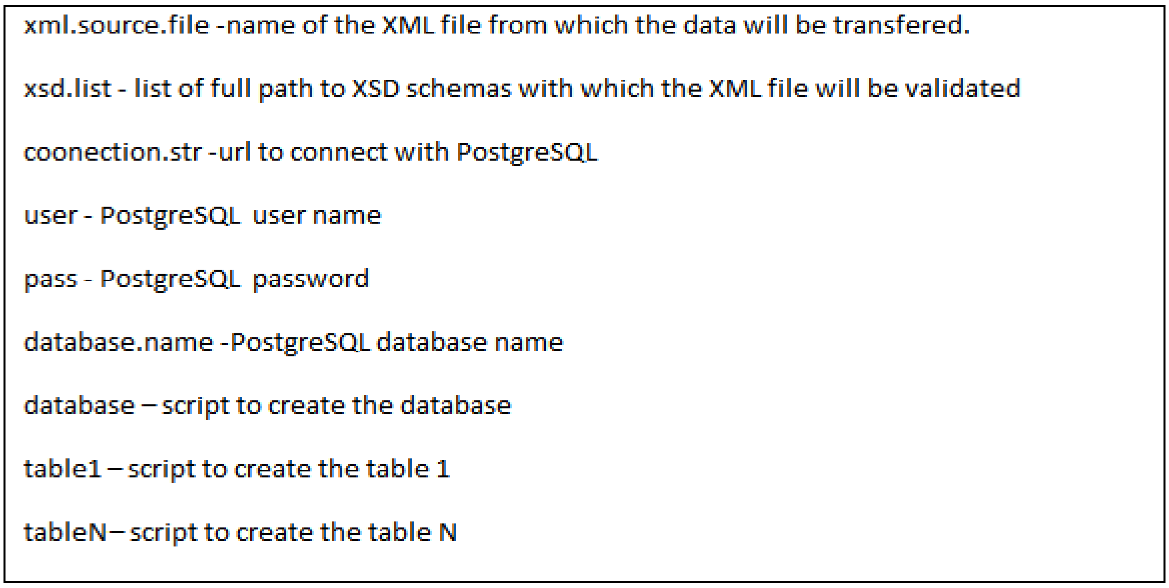
Data Migration Tool Documentation

The Data Migration Tool is designed to read data from *an XML file* and to import this data into a PostgreSQL. The tool can be used as a stand alone application.

Requirements

Data Migration Tool takes a file /init.properties/ as an input parameter. This property file has the following format.



xml.source.file -name of the XML file from which the data will be transfered.

xsd.list - list of full path to XSD schemas with which the XML file will be validated

coonection.str -url to connect with PostgreSQL

user - PostgreSQL user name

pass - PostgreSQL password

database.name -PostgreSQL database name

database – script to create the database

table1 – script to create the table 1

tableN– script to create the table N

When starting the tool, the database is created by the tool together with the necessary tables. Then the data transfer process starts.

Implementation

The tool has an API package "com.documaster.xml.analysis.api" in which the main interfaces are placed.

The implementation is located in the package "com.documaster.xml.analysis" and contains the implementation classes.

The input data from init.properties file are validated by the validate () method in the XmlMigrator class. Validation of the XML is performed by the Xserces library with validateXmlAgainstXsds( ) and parsing of the XML file is executed by the StaX library. The parsing of XML file is implemented in StAXParser class. There is an init() method in StAXParser class, that creates a Map from String and NodeProcessor classes. In this Map keps, wrapper classes have been created for each important Node of XML file.

The NodeProcessor class implements the CommandInterface interface. This interface sets command pattern to generate SQL requests and save the data in the database. This allows the code to be quite flexible when adding new objects, recording nodes and easily repairing SQL commands.

The NodeProcessor wrapper keeps for each node containing object a Map of child nodes and their values and a query with which they will be persisted in the database.

The execute() method from NodeProcessor completes the query with values and executes the request to persist data entry. The JDBCManager class connects to the database and persist the data record. With the method ctereateDataBase (),JDBCManager creates the database and the tables in it.

Build

The build performed with Maven and creates an exequtabal jar with all necessary libraries for starting the tool.

Used technologies

1.Java 8

2.Maven

3. StAX library

4. Xerces library

5. PostgreSQL library

Analysis of the XML file

The analysis of the XML file pointed out the following tags as containing information related to creation of the XML file - the time of file launching and the user who created it:

```
<systemID>c6812d9f-4149-403c-a925-64587..............</systemID>

<opprettetDato>2017-07-11T11:00:53</opprettetDato>

<opprettetAv>admin</opprettetAv>

<avsluttetDato>2017-07-11T11:00:54</avsluttetDato>

<avsluttetAv>admin</avsluttetAv>

<tilknyttetDato>2017-07-11T11:00:53</tilknyttetDato>

<tilknyttetAv>admin</tilknyttetAv>

<arkivertDato>2017-07-11T11:00:55</arkivertDato>

<opprettetDato>2017-07-11T11:00:54</opprettetDato>
```

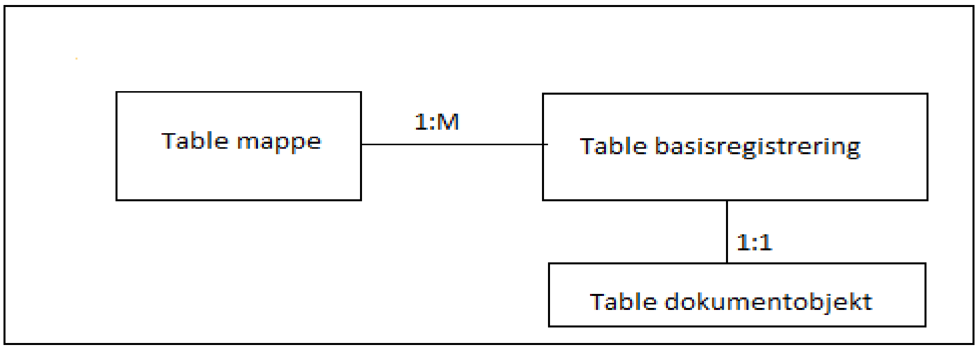
Moreover, these tags do not have a direct connection to the content of the data.

There is no information about the maximum size of the input XML file, but it is obvious that the data describe two tables / mappe, baseregistrering / with the following XML tags

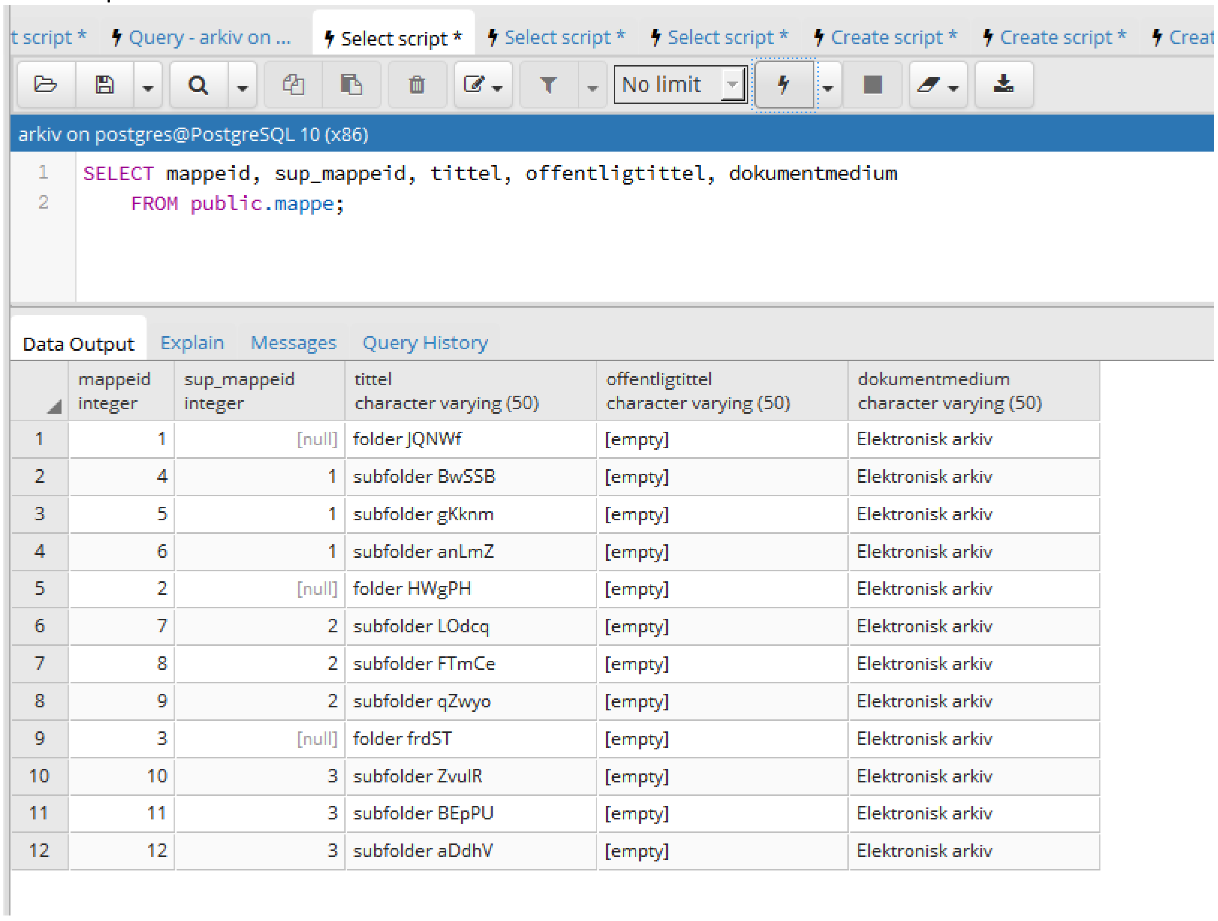
<mappe xsi:type="mappe">

<registrering xsi:type="basisregistrering">

The tag registrering may be divided into two tables baseregistrering and dokumentobjekt. Between the two tables / mappe and baseregistrering / has a 1:M relation, and between the baseregistrering and the dokumentobjekt the relation is 1: 1 as shown on the picture below.



The mappe table summarizes all mappe objects, and the relation between the super directories and sib directories is represented with the following mappeID and sup_mappeID columns displayed on the next picture:



```
SELECT mappeid, sup_mappeid, tittel, offentligtittel, dokumentmedium
    FROM public.mappe;
```

	mappeid integer	sup_mappeid integer	tittel character varying (50)	offentligtittel character varying (50)	dokumentmedium character varying (50)
1	1	[null]	folder JQNWf	[empty]	Elektronisk arkiv
2	4	1	subfolder BwSSB	[empty]	Elektronisk arkiv
3	5	1	subfolder gKknm	[empty]	Elektronisk arkiv
4	6	1	subfolder anLmZ	[empty]	Elektronisk arkiv
5	2	[null]	folder HWgPH	[empty]	Elektronisk arkiv
6	7	2	subfolder LOdcq	[empty]	Elektronisk arkiv
7	8	2	subfolder FTmCe	[empty]	Elektronisk arkiv
8	9	2	subfolder qZwyo	[empty]	Elektronisk arkiv
9	3	[null]	folder frdST	[empty]	Elektronisk arkiv
10	10	3	subfolder ZvulR	[empty]	Elektronisk arkiv
11	11	3	subfolder BEpPU	[empty]	Elektronisk arkiv
12	12	3	subfolder aDdhV	[empty]	Elektronisk arkiv

The sup_mappeID column are being added for the root mappe XML tag. The baseregistrering table contains the data from the tag registrering and has foreign key to the table mappe with primery key mappeID.
The table dokumentobjekt contains the data from the tag dokumentobjekt and has foreign key to the table baseregistrering with primery key registreringsid.

PostgreSQL: Server

The server from the development environment is initialized with the following command

```
initdb  -D "${ROOT_PATH}\PostgreSQL\10\data" --locale="Norwegian_Norway.1252"
```

this command sets:

-The database cluster will be initialized with locale "Norwegian_Norway.1252".

-The default database encoding has accordingly been set to "WIN1252".

-The default text search configuration will be set to "norwegian".

The tables are located in the ARKIV database, and it is created with the following parameters

```
ENCODING = 'WIN1252'

LC_COLLATE = 'Norwegian_Norway.1252'

LC_CTYPE = 'Norwegian_Norway.1252'
```

Indexes.

The optimal choice of indexes depends on the type of the performed searches.

It is possible the indexes created at the beginning are not to be effective enough which requires their correction or creation of new ones.

Table mappe

	Index Name	columns
1	mappe_idx	mappeid, sup_mappeid
2	public.mappe_idx_titel	tittel

Table basisregistrering

	Index Name	columns
1	basisregistrering_idx	registreringsid, mappeid
2	asisregistrering_idx_dok	dok_tittel
e	basisregistrering_idxt_titel	registrerings_tittel

Table dokumentobjekt

	Index Name	columns
1	dokumentobjekt_idx	id, registreringsid
2	dokumentobjekt_idxr	referansedokumentfil

Future improvements.

1. If the indexes turn out to be ineffective, they should be changed as needed.

2. If the current database model solution is not effective, the structure should be changed in order to remove the problem.

3.To validate the input data from init.properties file, use a JConsole framework.

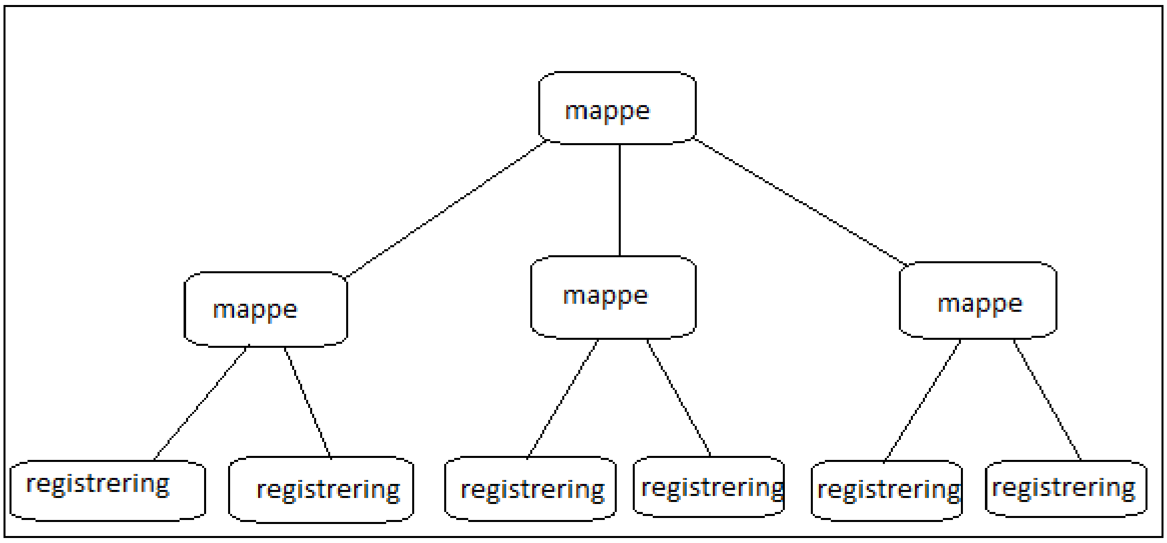
4.The Map of Nodeprocesor objects in StAXParser can be initialized from a proprietary file. The following sample file should have the following sample format.

```
nodeName=<node1>

insertQuery=<insert query1>

…………………………………………….

nodeName=<nodeN>

insertQuery=<insert queryN>
```

5. Creating indexes is initialized from a file and is executed automatically after the data is transferred.

6. The StAXParser class will accept parsing logic as a parameter.

One possible option is the Nodes of *the* XML file to be described with a tree structure for example:



, which contains the structure and the logic for the processing of the information in Nodes.

7. The current solution is working with one type of database. To be able to change the final database as a type.

8. Another possible upgrade – to be used a single connection to the database, different from the moment solution with new connection for every insert.

9. Another possible upgrade - to be used batches to save the data to the database (at the moment each insert has a separate request).