Document or database? The search for the perfect storage paradigm for lexical data.

Michal Měchura
Euralex 2022, Mannheim, Germany

Abstract

When building a dictionary writing system or indeed any software system for lexical data, one decision the software engineers need to make early in the project is, which storage format are we going to persist our data in? Are we going to store each entry as a separate XML document (or, more modernly, as a JSON object)? Or would it be better to store everything in a relational database with tables and relations between them? Or perhaps a graph database? An RDF triple store? Your answer to these questions will affect how easy or difficult certain things will be in the rest of the software, for example how easy or difficult it will be to search the dictionary, to perform bulk edits on multiple entries at the same time, to make changes to the entry schema, or to enforce referential integrity on cross-references. This paper will review the storage design patterns often found in lexicography and discuss their advantages and disadvantages.

Broadly speaking, there are two major patterns: the document pattern and the database pattern.

- In the document pattern, each entry is stored as a document in XML, JSON or some other markup/serialization language. The entries may and typically do have a highly formal internal structure which is made explicit in an entry schema such as a DTD, but this structure is unknown and invisible to the storage environment, apart from some indexes. Examples of the document pattern are file-based storage and XML databases, as well as relational databases where most tables are used only for storing indexes about the entries.

- In the database pattern, each entry is analyzed into its components (such as senses, example sentences, translations) and these are stored in the database as individual entities with their own identities. Examples of the database pattern are relational databases, graph databases and RDF triple stores. When a lexicographer is editing, he or she is not editing the entry as a whole, he or she is editing the individual entities. And each time an entry is to be shown to a human user, it is composed dynamically from the entities as an impermanent “view”.

The database pattern is strong where the document pattern is weak, and vise versa. For example, enforcing referential integrity on cross-references is easy in the database pattern (most database software has built-in support for that) and difficult in the document pattern (needs to be programmed manually). On the other hand, changing the entry schema and customizing the software for a different dictionary is easy in the document paradigm (we just need to provide an entry schema plus some rules for updating the indexes) and difficult in the database paradigm (the entry schema is hard-coded in the database structure).

The document pattern is very common in lexicography; practically all well-known dictionary writing systems are based on it, including Lexonomy, the IDM Dictionary Production System, and tLex. The database pattern is rare in lexicography and can mostly be found in single-purpose systems developed for a specific dictionary project where the schema is not likely to change. Hybrid approaches are ocassionally seen too, where some (but not all) entry components are analyzed into entities and stored à la the database pattern while the rest of the entry is left unanalyzed and stored à la the document pattern (this is how subentries are handled in Lexonomy, for example).

The paper will conclude by speculating that, while the document pattern is dominant currently, it is likely that lexicography will start embracing the database pattern more enthusiastically in the future, given current trends in the standardization of entry schemas and given a general tendency away from understanding dictionaries as static documents towards understanding them as dynamic repositories of structured content.