Automated metadata in PostgreSQL

Documentation (v.1.0 – 07/09/2012)

This document presents the main aspects related to the functionality of the automated metadata system in PostgreSQL and further developments that are needed. For more background information please refer to the internship report.

## General

Currently, the entire functionality is centred around the metadata table that contains several columns for each of the metadata elements and sometimes multiple fields for one metadata element as required (e.g. identifier has two sub-elements – code and namespace). For more details please see the internship report, specifically the Annex that describes the metadata standard, which is INSPIRE-based.

In addition, for the chain of triggers to start running, there is a need to insert in the metadata table a value for the dataset\_name field. To keep coherence, this value should be the name of the dataset (table) in the database. This can either be done manually in the SQL command line, or via the front-end QGIS plugin that was developed as part of this project. At the moment, the QGIS plugin supports the insertion of the resource\_abstract text, which is important for the language identifying module that populates the metadata\_language field, as well as lineage text.

## Look-up tables

At the moment there is one look-up table database\_groups, which holds contact details for the metadata contact and responsible organisation metadata elements. The table has to be manually filled with the contact details, while the system identifiers for the groups/users are automatically generated (IMPORTANT: when populating the group\_name column, the value has to be exactly the same with the name given to the group when initially created in the database).

## Permissions

Several permissions have to be granted at the moment for the created users/groups of the database.

* SELECT permission on the spatial\_ref\_sys table
* INSERT, UPDATE, DELETE on the metadata table
* all privileges on any existent SEQUENCE in case of serial id type (at the moment a sequence is created on the id field of the metadata table

## PERL (language identifying module)

The language identifying module that is used to fill in the metadata\_language can be found at <http://search.cpan.org/~ambs/Lingua-Identify-0.51/lib/Lingua/Identify.pm>

This module has to be installed on any new system/database, as well as the PERL procedural language

Although, not applied, this module can be used to identify the dataset attribute information language (i.e. resource\_language field). The only way to do that would be to concatenate together all text columns of a dataset/table and feed that into the Perl module as input (partly similar to keyword extraction – see below). The reason for that is that in this case there is no static text column (and long enough) like resource\_abstract in the metadata table. The input for the language identifying module is dynamic in this case with each different dataset.

## Current issues

First of all, there is a need to be able to automatically delete a metadata record from the metadata table, when a dataset (table) is deleted from the database. On the same event, the dynamic trigger functions that are created for each dataset when they are added to the database, have to be deleted as well (NOTE: the triggers themselves are deleted once the dataset/table is deleted).

The automatic generation of keywords has not been implemented yet, although different parts of the functionality, which would generate the keywords, has been developed. This is important because keywords are a mandatory metadata element. The different queries are attached in the Annex of this document and explained, being a matter of ‘gluing’ them together in a single trigger function.

There is still some functionality that needs to be added to the front-end QGIS plugin. First of all, it would be required to have a tick-box list of all the defined topic categories which the data can fall in. These categories can be found in the standard. The values have to be then inserted in the topic\_category field. This is also important as it is a mandatory element. Furthermore, it would require a drop-down list to select the coordinate system of the dataset to be imported, as at the moment the EPSG code of the coordinate system has to be inserted manually. It is also important to note that some trigger functions are written considering the default geometry column name for datasets is geom and not the\_geom for example. The plugin currently is default to the\_geom and this has to be changed every time to geom.

## Further (optional) metadata elements

There are few other metadata elements, which are not part of the standard, but were identified to be potentially useful. They come as an extension to the standard.

* Data rating – front-end functionality where user can rate quality of data (e.g. red, amber, green implemented as simple scale 1-2-3 via a lookup table or directly in the metadata table).
* Metadata rating – same principle as above
* Word cloud – based on keyword extraction methodology, but takes into account all words and their count in the attributes. Generated an image which can be attached in a separate metadata field and visualised in the front-end.
* Automatically create a copy of the dataset (zipped shapefile) when uploaded to the database and store on a given location on the server and generate access link. This link can then be inserted in the resource\_locator field, which is part of the metadata standard, but optional.
* A logging system for lineage
* Automatic identification of the coordinate system
* Spatial resolution (part of standard, but optional)
* Other elements as required by different projects – e.g. work package
* Standard elements that can be customised on a project basis: UCL originating controlled vocabulary for keywords, limitations on public access and conditions for access and use, etc.
* The other standard elements on ‘temporal reference’ – currently date of last revision implemented, as only one of the four is mandatory.

## Annex – Queries to generate keywords

### Query 1 – Keyword generation based on one column in one table

select string\_agg(keyword,',') as keyword from

(SELECT keyword, COUNT(keyword) from (select regexp\_split\_to\_table(name,E'\\s+') as keyword from uk\_counties\_kent) as keyword

group by keyword order by count desc limit 3) as keyword;

\*based on a table called ‘uk\_counties\_kent’ with a text column called ‘name’ >>> extract the top 3 occurring words

### Query 2 – Text columns selection from any table (input for query 3)

select (

select string\_agg(column\_name,' || '' '' || ') as column\_name from (

select column\_name from information\_schema.columns

where table\_name = 'uk\_counties\_kent' AND data\_type = 'character varying') as column\_name

) as query from uk\_counties\_kent limit 1

\*based on a table called ‘uk\_counties\_kent’ with a text column called ‘name’

### Query 3 – Concatenation of all text columns in a table

select --descript1 || ' ' ||

--type\_cod0 || ' ' ||

descript0 || ' ' ||

type\_code || ' ' ||

code || ' ' ||

file\_name || ' ' ||

descriptio || ' ' ||

area\_code || ' ' || name

as keyword from uk\_counties\_kent

\*based on a table called ‘uk\_counties\_kent’ with a text column called ‘name’

\*\*query 2 should generate the code required in Query 3 for any table. The problem is the output of Query 2 is a string, and in order to work as input for Query 3, it has to be transformed to regular code text when these are assembled together in a trigger function. The result is then provided as input to Query 1, which will aggregate everything, separate words by space, sort, and keep only the top occurring words.

IMPORTANT: - If any field of a text column is NULL, Query 3 will return NULL (that is why the first two columns are commented).

- Keywords will also have to be heavily filtered for acronyms, words like and, or, is, etc.