DATA BASE CONSTRUCTION

Work Package Title	Mapping and promoting Roman network of routes and settlements
Activity No. and Title	A.T3.2 Development of a GIS-based territorial Atlas of Roman routes legacy in DR
Deliverable	Data base construction – Deliverable D.T 3.2.2
Deliverable Responsible	ERDF PP8 Geodetic institute of Slovenia
Main Authors & Partner Acronyms	Marina Lovrić, Tomaž Žagar, Blaž Barborič PP8 (GI)
Co-Authors & Partner Acronyms	
Reviewed by: Name and Partner Acronyms	
Status	**Final (F)**
	Revised draft (RV)
	Final (F)
Length	5 pages

Project co-funded by European Union funds (ERDF, IPA)

Table of contents

Database construction 3
Conceptual model 3
Physical model - database object relations 4

Database construction

The Atlas database is constructed by merging data collected through the Catalogue delivered in WP_T1 and enriched with additional data from WP_T3, activity T3.1.

Conceptual model

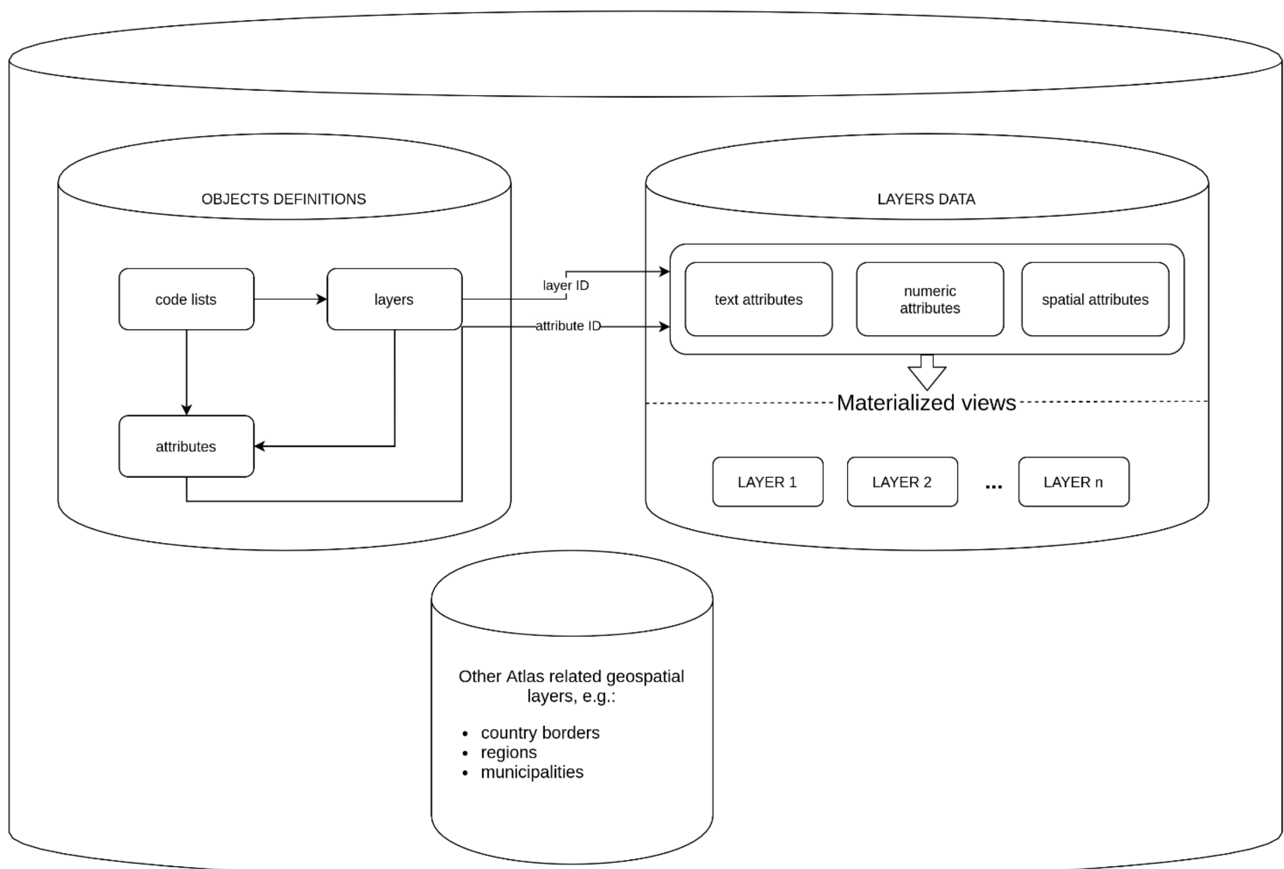
The database engine used for serving The GIS Atlas of Roman routes is PostgreSQL, which is a powerful, open source object-relational database – however – the Atlas database schema is constructed as a combination of two models:

- relational database objects, used for storing the layer definition data and code lists
- entity attribute value (EAV) model, used for storing individual layer's data

Entity–attribute–value model (EAV) is a data model to encode, in a space-efficient manner, entities where the number of attributes (properties, parameters) that can be used to describe them is potentially vast, but the number that will actually apply to a given entity is relatively modest (https://en.wikipedia.org/wiki/Entity-attribute-value_model).

EAV model was used to optimize the data storage, because the Atlas comprises layers which have substantial content differences and variable number of more or less unrelated attributes.

The conceptual schema is depicted in the following diagram:



The database is using PostGIS extension which enables efficient storage and work with geospatial data. Three multipart spatial data types are used:

- multipoints
- multilines
- multipolygons

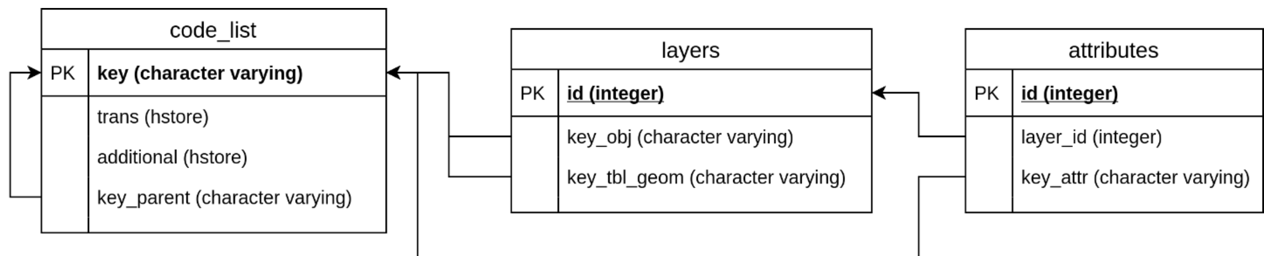
In this way we can easily assign one attribute to several points, lines or polygons.

After layers data is inserted or updated there are procedures implemented which build materialized views behaving pretty much like classical geospatial layers which can easily be queried or (spatially) spatially joined with other related geospatial layers in a relational database way.

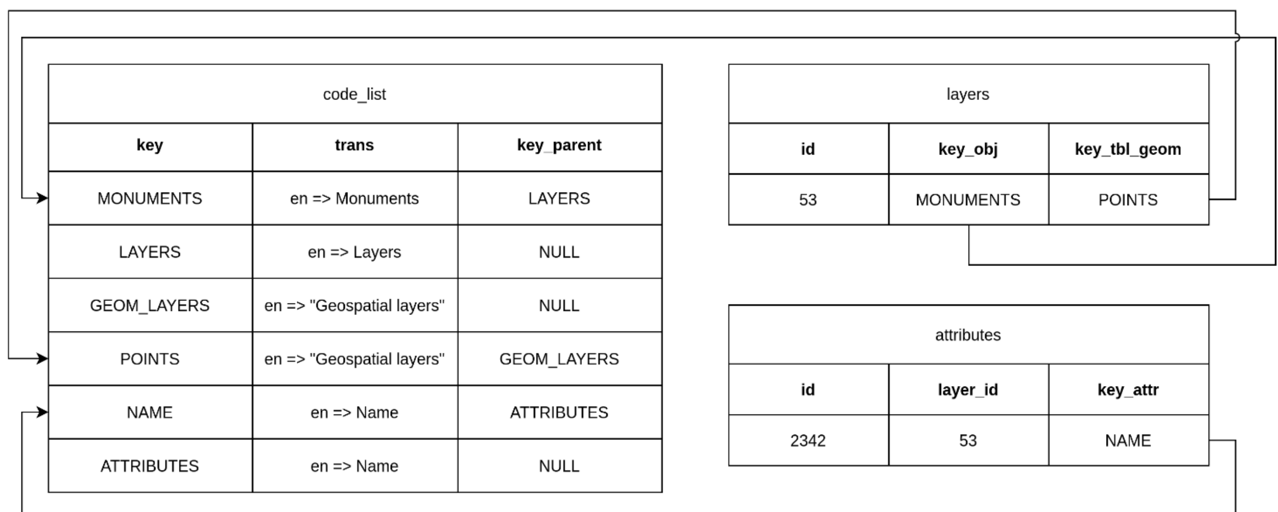
Physical model - database object relations

There are only 3 tables comprising physical database model:

- code_list (used for translations and object keys)
- layers (used to define a layer)
- attributes (used to define attributes belonging to a specific layer from the layers data table)



A dynamic EAV model is build programmatically based on the entries in those three tables. For example to create Atlas layer “Monuments” with an attribute “Name” we have to have at minimum the following records:



code_list

key	trans	key_parent
MONUMENTS	en => Monuments	LAYERS
LAYERS	en => Layers	NULL
GEOM_LAYERS	en => "Geospatial layers"	NULL
POINTS	en => "Geospatial layers"	GEOM_LAYERS
NAME	en => Name	ATTRIBUTES
ATTRIBUTES	en => Name	NULL

layers

id	key_obj	key_tbl_geom
53	MONUMENTS	POINTS

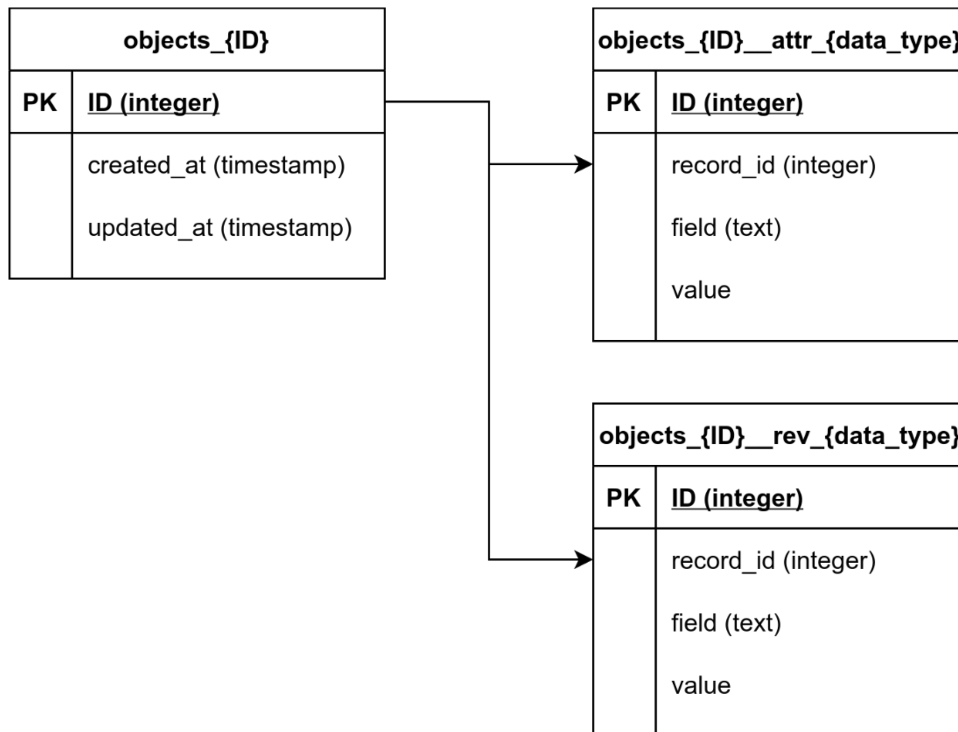
attributes

id	layer_id	key_attr
2342	53	NAME

The backend logic handles EAV model, ie. it creates a set of datatables:

- objects_{ID} (used to store EAV object entity record ID)
- objects_{ID}__attr_{data_type} (stores attribute data values)
- objects_{ID}__rev_{data_type} (stores attribute data values revision history)

With the following structure:



For example, if Monuments layer gets ID 101 in the table “layers”, then the backend will create tables objects_101 and objects_101__attr_txt, and if we store “some value” into the field (attribute) “name”, the value in the objects_101 is sequential record ID (let’s say 23) and corresponding record in the objects_101__attr_txt has values (record_id, field, value) = (23, ‘name’, ‘some value’).

In order to access the EAV data more easily in an object related manner materialized views are created when the model is rebuild. This creates objects_{ID}__mview data table, which is a typical object related spatial data table where each row has fields holding this row’s attribute and geometry values.