Deployment, management and exploitation of data warehouses with the R programming language

Author: Paul Taconet, French Institute for Research and Development (IRD)
2018-06-20

Introduction and rationale

The global increase in the volume of data collected, their heterogeneity, their complexity and the global demand for open and transparent data requires the implementation of tools to efficiently synthesize, anonymize, cross and disseminate these data in order to ease the extraction of information.

Data warehouses fill these needs: they are "central repositories of integrated data from one or more disparate sources. They store current and historical data in one single place that are used for creating analytical reports for workers throughout the enterprise "(Wikipedia definition).

We have set-up a methodology to deploy, load, update and exploit a data warehouse through the R programming language. The inputs of these scripts are mainly simple csv files. The R scripts that we have developed enable to i) deploy the physical model of the database, ii) manage the extraction - transformation - upload (ETL) process, iii) access the data in R, and iv) set-up the FAIR principles on the data stored on the DW, to improve their management (discovery, access, processing, information / visualization). The data warehouse model that we propose has got the following characteristics:

- Flexibility, meaning ability for the user to adapt the facts and dimensions of the data warehouse to his/her data ;
- Ability to store multiple reference data, including spatial reference data (managed through the PostGIS extension of PostgreSQL) ;
- Inclusion of a table dedicated to the metadata associated to each dataset loaded in the data warehouse.

The figure 1 provides a global overview of the workflow to setup and manage the data warehouse, from the extraction of the source datasets (heterogeneous, coming from multiple sources) to the management and exploitation of the data warehouse through the web services.

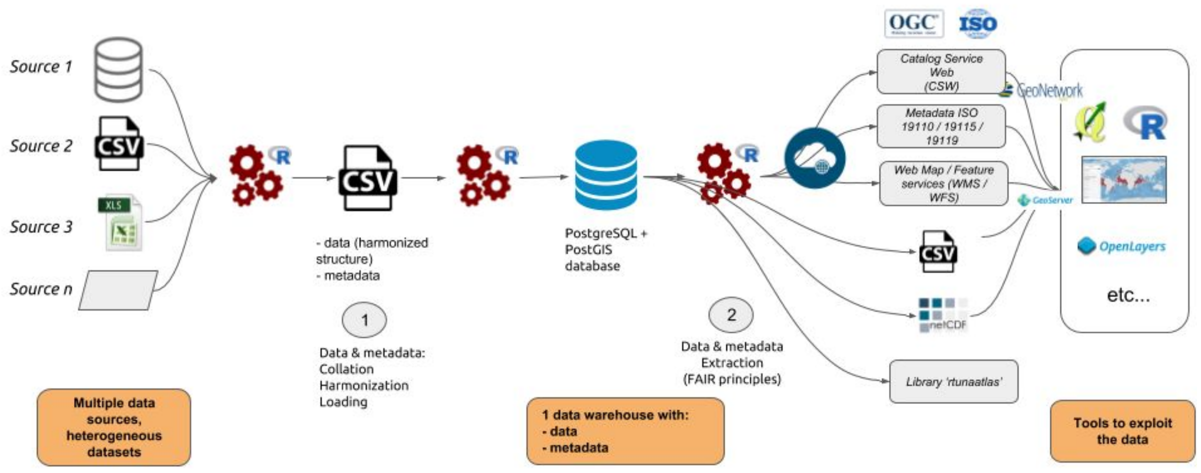

Source 1
Source 2
Source 3
Source n
Multiple data sources, heterogeneous datasets
- data (harmonized structure)
- metadata
1
Data & metadata: Collation Harmonization Loading
PostgreSQL + PostGIS database
1 data warehouse with:
- data
- metadata
2
Data & metadata Extraction (FAIR principles)
OGC
ISO
Catalog Service Web (CSW)
Metadata ISO 19110 / 19115 / 19119
Web Map / Feature services (WMS / WFS)
GeoNetwork
GeoServer
netCDF
Library 'rtunaatlas'
OpenLayers
etc...
Tools to exploit the data

***Figure 1**: Global overview of the workflow, from heterogeneous datasets to FAIR data services. Full-size figure available here.*

This documentation describes the data warehouse and the scripts that we have developed to manage and exploit it. The first section provides the conceptual, logical and physical data model of the database. The second section presents how to to deploy the data warehouse with the R scripts and load datasets. The last section presents the R scripts that enable to setup the FAIR services: the extraction from the data warehouse of metadata and data compliant with ISO/OGC standards.

1. The data warehouse

The main specifications for the data warehouse are the followings:

- Ability to store **3 kind of datasets**:
    - **Reference data** (also called *codelist*) : Reference data, i.e. code list, with codes, labels and any additional columns.
    - **Mapping between code lists** (also called *mapping*) : Correspondences between codes from two reference data. The mappings enable to express a dimension defined with a source reference data using another reference data (target).
    - **Multi-dimensional datasets** (also called *raw_dataset*) : The “core” of the DW. The datasets composed of a set of dimensions and one measure associated. The dimensions of the *raw_datasets* are expressed with the reference data stored in the DW (i.e. any raw_dataset that is loaded must have the reference data used in its columns already loaded in the DW).
- Ability to store **reference data with a spatial component** ;
- Ability to store **metadata** associated to each dataset loaded.

1.1. Conceptual data model

The data warehouse is organized around *facts*, *dimensions* and *metadata*. The *facts* are the variables measured. The *dimensions* are reference datasets that enable to express the facts: a fact table is composed of 1 to of ∞ dimension columns (1 column / dimension) and a column of measure (the variable measured). Finally, the *metadata* provide information on every dataset stored on the data warehouse, whether fact or reference data.

As an example :

- *Catch* is a fact. It provides the quantity of fish harvested.
- Each measure of catch (i.e. quantity of fish harvested) is defined over the *dimensions* that compose the catch: a flagging country, a fishing gear, a species, a time period, a spatial area, etc. For instance: France (dimension *country*), using purse seiners (dimension *gear*), in the year 2015 (dimension *time*), in the Atlantic Ocean (dimension *area*), has fished X tons (measure) of Albacore (dimension *species*).
- *Metadata* provide information about where the information on the catch is extracted from: who has generated the dataset? When? Who can we contact to get more information? etc.

The figure 2 presents the conceptual model of the data warehouse.

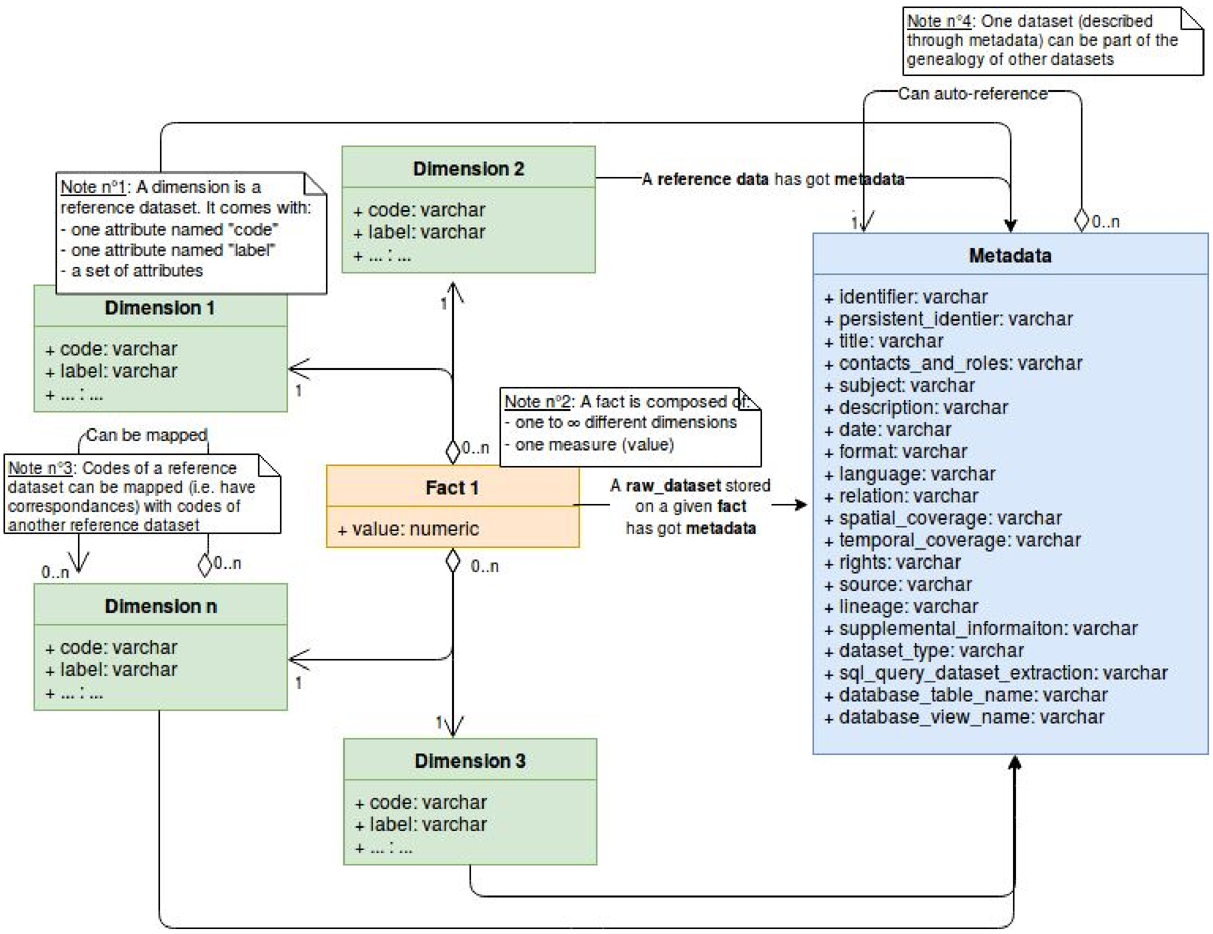

Note n°4: One dataset (described through metadata) can be part of the genealogy of other datasets
Can auto-reference
Dimension 2
+ code: varchar
+ label: varchar
+ ... : ...
A reference data has got metadata
1
0..n
Note n°1: A dimension is a reference dataset. It comes with:
- one attribute named "code"
- one attribute named "label"
- a set of attributes
Dimension 1
+ code: varchar
+ label: varchar
+ ... : ...
1
1
Note n°2: A fact is composed of:
- one to ∞ different dimensions
- one measure (value)
0..n
Can be mapped
Note n°3: Codes of a reference dataset can be mapped (i.e. have correspondances) with codes of another reference dataset
Fact 1
+ value: numeric
A raw_dataset stored on a given fact has got metadata
0..n
0..n
0..n
Dimension n
+ code: varchar
+ label: varchar
+ ... : ...
1
1
Dimension 3
+ code: varchar
+ label: varchar
+ ... : ...
Metadata
+ identifier: varchar
+ persistent_identier: varchar
+ title: varchar
+ contacts_and_roles: varchar
+ subject: varchar
+ description: varchar
+ date: varchar
+ format: varchar
+ language: varchar
+ relation: varchar
+ spatial_coverage: varchar
+ temporal_coverage: varchar
+ rights: varchar
+ source: varchar
+ lineage: varchar
+ supplemental_informaiton: varchar
+ dataset_type: varchar
+ sql_query_dataset_extraction: varchar
+ database_table_name: varchar
+ database_view_name: varchar

***Figure 2**: Conceptual model of the data warehouse - example with 1 fact and 4 dimensions*

It is noteworthy that:

- The DW is able to store 1 to ∞ facts ;
- The DW is able to store 1 to ∞ dimensions ;
- One fact can be linked to 1 to ∞ dimensions ;
- One dimension can be composed of 1 to ∞ reference datasets ;

1.2. Logical data model

In order to ease the visualization of the logical model, we provide it with 2 facts and 4 dimensions. In this example, each dimension is composed of 3 reference data. The fact 1 is composed of 4 dimensions (dimension 1, 2, 3, n) and the fact n is composed of 3 dimensions (dimension 1, 2, n). As noted in the previous section, remember that the DW can store 1 to ∞ facts and 1 to ∞ dimensions.

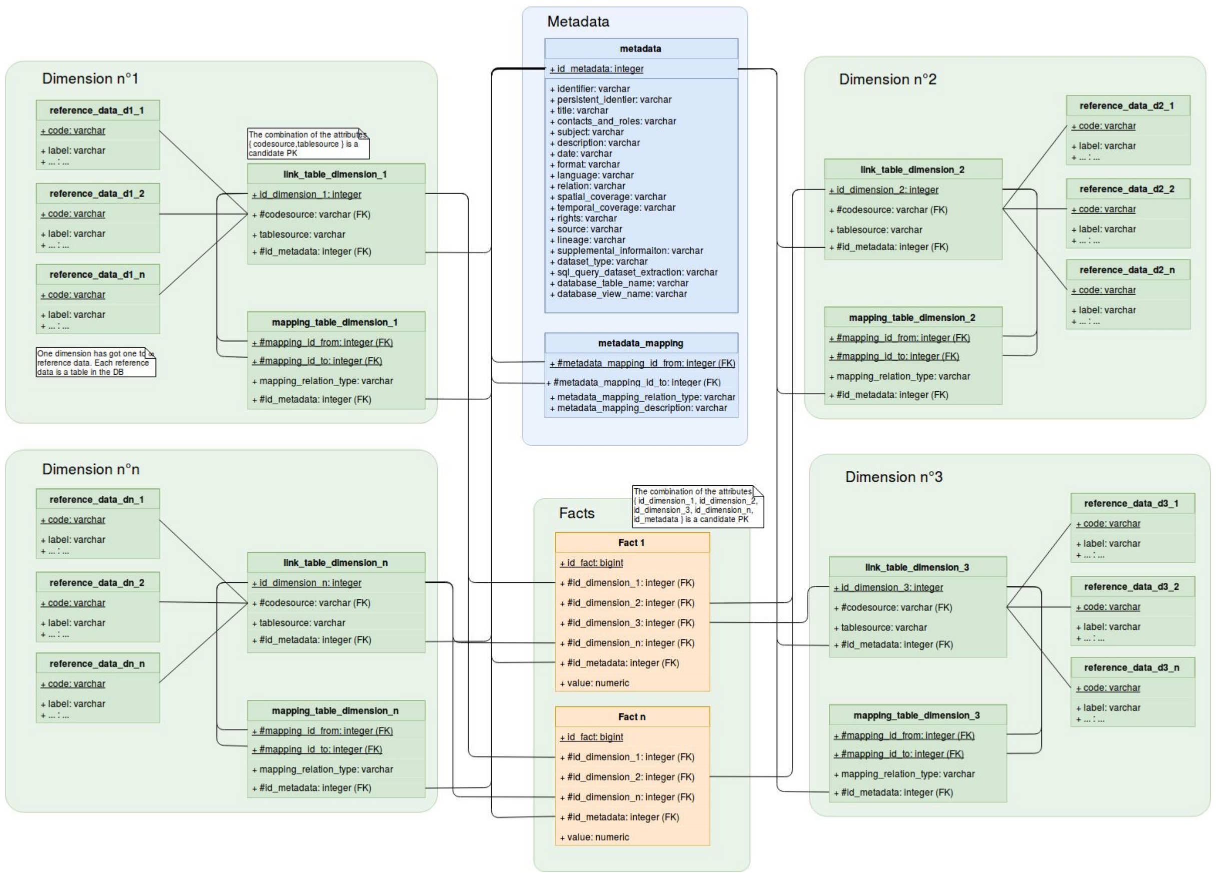

Metadata
metadata
+ id_metadata: integer
+ identifier: varchar
+ persistent_identier: varchar
+ title: varchar
+ contacts_and_roles: varchar
+ subject: varchar
+ description: varchar
+ date: varchar
+ format: varchar
+ language: varchar
+ relation: varchar
+ spatial_coverage: varchar
+ temporal_coverage: varchar
+ rights: varchar
+ source: varchar
+ lineage: varchar
+ supplemental_informaiton: varchar
+ dataset_type: varchar
+ sql_query_dataset_extraction: varchar
+ database_table_name: varchar
+ database_view_name: varchar
metadata_mapping
+ #metadata_mapping_id_from: integer (FK)
+ #metadata_mapping_id_to: integer (FK)
+ metadata_mapping_relation_type: varchar
+ metadata_mapping_description: varchar
Dimension n°1
reference_data_d1_1
+ code: varchar
+ label: varchar
+ ... : ...
reference_data_d1_2
+ code: varchar
+ label: varchar
+ ... : ...
reference_data_d1_n
+ code: varchar
+ label: varchar
+ ... : ...
One dimension has got one to ∞ reference data. Each reference data is a table in the DB
The combination of the attributes { codesource,tablesource } is a candidate PK
link_table_dimension_1
+ id_dimension_1: integer
+ #codesource: varchar (FK)
+ tablesource: varchar
+ #id_metadata: integer (FK)
mapping_table_dimension_1
+ #mapping_id_from: integer (FK)
+ #mapping_id_to: integer (FK)
+ mapping_relation_type: varchar
+ #id_metadata: integer (FK)
Dimension n°2
link_table_dimension_2
+ id_dimension_2: integer
+ #codesource: varchar (FK)
+ tablesource: varchar
+ #id_metadata: integer (FK)
mapping_table_dimension_2
+ #mapping_id_from: integer (FK)
+ #mapping_id_to: integer (FK)
+ mapping_relation_type: varchar
+ #id_metadata: integer (FK)
reference_data_d2_1
+ code: varchar
+ label: varchar
+ ... : ...
reference_data_d2_2
+ code: varchar
+ label: varchar
+ ... : ...
reference_data_d2_n
+ code: varchar
+ label: varchar
+ ... : ...
Dimension n°n
reference_data_dn_1
+ code: varchar
+ label: varchar
+ ... : ...
reference_data_dn_2
+ code: varchar
+ label: varchar
+ ... : ...
reference_data_dn_n
+ code: varchar
+ label: varchar
+ ... : ...
link_table_dimension_n
+ id_dimension_n: integer
+ #codesource: varchar (FK)
+ tablesource: varchar
+ #id_metadata: integer (FK)
mapping_table_dimension_n
+ #mapping_id_from: integer (FK)
+ #mapping_id_to: integer (FK)
+ mapping_relation_type: varchar
+ #id_metadata: integer (FK)
Facts
The combination of the attributes { id_dimension_1, id_dimension_2, id_dimension_3, id_dimension_n, id_metadata } is a candidate PK
Fact 1
+ id_fact: bigint
+ #id_dimension_1: integer (FK)
+ #id_dimension_2: integer (FK)
+ #id_dimension_3: integer (FK)
+ #id_dimension_n: integer (FK)
+ #id_metadata: integer (FK)
+ value: numeric
Fact n
+ id_fact: bigint
+ #id_dimension_1: integer (FK)
+ #id_dimension_2: integer (FK)
+ #id_dimension_n: integer (FK)
+ #id_metadata: integer (FK)
+ value: numeric
Dimension n°3
link_table_dimension_3
+ id_dimension_3: integer
+ #codesource: varchar (FK)
+ tablesource: varchar
+ #id_metadata: integer (FK)
mapping_table_dimension_3
+ #mapping_id_from: integer (FK)
+ #mapping_id_to: integer (FK)
+ mapping_relation_type: varchar
+ #id_metadata: integer (FK)
reference_data_d3_1
+ code: varchar
+ label: varchar
+ ... : ...
reference_data_d3_2
+ code: varchar
+ label: varchar
+ ... : ...
reference_data_d3_n
+ code: varchar
+ label: varchar
+ ... : ...

***Figure 3**: Logical model of the data warehouse - example with 2 facts and 4 dimensions. Full size here*

Details on the logical model :

The metadata

The metadata are composed of two tables:

- The table *metadata*. Each dataset available in the DW (whether raw dataset, reference data or mapping between code lists) is identified by a row in that table. The table *metadata* mainly has Dublin Core metadata, as well as some ISO/OGC 19115 metadata (e.g. spatial extent). The attribute *id_metadata* is the primary key. The attribute *identifier* is a candidate primary key.
- The table *metadata_mapping* enables to implement the genealogy. It explicits how the datasets are linked (i.e. which dataset(s) has (have) been used to generate one given dataset).

The dimensions

As specified in the conceptual data model, one fact is composed of i) a combination of dimensions and ii) one measure. The dimensions are reference datasets. In other words, the dimensions are physically represented in the DW by a set of tables that are reference datasets. One reference dataset belongs only to one dimension but one dimension can contain several (1 to ∞) reference datasets.

For instance, the global tuna atlas DW implemented with that methodology (see documentation here) has got, among others, a dimension named “species”. It contains many reference datasets for species, each coming with its own set of attributes. The DW stores the ASFIS list of species, as well as each of the reference datasets used by the tuna Regional

Fisheries Management Organizations. In total, there are seven reference datasets (i.e. seven tables) in the dimension “species” of that DW.

As presented in the logical data model above, the fact tables are not directly linked to the reference datasets. An intermediate table - called “*link_table*” in the schema - makes the link between the reference data and the fact tables. In the implementation of the model, that table is named after the dimension name. That table does not represent anything “real” - it has been built only to cope with some constraints that were set up for the DW. For each dimension, the link table is the union of all the primary keys (i.e. the codes) of all the reference tables of the dimension. Within the link table, a unique integer is assigned to each of these codes. The latter integer is used to store the data in the fact table for a given dimension, as a foreign key. The presence of the link table enables to store, within one single fact table, data that are expressed using multiple reference data for one given dimension.

For instance: in the global tuna atlas DW, “LLPB” is a code used in the ICCAT reference table for gears (table *gear_iccat*). In the link table of the dimension “gear” of the DW, that code was attributed the integer *37*. Hence, each raw dataset (stored in the fact tables) originally expressed with the code “LLPB” for gears will be stored as *37* in the gear column of the fact tables.

The DW also manages the storage of mappings (i.e. correspondences) between code lists. The mappings are stored in the table “mapping” of each dimension. The attribute *mapping_id_from* provides the source code, and the attribute *mapping_id_to* provides the target code of the mapping. The ids are the one of the link table.

<u>The facts</u>
In the DW, each fact is represented by one table. A fact table contains:

- 1 column by dimension that compose the fact. These columns are foreign keys to the *link table* of each dimension ;
- 1 column named *id_metadata*. This column is foreign key to the metadata table. It enables to get the metadata associated to the row ;
- 1 column named *value*. It provides the numerical measure of the fact, for the combination of dimensions in the row.

1.3. Physical data model

The data warehouse is implemented in PostgreSQL + PostGIS as follow:

- **1 to ∞ schemas “Dimension”**: Each dimension of the DW has got a dedicated schema whose name is the name of the dimension (e.g. “gear”, “species”, etc.). For each dimension, two tables are created when the DW is deployed :
  - The link table;
  - The mapping table.

  By default, 2 views are created in each dimension schema:
  - the view “*label*” provides the codes and labels (as character) for all the reference data of the dimension,
  - the view “*mapping_view*” provides the existing mappings (as character) in the dimension.
- **1 schema *fact_table*** : Contains the fact tables (e.g. catches, efforts, etc.).

- **1 schema *metadata*** : Contains the tables storing the metadata of the datasets loaded in the data warehouse (*metadata* and *metadata_mapping*).

It is noteworthy that the DW can store spatialized reference data through the PostGIS extension of the PostgreSQL database management system.

Annex 1 presents the dictionary of the tables of the DW (link tables and mapping tables of the dimension, metadata.metadata, metadata.metadata_mapping) as following :
- Annex 1.1 presents the dictionary of the tables ;
- Annex 1.2 presents the dictionary of the columns.

2. Deploy, load, update and exploit the data warehouse

The data warehouse is implemented as a PostgreSQL database with the PostGIS extension to manage the spatial reference datasets. A set of R scripts were developed to manage the DW, from its deployment on the database server to its exploitation.
We first present the R scripts to deploy the DW (section 2.1) and load datasets (section 2.2). We then present the workflow that enables to achieve both operations (deployment + load) within one single script (2.3). Then we present how to access the metadata and data available in the DW in R (2.4). Finally we present the R workflow that creates the FAIR principles (2.5).

2.1. Deploy the database on the server

The prerequisites to be able to deploy the data warehouse as a SQL database on a server are the followings:
- an empty PostgreSQL database (version 9.4.12) ;
- one administrator role ;
- one role with USAGE privileges ;
- the PostGIS extension enabled (version 2.3).

The DW can be deployed through a dedicated R function located here. The input parameters of the R function are described at its top. The R function uses a set of SQL scripts located here.

The dimensions to deploy on the DW, as well as the facts and the set of dimensions associated to each of them, are set in the function through dedicated input parameters (*db_dimensions* and *db_variables_and_associated_dimensions*). The names of the dimensions and the facts are free, with two exceptions: in case there is a time dimension it must be named “time”, and in case there is a spatial dimension it must be named “area” (this is due to the fact that the physical data model associated these two dimensions is slightly different than the one of the other “classical” dimensions).

Example: the following input parameterization :

db_dimension = "area,catchtype,unit,flag,gear,schooltype,sex,sizeclass,species,time,source" *db_variables_and_associated_dimensions* = *"catch=schooltype,species,time,area,gear,flag,catchtype,unit,source@effort=schooltype,time,area,gear,flag,unit,source@catch_at_size=schooltype,species,time,area,gear,flag,catchtype,sex,unit,sizeclass,source"*

will deploy a DW with:

- The following dimensions: area, catchtype, unit, flag, gear, schooltype, sex, sizeclass, species, time, source (1 schema / dimension) ;
- The following facts : catch, effort, catch_at_size (1 table / fact in the schema *fact_tables*)
- The following list of dimensions associated to each fact :
    - Catch = schooltype, species, time, area, gear, flag, catchtype, unit, source
    - Effort = schooltype, time, area, gear, flag, unit, source
    - Catch at size = schooltype, species, time, area, gear, flag, catchtype, sex, unit, sizeclass, source

2.2. Extract - Transform - Load datasets in the data warehouse

As previously mentioned, three types of datasets can be loaded in the DW: reference data, mapping between code lists and raw datasets.

Three R functions have been developed to load each of these types of datasets. The functions are available and described here. They take as input the following parameters :

- *con*: a wrapper of rpostgresql connection (connection to the DW with administration rights) ;
- *df_to_load*: data.frame of the dataset to load, with constraints on the structure and the column names depending on the type of dataset ;
- *df_metadata*: data.frame of the metadata associated to the dataset, with constraints on the structure (columns names and types) ;
- *df_codelists_input* (for *raw_dataset* only): in case of a *raw_dataset*, a data.frame of the identifiers of the reference data (*codelist*) associated to each column of the *df_to_load*

The constraints to load datasets in the DW are the followings:

- For each dataset (whether *codelist*, *mapping* or *raw_dataset*) the metadata must also be loaded ;
- In case of a *raw_dataset*, the reference data (*codelists*) associated to each column must be available in the DW

The structure of the datasets to load (columns names and types) are specified in the section “details” of the description of the function.

Two cases can be distinguished depending on the status of the dataset to load :

- **Case a)**: the dataset to load is already properly structured (i.e. the structure that is expected as input of the DW) and available as CSV ;
- **Case b)**: the dataset to load does not exist or is not properly structured; however, a R script has been developed to generate it.

A set of R functions manage the ETL process. These functions enable to extract/transform and load datasets in the DW iteratively. The main input parameter of the functions is a CSV file named "*metadata and parameterization file*" in this documentation. The metadata and parameterization file contains all the information useful to generate and load the datasets in the DW. Each row in the metadata and parameterization file represents one dataset to load in the database. The columns of the metadata and parameterization file can be divided into two categories :

- The metadata columns ;
- The parameterization columns.

The metadata columns provide the metadata of the dataset. They will be loaded in the metadata table of the database (title, description, contacts, etc.). In case the data is generated during the ETL process (case b), some facilities exist to automatically fill-in some metadata in function of the output of the script that generates the dataset (see annex 4).

The parameterization columns must be added by the user. They provide information to either locate (physically) the dataset to load (case a), or to generate it (case b), and to load it in the database. The columns to add to the metadata table depend on the cases:

- **For the case a)** (dataset already properly structured):
    - a column named *'path_to_dataset'* provides the path/URL where the dataset is stored.
- **For the case b)** (dataset to generate through a R script):
    - a column named '*path_to_script_dataset_generation*' provides the path/URL of the R script to execute to generate the dataset ;
    - a set of columns provide the values of each parameter of the latter R script to generate the dataset. The columns are named after the concatenation of the character string "parameter_" and the parameter names of the script.

  In other words, the dataset will be generated by executing the R script whose location is set in the column '*path_to_script_dataset_generation*' with the parameterization provided in the next columns.

The parameterization columns also include a column named '*path_to_codelists_used_in_dataset*'. This column must be filled with the path/URL of a csv file that contains the identifiers of the code lists used in the dimensions of the dataset to load. This column must be filled-in only for the *raw_datasets* (i.e. not the code lists and mappings, since by definition code lists and mapping do not have dimensions). The column can also be set to NA: in that case, the data.frame is extracted or generated elsewhere on the workflow, e.g. on the script that generates the dataset.

Note: Management of the spatial component of the DW

The spatial component of the DW is managed through the dimension named "area". In term of physical model, this dimension is strictly equivalent to any other dimension of the DW. In the dimension "area", the code lists might contain a PostGIS geometry column. When a dataset with a spatial component is loaded, two cases can be distinguished:

- The dataset is linked to a spatial reference data already loaded in the DW (as for the other dimensions): in that case, as for the other dimensions, the identifier of the code list must be specified in the *df_codelists_input* for the dimension "area";
- The spatial component of the dataset to load is a column in the Well Known Text (WKT) format. In that case, the identifier of the code list specified in the

df_codelists_input for the dimension “area” must be “area_wkt”. That table is created automatically in the DW when the physical model is deployed and can contain any WKT.

Annex 2 provides the dictionary of the metadata and parameterization file. Example of filled-in files can be found here:

- For case a) (dataset already properly structured): Reference data used in the global tuna atlas
- For case b) (dataset to generate before loading): Source datasets used in the global tuna atlas

Annex 3 is an activity diagram presenting how to fill-in the metadata and parameterization file, provided one or more datasets to load and the case for the status of the dataset(s) to load (a or b).

Annex 4 provides important and useful information to fill-in properly the metadata and parameterization file

Once the metadata and parameterization file properly filled-in, the scripts to (eventually generate and) load the dataset(s) in the DW can be executed. These functions mainly take as input information on the DB connection (argument “*con_parameters*”) and the URL/path to the metadata and parameterization file (argument “*metadata_and_parameterization*”). They are located here:

- Function to load dataset(s) (already properly structured and stored as csv file): here
- Function to generate and load dataset(s): here

2.3. All-in-one: workflow to deploy the data warehouse and load the datasets

3. Locate and access the datasets stored in the data warehouse

There are several ways to locate and access the datasets stored in the data warehouse.

3.1. Locate and access the datasets via the SQL database

Datasets can be accessed directly in the SQL database. The table *metadata.metadata* provides the metadata for all the datasets available in the database. Each row is a dataset available in the DW. Annex n°1.2 provides the dictionary of the metadata table. The column “sql_query_dataset_extraction” of the metadata table provides the SQL query to run to retrieve the dataset. The datasets for which the column “database_view_name” is not null are also available through views or materialized views in the database - their name is specified on that column.

3.2. Locate and access the datasets via *R*

The “rtunaatlas” package provides functions to locate and access the datasets stored in the DW. Among others:

- list_metadata_datasets: returns the metadata of the datasets available in the DW as data.frame (essentially the table metadata.metadata) ;
- extract_dataset: return a dataset as data.frame ;
- get_codelist_of_dimension: for given raw_dataset and dimension, returns the complete reference dataset used

Please refer the documentation of the package for additional information.

4. Deploy the FAIR services

5. Current limitations

Annexes

Annex 1: Dictionaries of the data warehouse

We provide the data dictionaries of the tables that are deployed by default when the DW is deployed. The fact tables and the reference datasets are by definition defined / provided by the user, hence no data dictionary can be provided in this documentation for these elements.

Annex 1.1: Dictionary of the tables

schema	**table**	**description**
metadata	metadata	This table contains the metadata associated to the dataset stored in the data warehouse. One row provides the metadata for one dataset. The metadata are mainly Dublin Core metadata extended with ISO 19115 metadata (for datasets with a spatial component)
metadata	metadata_mapping	This table enables to establish relationships between the datasets stored in the data warehouse (e.g. "was used to generate", "is a reference dataset", etc.
<dimension>	<dimension>	This table exists for each dimension (1 table by dimension available in the DW). It gathers all the primary keys (i.e. codes) of all the reference data of the dimension. It enables to make the link between the reference data and the fact tables that use these reference data (see DW documentation for more details).

<dimension>	<dimension_mapping>	This table exists for each dimension (1 table by dimension available in the DW). It stores the eventual mappings (i.e. correspondences) between code lists belonging to the dimension.

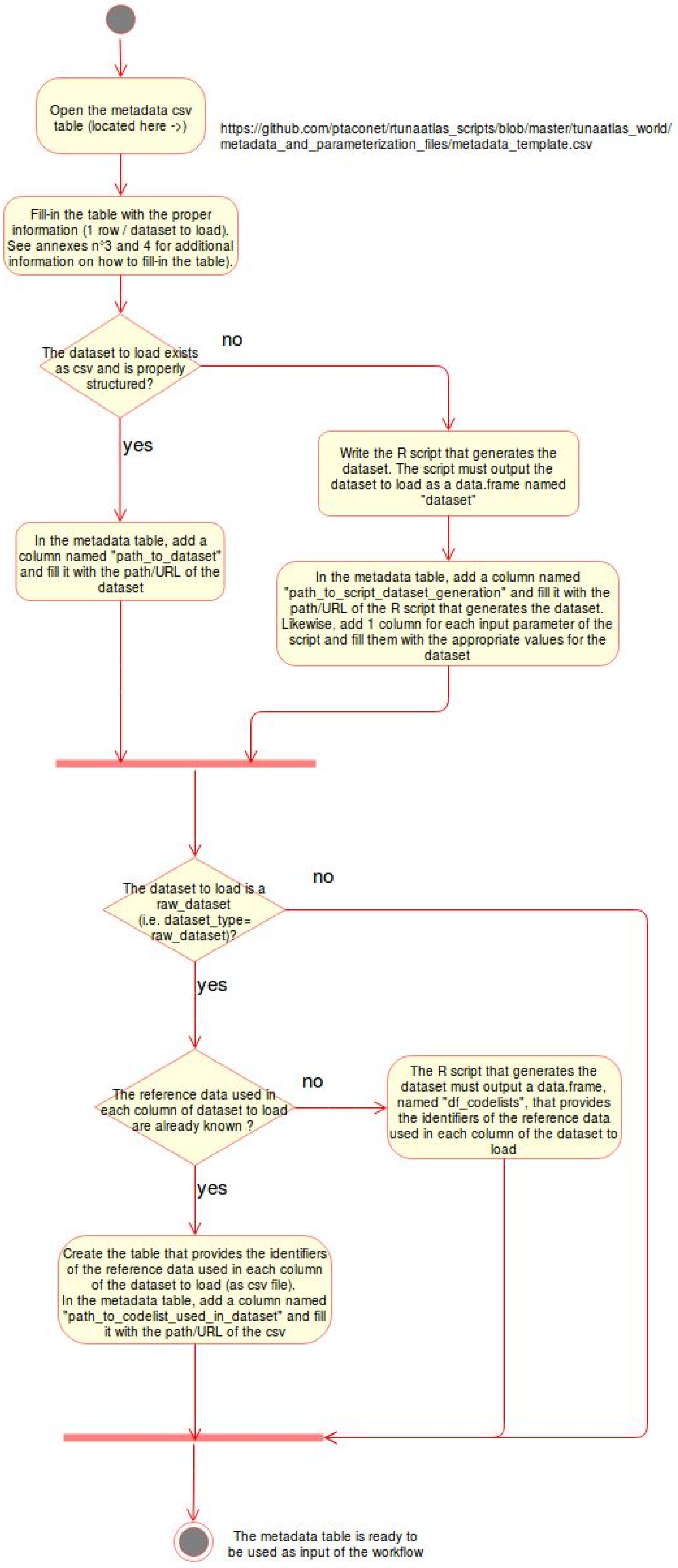
Annex 1.2: Dictionary of the columns

schema	table	field	type	definition	condition
metadata	metadata	id_metadata	integer	Primary key Technical numerical identifier of the dataset in the DW. Automatically generated when a new dataset is stored.	M
metadata	metadata	identifier	text	candidate primary key Text identifier of the dataset in the DW. The text identifier usually follows a nomenclature.	M
metadata	metadata	persistent_identifier	text	Persistent identifier of the dataset in the data warehouse. Identifier and persistent identifier can be different. For instance, a time covering the time frame t0 to t1 might have a given identifier, and the same time series covering the time frame t0 to t2 (e.g. the same time series with 1 additional year of data) might have the same persistent_identifier - both time series represent the same thing - must different identifiers - they are not the same time series.	M
metadata	metadata	title	text	Title of the dataset	O
metadata	metadata	contacts_and_roles	text	Contacts associated to the dataset. Syntax: contact_role=email_adress_of_the_contact . If multiple contacts, separate by a semicolon character (";"). Example: owner=ird@ird.fr;originator=ob7@ird.fr;metadata=paul.taconet@ird.fr;metadata=chloe.dalleau@ird.fr;	O
metadata	metadata	subject	text	List of keywords associated to the dataset. Syntax: THESAURUS_NAME = list_of_keywords_separated_by_commas . If multiple thesaurus, separate by a semicolon character (";") Example: COLUMNS = code, label, catchunitfr ;	O
metadata	metadata	description	text	Description of the dataset	O
metadata	metadata	date	text	Dates associated to the dataset. Syntax: date_type=date_format_YYYY_MM_DD . If multiple dates, separate by a semicolon	O

				character (";"). Example: publication=2018-02-21;download=2018-04-18;	
metadata	metadata	format	text	Format of the dataset	O
metadata	metadata	language	text	Language of the dataset	O
metadata	metadata	relation	text	List of useful URLs associated to the dataset. Syntax: url_type@url . If multiple URLs, separate by a semicolon character (";"). Example: source_download@http://www.marineregions.org/downloads.php;source_metadata@http://marineregions.org/sources.php;	O
metadata	metadata	spatial_coverage	text	Spatial coverage (bounding box) of the dataset in WKT format. Example: POLYGON((-31 -26,-31 29,104 29,104 -26,-31 -26))	O
metadata	metadata	temporal_coverage	text	Temporal coverage of the dataset. Syntax: *start=starting_time;end=ending_time* Example: start=1981-01-01 00:00:00Z;end=2017-12-31 00:00:00Z	O
metadata	metadata	rights	text	Rights associated to the dataset	O
metadata	metadata	source	text	Source institution that provides the dataset	O
metadata	metadata	lineage	text	Steps that have been achieved to generate the dataset. Syntax: *step n : description_of_the_step* . n is the step number (e.g. 1, 2, etc.)	O
metadata	metadata	supplemental_information	text	Any useful supplemental information	O
metadata	metadata	dataset_type	text	Type of dataset. Three types are currently accepted : - raw_dataset - codelist - mapping	M
metadata	metadata	sql_query_dataset_extraction	text	SQL query (type SELECT) to extract the dataset	O
metadata	metadata	database_table_name	text	Table useful only for DB management purposes. Name of the table in the database under which the dataset will be stored	M
metadata	metadata	database_view_name	text	Table useful only for DB management purposes. Name of the view in the database under which the dataset will be directly accessible with codes and labels. If empty, no view does exist for the dataset. It is possible to extract the dataset with the query stored under the column	O

				"sql_query_dataset_extraction".	
metadata	metadata_mapping	metadata_mapping_id_from	integer	Foreign key - References the table metadata. id_metadata (from the table metadata) of the source dataset	M
metadata	metadata_mapping	metadata_mapping_id_to	integer	Foreign key - References the table metadata. id_metadata (from the table metadata) of the target dataset	M
metadata	metadata_mapping	metadata_mapping_relation_type	text	Type of relation that exists between the source dataset and the target dataset (e.g. "is input of")	O
metadata	metadata_mapping	metadata_mapping_description	text	Description of the type of relation that exists between the source dataset and the target dataset	O
<dimension>	<dimension>	id_<dimension>	integer	Primary key Technical numerical identifier for each code used in the reference table of the dimension.	M
<dimension>	<dimension>	codesource_<dimension>	text	Conceptually references each reference table of the dimension (the reference is not technically implemented in the physical model). Code of the reference table.	M
<dimension>	<dimension>	tablesource_<dimension>	text	Identifier of the table that contains the code	M
<dimension>	<dimension>	id_metadata	integer	Foreign key - References the table metadata. Metadata of the reference table that contains the code	M
<dimension>	<dimension_mapping>	<dimension>_mapping_id_from	integer	Foreign key - References the table <dimension> id (from the table <dimension>) of the source dataset	M
<dimension>	<dimension_mapping>	<dimension>_mapping_id_to	integer	Foreign key - References the table <dimension> id (from the table <dimension>) of the target dataset	M
<dimension>	<dimension_mapping>	<dimension>_mapping_relation_type	text	Type of relation that exists between the source code and the target code (e.g. is strictly equivalent to, is part of, etc.)	M
<dimension>	<dimension_mapping>	id_metadata	integer	Foreign key - References the table metadata. Metadata of the mapping	M

Annex 2: Activity diagram: how to fill-in the metadata and parameterization file to load one or more datasets in the data warehouse


Open the metadata csv table (located here ->)
https://github.com/ptaconet/rtunaatlas_scripts/blob/master/tunaatlas_world/metadata_and_parameterization_files/metadata_template.csv
Fill-in the table with the proper information (1 row / dataset to load). See annexes n°3 and 4 for additional information on how to fill-in the table).
The dataset to load exists as csv and is properly structured?
no
yes
Write the R script that generates the dataset. The script must output the dataset to load as a data.frame named "dataset"
In the metadata table, add a column named "path_to_dataset" and fill it with the path/URL of the dataset
In the metadata table, add a column named "path_to_script_dataset_generation" and fill it with the path/URL of the R script that generates the dataset. Likewise, add 1 column for each input parameter of the script and fill them with the appropriate values for the dataset
The dataset to load is a raw_dataset (i.e. dataset_type= raw_dataset)?
no
yes
The reference data used in each column of dataset to load are already known ?
no
The R script that generates the dataset must output a data.frame, named "df_codelists", that provides the identifiers of the reference data used in each column of the dataset to load
yes
Create the table that provides the identifiers of the reference data used in each column of the dataset to load (as csv file). In the metadata table, add a column named "path_to_codelist_used_in_dataset" and fill it with the path/URL of the csv
The metadata table is ready to be used as input of the workflow

Annex 3: Dictionary of the metadata and parameterization file (only for the metadata columns)

Example of such filled-in tables can be found here:
- For case a) (dataset already properly structured): Reference data used in the global tuna atlas
- For case b) (dataset to generate before loading): Source datasets used in the global tuna atlas

Column	Mandatory	Definition
identifier	Yes	Identifier of the dataset in the DB
persistent_identifier	Yes	Persistent identifier of the dataset in the DB
dataset_type	Yes	Type of dataset. Three types are accepted: - raw_dataset - codelist - mapping
title	Yes	Title of the dataset
contact_owner	No	E-mail address of the party that owns the dataset. If multiple, separated by “;”
contact_originator	No	E-mail address of the party who created the dataset. If multiple, separated by “;”
contact_metadata	No	E-mail address metadata. If multiple, separated by “;”
contact_PointOfContact	No	E-mail address of the party who can be contacted for acquiring knowledge about or acquisition of the dataset. If multiple, separated by “;”
contact_PrincipalInvestigator	No	E-mail address of the key party responsible for gathering information and conducting research. If multiple, separated by “;”
contact_publisher	No	E-mail address of the party who published the resource. If multiple, separated by “;”
contact_processor	No	E-mail address of the party who has processed the data in a manner such that the resource has been modified. If multiple, separated by “;”
contact_data_structure_definition	No	E-mail address for data structure definition. If multiple, separated by “;”

subject	No	Keywords specific to the dataset. Some keywords will be calculated automatically (e.g. list of dimensions available)
description	No	Description of the dataset
date_publication	No	Date the originator (e.g. RFMO) released the dataset. Format: YYYY-MM-DD
date_download	No	Date the dataset was downloaded from the originator's website, or sent by the data manager. Format: YYYY-MM-DD
format	No	
language	No	
relation_source_download	No	URL of the web page where the source dataset was downloaded
relation_source_dataset	No	URL of the source dataset itself stored on the originator's (e.g. RFMO) website
relation_source_metadata	No	URL where the metadata for the dataset were extracted from
rights	No	Legal rights associated to the dataset
source	No	Originator of the dataset (e.g. RFMO's acronym)
lineage	No	Description of the processes applied to generate the dataset. The structure must be as follow: step1: [text] step2: [text] ...
supplemental_information	No	Any useful supplemental information not stated in the other metadata elements
database_table_name	Yes	Name of the table in the database under which the dataset will be stored
database_view_name	No	Name of the view in the database under which the dataset will be directly accessible with codes and labels. Works only for the *raw_dataset* type. If set to NA, no view will be created for the dataset.

Annex 4: Useful information to properly fill-in the metadata and parameterization file

The user can add as many *contact*, *relation* and *date* columns as he needs. In the ETL process, these columns will be concatenated. To add additional columns, the syntax is: [metadataElement]_[role]:

- [metadataElement] is one of {contact, relation, date}.
- [role] is the associated role name. There is no constraint for the role name.

Some examples : date_publication, contact_owner, contact_publisher, relation_source_dataset.

In case the dataset is generated through a R script (case b), the following information are useful:

- The R script that generates the dataset must return an object of type data.frame properly structured, i.e. the structure expected in input of the DW. That data.frame must be named "*dataset"*.
- The R script might also generate more than one dataset to load. In that case, the script must return an object of type list containing the datasets to load properly structured (each element in the list is one dataset to load). The list must be named "*dataset*".
- Some metadata can be generated dynamically in the R script that generates the dataset (e.g. starting and ending dates, etc.). For this to work:
    - The metadata that will be generated automatically must be indicated in the metadata table. There are two options:
        - The dynamic metadata generated are pasted in one of the column of the metadata table, where a text is already existing. In that case, the user should write the character string *@@automatically_generated@@* in the metadata column.
        - The dynamic metadata generated replaces characters in the metadata table, where a text is already existing. In that case, the user should write the text to replace framed by the percentage ("%") character (e.g. *%time_start%*)
    - The R script that generates the dataset must output an object of type list named "*additional_metadata*". These metadata will be included in the metadata of the dataset in the DW. For each text framed by the percentage character in the metadata table, there must be a corresponding element of the list *additional_metadata* with the same name. Example: if the user writes *%time_start%* somewhere in the metadata and parameterization file, then the list additional_metadata must have an element named *time_start*. The value contained in the latter element will replace all the occurrences of *%time_start%* in the metadata and parameterization file.
      If there are multiple datasets generated through the R script, there must be the same number of lists of additional metadata (each of these lists being one element in the list *additional_metadata*).