Installation

This document assumes that you’ve already installed an Oracle database with APEX installation upgraded.

# Jenkins

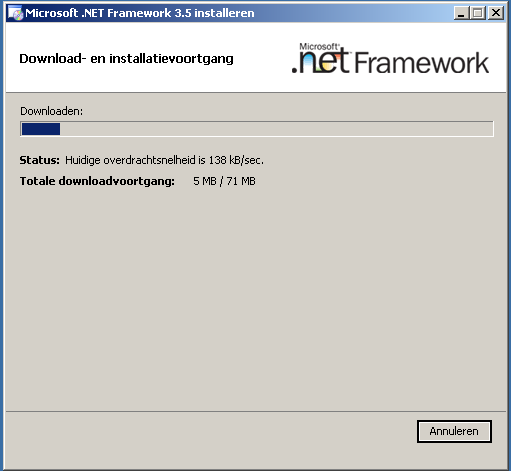
First install .NET 3.5. It’s very important to install the 3.5 version. You can find it over here: <https://www.microsoft.com/nl-be/download/details.aspx?id=21>

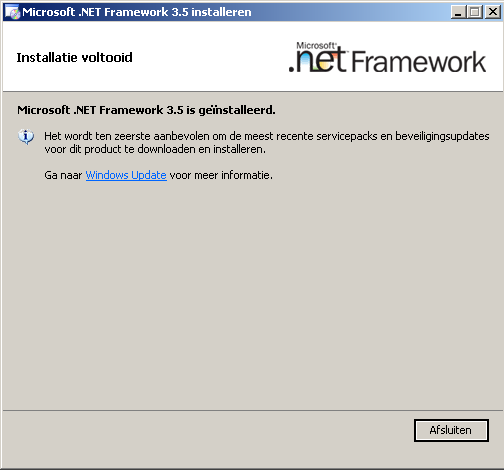


Pick download and choose **no thanks and continue** in the next option to continue.  
Execute the file





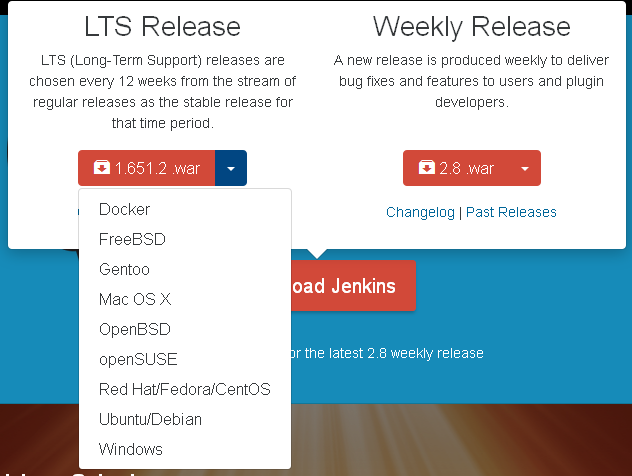




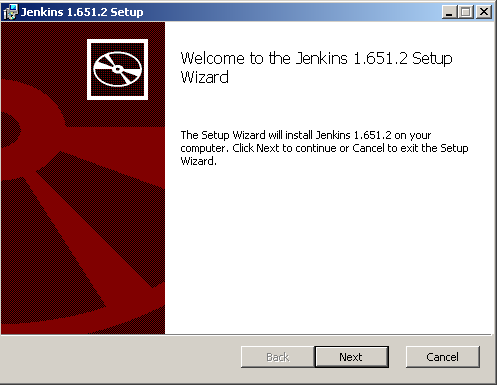
Click close and start the installation of Jenkinst

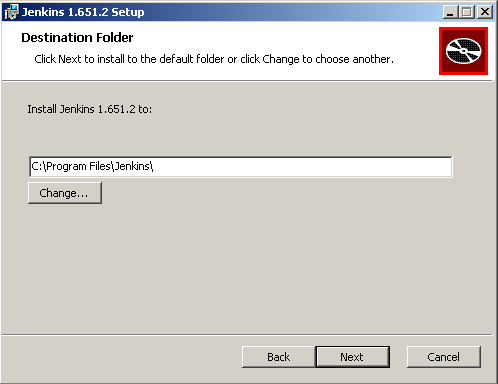


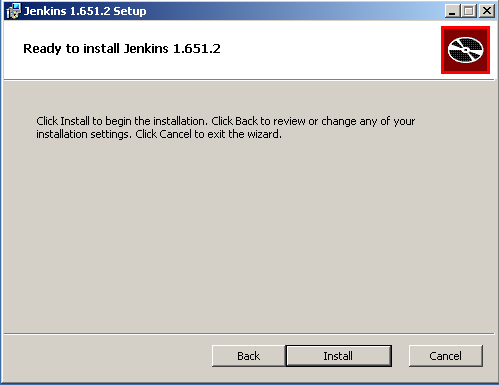
Use the windows version under LTS release



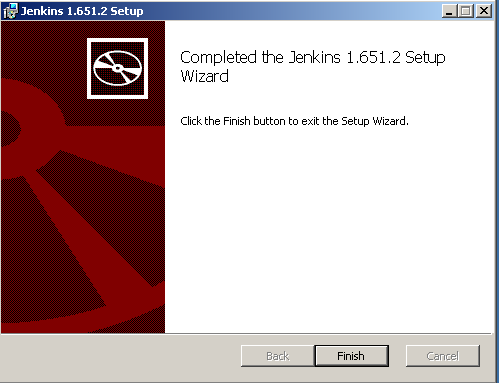
Save and unpack the zip. Run the msi installer.







Click install. Jenkins will be installed in the program files folder.

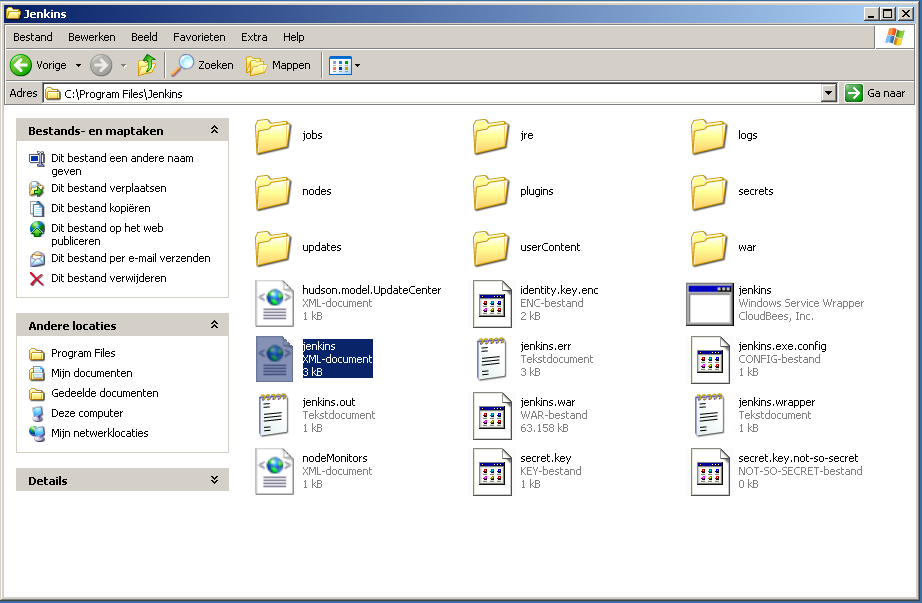


Jenkins should now be booting

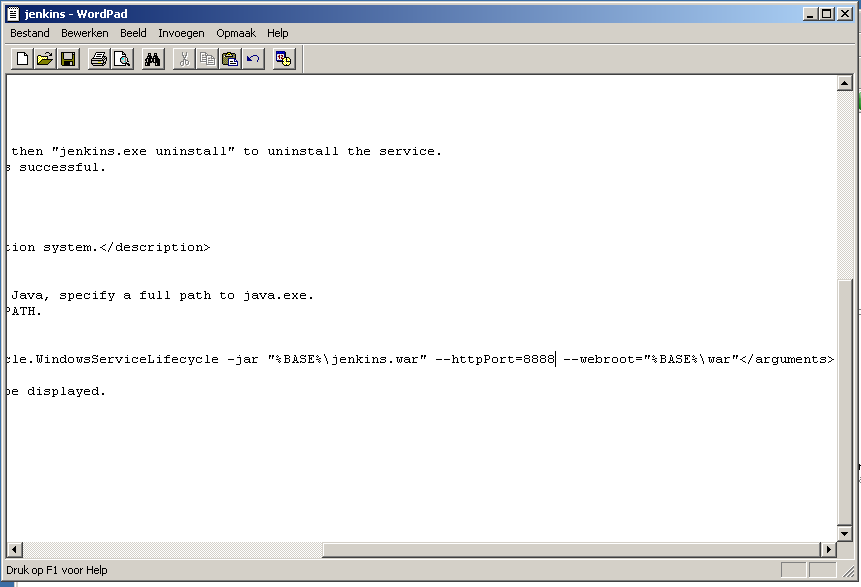


# Changing the port of Jenkins

Open the Jenkins.xml in c:\Program Files\Jenkins in wordpad



Find httpPort in the document and change it

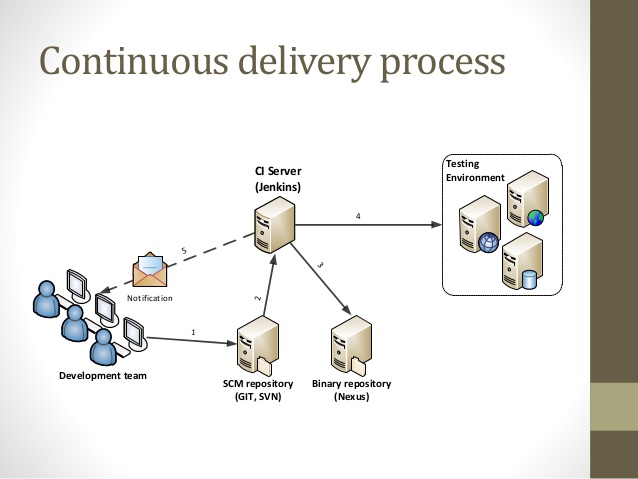


Reboot the pc to make the changes active.

User Manual

# The idea of building sql in a Continuous delivery process

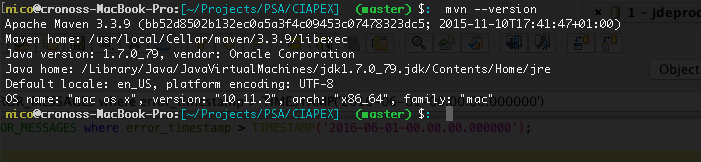
So first things first, what exactly is continuous delivery



As we see in the picture above, we have a development team that pushes changes to a SCM repository. Our CI server, in our case Jenkins, is listening on these remote changes and can trigger a build. We package this build with a tool called Jenkins and deploy it to our development street.

# Configuring your PC

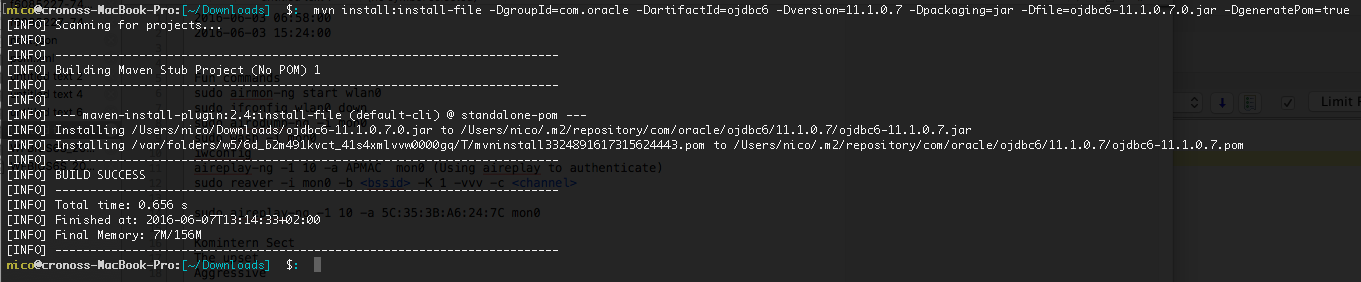
Installation of maven: Maven is our build tool.   
Follow the instructions: <https://maven.apache.org/install.html>  
Test maven with mvn –version in a command prompt or terminal



Because Oracle is not providing it’s jdbc driver by default in the maven repository we must install it manually. Download [ojdbc6-11.1.0.7.0.jar](https://maven-eu.nuxeo.org/nexus/content/groups/public/com/oracle/ojdbc6/11.1.0.7.0/ojdbc6-11.1.0.7.0.jar)

Install it with this command:

mvn install:install-file -DgroupId=com.oracle -DartifactId=ojdbc6 -Dversion=11.1.0.7 -Dpackaging=jar -Dfile=ojdbc6-11.1.0.7.0.jar -DgeneratePom=true



# First build of sample application

Create a folder structure within a sampleproject folder:  
-src/main/resources  
-src/main/sql  
-src/main/plsql

Create the main pom. This pom will contain properties of the servers. For the developpers this can be a pom without production properties. The build server should have information of all servers:  
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.project</groupId>

<artifactId>mainpom</artifactId>

<packaging>pom</packaging>

<version>1.0-SNAPSHOT</version>

<name>mainPom</name>

<profiles>

<profile>

<id>dev</id>

<activation>

<activeByDefault>true</activeByDefault>

</activation>

<properties>

<ojdbc.groupId>com.oracle</ojdbc.groupId>

<ojdbc.artifactId>ojdbc6</ojdbc.artifactId>

<ojdbc.version>11.1.0.7</ojdbc.version>

<jdbc.driverClassName>oracle.jdbc.driver.OracleDriver</jdbc.driverClassName>

<jdbc.url>jdbc:oracle:thin:@192.168.1.1:1521:orcl</jdbc.url>

<jdbc.username>devserver </jdbc.username>

<jdbc.password>devpassword</jdbc.password>

</properties>

</profile>

<profile>

<id>test</id>

<properties>

<ojdbc.groupId>com.oracle</ojdbc.groupId>

<ojdbc.artifactId>ojdbc6</ojdbc.artifactId>

<ojdbc.version>11.1.0.7</ojdbc.version>

<jdbc.driverClassName>oracle.jdbc.driver.OracleDriver</jdbc.driverClassName>

<jdbc.url>jdbc:oracle:thin:@192.168.1.1:1521:orcl</jdbc.url>

<jdbc.username>testserver </jdbc.username>

<jdbc.password>testpassword</jdbc.password>

</properties>

</profile>

</profiles>

</project>

Install this pom on the machine with the following command: mvn clean install –f mainPom.xml

You can use as many profiles as you want. In this example the dev profile is the default profile. The best practice is that the build server uses all profiles available (tst/acceptance/production) and the developers only use a development profile.

Now you can use a default pom file for your projects. Be sure that you use the same parent groupId, artifactId and version as you define in your main pom:

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.project</groupId>

<artifactId>test-project</artifactId>

<packaging>jar</packaging>

<version>1.0-SNAPSHOT</version>

<name>Test Project</name>

<parent>

<groupId>com.project</groupId>

<artifactId>mainpom</artifactId>

<version>1.0-SNAPSHOT</version>

</parent>

<build>

<plugins>

<plugin>

<groupId>org.liquibase</groupId>

<artifactId>liquibase-maven-plugin</artifactId>

<version>3.0.5</version>

<dependencies>

<dependency>

<groupId>${ojdbc.groupId}</groupId>

<artifactId>${ojdbc.artifactId}</artifactId>

<version>${ojdbc.version}</version>

</dependency>

</dependencies>

<configuration>

<changeLogFile>src/main/master.xml</changeLogFile>

<driver>${jdbc.driverClassName}</driver>

<url>${jdbc.url}</url>

<username>${jdbc.username}</username>

<password>${jdbc.password}</password>

<promptOnNonLocalDatabase>false<promptOnNonLocalDatabase>

<verbose>false</verbose>

</configuration>

<executions>

<execution>

<id>update</id>

<phase>process-resources</phase>

<goals>

<goal>update</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

In the scope of this project we will only be using Liquibase. Liquibase is a powerful tool to deploy sql and plsql scripts to a database. It protects the developer of unwanted changes in scripts that can only be executed once and keeps track of all executed scripts.

You can extend these pom files with much more tools like utPlsql for unit testing and Natural Doc for auto documentation of your packages and/or tables. This is outside the scope but it shows you how powerful using a building mechanism can be. When you push your code to the build server, Jenkins will also be using these pom files to build everything to the correct server.

As you can see the first script runned by liquibase is src/main/master.xml. In our demo application we will set this master to run some scripts. For now the content should just be:

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

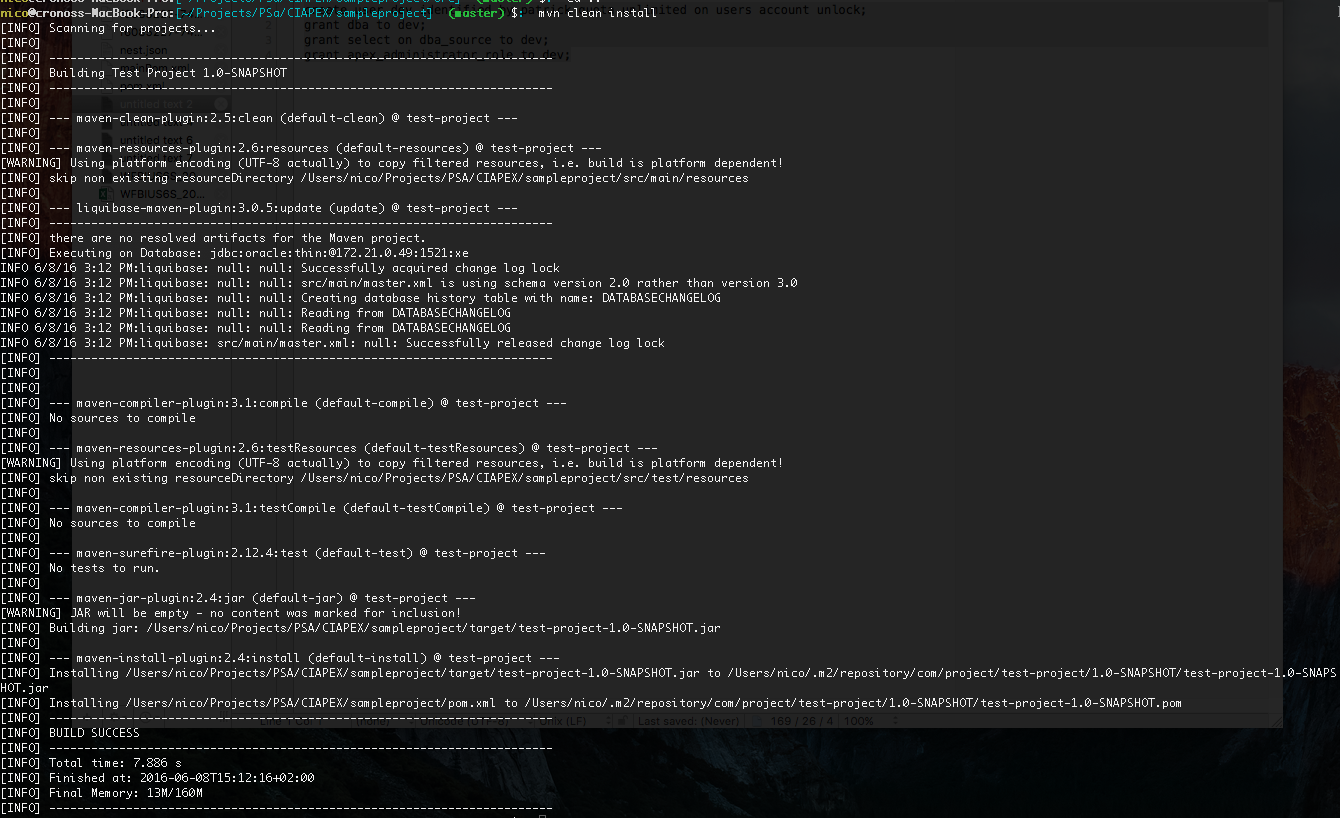
<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog"

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-3.0.xsd">

</databaseChangeLog>

Now you can test the build by executing mvn clean install in the sampleproject root directory. (The folder where pom.xml is). If you see something like this it means your pc is correctly installed.



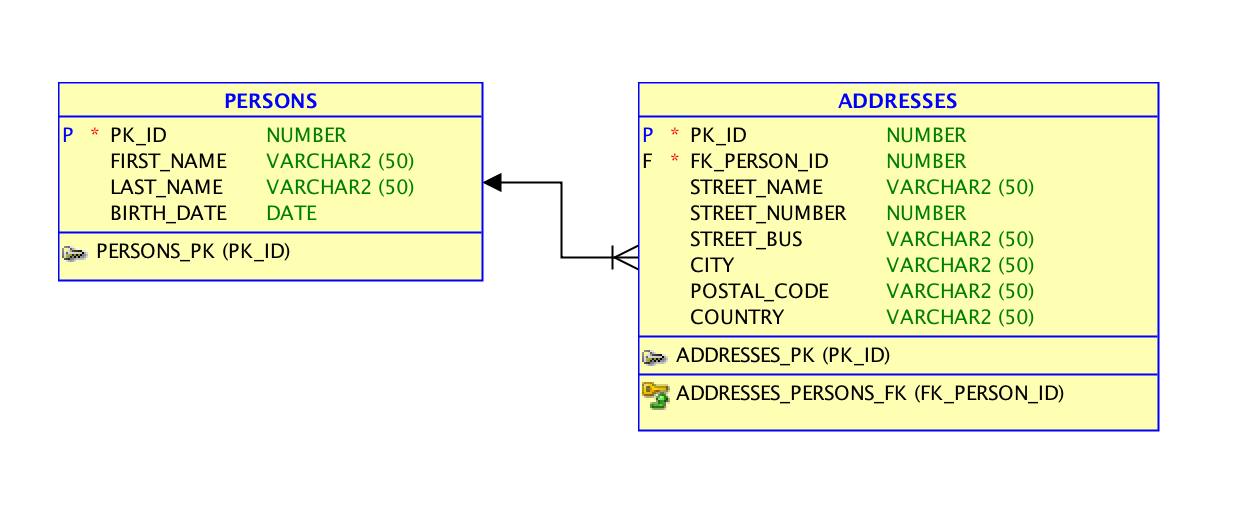
# Installing APEX application

# Installing supporting objects

# Sample Application

* Download the sampleproject template.
* Change the pom headers and the mainPom according to your liking.
* Install the mainPom with mvn clean install –f mainPom.xml

Lets’s assume following datamodel



Create for these tables the corresponding files in src/main/sql

**persons.sql**

create table persons(

pk\_id number primary key,

first\_name varchar2(50),

last\_name varchar2(50),

birth\_date date);

**addresses.sql**

create table addresses(

pk\_id number primary key,

fk\_person\_id references persons(pk\_id),

street\_name varchar2(50),

street\_number number,

street\_bus varchar2(50),

city varchar2(50),

postal\_code varchar2(50),

country varchar2(50));

Now you must add these scripts in the correct order to your master.xml in the src/main folder. Do this with following liquibase script between the databaseChangeLog tag:

<changeSet id="create\_persons" author="nico\_martens">

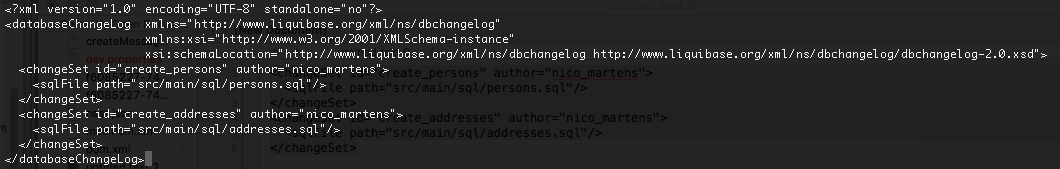
<sqlFile path="src/main/sql/persons.sql"/>

</changeSet>

<changeSet id="create\_addresses" author="nico\_martens">

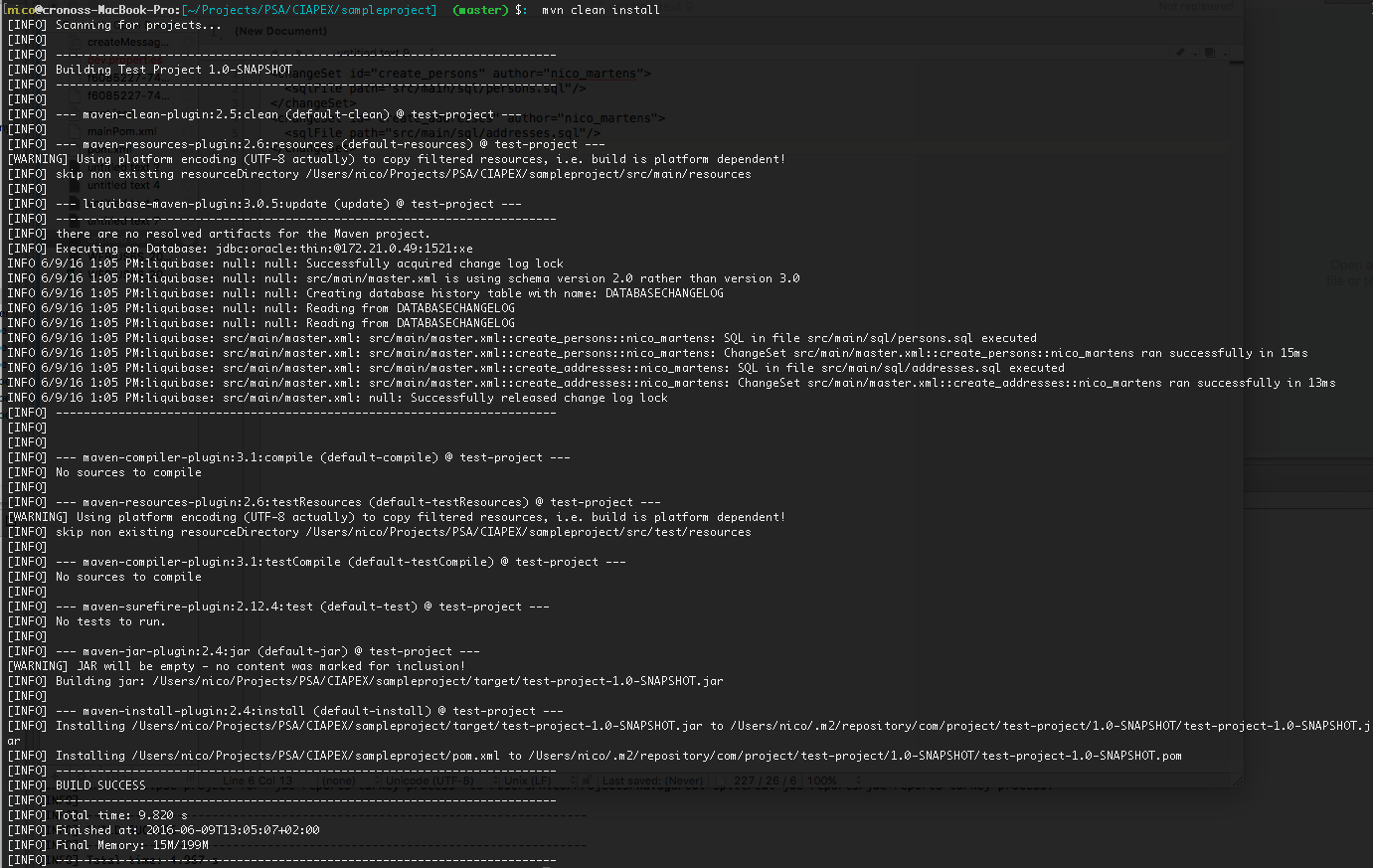
<sqlFile path="src/main/sql/addresses.sql"/>

</changeSet>



This changeSet will garantee that those create scripts will only be executed once in the build process.

Execute the maven build to install your scripts on the database with mvn clean install in the root folder of you sample project.



The tables should now be created in your database. Liquibase will remember what scripts you have executed so you can rerun your build with any problems.

Not all scripts are run once. You can add this as a parameter to your changeSet. We will see this in plsql code because you can run packages multiple time.

Now we create 2 more files for the sequences and 2 more for the triggers:

**seq\_persons.sql**

create sequence seq\_persons;

**seq\_addresses.sql**

create sequence seq\_addresses;

**trigger\_br\_i\_persons.sql**

create or replace trigger br\_i\_persons

before insert on persons

for each row

declare

l\_nextval pls\_integer;

begin

select seq\_persons.nextval

into l\_nextval

from dual;

:new.pk\_id := l\_nextval;

end br\_i\_persons;

**trigger\_br\_i\_addresses.sql**

create or replace trigger br\_i\_addresses

before insert on addresses

for each row

declare

l\_nextval pls\_integer;

begin

select seq\_addresses.nextval

into l\_nextval

from dual;

:new.pk\_id := l\_nextval;

end br\_i\_addresses;

For the sequences we will use the same changeSets. For the trigger we add runOnChange="true" and runAlways="true". For pl/sql you would also need the splitStatements="false" in the sqlFile tag. Else you would execute the code after the first semi colon.

**Master.xml**

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<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-3.0.xsd">

<changeSet id="create\_persons" author="nico\_martens">

<sqlFile path="src/main/sql/persons.sql"/>

</changeSet>

<changeSet id="create\_addresses" author="nico\_martens">

<sqlFile path="src/main/sql/addresses.sql"/>

</changeSet>

<changeSet id="create\_seq\_persons" author="nico\_martens">

<sqlFile path="src/main/sql/seq\_persons.sql"/>

</changeSet>

<changeSet id="create\_seq\_addresses" author="nico\_martens">

<sqlFile path="src/main/sql/seq\_addresses.sql"/>

</changeSet>

<changeSet id="trigger\_br\_i\_persons" author="nico\_martens" runOnChange="true" runAlways="true">

<sqlFile splitStatements="false" path="src/main/plsql/trigger\_br\_i\_persons.sql"/>

</changeSet>

<changeSet id="trigger\_br\_i\_addresses" author="nico\_martens" runOnChange="true" runAlways="true">

<sqlFile splitStatements="false" path="src/main/plsql/trigger\_br\_i\_addresses.sql"/>

</changeSet>

</databaseChangeLog>

Execute mvn clean install on the root. You should now have a valid table, sequence and trigger in a script that you can run as many times as you like.

These are the builds that will be executed on the build server to make CI possible.