctly
    SELECT COALESCE(@num / NULLIF(@i,0),@num);

    SET @i = @i - 1;

-- Test Case 5: The violation should not be reported, because the rule for
the operator
    SELECT @num = @num + col1/@i  /*IGNORE:SMP0001*/ FROM Table1

-- Test Case 6: The violation should not be reported, because the rule is
ignored for the line
    SELECT @num = @num + col1/@i FROM Table1 -- IGNORE:SMP0001(LINE)

-- Test Case 7: The violation should not be reported, because the rule is
ignored for the statement
    SELECT @num = @num + col1/@i
    FROM Table1   -- IGNORE:SMP0001(STATEMENT)

END
```

LINKS

The analysis rule created in the tutorial: http://ubitsoft.com/downloads/smp0001.zip

SQL Enlight for SSMS home page: http://ubitsoft.com/products/sqlenlight-for-ssms/

SQL Enlight online documentation: http://ubitsoft.com/products/sqlenlight/help_19/Index.html

Shared rules: http://ubitsoft.com/products/sqlenlight/analysis-rules.php