TALLINNA TEHNIKA ÜLIKOOL
Infotehnoloogia teaduskond
Informaatika instituut
Tarkvararatehnika õppetool

# Andmebaasi skeemi muudatuste verifitseerimise rakendus

Bakalaureusetöö

|  |  |
|---|---|
| Üliõpilane: | Sergei Kossik |
| Üliõpilaskood: | 112527IAPB |
| Juhendaja: | Kaarel Allik |

Tallinn
2015

# Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

| | |
|---|---|
| (*kuupäev*) | (*allkiri*) |

# Annotatsioon

Selle lõputöö eesmärk oli projekteerida ja rakendada prototüüp, mis kontrollib andmebaasi skeemi muudatusi. Loodud rakendus on käivitatav konsooli terminalist. Alguses loeb rakendus andmebaasi uuendust sisaldava koodi faili ja andmebaasi süsteemi tabeleid. Järgmises etapis võrdleb rakendus andmebaasi struktuuri ja koodi, mis pidi seda muutma. Rakendus annab tagasisidet logifailide kaudu. Logifailid sisaldavad nii edukalt kui mitteedukalt uuendatud objektide nimekirju.

Lõputöö kirjeldab rakendusele esitatavaid nõudeid ja pakub võimalikke lahendusi. Läbiviidud võimalike lahenduste analüüsi tulemusena on välja valitud kõige paremini sobiv lahendus. Lisaks on kirjeldatud rakendamiseks vajalikud detailid. Lõpuks on välja toodud rakenduse arendamise perspektiivid tulevikuks.

Lõputöö käigus sai püstitatud probleem lahendatud. Töötav prototüüp on projekteeritud, implementeeritud ja testitud. Valmis prototüüpi saab kasutada edasise arendustöö baasina.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 48 leheküljel, 4 peatükki, 9 joonist, 5 tabelit.

## Abstract

The aim of this thesis is to design and implement an application to verify database schema changes. The application is ran from a command line. The software scans update script file and database's system tables for updates. After that a comparison of the script and the database structure is performed. The application gives feedback by writing log files during validation. Log files contain list of database objects that were successfully updated or failed to update.

The thesis describes requirements to the software and possible solutions of database schema control. Analysis of possible database schema control methods was done in order to choose optimal solution. In addition, implementation details are provided as well as plans for the future modification.

During this work, a solution to a problem was found. Also, a working prototype was designed, implemented and tested. The working prototype can be used as a foundation for further development.

The thesis is in English and contains 48 pages of text, 4 chapters, 9 figures, 5 tables.

# Glossary of terms and abbreviations

| | |
|---|---|
| **XML** | ***Extensible Markup Language*** a markup language for documents containing structured information [1] |
| **SQL** | ***Structured Query Language*** is a programming language designed for managing data held in a relational database management system [2] |
| **Java SE** | ***Java Standard Edition*** Java Platform allows to develop and deploy applications for desktops and servers [3] |
| **HTML** | ***HyperText Markup Language*** is the Web's core language for creating documents and applications [4] |
| **Use case** | ***Use Case*** is a methodology used in system analysis to identify, clarify, and organize system requirements [5] |
| **Database entity** | ***Database entity*** entity is a thing or object of importance about which data must be captured [6] |
| **DDL trigger** | ***Data Definition Language trigger*** used to define database schema [7] |
| **DML trigger** | ***Data Manipulation Language*** used to modify data in database tables [8] |
| **JDBC** | ***The Java Database Connectivity*** an API for connectivity between the Java application and a database [9] |
| **DAO** | ***Data Access Object*** |

| | |
|---|---|
| | a pattern used to abstract and encapsulate all access to the database [10] |
| **Regular expression** | is a pattern that the regular expression engine attempts to match in input text [11] |
| **Lazy quantifier** | causes the regular expression engine to match as few occurrences as possible [12] |
| **StringBuilder** | a java class providing API for manipulation with text [13] |
| **JSP** | ***JavaServer Pages*** |
| | technology provides a simplified way to create dynamic web content [14] |
| **HashMap** | Java hash table based implementation, providing key-value container [15] |
| **JPA** | ***Java Persistence API*** |
| | provides persistence tools for accessing database [16] |
| **POJO** | ***Plain Old Java Object*** |
| | Ordinary Java object, not bound by any special restriction [17] |
| **JVM** | ***Java Virtual Machine*** |
| | An abstract computing machine that enables a computer to run a Java program [18] |
| **JRE** | ***Java Runtime Environment*** |
| | Java Virtual Machine implementation [18] |
| **JAR** | ***Java Archive*** |
| | JAR format enables to bundle multiple files into a single archive file [19] |

# Table of images

## Table of tables

# Table of Contents

# Introduction

The aim of this thesis is to design and implement console application to check database schema integrity after creation or update. Author received multiple complaints about unsuccessful database creation or update working as part of Profit Software OY [20] which is company designing pension and savings systems. Often corrupted database errors were left unnoticed and developers were inspecting web-tier code instead of checking database. Also, developers had very little feedback how client server works. Server administrators were not always able to share server logs with developers, because it could contain sensitive data. Finally the decision was made to develop application validating update process.

# 1. Problem

## 1.1 Problem description

Profit software OY [20] is a software developing company producing insurance products. Most of the products are savings and pension solutions. In these products clients invest money during many years and, when the time comes, receive their accumulated money back. This is the reason why product has to be safe and stable. Project owners often avoid changes in software because they are afraid that updates may lead to new errors. Also, workers will spent extra time adopting and learning the updated system. During this extra time there will be less agreements handled. In other words project owner might lose money. As a result company is dealing with some old legacy solutions where implementation cannot be changed so rapidly. Solutions become difficult to maintain.

One of the difficulties our company encountered was changing database schema of a product which is in production and being actively used. When product is ready it is packed into delivery package which consists of web-application archives, database creation scripts and parameters. Parameters change system behavior affecting localization, turning some content on or off and many more. Parameters are defined as XML file which is imported during installation into the database. During installation a database is populated by database objects from database creation scripts.

When a fix or a new version of a product is released, a delivery package is recreated and sent to the customer. Our company does not provide servers for customer products. This means our product is installed without our control to an unknown server. Often a server administrator forbids direct access to the server. After installation we are unable to get a quick feedback. Even getting log files takes time. In addition log files contain a great deal of extra information. When a problem is found in a log file another fix is prepared and sent to the database server administrator, who installs the fix again. This process is repeated until problems are eliminated.

Eventually decision was made to design and implement solution simplifying error tracing at the database level after update.

## 1.2 Thesis aims

Main goal of this thesis is to design and implement a software capable of checking status of a database schema after an update.

Subgoals:

1. Analyze requirements to software and database capabilities
2. Analyze possible solutions and choose the most suitable
3. Implement and test software

## 1.3 Thesis workflow

Thesis work is divided into 3 main parts.

Chapter 2 describes requirements to software. Use cases with description provided. Also, possible solutions with weak and strong sides are described. Finally chosen solution is declared.

Chapter 3 gives system overview of system components. Implementation details of components and theirs structure are presented. User interface and testing are described.

Chapter 4 describes plans and possibilities for future development and customization of a software.

# 2. Analysis

## 2.1 Requirements to the software

Main goal of the software is to verify accomplishment of database schema updates.

### 2.1.1 Functional requirements

- Solution is intended for a SQL relational database. IBM DB2 [21] is the target database.
- Software validates a file with SQL statements.
- Software verifies a database schema for updates.
- Software can be reconfigured to work with other databases.
- Result of validation is recorded in logs and reports.
- Application is run from command line.
- Application tracks CREATE, ALTER, DROP operations affecting database schema.
- Application tracks the structure of the following database objects: Table, Index, Trigger, Sequence, Function and Procedure.

### 2.1.2 Nonfunctional requirements

- Software must be runnable on both Linux [22] and Windows [23] systems.
- Software supports at least Java SE 7.
- Software must be configurable by external XML file.
- Logs are generated as text files.
- Reports are generated as HTML files.

- Software is lightweight.
- SQL script file must be checked for valid SQL statements and invalid statements should be marked in both log and report files.
- List of successful and failed updates is created and saved in both log and report files.
- Verification process takes less than a minute.
- At least 200 SQL statements can be handled at once.

## 2.2 Use cases

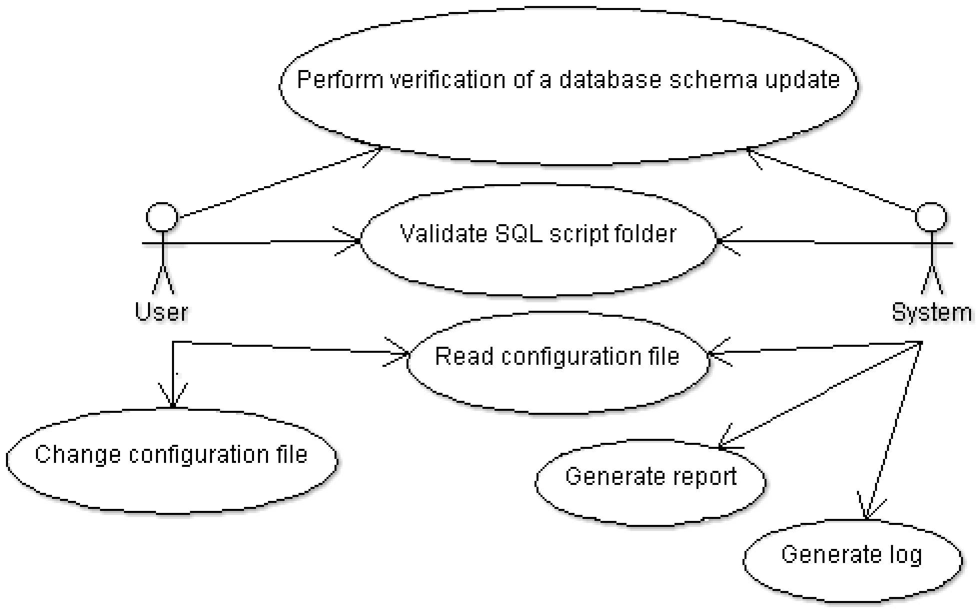

Perform verification of a database schema update
Validate SQL script folder
User
System
Read configuration file
Change configuration file
Generate report
Generate log

**Figure 1. Use case diagram**

**Use case title:** Perform verification of a database schema update

**Actors:** User, System

**Description:** User starts application, choses option to verify database for updates based on an update script. System scans SQL files, queries database, performs comparison of expected database schema to actual.

**Use case title:** Validate SQL script folder

**Actors:** User, System

**Description:** User starts application, choses option to validate folder containing files with SQL statements for syntax correctness. System validates a folder with SQL files for syntactical correctness based on parameters provided in configuration file.

**Use case title:** Read configuration file

**Actors:** User, System

**Description:** User can examine configuration XML file. System reads configuration file to set parameters of an application. For example, system generates templates for database objects based on configuration file.

**Use case title:** Change configuration file

**Actors:** User

**Description:** User can open the configuration XML file in a text editor, change parameters and save it. System reads configuration file and adjusts parameters of an application. For example, system generates templates for database objects based on the configuration file.

**Use case title:** Generate log

**Actors:** System

**Description:** System generates log during execution time to register application's workflow.

**Use case title:** Generate report

**Actors:** System

**Description:** System generates report. Report contains information about validation and verification processes.

## 2.3 Database objects

A Relational database is based on the relational model of E.F. Codd. [24] A database is a collection of tables. Tables represent objects or entities. Columns in a table represent fields and rows represent attributes of fields. Tables can relate to each other by referencing columns. However, there are other database objects like indexes, procedures, views, triggers, constraints, sequences and functions. These objects help to keep and retrieve data from a relational database more efficiently. The relational database structure is defined by Structured Query Language (SQL).

Below are provided general SQL clauses for database schema alteration. This means clauses are not database specific. In the scope of this thesis modification of the following objects' structure will be tracked:

**Table**

The database table can have 1-n columns. Each column can have properties like type, maximum length, default value and others.

Table structure can be modified by clauses:

*CREATE table_name (column_name1 data_type(size), ... , column_nameN data_type(size));*

*ALTER TABLE table_name ADD column_name new_data_type;*

*ALTER TABLE table_name DROP column_name;*

*ALTER TABLE table_name MODIFY column_name new_datatype;*

*DROP TABLE table_name;*

### View

The database view is a virtual table, which can be created from multiple tables and expose only some columns. The view is usually read-only.

View structure can be modified by clauses:

*CREATE OR REPLACE VIEW view_name AS clause;*

*ALTER VIEW view_name parameters;*

*DROP VIEW view_name;*

### Index

The index is used to improve data search in a table. An index can be created on one or more database columns. Index implementations can vary.

An index structure can be modified by clauses:

*DROP INDEX index_name;*

*CREATE INDEX index_name ON table_name (column_name1, ..., column_nameN) ...;*

Additional index creation parameters are omitted.

### Procedure

The stored procedure is a subprogram ran inside the database to fetch data. Additional conditional logic can occur in procedure to fetch only specific data. A stored procedure can return either multiple or one fetched row or single data like integer or string. It is not mandatory for procedure to return a value.

Procedure structure can be modified by clauses:

*DROP PROCEDURE procedure_name;*

*CREATE PROCEDURE procedure_name (procedure body)*

Procedure body details are omitted.

**Function**

The function is subprogram ran inside database very similar to procedure. However, a function must always return data.

Function structure can be modified by clauses:

*DROP FUNCTION procedure_name;*

*CREATE FUNCTION procedure_name (function body) END;*

Function body details are omitted.

**Trigger**

The trigger is a subprogram ran on a particular event. Usually when data in a row or table is modified, trigger is executed.

Trigger structure can be modified by clauses:

*CREATE TRIGGER trigger_name (trigger body) END;*

*ALTER TRIGGER trigger_name (trigger body);*

*DROP TRIGGER trigger_name;*

Trigger body details are omitted.

**Sequence**

The sequence is used to generate a sequence of unique integer values. A sequence can have adjustable start value, step size and cache size. In most of the cases a sequence is used for providing unique values for primary key columns.

*CREATE SEQUENCE sequence_name (sequence body);*

*ALTER SEQUENCE sequence_name (sequence body);*

*DROP SEQUENCE sequence_name;*

Sequence body details are omitted.

## 2.4 Possible solutions

Before starting the implementation, analysis of possible solutions was made. Below are provided solutions which were considered as possible candidates. The brief description with advantages and disadvantages (pros and cons) is also included.

### 2.4.1 Using database triggers

Database triggers allow to execute particular procedure before or after some event. The DDL trigger is fired by *CREATE, ALTER* and *DROP* statements, while DML trigger is fired by *INSERT, UPDATE* and *DELETE* statements. [25]

The idea was to alter the database schema and make triggers part of a project schema. All user-defined tables will get triggers injected. Updating the database schema will result in an event triggered. The event will write message into the log table.

After further investigation it became clear that the IBM DB2 does not support DDL triggers. Also, DML triggers have to be recreated if tracked table was erased. Triggers cannot track modification of non-table elements, for example, triggers, indexes, sequences and others. In addition, it is impossible to create a trigger on the system table where the database object structure is stored, because of permission and security issues.

### 2.4.2 Using database DDL export tools and comparing exported to imported

Most databases have built-in DDL export tools available. For example, IBM DB2 uses the **db2look** [26] command to extract database objects as a DDL script.

A possible solution could be a program scanning an update script file, determining types and names of objects to be modified. After that the **db2look** command with necessary arguments is prepared based on the scanned update script and executed. Result extracted by the **db2look** tool is saved into a file. After extraction file, generated by **db2look** tool**,** is parsed. Parsed result is compared to the original update script file. The result of comparison is saved into a log file.

| **Pros** | **Cons** |
|---|---|
| Preparations of a database are minimal. No need to inject any triggers or tables to monitor the database schema alteration. The software is fully external. | File exported with the db2look utility can be difficult to parse. Especially if vendor will decide to change format of the DDL file generation. |
| No need to query a database with SQL statements. Only command line arguments for db2look are provided. | The exported file generated with db2look utility contains some extra information. |
|  | Not all database objects store the time of creation and alteration. |

**Table 1. Comparing exported to imported with DDL**

### 2.4.3 Making custom installer

A different approach is making an installer to control whole update process. The software scans an update file for SQL statements. The connection with the database is established. Every SQL statement is executed through the JDBC driver. The result of execution for each statement is generated and saved in a log file.

| Pros | Cons |
|---|---|
| Preparations of a database are minimal. No need to inject any triggers or tables to monitor the database schema alteration. The software is fully external. | If an update was run manually without the installer, second run may generate errors that some objects are already created, updated or deleted. |
| A lightweight tool. | |
| Whole update process is under control. | |

**Table 2. Making custom installer**

### 2.4.4 Scanning technical tables for database objects and their attributes

Another solution is creating an application parsing SQL update script. Software gets type and name of the database object. Then a query to a technical table in a database is initialized. Technical tables contain the information about database objects, for example, creation time, alteration time, number and type of columns (in case of table) and many more. Next step is to compare the expected database object to the actual database object. Last step is to generate logs based on a result.

| Pros | Cons |
|---|---|
| Preparations of a database are minimal. No need to inject any triggers or tables to monitor the database schema alteration. The software is fully external. | The update script algorithm must be powerful enough to identify subtypes like constraints, columns, column length, column type and other important parameters. |
| A lightweight tool. | The configuration file size is estimated to be large. |
| | The SQL query complexity grows. It is required to return all parameters required for validation. |

**Table 3. Scanning technical tables for database objects and their attributes**

### 2.4.5 Scanning technical tables for database objects for create and alter time only

This solution is derived from the previous solution, but simplified. An assumption is made that if the update script was successful (finished without errors) the creation time and/or alteration time would change. This means if a database object was found in the database and the creation time or alteration time is within the period specified by a user, this means the object was successfully updated.

| **Pros** | **Cons** |
|---|---|
| Preparations of a database are minimal. No need to inject any triggers or tables to monitor the database schema alteration. The software is fully external. | Not all database objects store the time of creation and alteration. |
| A lightweight tool. | |

**Table 4. Scanning technical tables for database objects for create and alter time only**

### 2.4.6 Chosen solution

*Scan technical tables for database objects for create and alter time only* was eventually chosen as the solution to be implemented.

# 3. Solution

## 3.1 System overview

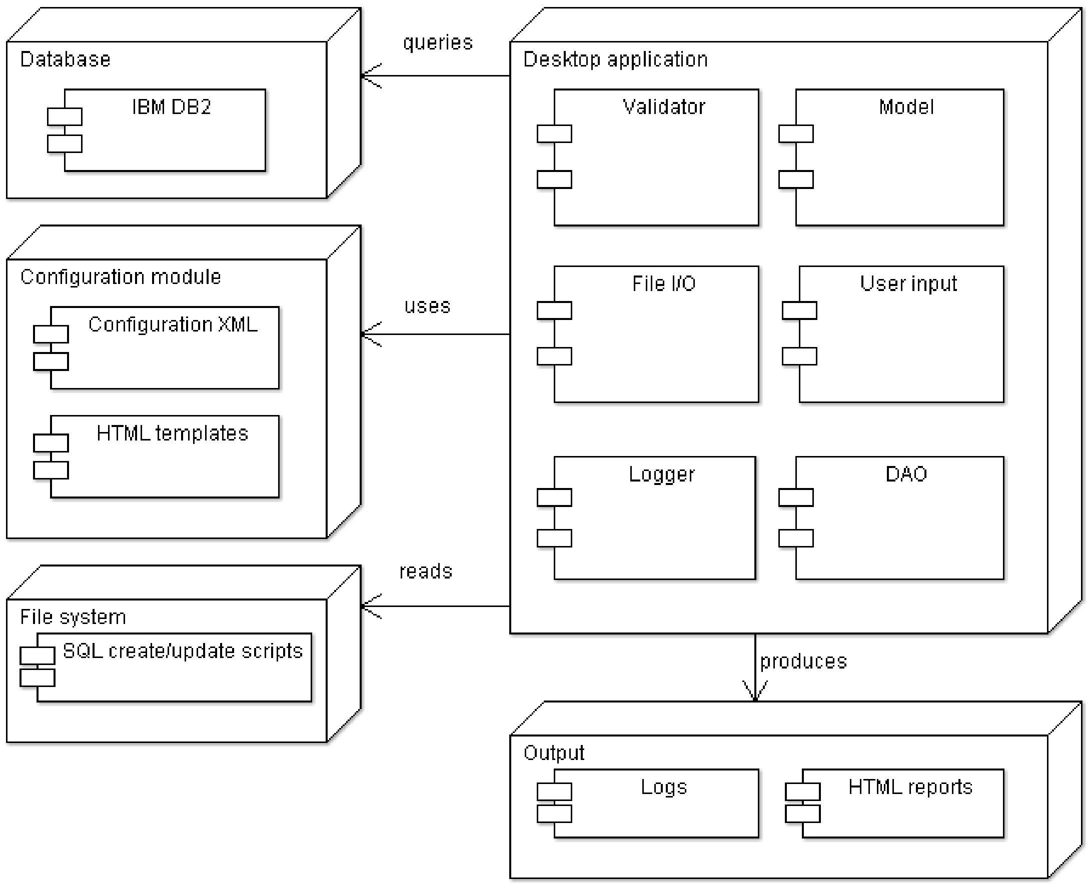


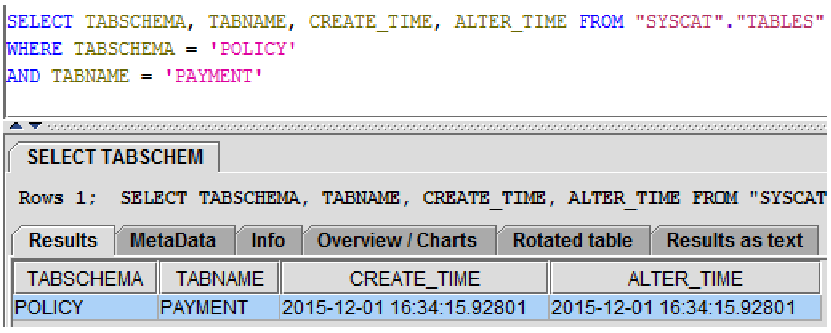
**Figure 2. System domain model**

The desktop application is configured by an XML file from the configuration module. Also, HTML templates are used from the same module to generate HTML report files. SQL scripts are read from a filesystem by the file input/output (I/O) [27] module. The path to the folder with scripts is specified in the configuration module. The output module provides generation of logs and html reports. The application accesses the database via the DAO submodule based on settings from the configuration file.

## 3.2 Getting objects from a database

After deeper investigation of the IBM DB2 database it became clear that the information about database objects can be queried from system tables. The *SYSCAT* schema [28] provides a set of views capturing objects' status in database.

For example, to get information about table *PAYMENT* in schema *POLICY* the following query to *SYSCAT.TABLES* is executed:



```
SELECT TABSCHEMA, TABNAME, CREATE_TIME, ALTER_TIME FROM "SYSCAT"."TABLES"
WHERE TABSCHEMA = 'POLICY'
AND TABNAME = 'PAYMENT'
```

```
Rows 1;  SELECT TABSCHEMA, TABNAME, CREATE_TIME, ALTER_TIME FROM "SYSCAT
```

| TABSCHEMA | TABNAME | CREATE_TIME | ALTER_TIME |
|---|---|---|---|
| POLICY | PAYMENT | 2015-12-01 16:34:15.92801 | 2015-12-01 16:34:15.92801 |

**Figure 3. Query to SYSCAT.TABLES view**

The list of views below contains the information about objects required to capture in the scope of this thesis:

- SYSCAT.INDEXES
- SYSCAT.PROCEDURES
- SYSCAT.TRIGGERS
- SYSCAT.TABLES
- SYSCAT.FUNCTIONS
- SYSCAT.SEQUENCES

## 3.3 User input state diagram

The user input is read from a command line. Some servers run under the Linux operating system without a graphical user interface. Also, this software is intended for people familiar with a command line workflow.

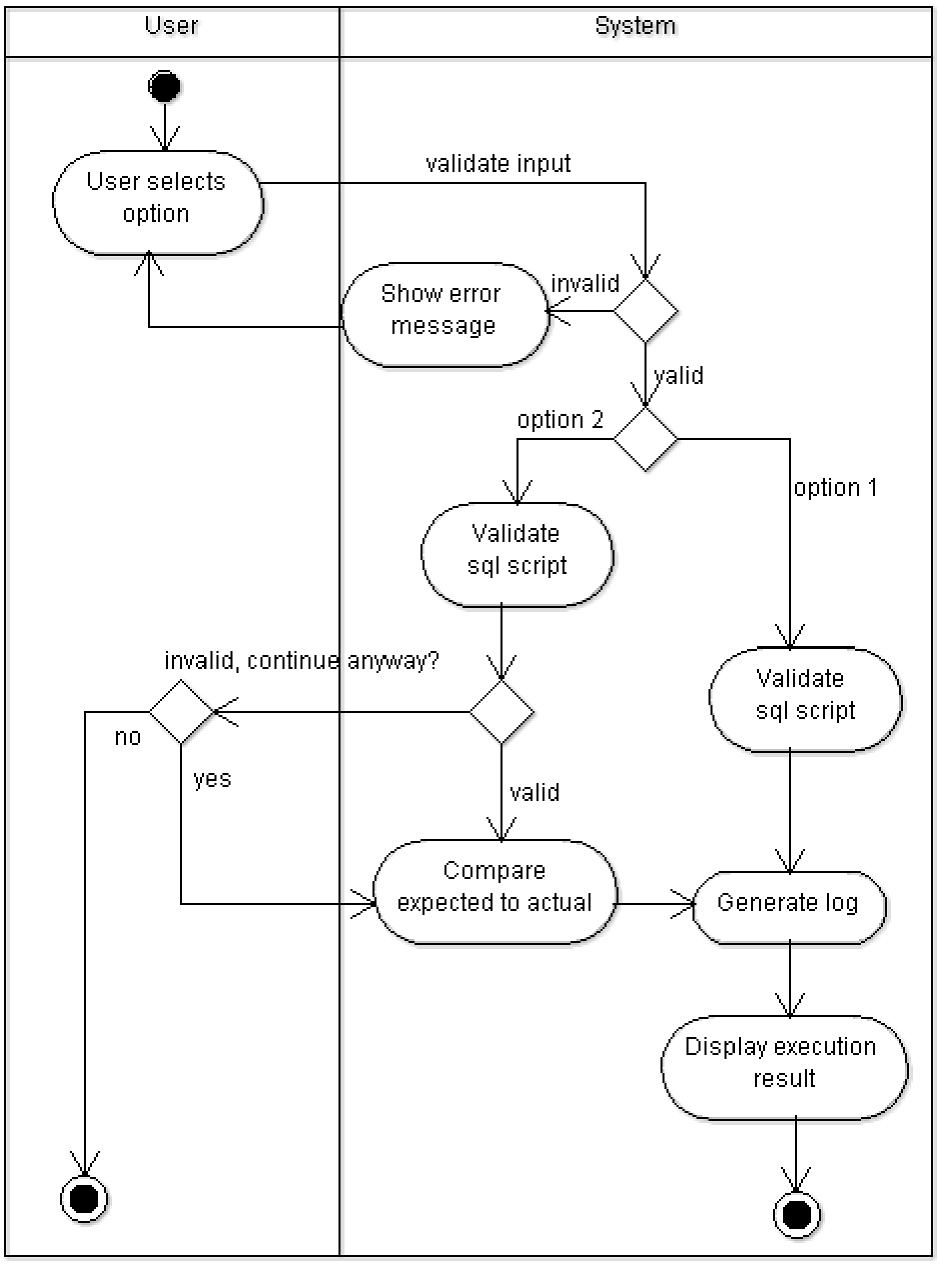


**Figure 4. User input state diagram**

## 3.4 Configuration file

The application configuration is specified in an XML file to avoid hard-coding of parameters in java classes. If the database implementation changes or it is required to validate a new type, developer can customize the existing XML file to satisfy new requirements and avoid recompiling Java modules. The configuration file consists of 3 main parts.

### 3.4.1 Database properties

Database properties are described between <*database*></*database*> tags. Properties declare database connection parameters. A hostname, a username, a password, a port, a database name and a driver type.

### 3.4.2 Global properties

Global properties are declared between <*global*></*global*> tags and reflect overall application parameters:

1. **The delimiting character between SQL statements** – in case of the IBM DB2 and this thesis it is the *"@"* (at) sign. The delimiting character is used to separate SQL statements in a text file so only one statement is executed at once. The delimiting character is enclosed by <*statementDelimiterCharacter*> tags.

2. **The time interval** – has attributes *intervalStart* and *intervalEnd* reflecting the time and the date interval when a SQL script was executed.

3. **The path to the SQL script folder** – Enclosed by <*sqlFolderLocation*> tags. Declares an absolute path to the folder. [29]

4. **Enable generation of HTML reports** – a boolean value either *true* or *false*. Enclosed by <*generateHtmlReport*> tag.

### 3.4.3 Database object mappings

The core of the configuration file is a database object mapping. The mapping is a set of templates. Each template maps to a specific SQL statement and defines validation methods. Regular expressions are used to identify and to extract required text values.

The template describes the following:

- The way a statement is identified. Regular expressions are used to find a match. For example, regular expression can tell the difference between *CREATE TRIGGER* and *DROP TABLE* statements.
- The text data being extracted from an SQL statement with the help of regular expressions.
- The SQL testing statement to check the status of an object in a database.
- Columns being selected from the SQL testing statement result set.
- Strategies for validating each column.

The full template for *CREATE FUNCTION* statement can be found in the Appendix 1. Below the *CREATE FUNCTION* template is explained as an example line by line:

```
<object type="FUNCTION" operation="CREATE" objectNameParameter="FUNCTION_NAME_IN"
objectSchemaParameter="FUNCTION_SCHEMA_IN">
```

**<object>** tag attributes definition:

***type*** - defines the object type. In the scope of the thesis only ***function, procedure, sequence, trigger, index*** and ***table*** objects are supported.

***operation*** – defines an allowed operation with this object. In the scope of this thesis only ***create, alter*** and ***drop*** operations are supported.

***objectNameParameter*** – a reference to the regular expression extracting an object name.

***objectSchemaParameter*** – a reference to the regular expression extracting an object schema.

```
<regExpStatement regExpType="VALIDATION">
          "^CREATE\s+FUNCTION\s+\w+\.\w+\s*\(\s*.+"
</regExpStatement>

<regExpStatement regExpType="FILTER_OBJECT_NAME" parameterName="FUNCTION_SCHEMA_IN"
groupNumber="1">
    "FUNCTION\s+(.+?)\.\w+\s*\(.+"
</regExpStatement>
```

**<regExpStatement>** tag attributes definition:

***parameterName*** – is referenced by *objectNameParameter* or *objectSchemaParameter*. Used only with filtering regular expressions.

***groupNumber*** – defines number of a group to be extracted by regular expression. Used only with filtering regular expressions.

***regExpType*** – marks the regular expression between tags to be used for validation or filtering.

***Validation regular expression*** (regExpStatement = *VALIDATION*) is used to identify the statement being parsed and choose the corresponding template. In the scope of this thesis there is no need to validate a whole statement, because only object's schema and name are required for validation. An assumption is made that statements in the SQL update script are valid and executable. For example, the regular expression ***^CREATE\s+FUNCTION\s+\w+\.\w+\s*\(\s*.+*** matches statement

***CREATE FUNCTION POLICY.FUNC1(...***

The part of a statement marked by dots is omitted and not validated.

***Filtering regular expression*** (regExpStatement = *FILTER_OBJECT_NAME*) is used to extract the text from the SQL statement. The group number is fetched from *groupNumber* attribute and used for a lazy qualifier in a regular expression processor.

For example, ***FUNCTION\s+(.+?)\.\w+\s*\(.+*** statement applied to *CREATE FUNCTION POLICY.FUNC1(...* will search a text for the first occurence of *„POLICY“* surrounded by ***FUNCTION(one or more whitespaces)*** and ***dot (one or more word characters) (zero or more whitespaces) (opening round bracket) (one or more any type of characters)***.

```
<sqlString>
SELECT FUNCSCHEMA, FUNCNAME, CREATE_TIME FROM "SYSCAT"."FUNCTIONS"
WHERE FUNCSCHEMA = '${FUNCTION_SCHEMA_IN}'
AND FUNCNAME = '${FUNCTION_NAME_IN}'
</sqlString>
```

<sqlString> tags enclose the SQL clause query to the technical tables in a database. The text in **${parameter_name}** pattern is parsed and replaced by corresponding value extracted by *regExpStatement*.

For example, from *CREATE FUNCTION POLICY.FUNC1(...* substring *POLICY* is extracted and *'${FUNCTION_SCHEMA_IN}'* is replaced by *'POLICY'*.

```
<outputParam id="FUNCNAME">
    <strategy>IS_NOT_NULL</strategy>
</outputParam>

<outputParam id="CREATE_TIME">
    <strategy>IS_NOT_NULL</strategy>
    <strategy>TIME_IN_INTERVAL</strategy>
</outputParam>
</object>
```

**<outputParam>** tag describes an output parameter and a validation strategy for a fetched value. The attribute *id* must be equal to one of the columns' name of a *SELECT* statement between <sqlString> tags.

**<strategy>** tags enclose the validating method.

In this thesis only following validation strategies are supported:

1. *IS_NULL* – expects value to be null. *OutputParam* value returned is *null* and not present in the database.
2. *IS_NOT_NULL* – expects value to be not null. *OutputParam* value returned is not *null*. Means an object exists in the database.
3. *TIME_IN_INTERVAL* – expects value to be in a *timestamp* format. Also, fetched date and time must be within interval specified in global properties.

## 3.5 XML parser

Converting the text data into objects described in the configuration file requires an XML parser. The XML parsing library should be able to read a text file, identify different tags and tag attributes. The configuration file contains relatively small amount of data required to parse. One element can contain text between tags. Tags can contain attributes.

### 3.5.1 XML parser types

XML parsers are divided into: *document streaming parser* and *document object model (DOM) parser*. [30]

DOM method loads entire XML tree into the memory, while streaming method reads file element by element, processes the data and occupies the memory less than DOM method.

The streaming parsing divides into two methods: the *push parsing* method and the *pull parsing* method. [30]

The streaming *push parsing* refers to a programming model in which an XML parser sends (pushes) XML data to the client as the parser encounters elements in an XML infoset – that is, the parser sends the data whether or not the client is ready to use it at that time. [30]

The streaming *pull parsin*g refers to a programming model in which a client application calls methods on an XML parsing library when it needs to interact with an XML infoset – that is, the client only gets (pulls) XML data when it explicitly asks for it. [30]

### 3.5.2 Choosing an XML parser

**SAX parser** – (Simple API for XML) is an event-driven XML streaming-parser based on the *push* method. Event-driven means when particular event is triggered, a developer must provide the handling code. If event was not handled, the information generated in this event will be lost. Events are, for example, encountering a beginning tag, an ending tag, or a text. When parsing is initialized, whole file is parsed with one run. There is no way to iterate step by step over the file. Also, it is impossible to go back and read previous tags. A SAX parser keeps in memory only open tags. After reaching a closing tag an element is removed from the memory. However, the advantage of this approach is a low memory consumption. [31]

**StAX parser** – (Streaming API for XML) is an XML streaming-parser based on the *pull* method. The main difference of StAX from SAX is ability to iterate the file step by step. It is possible to stop whenever necessary, however, it is still impossible to read previous elements. The mechanism of finding needed data is checking for starting and ending tags. Similar to a SAX parser a StAX has small memory consumption and is suitable for parsing large files. [30]

**DOM parser** – (Document Object Model). DOM parser is completely different from SAX and StAX parsers. DOM parser loads entire XML tree into the memory. As a result, it is possible to access any element in any order once a tree is loaded. A drawback of this parser is a significant memory consumption.

For a project where the memory footprint is small and a chance of running out of memory is small DOM parser is suitable. Also, ability to access any object in the object tree simplifies implementation.

## 3.6 Logging framework

After application has finished execution a summary text report must be generated. A summary report contains results of a database schema update verification and a list of failed and successfully updated database objects. Also, it is wise to generate the technical log during the program execution for future debugging. If error occurs in a client's environment, a client can send logs and a source of an error can be detected. A simple console output is also present to notify user about an execution status.

The logging information is produced by:

| | Implementation | Saved | Available at | Information type |
|---|---|---|---|---|
| Runtime output | Log4j | Yes | out/technical_log.log | All technical information during program execution |
| Summary output | Log4j | Yes | out/summary.log | Results of verification. |
| Console output | System.out | No | Console or terminal window | User interface. User dialog. |

**Table 5. Log generators**

Instead of implementing a logging framework from scratch the log4j [32] implementation with SLF4J [33] facade are used.

Log4j logger allows to output logs into multiple files in parallel. Also, verbose levels can be adjusted. Verbose levels filter log messages. Message will be printed only when message's verbose level is lower or equal to global verbose level. For example, if verbose level is set to *error* than only *error* and *fatal* log messages will be printed. *Trace*, *debug*, *info* and *warn* will be ignored.

Good practice is to use a logging interface with some general logging methods provided and allow end-user to choose a desired logging system. For this reason SL4J is used as simple facade interface and log4j as an implementation framework.

Another severe advantage of using SL4J is a performance increase. When log4j prints text with some variables concatenated to text, it does not actually concatenate, but creates new instance of String for every concatenation sign it encounters. While SL4J simply replaces "{}" symbol in a text with a variable.

Below is an example message being logged with log4j and SL4J:

```
// Example of logging with log4j. String concatenation is used with "+" sign. Slow
logger.trace("templateFullPath: " + templateFullPath);

// Example of logging with SL4J. No string concatenation. Fast
logger.trace("templateFullPath: {}", templateFullPath);
```

A problem arises when log message consists of many bits of text concatenated together and concatenation is running in a loop. A message gets constructed even if debugging level is set to high level, but a log message should be printed only at a very low level. This means every extra line using too many concatenation will decrease performance independent of current logging level.

Loggers are declared in the beginning of the class:

```
private static final Logger logger = LoggerFactory.getLogger(Main.class);
private static final Logger loggerSummary = LoggerFactory.getLogger("summary");
```

Setup above allows to write log messages into multiple files in parallel. One logger is configured to log messages into *technical_log.log* file and requires only class name as an argument. Second logger generates messages into *summary.log* file.

## 3.7 Generation of HTML reports using FreeMarker template engine

In addition to plain text logs HTML reports are generated. A main disadvantage of plain text logs is difficulty to visually find failed component. A log file is overloaded with extra information.

The initial plan was to generate HTML reports using Java’s StringBuilder and concatenating HTML content with values. However, this approach has many flaws.

First, HTML tags hardcoded in Java class make file look hard to read and difficult to maintain. Second, change of style would require to recompile Java class. Third, separation of concerns is violated.

A better approach is to use a template engine for HTML file generation. At first the most intuitive seems to be the JSP template. However JSP requires a servlet container to run. Also, JSP solution is too massive for a small application.

Finally, decision was made towards Apache FreeMarker template engine. FreeMarker provides the functionality of generating HTML web pages just like JSP and does not require a web container to run. FreeMarker is using the FreeMarker Template Language (FTL), which is similar to the JSP Expression language. [34] FreeMarker is a free software. [35]

Below is provided representation of a HTML table with FreeMarker

validation_html_template.ftl

```
<!-- rest of the code omitted -->
<table>
    <tr>
        <th>#</th>
        <th>Status</th>
        <th>Clause</th>
        <th>Validating regex</th>
    </tr>
    <#list validRows as validRow>
        <tr>
            <td>${count}</td>
            <#assign count = count + 1>
            <td class="success">${validRow.status}</td>
            <td>${validRow.clause}</td>
            <td>${validRow.validatingRegExp}</td>
        </tr>
    </#list>
</table>
<!-- rest of the code omitted -->
```

HtmlFileGenerator.java

```java
/* rest of the code omitted */
Configuration cfg = getConfig();
Map<String, Object> input = new HashMap<>();
input.put("validRows", validStatements);
Template template = cfg.getTemplate(VALIDATION_HTML_TEMPLATE);
Writer fileWriter = new FileWriter(new File(VALIDATION_OUTPUT_FILE_PATH));
template.process(input, fileWriter);
/* rest of the code omitted */
```

First of all FreeMarker gets configuration from *getConfig()* method. Secondly *List* with data is put into *input HashMap*. After that template is assigned. Next step initializes FileWriter opening file stream. Finally *process()* method compiles and writes processed HTML file based on *FileWriter*, *input HashMap* and template.

## 3.8 Database access object

After parsing and analyzing a script file and building objects based on a template, access to the database is required to compare expected objects to actual. The database access tool must be simple and lightweight.

Today there exist advanced database connectivity frameworks like JPA [16] and Hibernate [36] working in Java. These frameworks manage a lot of information in the background to simplify programmers work. Object-relational mapping frees programmer from writing a native SQL code. Hibernate Query Language (HQL) [37] and Java Persistence query language (JPQL) [38] allow programmer to write abstract queries. Queries are later translated into the native SQL depending on the chosen dialect and the database. Furthermore, a transaction management is done by framework as well as database exception handling and many more. Most of the frameworks have JDBC [9] running under the hood.

Despite the benefit frameworks bring, there is the other side of the coin. All extra processes running in the background result in overhead and performance penalties. Also, time is required to learn new framework.

Another solution is to use JDBC API. JDBC requires less time to initialize. Also, query execution time is smaller compared to Hibernate. [39] In addition, there is a win in the time required to learn JDBC. For reasons mentioned above JDBC is used to query database in this thesis.

## 3.9 Strategies for validation

Validation is performed by using the Strategy pattern. [40] A strategy type for validation is defined in the configuration file for each column returned from the database. (see 3.4.3 Database object mappings) Implementation for strategies is provided in the Strategy subclasses.

Strategy pattern allows with the use of inheritance to move implementation from Strategy class into subclasses. Behavior for each strategy is defined in separate class. When a new Strategy should be introduced, code modification of existing classes is not required. The new functionality is brought by extending the Strategy class.

Strategies are generated using Factory pattern. [41]

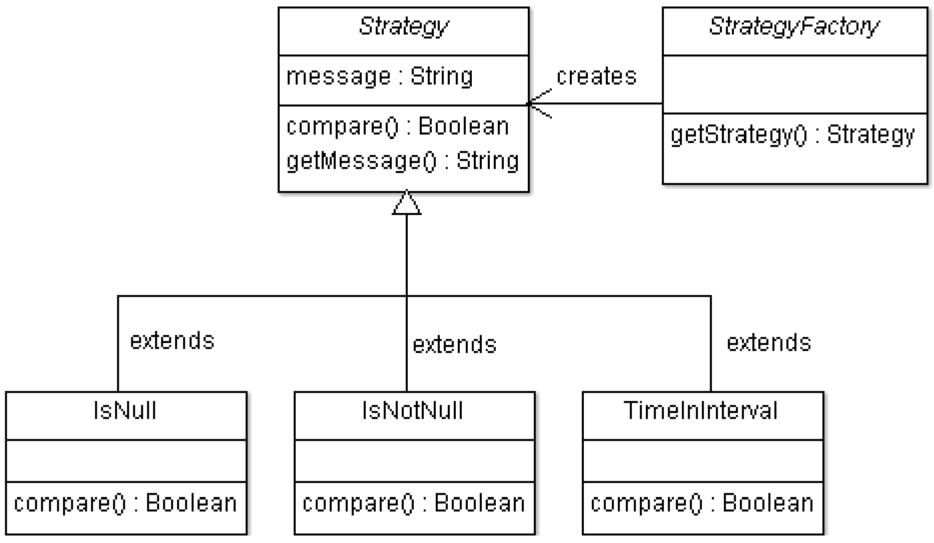
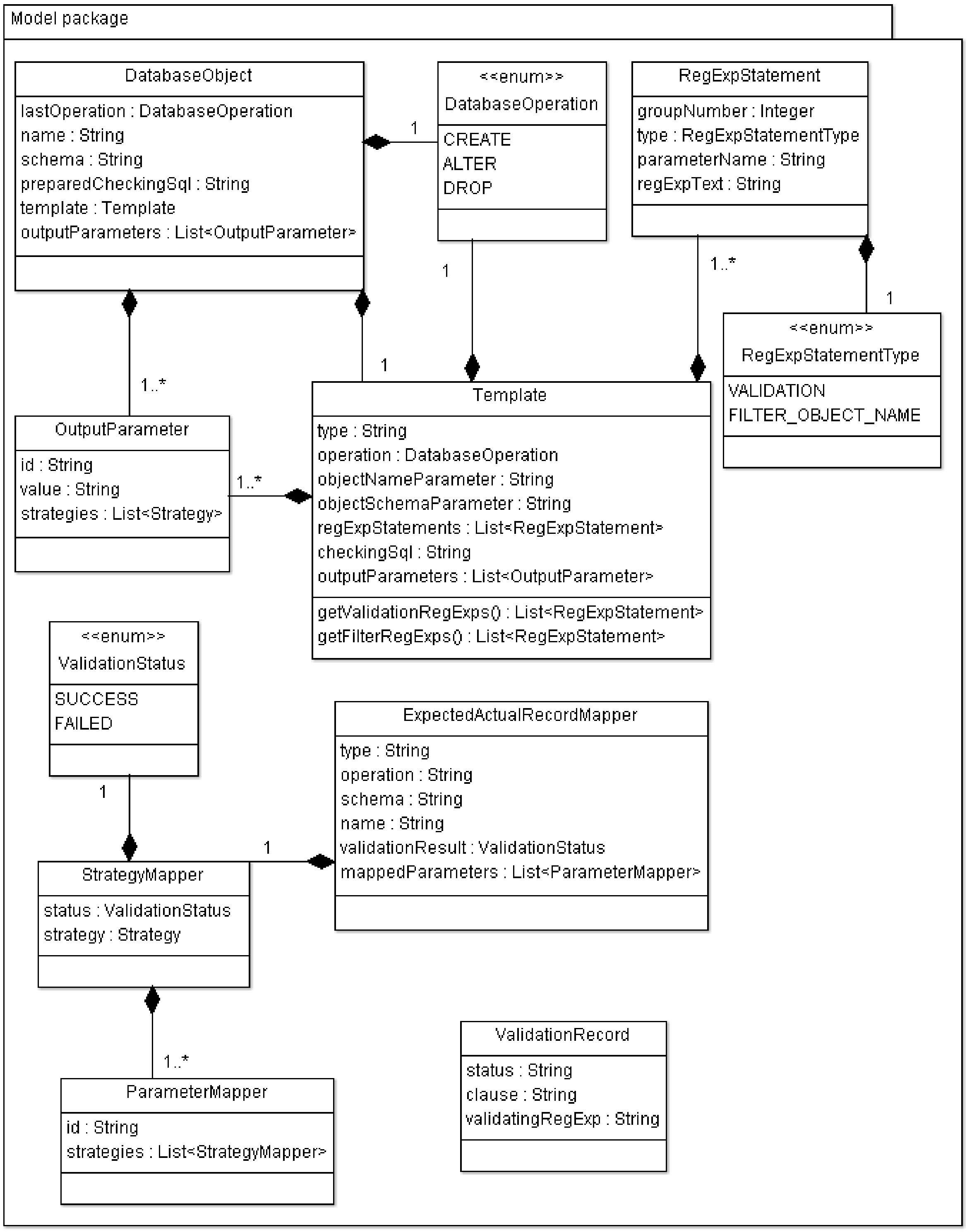


**Figure 5. Strategy class diagram**

## 3.10 Model package

The class diagram below describes fields and main functions of model package classes. Model classes are mainly POJO. [17] Getter and setter functions are omitted. Model classes are shared between all other classes of an application.



Model package
DatabaseObject
lastOperation : DatabaseOperation
name : String
schema : String
preparedCheckingSql : String
template : Template
outputParameters : List<OutputParameter>
<<enum>>
DatabaseOperation
CREATE
ALTER
DROP
RegExpStatement
groupNumber : Integer
type : RegExpStatementType
parameterName : String
regExpText : String
<<enum>>
RegExpStatementType
VALIDATION
FILTER_OBJECT_NAME
OutputParameter
id : String
value : String
strategies : List<Strategy>
Template
type : String
operation : DatabaseOperation
objectNameParameter : String
objectSchemaParameter : String
regExpStatements : List<RegExpStatement>
checkingSql : String
outputParameters : List<OutputParameter>
getValidationRegExps() : List<RegExpStatement>
getFilterRegExps() : List<RegExpStatement>
<<enum>>
ValidationStatus
SUCCESS
FAILED
ExpectedActualRecordMapper
type : String
operation : String
schema : String
name : String
validationResult : ValidationStatus
mappedParameters : List<ParameterMapper>
StrategyMapper
status : ValidationStatus
strategy : Strategy
ParameterMapper
id : String
strategies : List<StrategyMapper>
ValidationRecord
status : String
clause : String
validatingRegExp : String
1
1
1..*
1
1..*
1
1..*
1
1
1..*

**Figure 6. Model package diagram**

## 3.11 Making software runnable under Linux and Windows

Java can be run on any system, where system supports JVM. [18] However, small adjustments have to be made.

First of all, libraries being used in a project have to be provided within the jar file. Maven [42] initially keeps all the fetched libraries in a local repository separately from the project, so the Maven build has to be reconfigured accordingly.

Instead of calling JRE [18] from a command line and providing JAR [19] file as an argument, shell scripts for running in Linux and Windows environments were written.

runMe.cmd – for Windows:

```
java -jar "DbConsistencyChecker-1.0-SNAPSHOT-jar-with-dependencies.jar"

pause
```

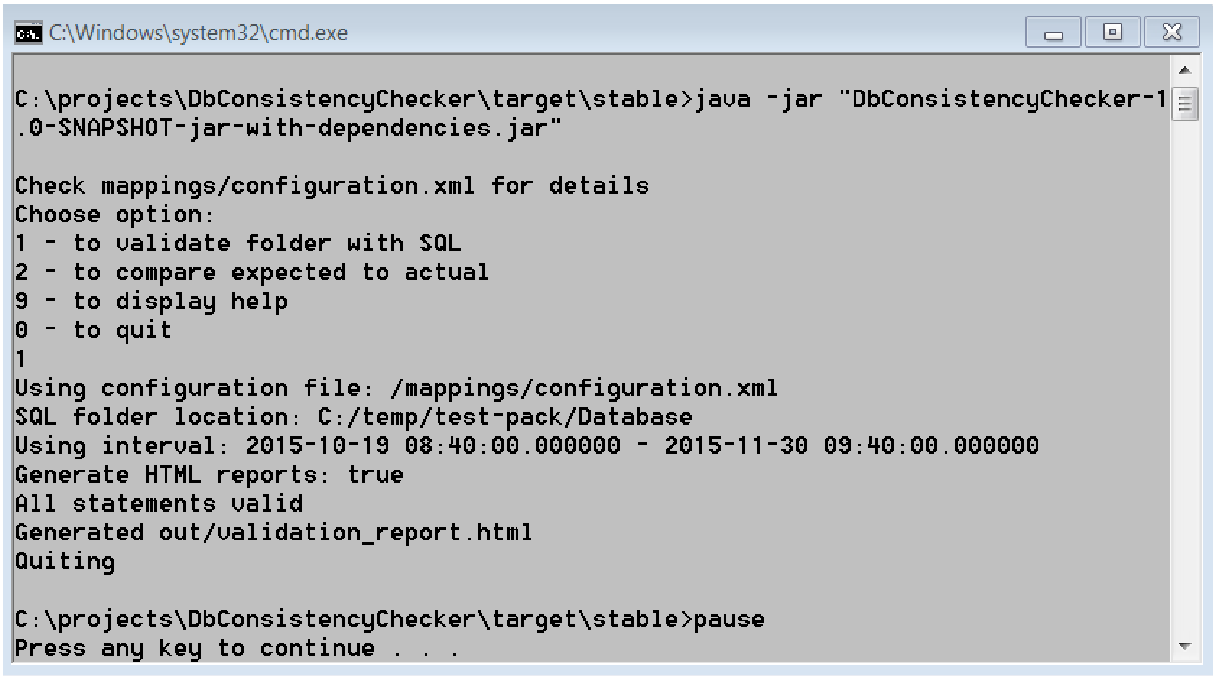
runMe.sh – for Linux:

```
#!/bin/bash

java -jar "DbConsistencyChecker-1.0-SNAPSHOT-jar-with-dependencies.jar"
```

In the script for Windows “pause” command is provided to prevent command line window from closing, in case if a user has run the script directly by clicking on it.

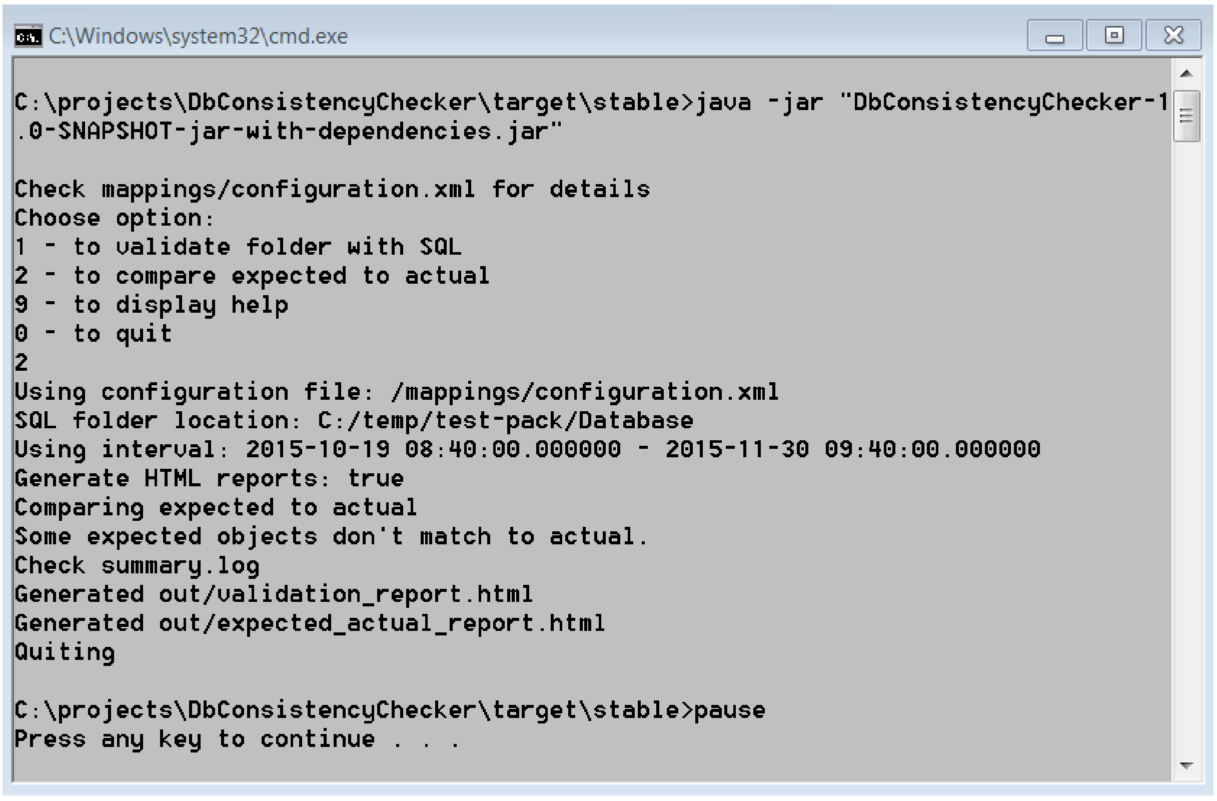
By calling *java –jar* assumption is made that $JAVA_HOME environment variable is pointing to the JRE.

## 3.12 User interface



```
C:\projects\DbConsistencyChecker\target\stable>java -jar "DbConsistencyChecker-1
.0-SNAPSHOT-jar-with-dependencies.jar"

Check mappings/configuration.xml for details
Choose option:
1 - to validate folder with SQL
2 - to compare expected to actual
9 - to display help
0 - to quit
1
Using configuration file: /mappings/configuration.xml
SQL folder location: C:/temp/test-pack/Database
Using interval: 2015-10-19 08:40:00.000000 - 2015-11-30 09:40:00.000000
Generate HTML reports: true
All statements valid
Generated out/validation_report.html
Quiting

C:\projects\DbConsistencyChecker\target\stable>pause
Press any key to continue . . .
```

**Figure 7. Executing SQL folder validation**



```
C:\projects\DbConsistencyChecker\target\stable>java -jar "DbConsistencyChecker-1
.0-SNAPSHOT-jar-with-dependencies.jar"

Check mappings/configuration.xml for details
Choose option:
1 - to validate folder with SQL
2 - to compare expected to actual
9 - to display help
0 - to quit
2
Using configuration file: /mappings/configuration.xml
SQL folder location: C:/temp/test-pack/Database
Using interval: 2015-10-19 08:40:00.000000 - 2015-11-30 09:40:00.000000
Generate HTML reports: true
Comparing expected to actual
Some expected objects don't match to actual.
Check summary.log
Generated out/validation_report.html
Generated out/expected_actual_report.html
Quiting

C:\projects\DbConsistencyChecker\target\stable>pause
Press any key to continue . . .
```

**Figure 8. Executing comparison of expected to actual**

## 3.13 Development tools used

IntelliJ Idea [43] is used as a Java integrated development environment (IDE). IntelliJ Idea IDE makes writing code faster, minimizes typographical errors by providing smart code completion and has other useful tools integrated to speed up developing process, for example, debugging tools, build automation tools and version control tools.

Apache Maven is used as a build automation tool. Maven gives control over the project's build lifecycle introducing phases. Phases can be executed separately or combined together. Also, Maven allows to fetch libraries from a repository located remotely on the server. There is no need to download all libraries manually into the local system and place them into a specific folder. A library version can be changed by modifying only one parameter in a pom.xml configuration file. Another advantage of Maven is standardized project layout. For example, in Ant [44] source folder, classes folder, target folder, folder with libraries have to be defined in a configuration file, while Maven already provides default layout, reducing the amount of parameters programmer has to be concerned about.

Git [45] was used as a version control system. Even for a small project like this using Git was beneficial. Version control saved time when it was required to revert to a particular version. Also, version control allowed to visualize development progress.

## 3.14 Testing

Testing is performed to ensure that functionality of a software meets the requirements. Also, by running tests a developer can ensure that his introduced code did not damage existing working components. Unit tests are written to test individual modules of a software. In this thesis the JUnit [46] framework is used.

JUnit framework is a testing framework written in Java. In addition, it is supported by many development environments, for example Eclipse and IntelliJ IDEA. Also, tests can be integrated into project build tools. For example, Ant or Maven.

Below is provided testing of HelperUtil class:

```java
public class HelperUtilTest {

    @Test
    public void testValidateText() throws Exception {
        String validText = "abc123";
        String pattern = "abc123";
        assertTrue(HelperUtil.validateText(validText, pattern));
        String invalidText = "bca321";
        assertFalse(HelperUtil.validateText(invalidText, pattern));
    }

    @Test
    public void testCutText() throws Exception {
        String testText = "I want to get THIS value";
        String pattern = "get (.+?) value";

        String result = HelperUtil.cutText(testText, pattern, 1);
        assertEquals("THIS" ,result);
    }

    @Test
    public void testPrepareSqlStatement() throws Exception {
        String checkingSqlClause = "Text with ${PARAM1} and ${PARAM2} replaced.";

        Map<String, String> inputParameters = new HashMap<>();
        inputParameters.put("PARAM1", "VALUE1");
        inputParameters.put("PARAM2", "VALUE2");

        String preparedSql = HelperUtil.prepareSqlStatement(checkingSqlClause,
inputParameters);
        String resultSql = "Text with VALUE1 and VALUE2 replaced.";
        assertEquals(resultSql, preparedSql);
    }

    @Test
    public void testGetParameterValueByKey() throws Exception {
        Map<String, String> inputParameters = new HashMap<>();
        inputParameters.put("PARAM1", "VALUE1");
        inputParameters.put("PARAM2", "VALUE2");
        String value = HelperUtil.getParameterValueByKey(inputParameters,
"PARAM2");
        assertEquals("VALUE2", value);
    }

    @Test
    public void testConvertStringToDateTime() throws Exception {
        DateTime dateTime = HelperUtil.convertStringToDateTime("11:22 27.11.2015",
"HH:mm dd.MM.yyyy");
        assertNotNull(dateTime);
        long actualMillis = dateTime.getMillis();
        long expectedMillis = 1448616120000L;
        assertEquals(expectedMillis, actualMillis);
    }

    @Test
    public void testConvertDateTimeToString() throws Exception {
        String pattern = "HH:mm dd.MM.yyyy";
        String dateTime = "11:22 27.11.2015";
        DateTimeFormatter formatter = DateTimeFormat.forPattern(pattern);
        DateTime dt = formatter.parseDateTime(dateTime);
        String dateTimeActual = HelperUtil.convertDateTimeToString(dt, pattern);
        assertNotNull(dateTimeActual);
        assertEquals(dateTimeActual, dateTime);
    }
}
```

# 4. Future development possibilities

The development and testing of an initial prototype has finished. Now the prototype can be used as a foundation for other similar projects.

Validation for nested parameters inside a statement will be added. For example, if *ALTER TABLE* is executed and a new column is added – column name, column type and column size should be validated.

Another example is altering a table and adding a constraint. In this case it is required to verify a constraint name, a constraint type and a referencing table name and a table column.

Finally, an important step is to add the support for other databases. IBM DB2 is not the only database on the market.

# Kokkuvõte

Lõputöö eesmärk oli luua töötav lahendus, mis kontrollib andmebaasi skeemi muudatusi. Esimeses etapis tuli projekteerida rakenduse arhitektuur. Seejärel tuli valida õiged vahendid ja sobivad teegid. Järgmisena tuli rakendada prototüüp. Viimases etapis toimus tarkvara testimine. Kõik kirjeldatud etapid on edukalt läbi viidud.

Selle töö tulemuseks on töötav prototüüp, mis on võimeline leidma andmebaasi skeemi tehtud muudatusi. Selles lõputöös kirjeldatud meetodid võivad olla kasulikud ka teistele arendajatele, kes puutuvad kokku andmebaasi skeemi muudatuste probleemidega. Töötav prototüüp on edasi arendatav. Lisaks võimaldas lõputöö autoril testida oma oskusi ja teadmisi, mis on omandatud ülikoolis õppimise ajal ja töötades erinevates IT firmades.  Samuti sai autor uusi teadmisi andmebaasi ja Java rakenduste arendamise valdkonnas.

# Summary

In this thesis it was required to find a solution for a database schema updates verification control. The next step was to design application architecture. After that suitable tools and libraries were chosen. Following that, the prototype was implemented. Lastly, the software was tested. All mentioned goals were successfully achieved during this work.

The result of this thesis is a working prototype capable to distinguish database schema changes. Methods described in this thesis can be beneficial for other developers encountering problems with database schema tracking. The working prototype can be further modified. Also this thesis allowed the author to test his skills and knowledge gained during studying at university and working in software developing companies. In addition, author gained new knowledge in databases and Java application development.

# Table of literature

[1] "Extensible Markup Language (XML)," W3C, [Online]. Available: http://www.w3.org/XML/. [Accessed 17 12 2015].

[2] "SQL - Wikipedia, the free encyclopedia," Wikimedia Foundation, Inc, [Online]. Available: https://en.wikipedia.org/wiki/SQL. [Accessed 17 12 2015].

[3] "Java SE | Oracle Technology Network | Oracle," Oracle, [Online]. Available: http://www.oracle.com/technetwork/java/javase/overview/index.html. [Accessed 17 12 2015].

[4] "W3C HTML," W3C, [Online]. Available: http://www.w3.org/html/. [Accessed 17 12 2015].

[5] "What is use case? - Definition from WhatIs.com," TechTarget, [Online]. Available: http://searchsoftwarequality.techtarget.com/definition/use-case. [Accessed 2015 12 18].

[6] "Database Entities," eWebArchitecture, [Online]. Available: http://ewebarchitecture.com/web-databases/database-entities. [Accessed 18 12 2015].

[7] "DDL Trigger," Microsoft, [Online]. Available: https://technet.microsoft.com/en-us/library/ms190989(v=sql.105).aspx. [Accessed 18 12 2015].

[8] "DML Trigger," Microsoft, [Online]. Available: https://technet.microsoft.com/en-us/library/ms191524(v=sql.105).aspx. [Accessed 18 12 2015].

[9] "Java SE Technologies - Database," Oracle, [Online]. Available: http://www.oracle.com/technetwork/java/javase/jdbc/index.html. [Accessed 21 12 2015].

[10] "Core J2EE Patterns - Data Access Object," Oracle, [Online]. Available: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html. [Accessed 19 12 2015].

[11] "Regular Expression Language - Quick Reference," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/az24scfc(v=vs.110).aspx. [Accessed 21 12 2015].

[12] "Quantifiers in Regular Expressions," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/3206d374(v=vs.110).aspx. [Accessed 21 12 2015].

[13] "StringBuilder (Java Platform SE 7 )," Oracle, [Online]. Available: http://docs.oracle.com/javase/7/docs/api/java/lang/StringBuilder.html. [Accessed 21 12 2015].

[14] "JavaServer Pages Technology," Oracle, [Online]. Available: http://www.oracle.com/technetwork/java/javaee/jsp/index.html. [Accessed 21 12 2015].

[15] "HashMap (Java Platform SE 7 )," Oracle, [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html. [Accessed 21 12 2015].

[16] "Java Persistence API," Oracle, [Online]. Available: http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html. [Accessed 21 12 2015].

[17] "Plain Old Java Object - Wikipedia, the free encyclopedia," Wikimedia Foundation, Inc., [Online]. Available: https://en.wikipedia.org/wiki/Plain_Old_Java_Object. [Accessed 21 12 2015].

[18] "Java virtual machine - Wikipedia, the free encyclopedia," Wikimedia Foundation, Inc., [Online]. Available: https://en.wikipedia.org/wiki/Java_virtual_machine. [Accessed 21 12 2015].

[19] "Lesson: Packaging Programs in JAR Files," Oracle, [Online]. Available: https://docs.oracle.com/javase/tutorial/deployment/jar/index.html. [Accessed 21 12 2015].

[20] "Profit Software," Profit Software OY, [Online]. Available: http://www.profitsoftware.com/. [Accessed 19 10 2015].

[21] "IBM DB2 database software," IBM, [Online]. Available: http://www-01.ibm.com/software/data/db2/. [Accessed 19 10 2015].

[22] "Linux.com | The source for Linux information," The Linux Foundation, [Online]. Available: https://www.linux.com/. [Accessed 21 12 2015].

[23] "Windows - Microsoft," Microsoft, [Online]. Available: https://www.microsoft.com/en-us/windows. [Accessed 21 12 2015].

[24] "Relational database - Wikipedia, the free encyclopedia," Wikimedia Foundation, Inc, [Online]. Available: https://en.wikipedia.org/wiki/Relational_database. [Accessed 17 12 2015].

[25] "Understanding DDL Triggers vs. DML Triggers," Microsoft, [Online]. Available: https://technet.microsoft.com/en-us/library/ms189599(v=sql.105).aspx. [Accessed 18 12 2015].

[26] "IBM Knowledge Center - db2look," [Online]. Available: https://www-01.ibm.com/support/knowledgecenter/SSEPGG_10.5.0/com.ibm.db2.luw.admin.cmd.doc/doc/r0002051.html. [Accessed 11 30 2015].

[27] "Input/output - Wikipedia, the free encyclopedia," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Input/output. [Accessed 18 12 2015].

[28] "IBM Knowledge Center," IBM, [Online]. Available: http://www-01.ibm.com/support/knowledgecenter/SSEPGG_9.7.0/com.ibm.db2.luw.sql.ref.doc/doc/r0008443.html?cp=SSEPGG_9.7.0%2F2-10-7. [Accessed 19 12 2015].

[29] "What is absolute path?," Computer Hope, [Online]. Available: http://www.computerhope.com/jargon/a/absopath.htm. [Accessed 21 12 2015].

[30] "Why StAX?," Sun Microsystems, [Online]. Available: https://docs.oracle.com/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/SJSXP2.html. [Accessed 16 12 2015].

[31] "Simple API for XML - Wikipedia, the free encyclopedia," Wikimedia Foundation, Inc, [Online]. Available: https://en.wikipedia.org/wiki/Simple_API_for_XML. [Accessed 21 12 2015].

[32] "Apache log4j 1.2," Apache Software Foundation, [Online]. Available: https://logging.apache.org/log4j/1.2/. [Accessed 21 12 2015].

[33] "SLF4J," QOS.ch, [Online]. Available: http://www.slf4j.org/. [Accessed 21 12 2015].

[34] "Expression Language," Oracle, [Online]. Available: http://docs.oracle.com/javaee/1.4/tutorial/doc/JSPIntro7.html. [Accessed 21 12 2015].

[35] "FreeMarker Java Template Engine," The FreeMarker Project, [Online]. Available: http://freemarker.incubator.apache.org/. [Accessed 7 12 2015].

[36] "Hibernate. Everything data. - Hibernate," Red Hat, [Online]. Available: http://hibernate.org/. [Accessed 21 12 2015].

[37] "Chapter 14. HQL: The Hibernate Query Language," RedHat, [Online]. Available: https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/queryhql.html. [Accessed 21 12 2015].

[38] "The Java Persistence Query Language - The Java EE 6 Tutorial," Oracle, [Online]. Available: http://docs.oracle.com/javaee/6/tutorial/doc/bnbtg.html. [Accessed 21 12 2015].

[39] "Hibernate vs JDBC performance," [Online]. Available: http://phpdao.com/hibernate_vs_jdbc/. [Accessed 21 12 2015].

[40] "Design Patterns Strategy Pattern," Tutorialspoint, [Online]. Available: http://www.tutorialspoint.com/design_pattern/strategy_pattern.htm. [Accessed 21 12 2015].

[41] "Design Pattern Factory Pattern," Tutorialspoint, [Online]. Available: http://www.tutorialspoint.com/design_pattern/factory_pattern.htm. [Accessed 21 12 2015].

[42] "Maven - Welcome to Apache Maven," The Apache Software Foundation, [Online]. Available: https://maven.apache.org/. [Accessed 21 12 2015].

[43] "IntelliJ IDEA the Java IDE," JetBrains, [Online]. Available: https://www.jetbrains.com/idea/. [Accessed 22 12 2015].

[44] "Apache Ant - Welcome," The Apache Software Foundation, [Online]. Available: http://ant.apache.org/. [Accessed 22 12 2015].

[45] "Git," Software Freedom Conservancy, [Online]. Available: https://git-scm.com/. [Accessed 22 12 2015].

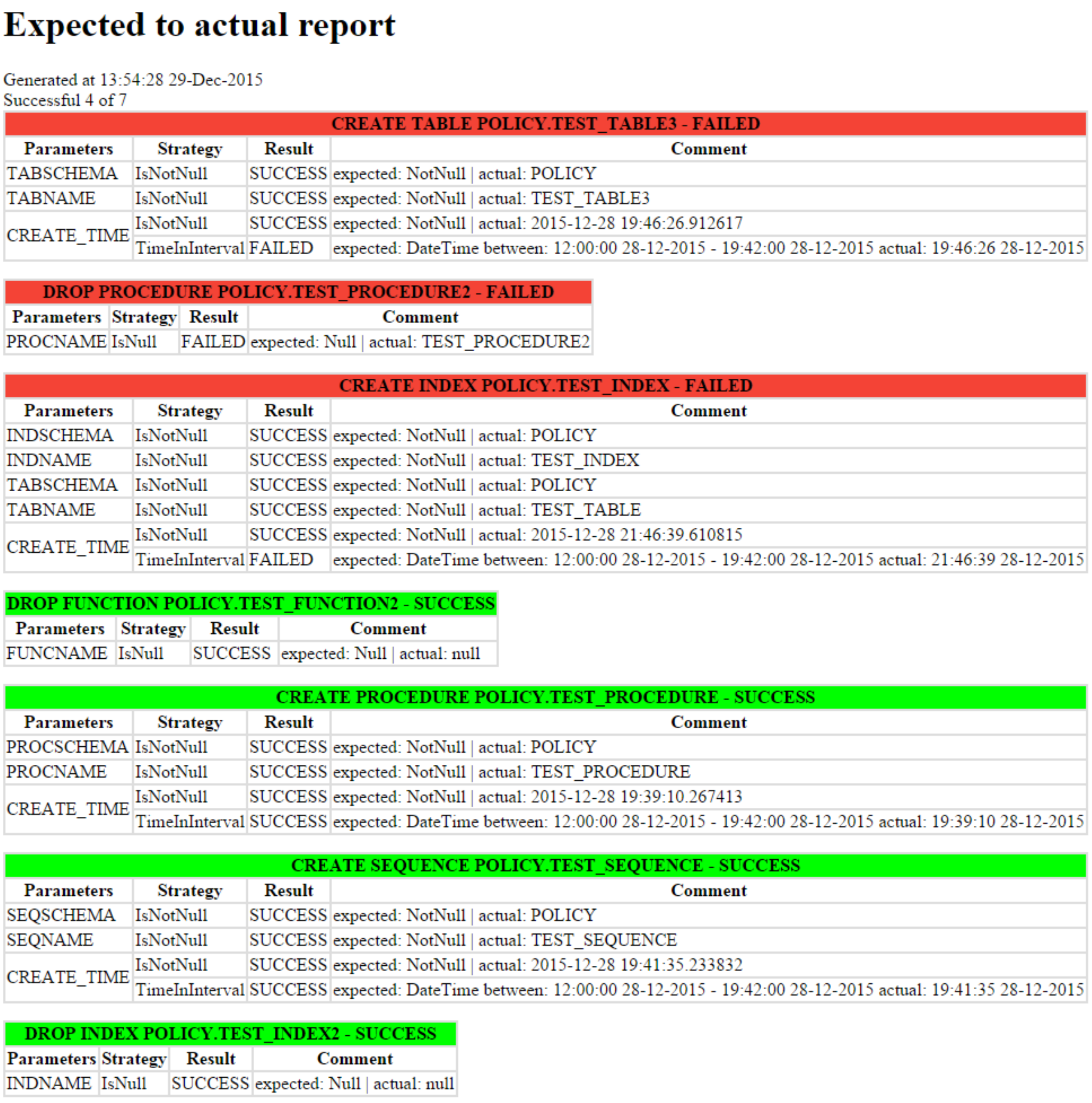
[46] "JUnit - About," JUnit, [Online]. Available: http://junit.org/. [Accessed 22 12 2015].

# Appendix 1

*CREATE FUNCTION* template from configuration.xml file

```
<object type="FUNCTION" operation="CREATE"
objectNameParameter="FUNCTION_NAME_IN"
objectSchemaParameter="FUNCTION_SCHEMA_IN">
   <regExpStatement regExpType="VALIDATION">
           "^CREATE\s+FUNCTION\s+\w+\.\w+\s*\(\s*.+"
   </regExpStatement>
   <regExpStatement regExpType="FILTER_OBJECT_NAME"
parameterName="FUNCTION_SCHEMA_IN" groupNumber="1">
           "FUNCTION\s+(.+?)\.\w+\s*\(.+"
   </regExpStatement>
   <regExpStatement regExpType="FILTER_OBJECT_NAME"
parameterName="FUNCTION_NAME_IN" groupNumber="1">
           "FUNCTION\s+\w+\.(.+?)\s*\(.+"
   </regExpStatement>
   <paramMapping>

       <outputParam id="FUNCSCHEMA">
           <strategy>IS_NOT_NULL</strategy>
       </outputParam>

       <outputParam id="FUNCNAME">
           <strategy>IS_NOT_NULL</strategy>
       </outputParam>

       <outputParam id="CREATE_TIME">
           <strategy>IS_NOT_NULL</strategy>
           <strategy>TIME_IN_INTERVAL</strategy>
       </outputParam>

   </paramMapping>
   <sqlString>
      SELECT FUNCSCHEMA, FUNCNAME, CREATE_TIME FROM "SYSCAT"."FUNCTIONS"
      WHERE FUNCSCHEMA = '${FUNCTION_SCHEMA_IN}'
      AND FUNCNAME = '${FUNCTION_NAME_IN}'
   </sqlString>
</object>
```

# Appendix 2



## Expected to actual report

Generated at 13:54:28 29-Dec-2015
Successful 4 of 7

**CREATE TABLE POLICY.TEST_TABLE3 - FAILED**

| Parameters | Strategy | Result | Comment |
|---|---|---|---|
| TABSCHEMA | IsNotNull | SUCCESS | expected: NotNull \| actual: POLICY |
| TABNAME | IsNotNull | SUCCESS | expected: NotNull \| actual: TEST_TABLE3 |
| CREATE_TIME | IsNotNull | SUCCESS | expected: NotNull \| actual: 2015-12-28 19:46:26.912617 |
| | TimeInInterval | FAILED | expected: DateTime between: 12:00:00 28-12-2015 - 19:42:00 28-12-2015 actual: 19:46:26 28-12-2015 |

**DROP PROCEDURE POLICY.TEST_PROCEDURE2 - FAILED**

| Parameters | Strategy | Result | Comment |
|---|---|---|---|
| PROCNAME | IsNull | FAILED | expected: Null \| actual: TEST_PROCEDURE2 |

**CREATE INDEX POLICY.TEST_INDEX - FAILED**

| Parameters | Strategy | Result | Comment |
|---|---|---|---|
| INDSCHEMA | IsNotNull | SUCCESS | expected: NotNull \| actual: POLICY |
| INDNAME | IsNotNull | SUCCESS | expected: NotNull \| actual: TEST_INDEX |
| TABSCHEMA | IsNotNull | SUCCESS | expected: NotNull \| actual: POLICY |
| TABNAME | IsNotNull | SUCCESS | expected: NotNull \| actual: TEST_TABLE |
| CREATE_TIME | IsNotNull | SUCCESS | expected: NotNull \| actual: 2015-12-28 21:46:39.610815 |
| | TimeInInterval | FAILED | expected: DateTime between: 12:00:00 28-12-2015 - 19:42:00 28-12-2015 actual: 21:46:39 28-12-2015 |

**DROP FUNCTION POLICY.TEST_FUNCTION2 - SUCCESS**

| Parameters | Strategy | Result | Comment |
|---|---|---|---|
| FUNCNAME | IsNull | SUCCESS | expected: Null \| actual: null |

**CREATE PROCEDURE POLICY.TEST_PROCEDURE - SUCCESS**

| Parameters | Strategy | Result | Comment |
|---|---|---|---|
| PROCSCHEMA | IsNotNull | SUCCESS | expected: NotNull \| actual: POLICY |
| PROCNAME | IsNotNull | SUCCESS | expected: NotNull \| actual: TEST_PROCEDURE |
| CREATE_TIME | IsNotNull | SUCCESS | expected: NotNull \| actual: 2015-12-28 19:39:10.267413 |
| | TimeInInterval | SUCCESS | expected: DateTime between: 12:00:00 28-12-2015 - 19:42:00 28-12-2015 actual: 19:39:10 28-12-2015 |

**CREATE SEQUENCE POLICY.TEST_SEQUENCE - SUCCESS**

| Parameters | Strategy | Result | Comment |
|---|---|---|---|
| SEQSCHEMA | IsNotNull | SUCCESS | expected: NotNull \| actual: POLICY |
| SEQNAME | IsNotNull | SUCCESS | expected: NotNull \| actual: TEST_SEQUENCE |
| CREATE_TIME | IsNotNull | SUCCESS | expected: NotNull \| actual: 2015-12-28 19:41:35.233832 |
| | TimeInInterval | SUCCESS | expected: DateTime between: 12:00:00 28-12-2015 - 19:42:00 28-12-2015 actual: 19:41:35 28-12-2015 |

**DROP INDEX POLICY.TEST_INDEX2 - SUCCESS**

| Parameters | Strategy | Result | Comment |
|---|---|---|---|
| INDNAME | IsNull | SUCCESS | expected: Null \| actual: null |

**Figure 9. Expected to actual HTML report generated after execution**