SylvaDB: A Polyglot and Multi-backend Graph Database Management System

Javier de la Rosa[1], Juan Luis Suárez[1] and Fernando Sancho Caparrini[2]
[1]*CulturePlex Lab, Dept. Modern Languages and Literature, Western University, Western Rd., London, Canada*
[2]*Ciencias de la Computacin e Inteligencia Artificial, CulturePlex Lab, University of Seville, Seville, Spain*

Keywords: Graph Databases, Polyglot Backends, Network Research.

Abstract: This paper presents SylvaDB, a graph database management system designed to be used by people with no technical knowledge. SylvaDB is based on flexible schema definitions and has been developed taking into account the need to deal with semantic information. It relies on the mathematical notion of property graph. SylvaDB is an open source project and aims at lowering the barrier of adoption for anyone using graph databases. At the same time, it is robust and scalable enough to support collaborative large projects related to knowledge management, document archiving, and research.

1 INTRODUCTION

After several years of work in an international and multidisciplinary project on cultural transfers, whose objectives included the analysis of several datasets of various types of cultural objects, we came to the conclusion that both the classic relational databases and the total dependence of researchers on programmers to deal with their data was detrimental to the purposes and objectives of the project.

We came to this conclusion for several reasons. First, we realized that non technical researchers presented a strong tendency towards resigning their own research autonomy and handing in all the decisions on database design and implementation to the professional programmer. However, this attitude was usually accompanied by the unsatisfactory results of the collaboration, as humanists, for example, were not able to extract or analyze their own results. And second, as the projects evolved, non technical researchers would come back to the programmers demanding changes in the structure of their databases, asking for new features during incremental design, corrections of mistakes in the database design, or modifications to reflect changes in the structure of the real world artefacts modelled in the database. But despite of the efforts of the database community to solve schema modification and evolution, as noticed by (Roddick, 1992), there is still a gap between the database administrator who manages to solve this issues, and the end user who needs them solved.

In an attempt to improve the interaction with databases, research efforts have been dedicated to provide graphical interfaces to relational databases such as HIBROWSE (Ellis et al., 1994), Query-By-Example (Zloof, 1975), or Santucci and Palmisano's Visualiser (Santucci and Palmisano, 1994). Also, some other works reveals moves from these complex graphical interfaces to more expressive and closer to natural languages alternatives (Hendrix et al., 1978; Wolfe et al., 1992; Popescu et al., 2003). However, there is no evidence of providing better and easier administration tools for non technical users that allow them to manage their data.

On the other hand, one of the most active areas of research at this moment is the study of networks (Knoke et al., 2008), what has brought the need to manage information with inherent graph-like nature (Angles and Gutierrez, 2008). Therefore, after surveying the landscape (Vicknair et al., 2010), we decided to build our own database management system, SylvaDB[1], in order to implement the graph model instead of the relational and to provide researchers with the tools they need as well as to reduce the dependency on programmers and database administrators.

2 DESIGN PRINCIPLES

The main goals that guided the design of this graph

[1]http://sylvadb.com

de la Rosa J., Suárez J. and Sancho Caparrini F..
SylvaDB: A Polyglot and Multi-backend Graph Database Management System.
DOI: 10.5220/0004475702850292
In *Proceedings of the 2nd International Conference on Data Technologies and Applications* (DATA-2013), pages 285-292
ISBN: 978-989-8565-67-9

database management system were:

- to provide technological autonomy to non technical researchers,
- to allow for modifications and evolutions of the schema at any time of the life cycle of the database,
- to be scalable and therefore useful for both the single researcher working on a relatively small project and the requirements of large teams,
- to manage different datasets under the same system,
- to allow collaborative work on single databases,
- to support as many as possible types of objects,
- to present the information by using an intuitive interface.

Out of our own experience in Complex Systems and Cultural Analysis, we realized that topics and relations among them are at the same level of interest. Consequently, and from previous goals, the following conceptual and technological features were mandatory for us:

- No tables, only objects and relations.
- Arbitrary number of attributes on objects and relationships.
- Support for multimedia content natively (image, audio, video and documents).
- Users and permissions management.
- Ability to perform complex queries to retrieve and analyze information.
- Interactive graphic visualization of the content.
- Connection to external tools for visualization and analysis.

3 ARCHITECTURE

SylvaDB relies on the paradigm of Database as a Service (DBaaS), as described by Hacigumus (Hacigumus et al., 2002). Written using the Python programming language, and built through the Model-View-Controller web framework Django (Holovaty and Kaplan-Moss, 2009), SylvaDB is a web platform that can be used in modern browsers such as Google Chrome or Mozilla Firefox.

In SylvaDB a graph database consists of a schema with information on data types and properties for nodes and relationships, and the real data stored on them. Because some graph backends are schema-free and some others are schema-restricted, and in order to guarantee compatibility with as many of them as possible, we store the schema (if any) in a relational database while all data is actually stored in the graph database backend; properties for both nodes and relationships are also stored in the graph backend; and the set of media files and other static files are stored in Amazon Simple Storage Service (S3)[2].

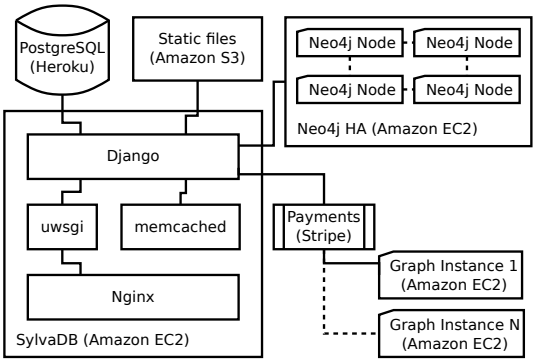

Figure 1: General Architecture of SylvaDB.

SylvaDB's architecture, as depicted in Figure 1, is divided into several parts in terms of machine isolation, so any of them can be replaced or clustered with no major problems[3].

In short, there is a main HTTP server that serves the Django application SylvaDB through an uwsgi-compliant Python process. All statics files (including user uploads and client-side Javascript and styles) are handled in Amazon S3 instances. The relational database to manage users, sessions, permissions and graphs metadata like their schemas, is managed in PostgreSQL instances by the Platform as a Service provider Heroku (Malliga, 2012). Actual data for graphs in the shared graph backend is stored in a Neo4j High Availability (HA) cluster, and graphs in custom instances may be in different kinds of graph backends as described in section 3.2. On the other hand, both low level and view level cache are implemented in a distributed memcached environment (Fitzpatrick, 2004).

This setup confers reliability and performance to SylvaDB at the time that makes scalability a problem easy to solve.

3.1 HTTP Server

Nginx[4] is an asynchronous HTTP and reverse proxy server that natively includes support for upstream

[2]https://aws.amazon.com/en/s3/

[3]SylvaDB also gives support to services such as registration and messaging, but we have omitted them in the diagram, as they play an auxiliary role in our architecture.

[4]http://nginx.com/

servers speaking the WSGI[5] protocol, such as uWSGI[6] does for Django. The main advantage of the asynchronous approach is scalability. In a process-based server, each simultaneous connection requires a thread which incurs significant overhead. An asynchronous server, on the other hand, is event-driven and handles requests in a single, or very few threads. While a process-based server can often perform on par with an asynchronous server under light loads, if heavier loads occur the performance degrades significantly as RAM consumption increases.

3.2 Graph Model

At the core of the system, we use the Object-Relational Mapping (ORM) (Burzańska et al., 2010) provided by Django to manipulate the programming objects in the code, and a set of *mixins* to abstract graph backends (Esterbrook, 2001).

Figure 2 illustrates a simplified UML diagram of the elements implied in the Graph model. Django terminology uses *model* when refering to Python classes that inherit from the base class that does the mapping between the code and the database; it provides some helpers to avoid the manual manipulation of tables, foreign keys and many to many relationships. To emulate a similar behaviour in relation to the graph backends, and to make it easier to develop SylvaDB, we wrote some mixins to abstract all the graph databases primitives. In the diagram we omit inheritance from Django models. Therefore, SylvaDB makes use of Graph object instances that cover both the schema and the backend.

Our graph structure is based on the property graph, i.e., a multigraph data structure where graph elements, nodes and relationships, can have properties (attributes) and can be typed. Formally, the property graph is defined as a tuple $G = (V, E, P, D_V, D_E, \nu_V, \nu_E)$, where V is a set of vertices, E is a multiset of directed edges, P is a domain of properties, D_V, D_E are the domain of allowed property values for vertices ad edges, respectively, $\nu_X : X \times P \to \mathcal{P}_f(D_X)$ ($X = V$ or $X = E$) is a function that maps properties of elements of X to their values ($\mathcal{P}_f(D_X)$ is the collection of finite subsets of D_X, meaning that a property can be associated with multiple items from D_X).

3.2.1 Flexible Schemas

The first component of a Graph object is the schema,

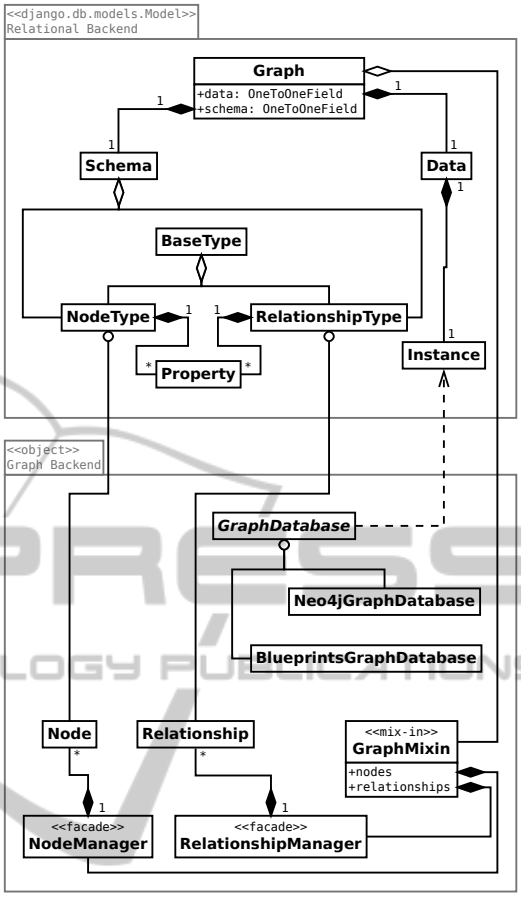

Figure 2: Simplified UML Diagram.

albeit it is not required of the graph to have one. Together with the fact that some users may need no schema in their data structure, the idea behind this flexible schema is to design the platform as independent as possible from the graph backend. Since some graph bakends may require a schema and others do not, we move the schema out of the graph backend and made it optional.

Even when this decision may involve a manual managing of all triggers commonly handled by relational databases, at the same time provide us with a powerful and fine grained control over schema evolution (the ability of a database system to respond to changes in the real world by allowing the schema to evolve (Roddick, 1992)). Relational databases usually have fixed schemas, but SylvaDB does not have this limitation because is based on graphs, and schema information is only tied to graph data through the application logic.

SylvaDB makes users owners and responsible for their schema evolution, at the time it provides them with the tools to decide. Before adding nodes to a graph, a schmea must be defined by creating node

[5]http://wsgi.readthedocs.org/

[6]http://uwsgi-docs.readthedocs.org/en/latest/Protocol.html

types and relationship types. By using web forms generated by SylvaDB, the user is able to add properties to the types and build the schema. Every time a user makes a change in a graph schema, he is prompted to choose what to do depending on the operation he wants to perform:

- If the user renames a node type or relationship type, nothing will happen since types have internal numerical identifiers.
- If the user renames a property, SylvaDB can keep the previous property name internally in the item (node or relationship), can remove the old property from the item and consider the new property as new and empty, or can try to rename the last property with the new one.
- If the user removes a node type or relationship type, will be asked to decide if all related items must be removed on cascade or just the type.
- If the user removes a property, SylvaDB can also keep the value internally in the item or remove it from all the items affected.

3.2.2 Polyglot Graph Backend

The second component of a Graph object is a (mandatory) graph backend. From its inception, SylvaDB was designed to support different technologies as backend, what we call polyglot graph backend, since there exist different graph databases providers for different needs. For example, Titan[7] is intended for distributed and massive-scale graphs, whereas Neo4j is faster but does support a slightly lower number of nodes. Currently, the graph databases providers landscape is not as huge and diverse as the relational, and unfortunately most of them are only suitable to be used in Java language programming.

Table 1: Summary of graph databases, query methods and Python bindings.

Graph Database	Query Method	Python Binding
Neo4j	Cypher, Gremlin, Traversal	Blueprints, Native, REST
OrientDB	Cypher Traversal	Blueprints
HyperGraphDB	HGQuery Traversal	*No*
DEX	Traversal	Blueprints
Titan	Gremlin	Blueprints
InfiniteGraph	Gremlin	*No*

To overcome this limitation, and having in mind that SylvaDB base code is Python, we resorted to REST (Representational State Transfer) interfaces (Fielding, 2000). Specifically, SylvaDB makes use of the Blueprints API[8], a generic graph Java API that binds to various graph database backends. Many graph processing and programming frameworks are built on top of it, e.g., Gremlin[9], a graph traversal language, and Rexster[10], a graph server that exposes any Blueprints graph through a REST interface. Not all graph database backends have a built-in support for the Blueprints API, as the Table 1 shows, and others have their own interfaces, like Neo4j.

Figure 3 shows how, on top of these REST interfaces (Blueprints and Neo4j), SylvaDB defines a set of abstraction layers to operate with them exposing a common and unique set of methods and attributes, the `GraphDatabase` interface. Any class able to implement this interface, will be suitable as a graph backend in SylvaDB. Nevertheless, only Neo4j and Blueprints-compliant graph databases are available now, thanks to client libraries like neo4j-rest-client[11] and pyblueprints[12], respectively.

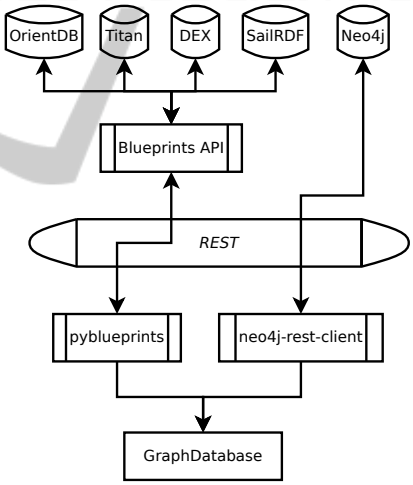

Figure 3: Polyglot graph backend diagram.

Since each graph backend run on its own service, SylvaDB is able to deploy backends on demand, by using Amazon CloudFormation[13] templates, and offer them to users with different needs.

3.3 Queries

For the purposes SylvaDB has been built, easy ways

[7] http://thinkaurelius.github.io/titan/

[8] http://blueprints.tinkerpop.com
[9] http://gremlin.tinkerpop.com
[10] http://rexster.tinkerpop.com
[11] http://pypi.python.org/pypi/neo4jrestclient/
[12] http://pypi.python.org/pypi/pyblueprints/
[13] http://aws.amazon.com/cloudformation

for retrieval information from data was as important as data itself, making it mandatory to provide tools to build powerful queries to the user.

However, although it is not a new topic, recent research and implementations are finally reaching a tipping point with regards to providing natural language interfaces for querying databases (Popescu et al., 2003). In this context, SylvaDB tries to be a step ahead by generating schema-based grammars to process users' queries.

Although the grammar depends on the graph schema and is domain-specific and quite limited yet, it is able to process approximate natural language queries as inputs and produce backend-specific queries as outputs. As an example, let's suppose we have a graph called "Workers" with a Neo4j backend, which supports Cypher Query Language[14], then the query:

```
Person who lives in country with name Spain
```

will produce the following Cypher query:

```
START node=node:graph(label='Workers')
MATCH person-[:lives_in]->country
WHERE country.name = 'Spain'
      AND person.type = 'person'
      AND country.type = 'country'
RETURN person
```

Unfortunately, for more complex queries (such as returning nodes in the middle of a path) or to support schema-free graphs, the only way around is by using the query language available for the specific backend. After having explored the options to implement a visual query system (Catarci et al., 1997), it is clear that SylvaDB has considerable room to improve in this area.

3.4 Visualization

For graphs with defined schemas, paginated table-based views are provided for each type in the schema.

Since the data model is based on graphs, SylvaDB may provide a proper way to visualize users' data in a more graphical manner. There are so far two different kinds of visualization: the first is our own development, and it allows users to expand relationships by clicking on the nodes. However, for graphs with more than a couple of thousand nodes this visualization, although more complete, is not very useful. The second one is based on sigma.js[15], an open-source lightweight JavaScript library to draw graphs, and using the HTML5 canvas element[16].

[14]http://docs.neo4j.org/chunked/stable/cypher-query-lang.html

[15]http://sigmajs.org/

[16]http://w3.org/TR/html5/embedded-content-0.html

For other general purposes, the system provides exporting functionality to GEXF[17] files, a graph format file defined for Gephi (Bastian et al., 2009).

3.5 Collaboration

Fine grained permissions is also an important feature that SylvaDB includes. Implemented to graph object level, graph owners can grant up to 9 different permissions for their graphs to other users:

- over the graph: view, change collaborators, change graph properties (information description and visibility),
- over the data: all CRUD operations for nodes and relationships,
- and over the schema: view and change.

4 APPLICATIONS

Thanks to the versatility of the conceptual storage data type supported, the property graph, the uses of SylvaDB vary from a tool for standard storage, or an implementation system for ontologies, glossaries, conceptual maps and relaxed topic maps, to an analysis tool useful to find non explicit relations between topics or discover hidden knowledge through queries that traverse the graph.

Below follow three well studied cases that have proven how useful SylvaDB can become.

4.1 Formal Ontologies: Baroque Paintings Network

This research on baroque paintings is part of a 7-year project that deals with the issues of cultural transmission and assimilation and community formation during the baroque period in the territories of the former Hispanic Monarchy. One of the important aspects of this historical case lies in the fact that the political structure that supported or accompanied the studied cultural transfers reached almost global proportions, from Europe to American and Asia. Thus, it offered an excellent benchmark to test some of the prevalent theories in the field taking advantage of a dataset that included over 13,000 paintings, 1636 creators, 405 series of paintings, 195 schools and 2482 geographical locations from different territories and cultural areas.

So far, extensive research have been done on three main issues. First, an analysis of the data set to answer questions related to the formation and sustainability of large cultures, the semantic content of the

[17]http://gexf.net/format/

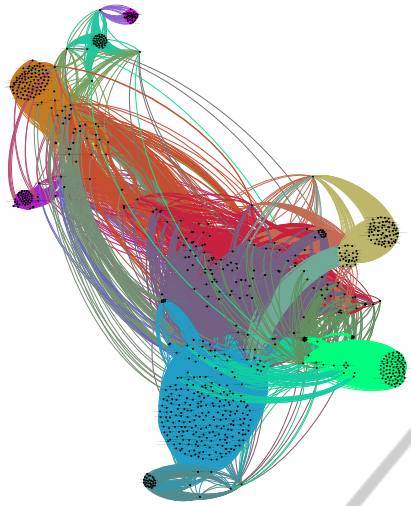

Figure 4: Example of graph generated by SylvaDB, exported to GEXF format and visualized and styled in Gephi. This graph represents the similarity network of Hispanic-American paintings in the period from 1600 to 1625. Colors designate modularity classes.

network of paintings, and the role of art as an institution that contributes to sustain large-scale societies (Suárez et al., 2012). From a set of 211 keys or descriptors it was carried out a manual semantic annotation of all artworks (with an average of 5.85 descriptors/work and peaks of 14 per work).

This process of annotation required a special level of reliability in order to avoid data conflicts. A team of annotators was setup following the next hierarchy: administrators (with global permissions over the graph), reviewers (with all CRUD permissions over the graph data, but not over the schema), and annotators (with creation and edition permissions only). The fact that SylvaDB has a built-in support for collaborative work and detailed permissions management (both essential in this kind of research), was crucial to the success of the project.

On the other hand, this experience allowed us to know how real users face the interface, resulting in a positive feedback about its usability and easiness when it comes time to handle complex data or modify the schema according to the necessities in real time. Altogether, more than 30 people were using SylvaDB in creating the Hispanic Baroque paintings network, and currently it is available for researchers all over the world to enhance and validate the content.

The research focuses on the network of paintings resulting from an analysis of the edges that connected them (Suárez et al., 2011) (see Figure 4). These edges are part of a detailed ontology that allowed the research team to fully categorize artworks from various provenances and geographic contexts. The network of baroque paintings proved to be a resilient one that allowed for the integration of many local aspects and techniques while keeping all those technical and thematic features that made a painting from Cusco (Peru) similar to one from Puebla (Mexico), Madrid (Spain), or Antwerp (Belgium).

4.2 Knowledge Influence: elBulli Graph

This case proposes a formalization of the evolutionary method of creation developed by Ferran Adrià in his restaurant, elBulli, over many years of work that took it to be considered the best restaurant in the world by expert media and the public. If a recipe is a sort of algorithm that explicitly gives a set of instructions to reproduce a given dish, then they are excellent for reproducing existing culinary knowledge. But if reproduction is not the objective any more, as Adrià discovered early on, then the recipe would be just a byproduct of the whole method, not the corner stone. For the new method of creative cuisine developed by Adrià, creativity was to be anchored at the levels of preparations, techniques and concepts as it is at these levels that creativity in the kitchen can become evolving creativity. Figure 5 shows the general schema of a recipe as conceived by Adrià's cooking style.

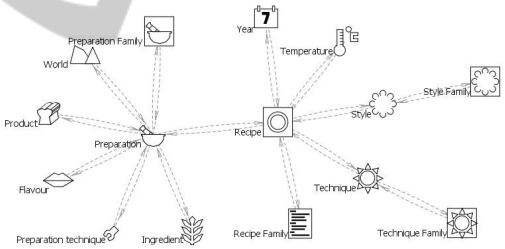

Figure 5: Schema of a Recipe in elBulli context.

What Adrià and his team did was to develop and implement a method to achieve what Horng and Lin (Horng and Lin, 2009) have described as the objective of a truly creative form of culinary art, that is, a "wider variety of dishes, as well as more innovative, aesthetically, and culturally innovative individual dishes". The development of new techniques, the inclusion of new concepts, the constant addition of new products and, specifically, the combination, mix and merger of all elements would open the floodgates of discovery into the creation of new preparations and, along with it, a considerable increase in the rate of innovation. All these elements become nodes in a complex network that keep most of their own instances connected, a network that allows for the creation of many paths and the integration of new elements, specially *products*, *techniques*, and *preparations*. The

combinatory potential of the method is unquestionable, but even more important are the effects provoked by the integration of new elements connected in surprising ways with the existing structure. This increase in innovation can be seen in the growth in the numbers of hubs and relations in their method that would result in a multiplication of the dishes and recipes produced at the end of any given season.

Using the SylvaDB query system, in this case running real Cypher queries for a Neo4j graph backend, conclusions about elBulli artistic evolution and the elements involved in its creative process, just came up. In this way, we could prove how useful is the representation of data in graphs, not only at conceptual level, but from the point of view of storage backend and querying.

4.3 Social Network Analysis: Preliminaries Project

Finally, the Preliminaries project uses SylvaDB to store, visualize and analyze a large data set dealing with information contained in the 17^{th}-century books that allow us to unearth the social network of publishers, writers, bureaucrats, and noblemen involved in the publishing industry at both sides of the Hispanic Atlantic.

The "preliminary" of a 17^{th}-century book encompasses all pages appearing in the printed text before the beginning of the work itself. This information is divided into five different types: details of publication, documentation of censorship (both civil and ecclesiastical), licensing, selling price and dedications. The important thing for this project lies in the fact that all sections usually include names of the officers or individuals signing the specific documentation, dates and places of the issuing of the documents, i.e., in a few pages these preliminaries give us a very complete image of the individuals, places, dates, councils, and institutions involved in the publication of each specific title of literature. In this process, SylvaDB played an important role by allowing an incremental design of the schema without losing any previous data. By compiling all this information into a graph database, and subsequently performing different kinds of queries according to the various research questions, the dataset became a valuable source of information about the historical networks that influenced the publication of early Modern Spanish literature.

This project has produced a research model that combines network-based analysis with quantitative and qualitative studies of early Modern cultural production. It has also explored the concept of cultural network within the framework of complexity theory and with new data covering various historical periods. Last, it has given evidence of the interaction between political structures and cultural production through social network analysis.

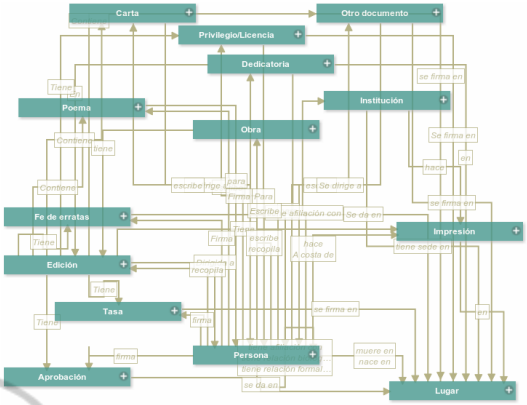

Figure 6: Screenshot of the SylvaDB schema for the Preliminaries Project graph.

The cultural objects considered in this study are complex and variable (see Figure 6). Therefore, the flexibility of SylvaDB, that allows for the adaptation of the schema in real time, was a key factor to achieve the goals of the project.

5 CONCLUSIONS AND FUTURE WORK

SylvaDB, although still in private beta, is currently being used in several historical and cultural projects, and it will soon be open for any researcher or group to use. Also, it is open source software (GNU Affero General Public License[18]) and the source code can be easily found and forked[19].

As fields like Digital Humanities grow and evolve, more and more researchers with no training in computer science feel the need to work with large datasets and combine their traditional object-focused research with data analysis and visualization. SylvaDB aims at lowering the barrier for the adoption of database systems into the practices of those fields. Also, it is designed to support the storage and analysis required by large, advanced research projects looking for a powerful and intuitive data and document management system based on flexible schemas.

As any other existing piece of software, SylvaDB is far of being finished in terms of features. A non exhaustive list of proposed features is shown below.

[18] http://www.gnu.org/licenses/agpl-3.0.html
[19] http://github.com/CulturePlex/Sylva

- A visual editor for queries. Even if the natural language query input is powerful and intuitive enough, a more complete and customizable system that enables users to build their own complex queries is in our plans, due to the same criteria of usability and low barriers that got the project started in the first place.
- Implementation of topological pattern matching protocols as a complement for the query module.
- Better visualizations. Full-screen mode, integration with queries system, and basic interaction with properties of drawn elements such as shape, color, and size of nodes, relationships and labels.
- A battery of well-known algorithms. Including the most common ones, like Page Rank, HITS or Dijkstra (Ding et al., 2002), could be a valuable help for users to perform first level analysis using the same interface.

We are also working to improve tests coverage to reach at least a 90% of the code covered.

ACKNOWLEDGEMENTS

We acknowledge the support of the Social Sciences and Humanities Research Council of Canada through a Major Collaborative Research Initiative on The Hispanic Baroque. And the Canada Foundation for Innovation through the Leaders Opportunity Fund.

REFERENCES

Angles, R. and Gutierrez, C. (2008). Survey of graph database models. *Computing Surveys*, 40(1):1.

Bastian, M., Heymann, S., and Jacomy, M. (2009). Gephi: An open source software for exploring and manipulating networks.

Burzańska, M., Stencel, K., Suchomska, P., Szumowska, A., and Wiśniewski, P. (2010). Recursive queries using object relational mapping. *Future Generation Information Technology*, pages 42–50.

Catarci, T., Costabile, M., Levialdi, S., and Batini, C. (1997). Visual query systems for databases: A survey. *Journal of visual languages and computing*, 8(2):215–260.

Ding, C., He, X., Husbands, P., Zha, H., and Simon, H. (2002). Pagerank, hits and a unified framework for link analysis. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 353–354. ACM.

Ellis, G., Finlay, J., and Pollitt, A. (1994). Hibrowse for hotels: bridging the gap between user and system views of a database. In *IDS'94 Workshop on User Interfaces to Databases*, pages 45–58.

Esterbrook, C. (2001). Using mix-ins with python. *Linux Journal*, 2001(84es):7.

Fielding, R. (2000). *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California.

Fitzpatrick, B. (2004). Distributed caching with memcached. *Linux journal*, (124):72–74.

Hacigumus, H., Iyer, B., and Mehrotra, S. (2002). Providing database as a service. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 29–38. IEEE.

Hendrix, G., Sacerdoti, E., Sagalowicz, D., and Slocum, J. (1978). Developing a natural language interface to complex data. *ACM Transactions on Database Systems (TODS)*, 3(2):105–147.

Holovaty, A. and Kaplan-Moss, J. (2009). *The Definitive Guide to Django: Web Development Done Right*. Apress.

Horng, J. and Lin, L. (2009). The development of a scale for evaluating creative culinary products. *Creativity Research Journal*, 21(1):54–63.

Knoke, D., Yang, S., and Kuklinski, J. (2008). *Social network analysis*, volume 2. Sage Publications Los Angeles, CA.

Malliga, P. (2012). Database services for cloud computing–an overview. *Database*, 2(3).

Popescu, A., Etzioni, O., and Kautz, H. (2003). Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 327–327. ACM.

Roddick, J. (1992). Schema evolution in database systems: an annotated bibliography. *ACM sIGMOD record*, 21(4):35–40.

Santucci, G. and Palmisano, F. (1994). A dynamic form-based data visualiser for semantic query languages. *Interfaces to Database Systems*, pages 249–265.

Suárez, J., Sancho, F., and de la Rosa, J. (2011). The artspace of a global community: the network of baroque paintings in hispanic-america. In *Culture and Computing (Culture Computing), 2011 Second International Conference on*, pages 45–50. IEEE.

Suárez, J., Sancho, F., and de la Rosa, J. (2012). Sustaining a global community: Art and religion in the network of baroque hispanicamerican paintings. *Leonardo*, 45(3):281–281.

Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., and Wilkins, D. (2010). A comparison of a graph database and a relational database: a data provenance perspective. In *Proceedings of the 48th annual Southeast regional conference*, page 42. ACM.

Wolfe, R., Needels, M., Arias, T., and Joannopoulos, J. (1992). Visual revelations from silicon ab initio calculations. *Computer Graphics and Applications, IEEE*, 12(4):45–53.

Zloof, M. (1975). Query by example. In *Proceedings National Computer Conference*, pages 431–438. ACM.