jamk.fi

Liquibase

Version Control for Database Schema

Radosław Dziadosz

Bachelor’s thesis
May 2017
School of Technology, Communication and Transport
Bachelor's Degree Program in Information and Communications Technology

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Description

Author(s)	Type of publication	Date
Dziadosz, Radosław	Bachelor's thesis	29.05.2017
		Language of publication: English
	Number of pages 46	Permission for web publication: x

Title of publication
Liquibase
Version Control for Database Schema

Degree programme
Information and Communications Technology

Supervisor(s)
Salmikangas, Esa

Assigned by
Landis+Gyr Oy

Abstract

The main task was to add a version control system to an existing Oracle database. The database was complex and migrations were very complicated and difficult to develop without control version system for database schema.

Creating patches or improvements to the database schema was taking a great amount of time, because it was needed to take care of error handling manually. It was not sure if the script would be executed a few times or once, therefore, any exception that could occur had to be handled in SQL script. All scripts had to be executed in the correct order, because existing data could be corrupted.

Another goal was also to make it easy for developers to create a database instance. They were sharing resources, which sometimes caused conflicts while working.

In order to solve the above problems, it was decided to add Liquibase to the existing database. Liquibase was added to the project in an automated method only in some places due to the complexity of the database.

As a result, the management of database migration has become much easier. The migration process was accelerated by not executing scripts again in the upgrade process and by adding Continuous Integration.

Liquibase should have been added much earlier. It would be good to think about how to manage the changes at the very beginning of creating the database structure, because adding a version control system to a schema in a complex database is much more difficult.

Keywords/tags
versioning software, version control, VCS, databases, Liquibase

Miscellaneous

Contents

Figures

List of acronyms

CI	Continuous Integration
CLI	Command Line Interface
CSV	Comma-Separated Values
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
MD5	Message Digest 5
SQL	Structured Query Language
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

1 Introduction

1.1 Databases

Database is usually a large, organized collection of data in form of tables, schemas and other objects. Today, no application can run or even exist without data. Database is even more important than code because an algorithm may change but data is forever. (Database - Definition of database)

Databases are the central elements of most current software systems. They contain business information, however, they are often not treated with due care. The databases are as complex and changing as the systems themselves.

The "real" databases are more than just a schema with few tables and a small number of records. Real databases contain hundreds of tables, with hundreds of thousands of records each. In addition, databases contain triggers, procedures, functions and other objects needed to streamline the operations and keep the system functional. (Oracle Database Concepts: Schema Objects)

During the software development process, the database that is created at the beginning is always different from the one that will be released. Also, after the software release the data structure may change. New features and bug fixes can be added, which affects not only the source code but also the database, thus, the data structure or stored procedures need to be modified.

1.2 Managing changes manually

Managing changes in a database is not a task that can be done manually or by a single individual, because it can easily cause errors or inconsistent data. The process is shown in Figure 1. Changing data directly is ineffective, and reversing the changes is difficult. It may cause data mismatches. (SQL Server Database Change Management with Liquibase, 2016)

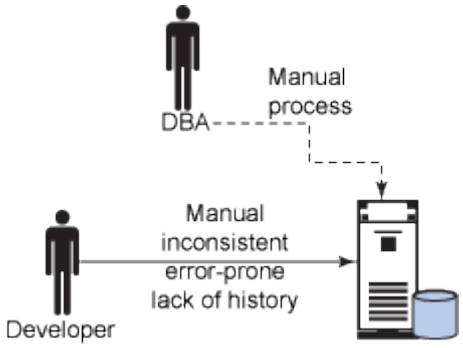


Figure 1 Applying changes directly to the database
(adapted from LiquiBase를 사용하여 데이터베이스 변경 관리하기)

The problem with the version control of databases is very extensive, especially when there are many people working on the software. In this case, there are many development environments, many test and production databases, each of which may have a different database schema version on it.

Moreover, if the schema changes, then it is likely that an existing data transform will be required. It is simply an operation involving changing existing data to another using a specific algorithm. The more complex the database is, the more difficult it is to manage transformations. (A simple introduction to database change management with SQLite and Liquibase, 2011)

1.3 Versioning tools

There are many open source libraries and frameworks that can help with the versioning of a database. One of the most popular ones is the Liquibase. The project started in 2006, and Liquibase library is constantly being improved by many contributors. (Continuous database migration with Liquibase and Flyway, 2013)

Figure 2 shows the process of database migration using Liquibase and a continuous integration (CI) server. The version control repository such as Git, is read by the CI server and if there are any changes to apply to database, Liquibase modifies the database.

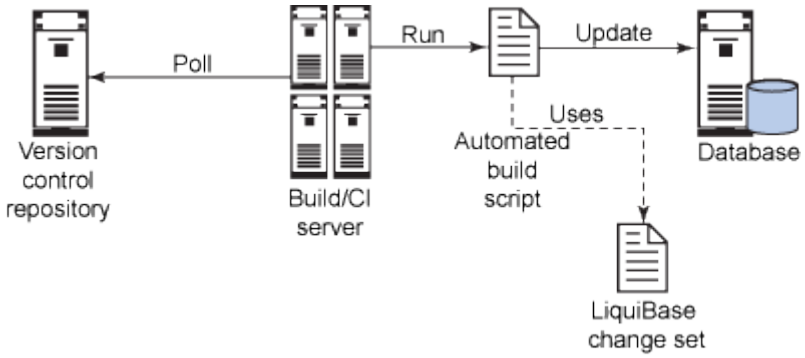


Figure 2 Automating database migration
(adapted from LiquiBase를 사용하여 데이터베이스 변경 관리하기)

Using the same process as shown in Figure 2, all team members can make the same process to the databases. It can be a local or shared database server. This process can also be automated, and changes can be easily applied to many databases using automation scripts.

Liquibase has following basic capabilities (Liquibase Official Website):

- Database schema versioning
- Code branching and merging
- Database change history tracking
- Support multiple database types
- Supports multiple developers
- Rollback changes until a particular tag, date or changelog number
- Performing precondition check

2 Liquibase - theoretical basis

2.1 How does Liquibase work?

The main operation of Liquibase is the changeLog file. This is a text file containing one or more changeSets. A changeLog can include other changeLogs, which creates a hierarchy of files. (Liquibase documentation: Database Change Log File, 2014)

A changeSet is the basic unit of change for a database. A changeSet include two types of information: the metadata about the change and the change instruction itself. Instruction can contain one or more atomic changes (adding table, modifying column, adding constraint, inserting rows, etc.), however, when it runs, Liquibase considers all those changes as part of a single changeSet. (Liquibase documentation: <changeSet> tag, 2015)

Below is an example of changeLog in XML format containing two changeSets:

```
<databaseChangeLog
        xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog

http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.5.xsd">

    <changeSet id="1" author="rdziadosz">
        <createTable tableName="news">
            <column name="id" type="number"/>
            <column name="title" type="varchar(255)"/>
        </createTable>
    </changeSet>

    <changeSet id="2" author="rdziadosz">
        <insert tableName="news">
            <column name="id" value="1"/>
            <column name="title" value="Hello world!"/>
        </insert>
    </changeSet>

</databaseChangeLog>
```

Metadata allows to uniquely identify the changeSet (information about identifier, author, and filename) and to define how the change is to be applied. (Liquibase documentation: <changeSet> tag, 2015)

On the first run, Liquibase creates two tables in the database as shown in Figure 3. The first one is DATABASECHANGELOG containing data about successful migrations. Each row in this table contains information about the executed changeSets such as identifier, author, filename, checksum and execution date. (Liquibase documentation: DATABASECHANGELOG table, 2015)

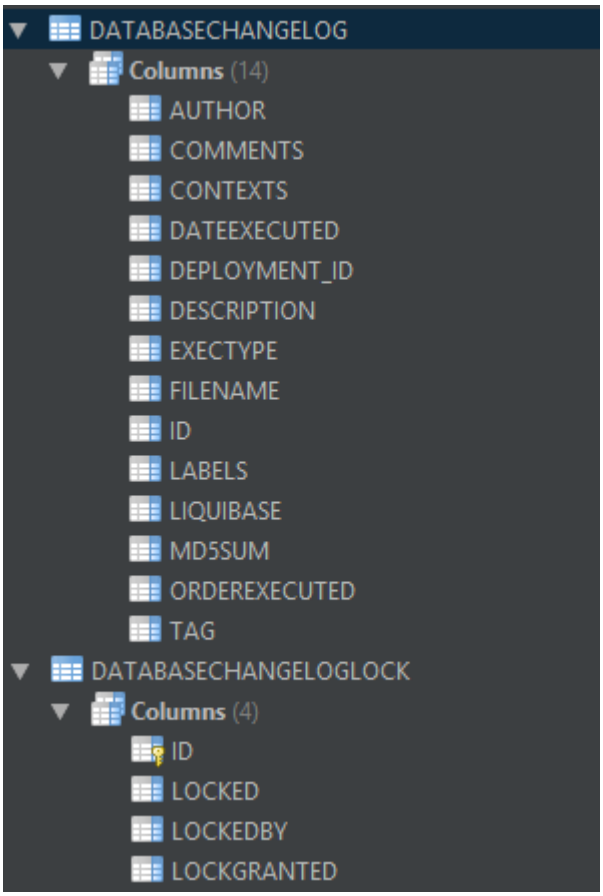


Figure 3 Liquibase tables

If Liquibase is run again, it will be able to recognize if the change has already been applied. By default, Liquibase will only apply new changes. Old changes will be omitted, however, their MD5 checksum is checked during this process. If the checksum is different, it means that changeSet has been modified from the previous run. This is usually an undesirable effect and can lead to unwanted changes to the database. In this case, Liquibase stops and returns the information about the wrong

checksum. That is why every new change should be a part of a new changeSet. (Liquibase documentation: <changeSet> tag, 2015; Liquibase documentation: Updating the Database, 2014)

Another table created by Liquibase is DATABASECHANGELOGLOCK, the purpose of which is to prevent modifying one database instance by multiple users at the same time. (Liquibase documentation: DATABASECHANGELOGLOCK table, 2015)

2.2 Liquibase is not difference-based system

The fact that Liquibase uses the DATABASECHANGELOG table rather than a difference-based system has many advantages. The most important is the semantics of changes in the database schema. Some of the changes to the database schema can be done in several ways. Some of them may cause modification or change of related data. Liquibase allows to specify the exact way in which changes are to be made. (The problem with database diffs, 2007)

Another reason why Liquibase is not based on differences is performance. In huge databases, it is not possible to compare data within a reasonable time to make changes. (The problem with database diffs, 2007)

2.3 Development environment

Liquibase is a cross platform tool; therefore, it works on Linux, Windows, or Mac OS. It can work on both the workstation and the server. (Liquibase documentation: Installation, 2016)

Liquibase is written in Java language. The latest version of Liquibase (3.5.3) requires Java 1.6 or higher. (Liquibase documentation: Installation, 2016)

Liquibase connects to the database using the JDBC driver. For example, for Oracle Database Express Edition 11g Release, the Oracle Database 12.1.0.1 JDBC Driver can be used. (Liquibase documentation: Supported Databases, 2014)

2.4 Database

Liquibase has many advantages and a very important one is its compatibility with multiple database types. The following list shows the databases with which it is compatible (Liquibase documentation: Supported Databases, 2014):

- MySQL
- PostgreSQL
- Oracle
- Sql Server
- Sybase_Enterprise
- Sybase_Anywhere
- DB2
- Apache_Derby
- HSQL
- H2
- Informix
- Firebird
- SQLite

Liquibase can support other database systems by using extensions. (Liquibase documentation: Supported Databases, 2014).

2.5 Deployment

Liquibase can be executed in several ways: it may be an Apache Ant task or a direct Java call but most often used is the command line, which offers all possibilities of Liquibase. (A simple introduction to database change management with SQLite and Liquibase, 2011)

Running using command line

Liquibase uses the following syntax:

```
liquibase [options] [command] [command parameters]
```

The command can be executed either at the terminal on Linux or Mac Os or at the Windows command line. (A simple introduction to database change management with SQLite and Liquibase, 2011)

All commands are checked by the command line processor. In case of a wrong or not allowed command, Liquibase will return an error message and terminate the operation. (Liquibase documentation: Liquibase Command Line, 2015)

Below is an example of a code that runs standard migration:

```
java -jar liquibase.jar \
      --driver=oracle.jdbc.OracleDriver \
      --classpath=ojdbc6.jar \
      --changeLogFile=changelog.xml \
      --url="jdbc:oracle:thin:@localhost:1521:xe" \
      --username=rdziadosz \
      --password=secret \
      update
```

Using Microsoft Windows to be able to use Liquibase in any directory using the CLI, Liquibase directory should be added to the environment variable. Using Liquibase will be much more convenient than typing the entire path to Liquibase in each run. (SQL Server Database Change Management with Liquibase, 2016)

To change the environment variables, the following settings need to be opened:

```
Control Panel -> System and Security-> System -> Advanced system
settings -> Advanced -> Environments variables...
```

In the window shown in Figure 4, the path to the Liquibase directory should be entered. (Microsoft Dev Center: Environment Variables, 2017)

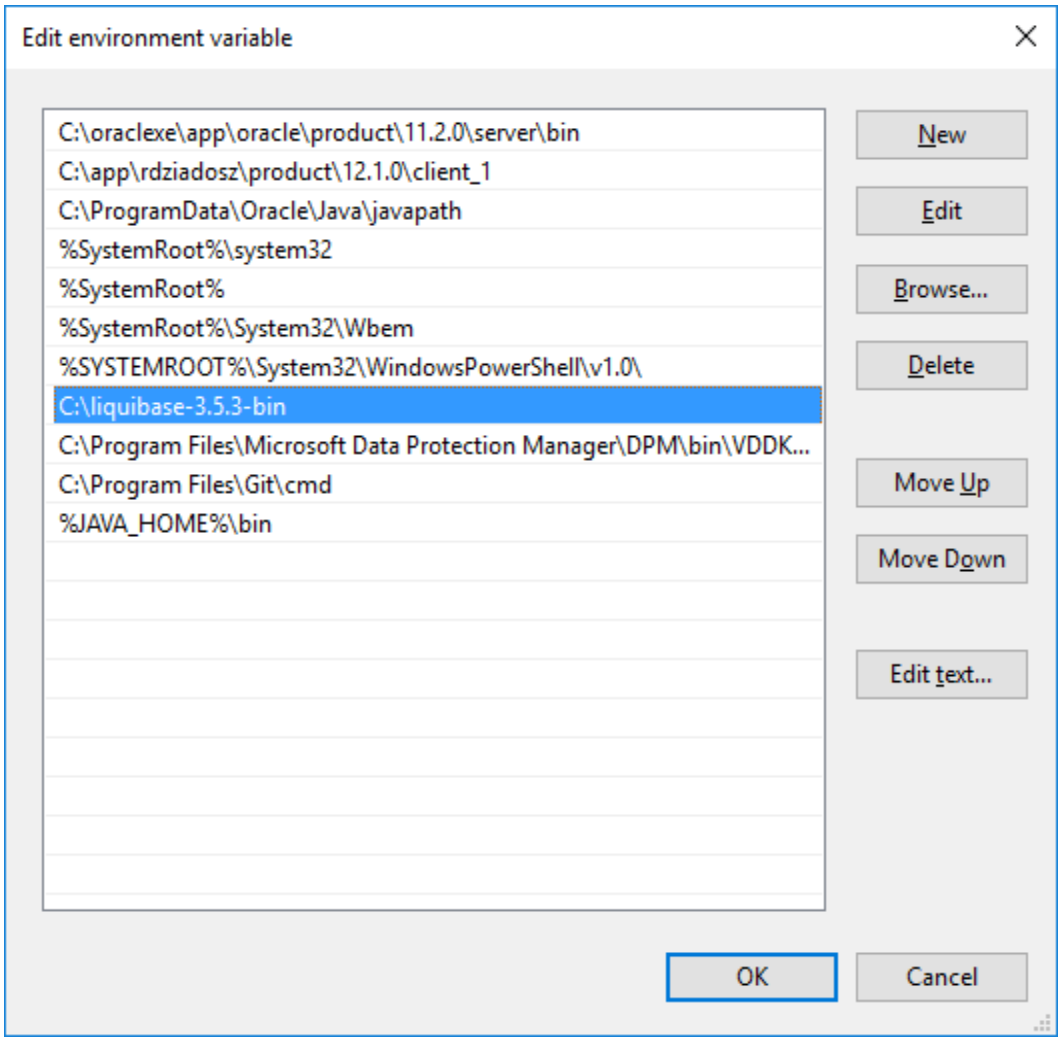


Figure 4 Editing environment variables in Windows

Running using command line and properties file

It is not necessary to enter all options every time, because that is error prone and tedious. Instead, it is better to create a file with default values. (SQL Server Database Change Management with Liquibase, 2016)

The file called “liquibase.properties” should be located in the directory where Liquibase is run, or the location of the file must be specified as a parameter. (Liquibase documentation: Liquibase Command Line, 2015)

The sample source code for the properties file is shown below:

```
#liquibase.properties
driver: oracle.jdbc.OracleDriver
classpath: ojdbc6.jar
url: jdbc:oracle:thin:@localhost:1521:xe
changeLogFile: changelog.xml
username: rdziadosz
password: secret
```

When the following command is executed in the directory that contains the shown liquibase.properties file, the same effect will be reached as for the command shown in the previous chapter:

```
liquibase update
```

In case that the properties file will be located in another directory the command will be as follows:

```
liquibase --defaultsFile=”C:\project\liquibase.properties” update
```

Parameter “defaultsFile” defines the path to the properties file. (Liquibase documentation: Liquibase Command Line, 2015)

2.6 ChangeLog file formats

ChangeLogs can be written in many different formats. Liquibase supports the following file formats (Liquibase Official Website):

- XML Format
- YAML Format
- JSON Format
- SQL Format
- Other Formats

The main advantage of the XML, YAML and JSON formats is that they support all Liquibase capabilities. Using the SQL format imposes some limitations, such as the lack of automatic rollback generation. (Liquibase documentation: XML Format, 2014; Liquibase documentation: YAML Format, 2014; Liquibase documentation: JSON Format, 2014; Liquibase documentation: Formatted SQL Changelogs, 2014)

It is possible to create one's own changeLog files formats using the extension system. The Liquibase community develops formats such as Groovy Liquibase and Clojure Liquibase Wrapper (Liquibase documentation: Other Changelog formats, 2013)

2.7 Rollback

One of the most usable key features of Liquibase is the ability to undo changes. This can be done automatically or using a defined SQL script. (Liquibase documentation: Rolling Back ChangeSets, 2014)

Using rollback generated automatically is possible for commands such as "CREATE TABLE", "CREATE VIEW", "ADD COLUMN", and so on. However, some changes cannot be undone, for example, "DROP TABLE". In that case, rollback must be defined manually. (Liquibase documentation: Rolling Back ChangeSets, 2014)

The following changeSet allows to remove the table news and rollback it:

```
<changeSet id="drop_news" author="rdziadosz">

    <dropTable tableName="news"/>

    <rollback>

        <createTable tableName="news">

            <column name="id" type="number"/>
            <column name="title" type="varchar(255)"/>

        </createTable>

    </rollback>

</changeSet>
```

In case the rollback tag is left empty this means that rollback is not possible or needed. (Liquibase documentation: Rolling Back ChangeSets, 2014)

It is also possible to specify changesets to be made during rollback:

```
<changeSet id="drop_news2" author="rdziadosz">

    <dropTable tableName="news"/>
    <rollback changeSetId="news_create"
changeSetAuthor="rdziadosz"/>

</changeSet>
```

3 Existing project and Liquibase

3.1 Approaches to adding Liquibase

Adding Liquibase to a new project is uncomplicated process. A new, empty changeLog matches the empty database. However, addition of Liquibase to an existing database is more demanding.

Unfortunately, there is no single method for adding Liquibase to an existing project, because each project is different. Liquibase provides some tools that help with this process.

There are two ways to add Liquibase: the first of them is just to start using Liquibase only for new changes, the second is to make it look like Liquibase was used from the beginning. (Liquibase documentation: Adding Liquibase on an Existing project, 2014)

3.2 Liquibase only for new changes

This approach consists of creating a changeSet only for new changes made to the database. It is not need to create a changeSet for changes made in the past. This approach is only to declare that from that moment Liquibase is in use. This method is the simplest to set up - it is just a mandate. (Liquibase documentation: Adding Liquibase on an Existing project, 2014)

Because Liquibase is not a difference-based application, it only looks at the DATABASECHANGELOG table to check which changeSets needs to be run. All existing objects in the database will be left unchanged, only the new changeSets will be run. (The problem with database diffs, 2007)

Usually the best moment to add Liquibase using this method is when moving from one version of software to the next; the database is usually in a reasonably consistent state. (Liquibase documentation: Adding Liquibase on an Existing project, 2014)

The biggest disadvantage of this method is that it requires other tools to bootstrap a new database. First, all pre-Liquibase changes must be made using other tools.

Snapshot of pre-Liquibase database should be created using the backup tool to create a script, which helps in new database bootstrapping. When a new database needs to be created, first, the snapshot should be loaded and then Liquibase update can be run. (Liquibase documentation: Adding Liquibase on an Existing project, 2014)

Because databases may vary between schemas, it is useful to use Liquibase functions such as preconditions, mark the changes run or contexts to standardize those variations. (Liquibase documentation: Adding Liquibase on an Existing project, 2014)

3.3 Make it look like Liquibase was used from the beginning

In this approach, the main goal is to create a changelog, which allows creating a database identical to the current state of the database if it is run against empty database. This approach is best in the long term; however, it requires more work at the beginning. (Liquibase documentation: Adding Liquibase on an Existing project, 2014)

The process of adding Liquibase support to existing database in this case is described in the following chapters. (Liquibase documentation: Adding Liquibase on an Existing project, 2014):

Creating changeLog files

Create changeLog files manually or automatically using CLI command generateChangeLog. This tool is especially good for large databases, however, all generated changeSets need to be gone through to ensure that they are correct. (Liquibase documentation: Generating Change Logs, 2014; Liquibase documentation: Adding Liquibase on an Existing project, 2014)

Liquibase allows reversing engineering the schema with the following command (Liquibase documentation: Liquibase Command Line, 2015):

```
liquibase --driver=oracle.jdbc.OracleDriver \
      --classpath=ojdbc6.jar \
      --changeLogFile=changelog.xml \
      --url="jdbc:oracle:thin:@localhost:1521:xe" \
      --username=rdziadosz \
      --password=secret \
      generateChangeLog
```

However, the use of this tool has some limitations. It is not possible to export stored procedures, functions, packages, triggers. Some objects specific to some database systems are also missing. (Liquibase documentation: Generating Change Logs, 2014)

For instance, in the case of Oracle Database it is not possible to export nested tables. In addition, details such as “not clustered” in the case of indexes may be missing, and types may vary. (Liquibase documentation: Generating Change Logs, 2014; Liquibase documentation: Adding Liquibase on an Existing project, 2014)

Running changeSets

The next step is to make sure that pre-Liquibase changes will only be applied to an empty database. There are several ways to do this:

a. The CLI command changeLogSync can be used. The information that the scripts have been executed will be added to DATABASECHANGELOG table, however, no changes to the database will be applied. (Liquibase documentation: Liquibase Command Line, 2015)

b. Context can be added to each of the pre-Liquibase changeSet, for instance “legacy”:

```
<changeSet id="1" author="rdziadosz" context="legacy">
    <createTable tableName="news">
        <column name="id" type="number"/>
        <column name="title" type="varchar(255)"/>
    </createTable>
</changeSet>
```

Then, on the new database, the script should be running using the context "legacy":

```
liquibase --driver=oracle.jdbc.OracleDriver \
      --classpath=ojdbc6.jar \
      --changeLogFile=changelog.xml \
      --contexts=legacy \
      --url="jdbc:oracle:thin:@localhost:1521:xe" \
      --username=rdziadosz \
      --password=secret \
      generateChangeLog
```

On the existing database the context should be different, it cannot remain empty or not set, because then all scripts will be executed and the execution will fail. (Liquibase documentation: Contexts, 2016)

```
      --contexts=nonLegacy
```

c. Another possibility is to add precondition tag to each generated changeSet.

```
<changeSet id="1" author="rdziadosz">

    <preConditions onFail="MARK_RAN">
        <not>
            <tableExists tableName="news"/>
        </not>
    </preConditions>

    <createTable tableName="news">
        <column name="id" type="number"/>
        <column name="title" type="varchar(255)"/>
    </createTable>

</changeSet>
```

A precondition is added to the changeLog. It checks for an existing element in the database, and if present, it skips the execution of script and marks changeSet as ran. (Liquibase documentation: Preconditions, 2014)

This method is time consuming, because it requires many different changes in the generated changeSets. Two previous methods can be easily automated.

This method can cause serious performance problems. Each precondition must be checked at the first execution that can take a long time. Therefore, this method is used in individual cases. (Liquibase documentation: Preconditions, 2014; Liquibase documentation: Adding Liquibase on an Existing project, 2014)

4 Assignment

4.1 Current situation and goals

The task it to add a version control system to an existing Oracle database. This complex database has been used a long time. Without Liquibase process of database, the upgrade is very complicated and difficult to develop.

It takes a great amount of time to create patches and improvements, because the error handling needs to be taken care of. It is not sure if the script will be executed a few times or once, therefore, any exception that can occur must be handled in SQL script.

All scripts must be executed in the correct order, because existing data could be corrupted.

Another goal is also to make it easy for developers to create a database instance. Now they share resources, which sometimes causes conflicts while working.

The next improvement is to accelerate the migration process by not executing scripts again in upgrade process and by adding CI.

The expected migration process is illustrated in Figure 5.

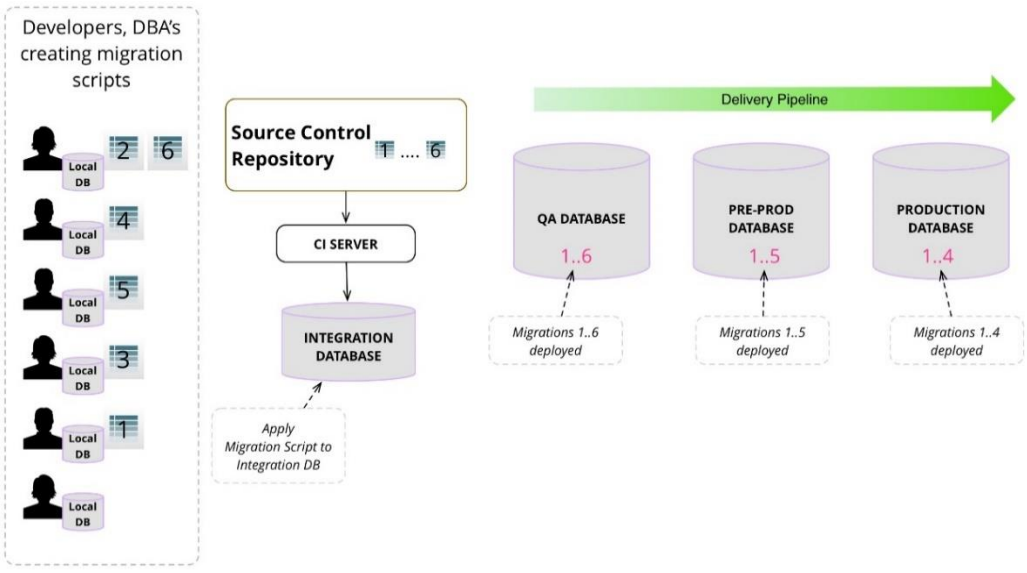


Figure 5 Life of a migration script (adapted from Evolutionary Database Design, 2016)

4.2 Initial idea for improvement

The first idea was to use the generateChangeLog command to the reverse engineering of the database. ChangeSets for objects such as tables, indexes, sequences, foreign keys, or views could be generated. Then the plans were to add the missing scripts that cannot be generated.

However, after the first tests, the results showed that much of the information is skipped and due to the huge amount of code, the generated XML files are very unreadable. In addition, the code generated by Liquibase ignored the comments that were previously in scripts making the code very difficult to maintain.

Some scripts must be executed in the correct order, which was not possible with the generated file. The code generated by this tool was not acceptable. It would take plenty of time and resources to correct this code.

4.3 Another approach to adding Liquibase

Due to the aforementioned problems, the decision was to manually or semi-automatically make the changes to the scripts to add Liquibase support in the project.

Some major architectural changes needed to be made. All of these changes are covered in the subsequent chapters.

5 Format of the changeLogs

5.1 SQL Formatted changeLog

The solution chosen was to use two formats of changeLogs XML and SQL. SQL format is compatible with Liquibase. The task required to do was to add Liquibase metatags to the existing scripts.

The SQL file may remain unmodified and it is compatible with Liquibase (one has to be aware that Liquibase does not support SQL Plus commands, and the scripts should be modified if used). However, to take full advantage of Liquibase it is worth adding Liquibase metatags to files.

Each SQL formatted changeLog must begin with the following line:

```
--liquibase formatted sql
```

(Liquibase documentation: Formatted SQL Changelogs, 2004)

5.2 SQL Formatted changeSet

Before each changeSet is metatag which in the sql file has the following format:

```
--changeset author:id attribute1:value1 attribute2:value2 [...]
```

Attributes can be, for example, runOnChange, runAlways or a context that allows using more advanced Liquibase functions. (Liquibase documentation: Formatted SQL Changelogs, 2004)

5.3 Adding metatags to multiple files

In order to speed up the work, a bash script was used to automate the tasks by adding meta tags to multiple files at once. The scripts of objects such as triggers or views were in separate files and had a unified structure.

Below is the code of bash script:

```
#!/usr/bin/env bash
#script appends liquibase metadata at the beginning of all *.sql
files in the directory
#create_filename is id of changeset (filename without .sql
extension)
for f in *.sql; do printf -- "--liquibase formatted sql\n--changeset
rdziadosz:${f::-4} runOnChange:true\n" | cat - "$f" > temp && mv
temp "$f"; done;
```

ChangeSets are created with the same id as the file name (without the extension).

After executing this script, the sample SQL script containing the view looks like the following:

```
--liquibase formatted sql
--changeset rdziadosz:locations_v runOnChange:true
 CREATE OR REPLACE FORCE VIEW locations_v AS
   SELECT c.country_id, c.country_name, l.location_id, l.city
   FROM countries c, locations l
   WHERE c.location_id = l.location_id;
```

5.4 XML formatted files

XML files are files such as baseline.xml or triggers.xml. They contain mostly commands not available in SQL files.

However, new changes will only be made through XML files, as they provide more capabilities, such as automatic rollback generation.

6 Organizing the changeLogs

6.1 Include and include all

Because the database has many objects, the code is divided into smaller parts, which allows managing them easier.

Liquibase has an include tag that allows creating changeLogs trees. It is important not to use XML build-in include, because the Liquibase parser recognizes the document as a single large XML file. It is known that Liquibase uses the file path to identify changeSet. Therefore, the use of such a solution could cause validation errors. (Liquibase documentation: Include, 2014)

A similar tool for the include tag is an includeAll tag. It allows adding multiple files at the same time. IncludeAll is looking for files with the *.xml and *.sql extension and adds them to changeLog. It is important that the files are executed in alphabetical order. (Liquibase documentation: IncludeAll, 2015)

The code below shows the use of include tag with two files and use includeAll for the whole directory “views”:

```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog
  xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
  http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.5.xsd">

  <include file="create_tables.xml" relativeToChangelogFile="true"/>
  <include file="constraints.sql" relativeToChangelogFile="true"/>
  <includeAll path="views" relativeToChangelogFile="true"/>

</databaseChangeLog>
```

6.2 “Replaceable” database objects

What can be replaced in Oracle?

Liquibase runs changeSets only once by default. If migrations are run again, they will be ignored. Sometimes this is not a wanted action. In the Oracle database objects such as

- stored procedures,
- functions,
- views,
- packages,
- synonyms,
- triggers

can be easily replaced using OR REPLACE statement. These objects need not to be deleted from the database each time before recreating them. (Oracle Database SQL Language Reference)

runOnChange

In case when changeSet has been changed since it was run against the database, Liquibase returns the checksum error. If it is the intended action, runOnChange attribute can be used.

After launching Liquibase with the changeSet below, DATABASECHANGELOG table contained the entry shown in Figure 6.

```
--liquibase formatted sql
--changeset rdziadosz:locations_v runOnChange:true
 CREATE OR REPLACE FORCE VIEW locations_v AS
   SELECT c.country_id, c.country_name
   FROM countries c;
```

ID	AUTHOR	FILENAME	ORDEREXECUTED	EXECTYPE	MD5SUM	DESCRIPTION	DATEEXECUTED
locations_v	rdziadosz	views/locations_v.sql	5503	EXECUTED	7:a7ff218e170206475b969e5d7e1fee55	sql	03.05.2017 17:45:46,732000

Figure 6 Entry in the DATABASECHANGELOG table after the first run

Then the view code was changed. Liquibase was executed again.

```
--liquibase formatted sql
--changeset rdziadosz:locations_v runOnChange:true
 CREATE OR REPLACE FORCE VIEW locations_v AS
   SELECT c.country_id, c.country_name, l.location_id, l.city
   FROM countries c, locations l
   WHERE c.location_id = l.location_id;
```

As a result, the view was updated, and the entry in the table looked as shown in Figure 7.

ID	AUTHOR	FILENAME	ORDEREXECUTED	EXECTYPE	MD5SUM	DESCRIPTION	DATEEXECUTED
locations_v	rdziadosz	views/locations_v.sql	5515	RERAN	7:c5c946dcc34c2f0c147e864b408c10fa	sql	03.05.2017 17:51:48,657000

Figure 7 Entry in the DATABASECHANGELOG table after the second run

Using runOnChange attribute the change will only be applied if the changeSet is changed. Liquibase compares the changeSet checksum with the checksum in the database. If the MD5 checksum was deleted from the database (e.g. by clearCheckSums command), the change is also executed. (Liquibase documentation: <changeSet> tag, 2015)

runAlways

Sometimes it is necessary to run changeSet each time. In this case, the runAlways attribute is used. (Liquibase documentation: <changeSet> tag, 2015)

However, if changeSet is be changed, the validation will fail, unlike runOnChange. If this is intentional, tag validCheckSum is used, as in the case below:

```
<changeSet author="rdziadosz" id="clear_table" runAlways="true">
    <validCheckSum>any</validCheckSum>
    <sql>call CLEAR_TABLE('loactions_temp');</sql>
</changeSet>
```

(Liquibase documentation: <changeSet> tag, 2015)

6.3 Directory structure

Because there are several thousand XML and SQL files in the project, their organization is very important. It is known that over time the project will be further developed, which means that there will be even more of them.

The directory structure used in the project is shown in Figure 8. Scripts to create non-replaceable objects are located in the “install” directory. Each type of object is in a separate directory.

The “latest” directory has been created for files where “CREATE OR REPLACE” statement can be used (triggers, views, etc.). It means that the updated version of the object can simply replace an old one. Because modifications are made in one directory, it is easy to track history changes using a version control system such as Git.

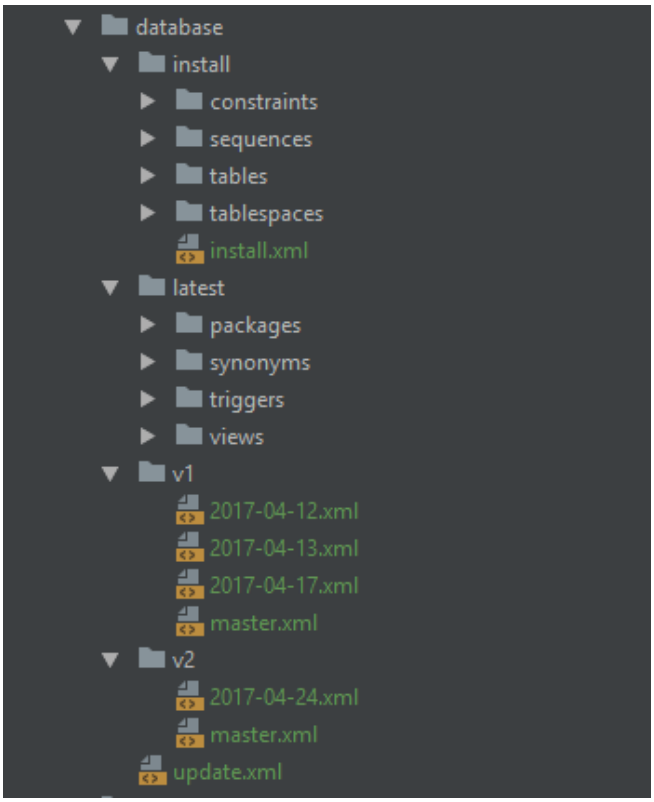


Figure 8 Directory structure

Directories where the name starts with “v” (eg. v1, v2, etc.) contain changeLogs which upgrade the data from one version to another. A directory tree organized in this way makes branching and merging easy. (Tutorial Using Oracle, 2015)

7 Inserting Setup Data

7.1 Loading Data into a Table

One of the tasks was to improve the addition of the initial data. These data are fixed, same for each created database. Sometimes, with the application update, it is necessary to change this data.

The old method of data entry was that each time the table was completely cleaned, and then the data was entered using thousands of INSERT INTO statements.

Even if the data was not changed, this process was performed, which unnecessarily increased the update time.

7.2 Load Update Data

Liquibase has made this process easier. The loadUpdateData function allows to add or update data that is loaded from a CSV file. (Liquibase documentation: Change: ‘loadUpdateData’, 2014)

To use this feature, a CSV file with data needs to be prepared. The following snippet contains a header and several entries to be inserted into the table:

```
ID,LON,LAT,PROBABILITYID,LON,LAT,PROBABILITY
"41","50.083333","19.916667","13.5"
"42","49.947942","20.242758","26.2"
"43',"53.84247,"20.947564","24.2"
"44","53.454207,"22.424624","16.2"
```

The code below shows the loadUpdateData tag. It is necessary to declare the table name to which data is to be entered, the location of the CSV file, and the column that is the primary key. (Liquibase documentation: Change: 'loadUpdateData', 2014):

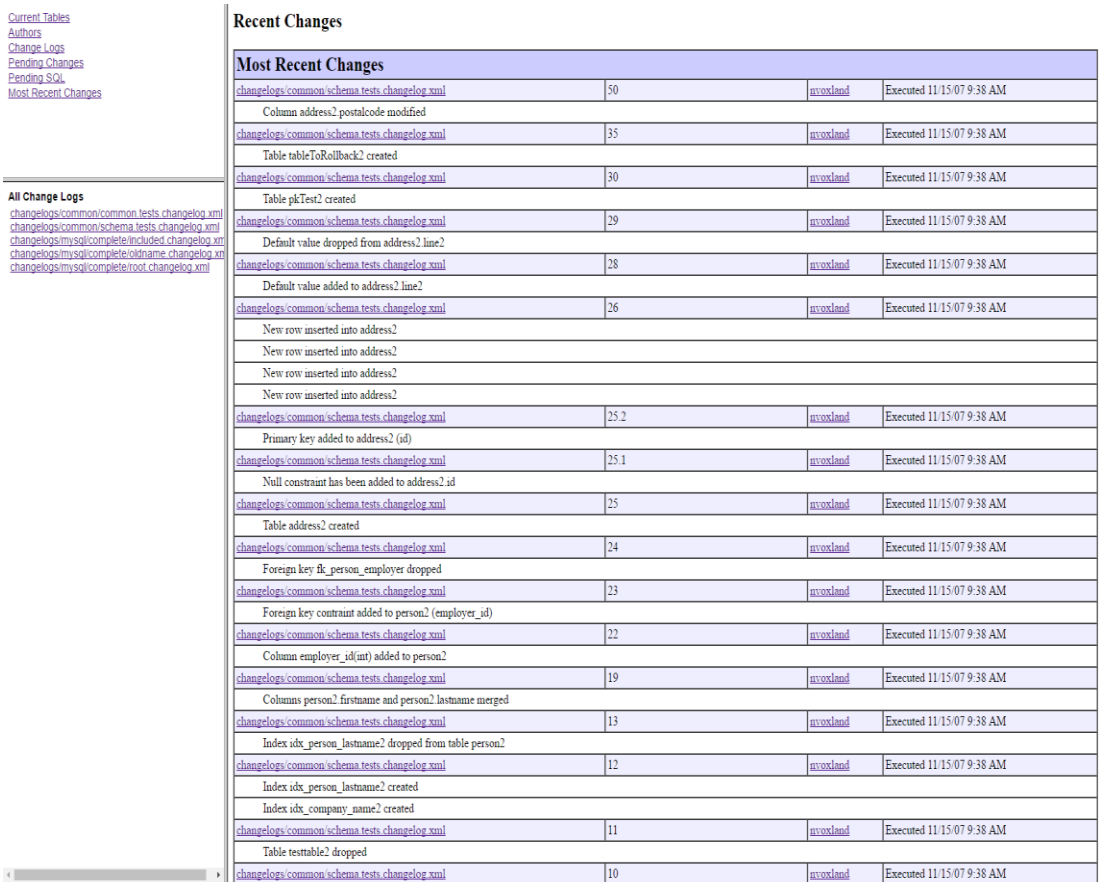
```
<changeSet author="rdziadosz" id="loadData-settings"
runOnChange="true">

    <loadUpdateData catalogName="world"
                primaryKey="ID"
                encoding="UTF-8"
                file="world_data.csv"
                quotchar="&quot;"
                schemaName="world"
                separator=","
                tableName="settings">
    </loadUpdateData>

</changeSet>
```

During first execution, data will be added to the database. Due to the runOnChange attribute, Liquibase will update the data in next executions if the file will be changed. However, one needs to keep in mind that if an entry will be deleted from a file, it will still remain in the database. In this case, the delete tag is needed to remove one or more entries from the table. (Liquibase documentation: Change: 'loadUpdateData', 2014)

8 Database Documentation

Database Documentation Generator is a very useful tool especially in teamwork. It allows creating documentation in HTML format. This tool can only be used by the CLI. (Liquibase documentation: Database Documentation Generator, 2014)

The documentation contains information on the executed and pending changes, information about the current and past structure of the database, and the time and authors of the changes that have been made. The generated documentation is shown in Figure 9.



Current Tables
Authors
Change Logs
Pending Changes
Pending SQL
Most Recent Changes

All Change Logs
changelogs/common/common.tests.changelog.xml
changelogs/common/schema.tests.changelog.xml
changelogs/mysql/complete/included.changelog.xml
changelogs/mysql/complete/oldname.changelog.xml
changelogs/mysql/complete/root.changelog.xml

Recent Changes

Most Recent Changes			
changelogs/common/schema.tests.changelog.xml	50	nvoxland	Executed 11/15/07 9:38 AM
Column address2.postalcode modified			
changelogs/common/schema.tests.changelog.xml	35	nvoxland	Executed 11/15/07 9:38 AM
Table tableToRollback2 created			
changelogs/common/schema.tests.changelog.xml	30	nvoxland	Executed 11/15/07 9:38 AM
Table pkTest2 created			
changelogs/common/schema.tests.changelog.xml	29	nvoxland	Executed 11/15/07 9:38 AM
Default value dropped from address2.line2			
changelogs/common/schema.tests.changelog.xml	28	nvoxland	Executed 11/15/07 9:38 AM
Default value added to address2.line2			
changelogs/common/schema.tests.changelog.xml	26	nvoxland	Executed 11/15/07 9:38 AM
New row inserted into address2			
New row inserted into address2			
New row inserted into address2			
New row inserted into address2			
changelogs/common/schema.tests.changelog.xml	25.2	nvoxland	Executed 11/15/07 9:38 AM
Primary key added to address2 (id)			
changelogs/common/schema.tests.changelog.xml	25.1	nvoxland	Executed 11/15/07 9:38 AM
Null constraint has been added to address2.id			
changelogs/common/schema.tests.changelog.xml	25	nvoxland	Executed 11/15/07 9:38 AM
Table address2 created			
changelogs/common/schema.tests.changelog.xml	24	nvoxland	Executed 11/15/07 9:38 AM
Foreign key fk_person_employer dropped			
changelogs/common/schema.tests.changelog.xml	23	nvoxland	Executed 11/15/07 9:38 AM
Foreign key contraint added to person2 (employer_id)			
changelogs/common/schema.tests.changelog.xml	22	nvoxland	Executed 11/15/07 9:38 AM
Column employer_id(int) added to person2			
changelogs/common/schema.tests.changelog.xml	19	nvoxland	Executed 11/15/07 9:38 AM
Columns person2.firstname and person2.lastname merged			
changelogs/common/schema.tests.changelog.xml	13	nvoxland	Executed 11/15/07 9:38 AM
Index idx_person_lastname2 dropped from table person2			
changelogs/common/schema.tests.changelog.xml	12	nvoxland	Executed 11/15/07 9:38 AM
Index idx_person_lastname2 created			
Index idx_company_name2 created			
changelogs/common/schema.tests.changelog.xml	11	nvoxland	Executed 11/15/07 9:38 AM
Table testtable2 dropped			
changelogs/common/schema.tests.changelog.xml	10	nvoxland	Executed 11/15/07 9:38 AM

Figure 9 Generated documentation

The documentation can be generated by running the following command:

```
liquibase --driver=oracle.jdbc.OracleDriver \
      --classpath=ojdbc6.jar \
      --changeLogFile=changelog.xml \
      --url="jdbc:oracle:thin:@localhost:1521:xe" \
      --username=rdziadosz \
      --password=secret \
      dbDoc \
      /documentation
```

The output files will be in the directory specified after the command name (in this case “/documentation”).

The advantage of this documentation is that it is precise and error-free. All changes made in the past will be described there. It takes a few seconds to generate it. Good documentation makes developers’ work much easier.

9 Contexts

9.1 Different features of the runtime environment

In the created project, some databases differed according to customer requirements. Sometimes it was necessary to run only some changeSets or skip them.

Creating separate changeLogs for each database would be very laborious. However, Liquibase offers the opportunity to use contexts. (Contexts Vs. Labels, 2014)

9.2 Adding contexts to changeSets

To use contexts, the context attribute needs to be added to changeSet. Then, during executing Liquibase, a context needs to be specified, depending on the environment to be created. (Liquibase documentation: Contexts, 2016)

The following snippets show example changeSets with specific contexts:

```
<changeSet author="rdziadosz" id="clear_table" runAlways="true"
context=”locations”>

    <validCheckSum>any</validCheckSum>
    <sql>call CLEAR_TABLE('loactions_temp');</sql>

</changeSet>
```

```
<changeSet id="1" author="rdziadosz" context=”legacy”>

    <createTable tableName="news">
        <column name="id" type="number"/>
        <column name="title" type="varchar(255)"/>
    </createTable>

</changeSet>
```

9.3 Running Liquibase with a specific context

To run changes, the --contexts parameter is used on the command line interface. When Liquibase with needs to be run with many contexts, they should be separated with commas. (Liquibase documentation: Contexts, 2016)

An example command to run Liquibase with two contexts is given below:

```
liquibase --driver=oracle.jdbc.OracleDriver \
      --classpath=ojdbc6.jar \
      --changeLogFile=changelog.xml \
      --contexts="locations,legacy" \
      --url="jdbc:oracle:thin:@localhost:1521:xe" \
      --username=rdziadosz \
      --password=secret \
      update
```

If the context remains empty or not set, then all changeSets will be executed. If all changeSets without the context set need to be run, the --contexts = x parameter should be set, where x is different from any context added to changeSets. (Liquibase documentation: Contexts, 2016)

10 Parameters

Parameters are a very useful function. Parameters allows inserting text fragments into changeLogs or changeSets. For example, the project needed to create a user, but for each database users had to have a different password - given when creating the database. (Change Log Parameters, 2014)

The following code snippet is used to create a user:

```
--liquibase formatted sql
--changeset rdziadosz:create_master
CREATE USER MASTER IDENTIFIED BY "${master.password}";
```

Liquibase dynamically replaces parameters with syntax ${name} to given parameter. Parameters can be included in the XML file or through the CLI. (Change Log Parameters, 2014)

In the changeLog file the parameter tag is used. The following snippet allows including a parameter in an XML file (Change Log Parameters, 2014):

```
<property name="master.password" value="secret"/>
```

On the command line, the parameter must be preceded by the letter D. The following fragment allows inserting a parameter in CLI (Liquibase documentation: Command Line, 2015):

```
liquibase --driver=oracle.jdbc.OracleDriver \
     --classpath=ojdbc6.jar \
     --changeLogFile=changelog.xml \
     --url="jdbc:oracle:thin:@localhost:1521:xe" \
     --username=rdziadosz \
     --password=secret \
     update \
     -Dmaster.password=secret
```

11 Testing

After the process of creating the changeLogs and running them, it was necessary to test if the database schema is identical to the current production database version.

Tests were done with Toad for Oracle. The Compare Schema function was used to check the database, shown in Figure 10. All discrepancies were manually checked and corrected in changeSets. Compare Schema allows comparing one schema in multiple databases. (Toad for Oracle 12.1 - Guide to Using Toad for Oracle)

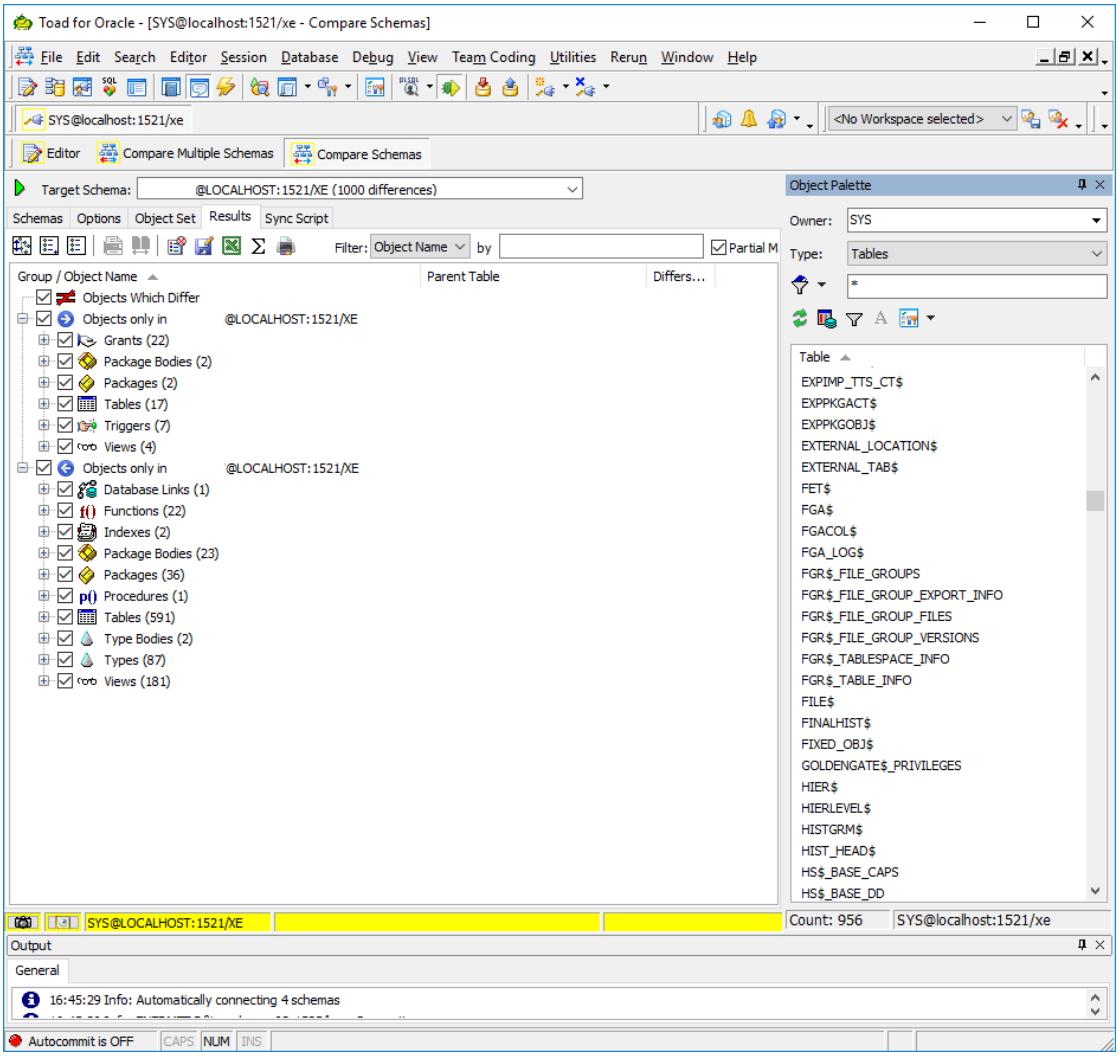


Figure 10 Compare Schemas in Toad for Oracle

For a more complex database, Compare Multiple Schemas function is useful. It is similar to Compare Schema but allows working with multiple schemas in two databases. Compare Multiple Schemas is shown in Figure 11. (Toad for Oracle 12.1 - Guide to Using Toad for Oracle)

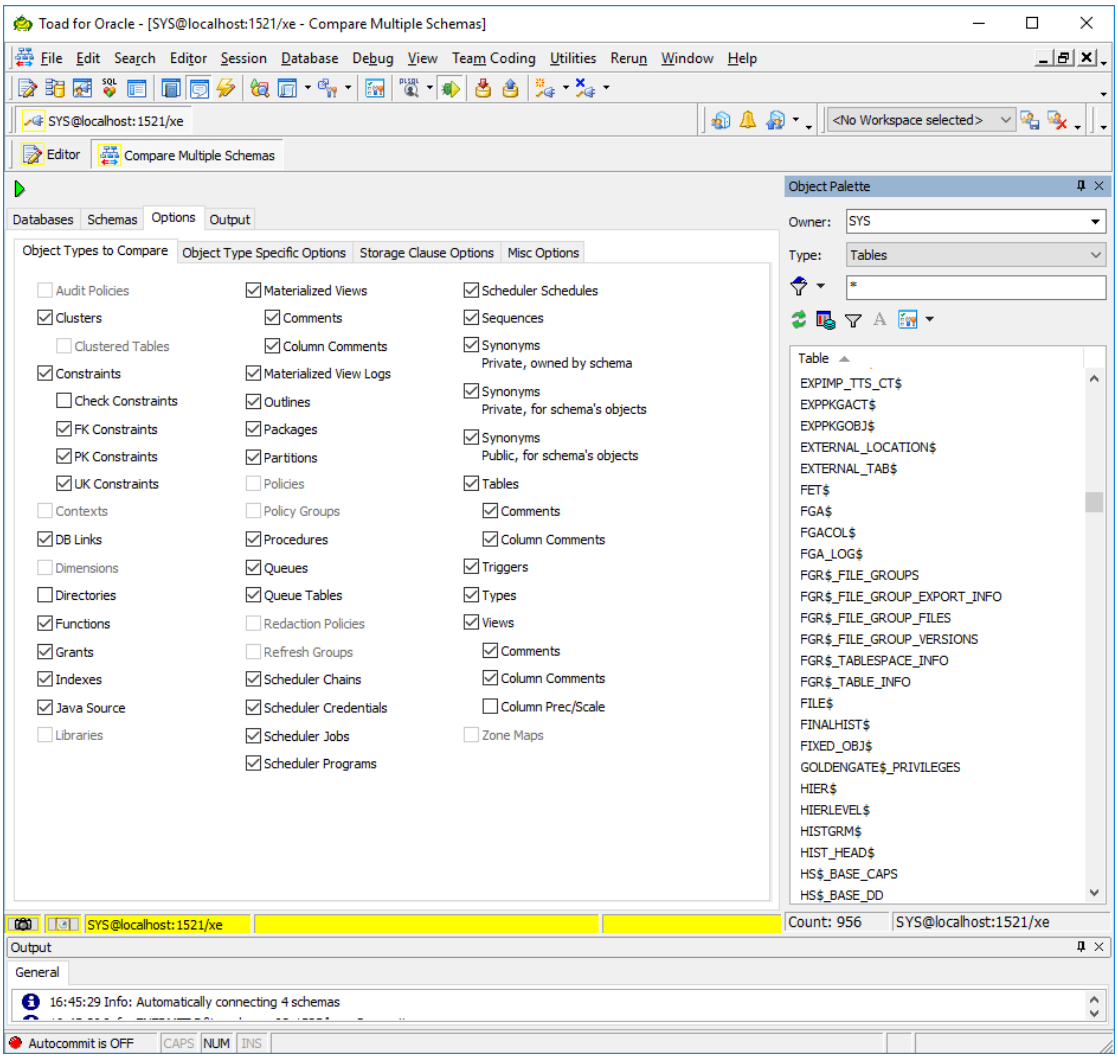


Figure 11 Compare Multiple Schemas in Toad for Oracle

12 Conclusions

The purpose of this thesis was to add a version control system to the Oracle database. The migration process was complex and error-prone. Another goal was also to make it easy for developers to create a database instance. Previously, conflicts were sometimes caused by work on a shared base.

All of the aforementioned goals were achieved using the open source tool, Liquibase. It allows easily management of schema migrations, control of database versions and automatic generation of documentation about the changes.

As a result of the actions taken, the process of making changes to the database schema was simplified. In addition, migrations history is saved and documentation is created automatically. The database is easy to be set up on the developers' devices so they did not have to share it.

The current version of the database that runs with Liquibase is in the testing phase. The branch will be merged into the main branch in the forthcoming weeks - when the development of the newest version of software developed by the company starts.

The problems that came up when working with Liquibase were, among other things, the lack of support for multiple connections to the database. This would sometimes be useful when other privileges need to be used in the database. Another disadvantage was the lack of support for SQL Plus commands, which required some code changes. Some of the problems are planned to be solved in the future Liquibase releases.

Further development will undoubtedly consist of creating new changes using only Liquibase. After each release, the updates will be merged into the base version to speed up the migration process and make the Liquibase changeLogs files simpler.

Working with Liquibase was a new useful experience helped me to increase my skills. Especially analytical skills and good organization were important. Due to the complexity of the database, it took plenty of time to understand how it worked.

A very important aspect of work is the contact with the team. Due to the co-operation with experienced programmers, I was able to resolve the problems quickly. The help of the entire team was invaluable.

After this experience, I can say that the versioned database is almost a necessity in a project that is continuously deployment. Versioning makes it very easy for both application development and deployment of the product. It is also worth remembering to implement solutions like Liquibase as early as possible. This can save programmers plenty of work.

References

Hussain, S. 2016. *SQL Server Database Change Management with Liquibase* Accessed on 01 April 2017. Retrieved from https://www.mssqltips.com/sqlservertip/4340/sql-server-database-change-management-with-liquibase/

Köbler, N. 2013. *Continuous database migration with Liquibase and Flyway* Accessed on 21 April 2017. Retrieved from http://www.h-online.com/developer/features/Continuous-database-migration-with-Liquibase-and-Flyway-1860080.html

Merriam-Webster *Database - Definition of database* Accessed on 07 May 2017. Retrieved from https://www.merriam-webster.com/dictionary/database

Microsoft 2017. *Microsoft Dev Center: Environment Variables* Accessed on 22 April 2017. Retrieved from https://msdn.microsoft.com/en-us/library/windows/desktop/ms682653(v=vs.85).aspx

Oracle *Oracle Database Concepts: Schema Objects* Accessed on 07 May 2017. Retrieved from https://docs.oracle.com/cd/B28359_01/server.111/b28318/schema.htm#CNCPT010

Oracle *Oracle Database SQL Language Reference: CREATE FUNCTION* Accessed on 03 May 2017. Retrieved from http://docs.oracle.com/database/122/LNPLS/CREATE-FUNCTION-statement.htm#LNPLS01370

Oracle *Oracle Database SQL Language Reference: CREATE PACKAGE* Accessed on 03 May 2017. Retrieved from http://docs.oracle.com/database/122/SQLRF/CREATE-PACKAGE.htm#SQLRF01306

Oracle *Oracle Database SQL Language Reference: CREATE PROCEDURE* Accessed on 03 May 2017. Retrieved from http://docs.oracle.com/database/122/LNPLS/CREATE-PROCEDURE-statement.htm#LNPLS01373

Oracle *Oracle Database SQL Language Reference: CREATE SYNONYM* Accessed on 01 April 2017. Retrieved from http://docs.oracle.com/database/122/SQLRF/CREATE-SYNONYM.htm#SQLRF01401

Oracle *Oracle Database SQL Language Reference: CREATE TRIGGER* Accessed on 03 May 2017. Retrieved from http://docs.oracle.com/database/122/LNPLS/CREATE-TRIGGER-statement.htm#LNPLS01374

Oracle *Oracle Database SQL Language Reference: CREATE VIEW* Accessed on 01 April 2017. Retrieved from http://docs.oracle.com/database/122/SQLRF/CREATE-VIEW.htm#SQLRF01504

Quest *Toad for Oracle 12.1 - Guide to Using Toad for Oracle* Accessed on 03 May 2017. Retrieved from https://support.quest.com/technical-documents/toad-for-oracle/12.1/guide-to-using-toad-for-oracle/

Ranch, H. 2011. *A simple introduction to database change management with SQLite and Liquibase* Accessed on 07 May 2017. Retrieved from http://henryranch.net/tutorials/a-simple-introduction-to-database-change-management-with-sqlite-and-liquibase/

Sadalage, P., Fowler, M. 2016. *Evolutionary Database Design* Accessed on 01 April 2017. Retrieved from https://www.martinfowler.com/articles/evodb.html

Voxland, N. 2014. *Contexts Vs. Labels* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/2014/11/contexts-vs-labels.html

Voxland, N. 2015. *Liquibase documentation: <changeSet> tag* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/documentation/changeset.html

Voxland, N. 2014. *Liquibase documentation: Adding Liquibase on an Existing project* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/documentation/existing_project.html

Voxland, N. 2014. *Liquibase documentation: Change Log Parameters* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/documentation/changelog_parameters.html

Voxland, N. 2014. *Liquibase documentation: Change: 'loadUpdateData'* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/documentation/changes/load_update_data.html

Voxland, N. 2016. *Liquibase documentation: Contexts* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/documentation/contexts.html

Voxland, N. 2013. *Liquibase documentation: Database Change Log File* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/documentation/databasechangelog.html

Voxland, N. 2014. *Liquibase documentation: Database Documentation Generator* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/documentation/dbdoc.html

Voxland, N. 2014. *Liquibase documentation: DATABASECHANGELOG table* Accessed on 09 April 2017. Retrieved from http://www.liquibase.org/documentation/databasechangelog_table.html

Voxland, N. 2015. *Liquibase documentation: DATABASECHANGELOGLOCK table* Accessed on 09 April 2017. Retrieved from http://www.liquibase.org/documentation/databasechangeloglock_table.html

Voxland, N. 2014. *Liquibase documentation: Formatted SQL Changelogs* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/documentation/sql_format.html

Voxland, N. 2014. *Liquibase documentation: Generating Change Logs* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/documentation/generating_changelogs.html

Voxland, N. 2014. *Liquibase documentation: Include* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/documentation/include.html

Voxland, N. 2015. *Liquibase documentation: IncludeAll* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/documentation/includeall.html

Voxland, N. 2016. *Liquibase documentation: Installation* Accessed on 12 April 2017. Retrieved from http://www.liquibase.org/download/

Voxland, N. 2014. *Liquibase documentation: JSON Format* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/documentation/json_format.html

Voxland, N. 2015. *Liquibase documentation: Liquibase Command Line* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/documentation/command_line.html

Voxland, N. 2013. *Liquibase documentation: Other Changelog formats* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/documentation/other_formats.html

Voxland, N. 2014. *Liquibase documentation: Preconditions* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/documentation/preconditions.html

Voxland, N. 2014. *Liquibase documentation: Rolling Back ChangeSets* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/documentation/rollback.html

Voxland, N. 2014. *Liquibase documentation: Supported Databases* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/databases.html

Voxland, N. 2014. *Liquibase documentation: Updating the Database* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/documentation/update.html

Voxland, N. 2014. *Liquibase documentation: XML Format* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/documentation/xml_format.html

Voxland, N. 2014. *Liquibase documentation: YAML Format* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/documentation/yaml_format.html

Voxland, N. *Liquibase Official Website* Accessed on 06 April 2017. Retrieved from http://www.liquibase.org/index.html

Voxland, N. 2007. *The problem with database diffs* Accessed on 01 April 2017. Retrieved from http://www.liquibase.org/2007/06/the-problem-with-database-diffs.html

Voxland, N. 2015. *Tutorial Using Oracle* Accessed on 03 May 2017. Retrieved from http://www.liquibase.org/tutorial-using-oracle

검린 2009. *LiquiBase를 사용하여 데이터베이스 변경 관리하기* Accessed on 16 April 2017. Retrieved from http://mirnae.blog.me/100065640596