Kinship and Conflict Visualization in European Royalty using JanusGraph

JOSÉ SILVA, FEUP, Portugal
MARIA CARNEIRO, FEUP, Portugal
SÉRGIO ESTÊVÃO, FEUP, Portugal

The goal of this research is to assess the potential of JanusGraph, a distributed graph database developed for large-scale graph dataset management. A complete explanation of the technology, including license, cost, and community, is provided. The research looks at JanusGraph's data model, APIs, and data processing features and compares them to comparable graph and non-graph database systems. Real-world use cases and problematic scenarios are also examined, as are the results of benchmark testing versus competing technologies. A JanusGraph prototype implementation is also being created to illustrate its potential use in organizing and querying complex graph databases. Overall, this paper gives a comprehensive examination of JanusGraph, outlining its merits and limitations as well as exploring its possible significance in future data management and analysis.

CCS Concepts: • **Database systems**; • **Graph-based database models**; • **Graph algorithms**; • **Distributed database systems**; • **Visualization**; • **Information visualization**; • **Graph partitioning**; • **Humanities computing**;

Additional Key Words and Phrases: JanusGraph, distributed database, graph database, kinship visualization, conflict visualization, graph partitioning, visualization, information visualization, humanities computing

ACM Reference Format:
José Silva, Maria Carneiro, and Sérgio Estêvão. 2023. Kinship and Conflict Visualization in European Royalty using JanusGraph. 1, 1 (May 2023), 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The growth of big data has made managing and querying large-scale databases increasingly difficult. Because traditional relational databases have limitations in handling complex relationships and graph data, specialized graph databases have emerged, intended to store, manage, and query data in the form of graphs.

Graph databases are appropriate for organizing data with complex relationships, such as those found in social networks, recommendation systems, and knowledge graphs. They allow for effective data searching that includes traversing several connections, which is sometimes difficult and wasteful in standard relational databases. Due to their capacity to manage complicated data models and interactions, graph databases have grown in popularity in recent years.

JanusGraph is a distributed graph database that is classified as a property graph database. It is intended to provide a scalable and adaptable solution for handling massive graph datasets, making it a viable technology for a wide range of applications. The purpose of this research is to assess JanusGraph's potential as a graph database

Authors' addresses: José Silva, FEUP, Porto, Portugal, up201904775@up.pt; Maria Carneiro, FEUP, Porto, Portugal, up201907726@up.pt; Sérgio Estêvão, FEUP, Porto, Portugal, up201905680@up.pt.

XXXX-XXXX/2023/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

solution and to investigate its strengths and limits. To do this, we will present a thorough review of JanusGraph's characteristics, such as its data model, APIs, data processing capabilities, and its unique features.

In addition, we will examine real-world use cases and problematic circumstances to illustrate the benefits and limitations of utilizing JanusGraph vs competing graph and non-graph database systems. Finally, we will create a JanusGraph prototype to show its potential applicability in organizing and querying complicated graph databases. Overall, the goal of this study is to contribute to the developing field of graph database research and give insights into JanusGraph's potential for future data administration and analysis.

2 BACKGROUND

In this section, we will give the necessary information to understand the contents of this report, namely the NoSQL and Graph Database paradigms.

2.1 Graph Databases

Before approaching the concept of graph databases, we must first define the concept of NoSQL databases. Although one would suggest, a NoSQL database does not mean it does not use SQL, in reality, it means that a database uses a distinct storage structure and data schema from the SQL counterpart. The most prevalent NoSQL database systems can be organized in document, column, key-value stores, vector, and graph databases.

Graph databases are organized as graphs with nodes and edges, which correspond respectively to the entities and their relationships. Edges correspond to a link between a source node and a destination node. Furthermore, both these structures can have properties, and additional metadata that is arranged in key-value pairs and can be singular or multiple. As mentioned in [2], depending on the implementation, these properties can be types or labels. While typed objects are unique and behave like classes, labeled objects are flexible and can be used for grouping or categorization.

Graph databases are generally built for use with transactional (OLTP - Online Transaction Processing) systems. Accordingly, they are normally optimized for transactional performance, and engineered with transactional integrity and operational availability in mind [7].

3 TECHNOLOGY OVERVIEW: JANUSGRAPH

In this section, we will give an overview of JanusGraph, a distributed, scalable, and open-source graph database. We will begin by examining JanusGraph's origins and history, including its development and progression from TitanDB. We will also discuss JanusGraph's license strategy, price, and community engagement.

3.1 JanusGraph Origin

JanusGraph is an open-source graph database project that began in 2017 as a clone of the TitanDB project, which had its development discontinued by the closure of the company that built it, Aurelius, creating a market gap for a high-performance, scalable graph database.

Continuing the TitanDB mystical theme, JanusGraph takes its name from the Roman deity Janus, who is frequently represented with two faces gazing in opposing directions. This represents the two data models that JanusGraph supports: property graphs and RDF (Resource Description Framework) graphs, as well as its ability to manage both OLTP (Online Transaction Processing) and OLAP (Online Analytical Processing) workloads. The "J" in Janus refers also to the Java programming language used to create JanusGraph.

JanusGraph was founded by a group of developers from the open-source community to fill this void by offering a more actively maintained, community-driven alternative to TitanDB. This new database was brought under the open governance in the Linux Foundation in 2017 and was designed to be a scalable, adaptable, distributed graph database that can manage enormous volumes of data and be utilized for a variety of applications such as recommendation engines, fraud detection, and social networking platforms. It supports many storage backends, including Apache Cassandra, Apache HBase, and Oracle BerkeleyDB, and may be readily modified to handle new data sources [5].

3.2 Licensing, Pricing, and Community

JanusGraph is a free and open-source project distributed under the Apache 2.0 license. This implies that the program can be used, modified, and distributed freely, with no licensing costs or limitations. Users may download and utilize JanusGraph for any purpose, including commercial usage, without incurring any expenses.

Furthermore, JanusGraph is a user- and developer-driven project with a varied and active user and development community. The Apache Software Foundation, a non-profit organization that offers governance and infrastructure support for open-source software projects, maintains the project. Developers, users, and contributors from diverse companies and backgrounds cooperate and contribute to the project through code contributions, bug reports, documentation, and community assistance.

4 FEATURES AND CAPABILITIES

JanusGraph is a feature-rich graph database that allows for the storage, querying, and management of large-scale graphs. In this section, we will go more in-depth about JanusGraph's important features and capabilities, including its data model and operations, APIs, client libraries, and data processing functions, as well as its consistency and replication capabilities. First, we'll look at JanusGraph's data model and operations, which include support for vertex and edge properties, indexes, and a flexible query language. Following that, we'll look at JanusGraph's architecture, namely its APIs, client libraries, and data processing features, such as support for a number of programming languages and frameworks and interaction with other data processing tools. Finally, we'll go through JanusGraph's consistency and replication capabilities, which allow for high availability and fault tolerance, as well as data distribution and load balancing.

4.1 Data Model and Operations

JanusGraph, a distributed graph database, provides a flexible data model and a comprehensive set of operations to manage and query graph data effectively.

As mentioned in [5], JanusGraph adopts a property graph data model, which consists of vertices, edges, and properties associated with both vertices and edges. This model allows for representing complex relationships and storing rich data within the graph. The vertices represent entities or objects, while the edges define the relationships between these entities. Both vertices and edges can have associated properties, which store additional information about them.

JanusGraph data model enables the user to assign multiple labels to vertices and edges, used for classification and organization of the elements in the graph, this also makes it easier to query and analyze specific subsets on the graph.

Regarding operations, JanusGraph provides a rich set of operations for traversing, querying, and modifying graph data. It provides the typical CRUD operations, and graph traversal, by using the integrated Gremlin traversal language, flexible graph querying, supporting property-based and index-based queries, and ACID transactions.

Related to deployment and architecture, JanusGraph offers single and multi-server deployments with high availability and distributed deployment across distinct data centers, the ability to instance graphs with customs architectures and configurations, which are going to be addressed in the following sections.

Finally, regarding processing and data handling, JanusGraph provides Bulk Loadings, a specific cache that stores frequently accessed graph data in memory, reducing the need to fetch data from disk, mechanisms to handle transaction failure and system recovery in cluster systems, ensuring system robustness no loss of performance while the failed instance is restarted, and it enables migrations to Titan, its graph database predecessor, and Apache Thrift, an interface that provides clean abstractions for data transport, data serialization, and application level processing.

4.2 Architecture

One of the distinct features of JanusGraph is its modularity and flexibility, as it can be configured with multiple applications to work in different environments.

JanusGraph can be integrated into both Online analytical processing (OLAP) and online transaction processing (OLTP) data processing systems. As regarded in [5], on one hand, applications can interact with JanusGraph in a OLAP system, Gremlin is utilized to perform complex analytical queries on the graph, within the same JVM, on the other hand, considering a OLTP system, since JanusGraph natively supports the Gremlin Server component of the Apache TinkerPop stack, application can interact directly with a local or remote instance of JanusGraph.

JanusGraph also uses external databases to store information, giving the user the opportunity to choose one accordingly to their needs, it can be configured with Cassandra and HBase, both Column-Family Databases with catered for different scenarios, and BerkeleyDB, a key-value store database, and others, such as ScyllaDB, another Column-Family Databases, known for being able to map CPU cores to Scylla shards, achieving lower latency queries compared to most databases, and consequently JanusGraph can take advantage of these qualities when both are combined [1].

Finally, JanusGraph can also support indexing backends, note that, as mentioned in its documentation [5], while JanusGraph's composite graph indexes are natively supported through the primary storage backend, mixed graph indexes require that an indexing backend is configured. Mixed indexes provide support for geo, numeric range, and full-text search. Both ElasticSearch and Apache Solr are ideal for distributed JanusGraph systems, while Apache Lucene is better for single-machine usage.

In the following, it is possible to visualize a diagram of the JanusGraph architecture:

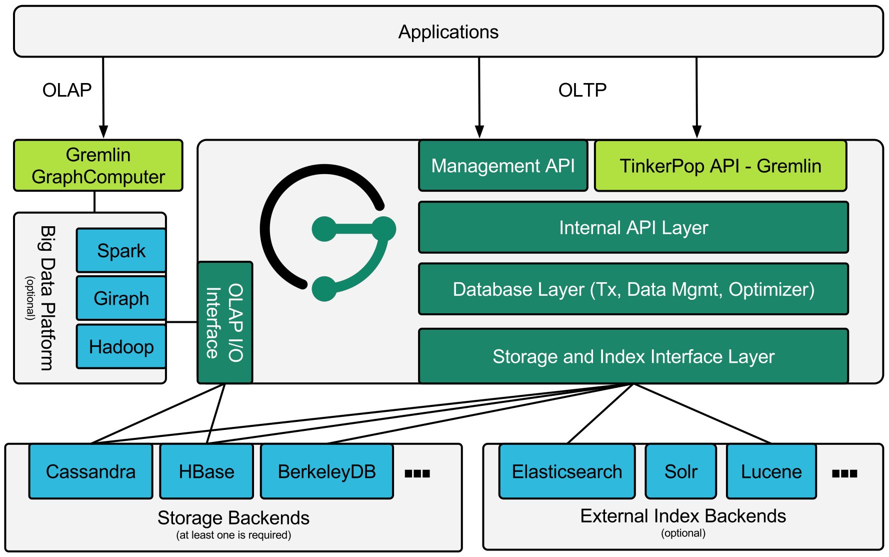

Applications
OLAP
OLTP
Gremlin GraphComputer
Management API
TinkerPop API - Gremlin
Internal API Layer
Database Layer (Tx, Data Mgmt, Optimizer)
Storage and Index Interface Layer
Big Data Platform (optional)
Spark
Giraph
Hadoop
OLAP I/O Interface
Cassandra
HBase
BerkeleyDB
Storage Backends (at least one is required)
Elasticsearch
Solr
Lucene
External Index Backends (optional)

Fig. 1. JanusGraph architecture. *Source* [5]

4.3 Consistency and Replication Features

JanusGraph, as a distributed graph database system, follows the concepts of the CAP theorem, which claims that it is impossible to maintain consistency (C), availability (A), and partition tolerance (P) in a distributed system at the same time. According to the CAP theorem, a trade-off must be made between these three traits. The choice of backend storage technology has a direct influence on how the CAP theorem is achieved in the context of JanusGraph. JanusGraph offers a variety of backend alternatives, including Apache Cassandra, Apache HBase, and Oracle BerkeleyDB, each with its unique set of features. Due to the nature of JanusGraph, depending on the selected backend different features of the CAP theorem may be emphasized. Apache Cassandra, for example, has high availability and partition tolerance, making it appropriate for applications that value scalability and fault tolerance. Apache HBase, on the other hand, provides excellent consistency assurances at the tradeoff of some availability during network splits. JanusGraph's backend option enables customers to align their data model and operational needs with desired trade-offs in consistency, availability, and partition tolerance.

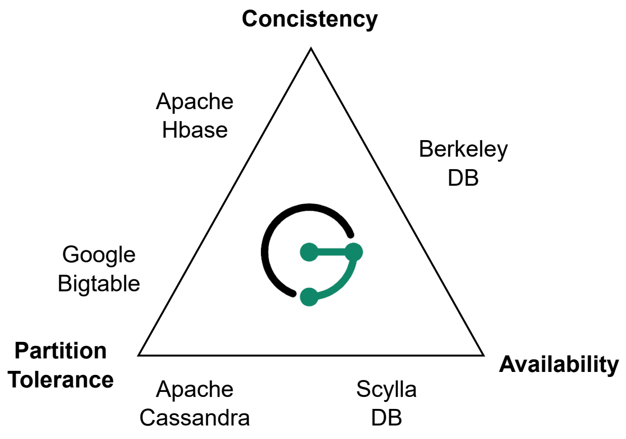


Fig. 2. JanusGraph related to the CAP theorem

5 REAL-WORLD USE CASES AND SCENARIOS

Due to its open-source nature and flexibility, JanusGraph has seen a wide range of uses across multiple scenarios, being deployed in production code of, for example, *IBM*, *Red Hat*, and *Ebay* projects.

Notable scenarios where JanusGraphs is ideal are the ones that involve modeling and analyzing complex relationships and networks (Graph problems), distributed and highly available environments, such as clusters or cloud-based architectures, and graph analytics applications, because JanusGraph integration with graph analytics frameworks, such as Apache Spark and Apache Giraph, allows the database to excel in these scenarios.

Some notable applications of JanusGraph are in spatiotemporal analysis [3], which involves the managing of large datasets that often are scattered. In this example, the authors investigate how to effectively store multi-dimensional, interconnected data, and whether or not graph database technology is better suited when compared to the extended relational approach, concluding that JanusGraph outperforms the querying of datasets with this kind of data. Another notable application is in cloud architecture, in [6] JanusGraph is setup with Google's Bigtable service as a backend, and orchestrated using Kubernetes, making this integration a viable serverless distributed graph database that can easily scale, thanks to Bigtables capability of handling large volumes of data.

6 COMPARISON TO OTHER SOLUTIONS

In this section, we will compare and evaluate JanusGraph in light of other graph database solutions. We'll start by drawing the advantages JanusGraph brings and its drawbacks, after that we'll compare JanusGraph to other graph database alternatives, namely Neo4j, and finally, we'll evaluate the performance and query efficiency of different JanusGraph setups.

6.1 Advantages and Drawbacks of JanusGraph

As with all databases, JanusGraph has its specific advantages and drawbacks, and understanding these is crucial to access whether this database is a suitable a good fit for a specific use case and needs.

JanusGraph advantages, as they have been mentioned in this report, can be summarized as follows:

- Scalability;
- Flexibility;
- High Performance;

- Unique Ecosystem;
- Transaction.

As for the drawbacks, JanusGraph trades its characteristic flexibility and scalability for:

- Complexity: Because it is a distributed graph database, it brings complexity in comparison to typical relational databases. Moreover, it necessitates additional configuration and maintenance of a dispersed environment, which might be difficult for some users;
- Learning Curve and Effort: Due to its uniqueness, comprehending graph data modeling, traversal, and querying principles may include a learning curve and it may take some time to comprehend JanusGraph's distinct qualities and capabilities;
- Lack of Maturity: Compared to similar databases, *Neo4j*, which was launched in 2007, JanusGraph is still considered relatively young in the market. This means it may not have the same level of maturity, community support, and extensive ecosystem of tools and libraries as more established graph databases.

6.2 Comparison to Other Graph Databases

If we want to compare JanusGaph with other graph databases, the first one that comes to our mind would probably be Neo4J.

Neo4J is, by far, the most used graph database. In contrast with JanusGraph, Neo4J uses a native graph storage that makes it easier to use and makes the processing become faster. Its community is also bigger and more active resulting in an easier and more enjoyable graph database to learn. On the other hand, while Neo4j is not tolerant to partitions, the CAP theorem can be applied to JanusGraph in different ways due to its ability to use different kinds of backends.

6.3 Benchmarking

We evaluated the performance of the three most common JanusGraph backend options, Cassandra, HBase, and Berkeley, by running the same 4 queries in all of the systems and measuring the return time. Comparing the results (Fig. 3) we realized that Berkeley was significantly slower than the others, which we didn't expect since, in contrast with the others, this one runs in the same JVM as JanusGraph.

Although Hbase was the fastest, both Hbase and Cassandra had a similar performance. All of this was expected since these two databases are very similar and the indexing and ordering of Hbase might not be very relevant in datasets of this size.

The queries used to evaluate each version are as follows:

- Query 1:

$$g.V().hasLabel(''Royals'').project(''id'', ''name'').by(T.id).by(''name'').limit(10000).toList()$$

- Query 2:

$$g.V().hasLabel(''Conflicts'').valueMap('type').dedup().toList()$$

- Query 3:

$$g.V().hasLabel(''Countries'').project(''id'', ''name'').by(T.id).by(''name'').toList()$$

- Query 4:

$$g.V().hasLabel(\text{"}Countries\text{"}).out(\text{"}participated\_in\text{"}).valueMap().toList()$$

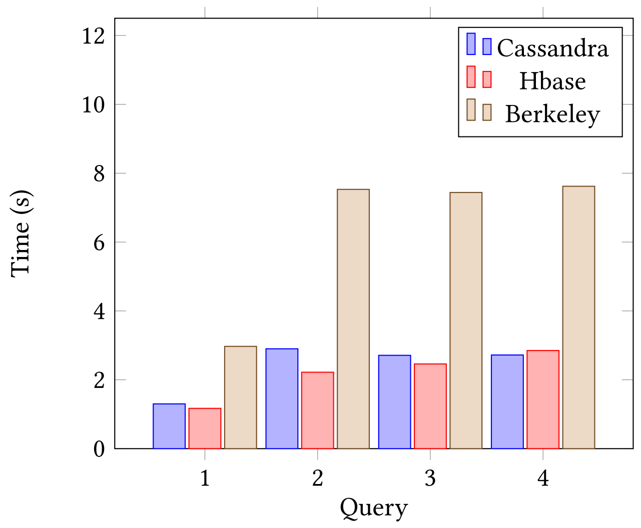


Fig. 3. Time execution of three distinct JanusGraph setups running four different queries.

7 PROTOTYPE - NETWORK OF THRONES

To test the capabilities and experiment with JanusGraph, we created a prototype named **Network Of Thrones**, a web app, supported by a backend running JanusGraph, that allows the user to visualize and query the information on the relationships, the wars, and conflicts of the Monarchy. In this section, we will explore the prototype developed and its main features.

7.1 Topic and Dataset Overview

The dataset used [4] had information on Kinship And Conflict In Europe, from 1418 to 1918, containing a large number of data relating monarchies and their conflicts. The dataset consisted of multiple .csv files that gathered relevant information for the exploration of the topic. We choose to focus on analyzing the royals, conflicts, wars, and countries, and the relationships between them, gathering relevant information from the multiple .csv files in the dataset. A detailed explanation of how we obtained the data can be seen in the next section.

7.2 Data Preparation and Implemented Data Structures

With the conceptual data model that we got from the original dataset, we aggregated some information in order to have a more concise data model that would fit better a graph database.

In this prototype, the user can access information regarding royals, conflicts, wars, and countries, which represent the entities present in the dataset. Notably, we added more information about royals by parsing a .GED file present in the data collected, from where we extracted the royals using a modified version of a Perl script also present in the original data. That allowed us to have significantly more amount of royals than we had previously (from 273 to 58674). The rest of the entities were extracted directly by parsing the given .csv and organizing them using Python scripts to obtain relevant information.

As for our data model, we organized our dataset into main entities, as mentioned before, wars, countries, conflicts, and royals. As can be seen in both the graph and conceptual models, royals are related to other royals that can rule countries, that participate in conflicts that are part of wars. This simplification over the original dataset allowed us to have a powerful but simple base to test JanusGraph with. We ended up having 58674 `royals`, 101 `wars`, 55 `countries` and 2614 `conflicts`, as well as 44314 `related_with`, 5228 `participated_in`, 426 `ruled_by` and 102 `part_of` relationships.

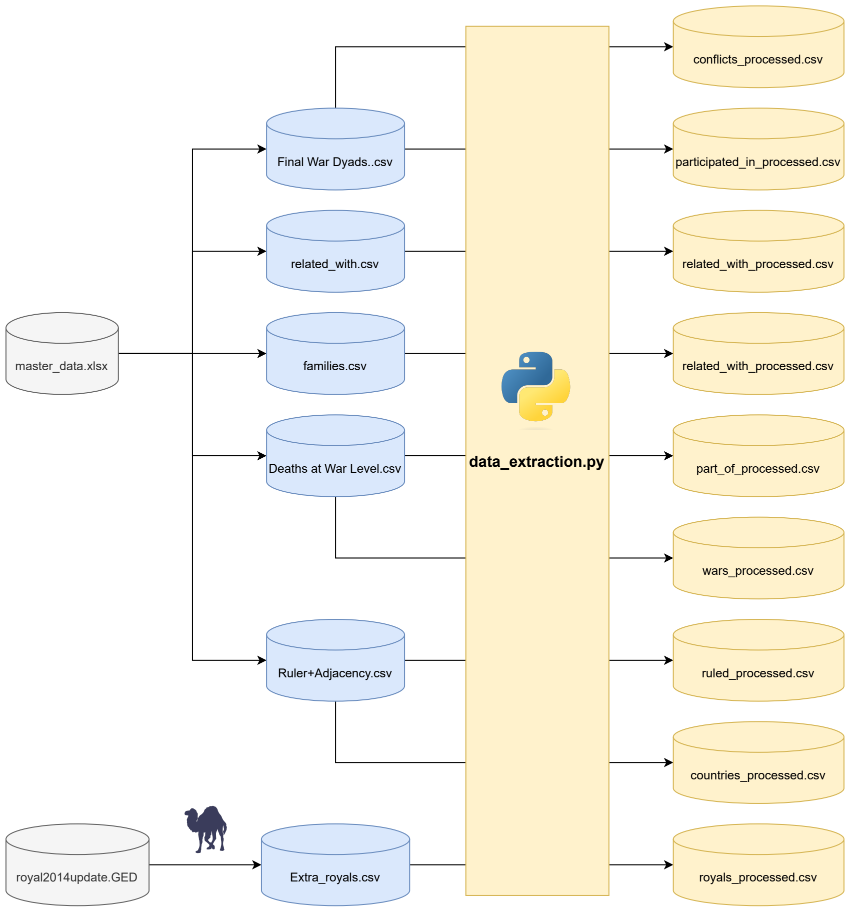
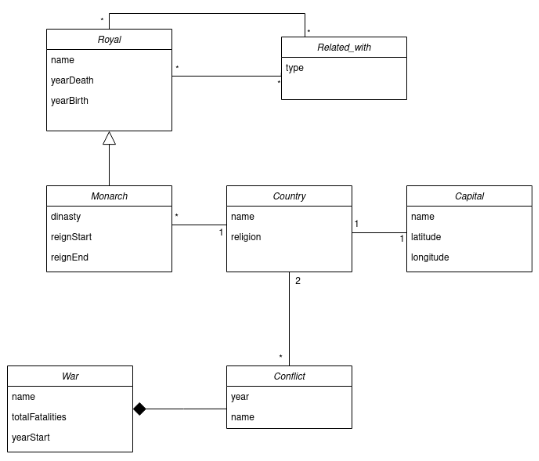
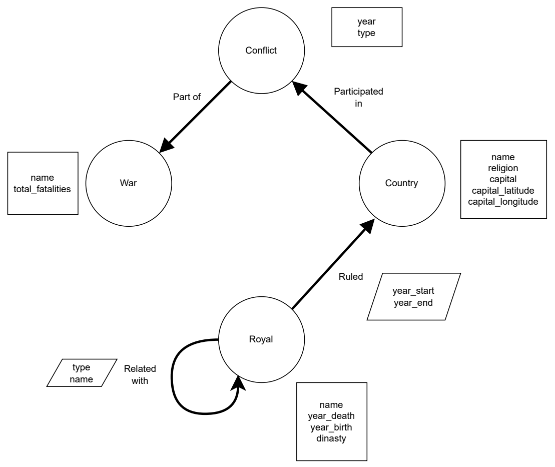


Fig. 4. Data Preparation Diagram



(a) Conceptual Data Model



(b) Graph Data Model

Fig. 5. Data Models

7.3 Prototype Architecture

The prototype is built by using a React.js frontend and a Flask backend that queries the JanusGraph containers. Finally, JanusGraph is built on top of 3 Cassandra containers, that act as the backend: 2 of the containers act as peers, and the other is a replication of the first Cassandra container, as seen in figure 6. This architecture is able to emulate, at a small scale, a cluster system. Cassandra was chosen for the backend because, even though we verified in section 6.3, that HBase had better overall execution times, Cassandra was more appropriate for the nature of our dataset, application, and objectives: we wanted to test a partitioned graph database and availability was desired over consistency, which in this particular could be neglected and would not have a significant impact due to the small scale of this project.

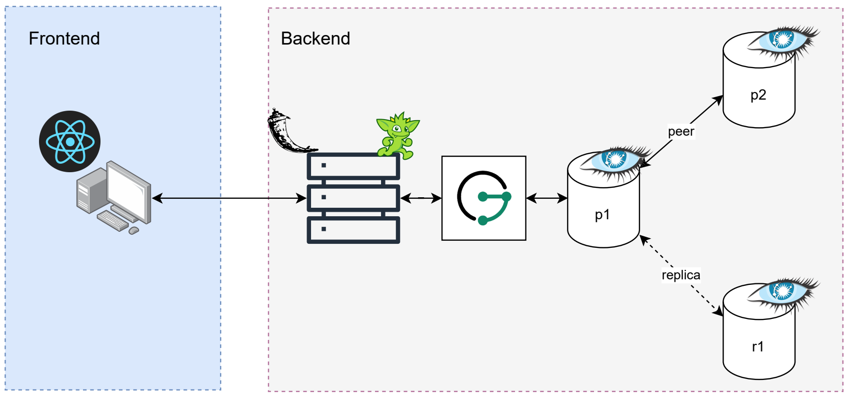


Fig. 6. Prototype physical data model

7.4 Prototype Features

The prototype developed aimed at querying the data so that relationships between both royals and countries and conflicts could be analyzed. For that, we created singular pages for each royal and country that contained special filters tailored for each entity:

- **Descendants and Ascendants by Generation and Year**: filtered royals by their descendants and ascendants up to 3 generations, or by a range of years.
- **Contemporaries**: filtered the royals that were alive at the same time as the given royal.
- **Siblings**: filtered siblings for each royal.
- **Country**: filtered conflicts by country - conflicts that both the given country and the filtered participated in at the same time
- **Type**: filtered conflicts by a given country through their specific type.
- **Year**: filtered conflicts by a given country through an year range.

The results of the filters are shown on a table next to them and we also intended to show a graph view of the nodes, but we didn't implement them fully due to time restrictions. Besides the filters, we also queried the data to obtain static information for royals and countries, like the total number of fatalities for a given country (relates to countries with wars and conflicts) or the ruling period of a monarch (relates royals to countries).

Images of the prototype can be found on the Appendix [9] and the endpoints related to each query can be found on `app.py` on the backend folder of the source code.

8 CONCLUSION

With this work, we studied in-depth a graph database, JanusGraph, not so known compared to the more popular counterparts, but of course not less important. JanusGraph comes with the particularity of being a distributed graph database, as moreover having a modular ecosystem that makes it able to integrate different backends, such as Cassandra and HBase, and indexes backends, such as ElasticSearch and Apache Solr. Although its steep learning curve and complexity, JanusGraph stands as an adaptable graph database that can be used in different scenarios, due to its flexibility.

With this study JanusGraph, we were able to understand the NoSQL paradigm and how graph databases offer a crucial way to store and access data, making it to have systems that scale better, and perform at a higher rate, essential characteristics in a time where large datasets come to be the norm.

REFERENCES

[1] [n. d.]. ScyllaDB Documentation: JanusGraph Integration. https://docs.scylladb.com/stable/using-scylla/integrations/integration-janus.html. Accessed on May 15, 2023.

[2] Bryant Avey. 2021. Labeled vs Typed Property Graphs–All Graph Databases are not the same. *Geek Culture* (14 6 2021). https://medium.com/geekculture/labeled-vs-typed-property-graphs-all-graph-databases-are-not-the-same-efdbc782f099 Accessed: May 14, 2023.

[3] Sedick Baker Effendi, Brink van der Merwe, and Wolf-Tilo Balke. 2020. Suitability of Graph Database Technology for the Analysis of Spatio-Temporal Data. *Future Internet* 12, 5 (2020). https://doi.org/10.3390/fi12050078

[4] Seth G. Benzell and Kevin Cooke. [n. d.]. A network of thrones: Kinship and conflict in Europe, 1495–1918. https://www.aeaweb.org/articles?id=10.1257%2Fapp.20180521

[5] JanusGraph contributors. 2021. JanusGraph Documentation. https://github.com/JanusGraph/janusgraph. Accessed on May 14, 2023.

[6] Google Cloud. 2021. Running JanusGraph with Bigtable. Google Cloud Architecture. https://cloud.google.com/architecture/running-janusgraph-with-bigtable Accessed: 12-05-2021.

[7] Ian Robinson, Jim Webber, and Emil Eifrem. 2015. *Graph Databases* (1st ed.). O'Reilly Media, Sebastopol, CA.

9 APPENDIX

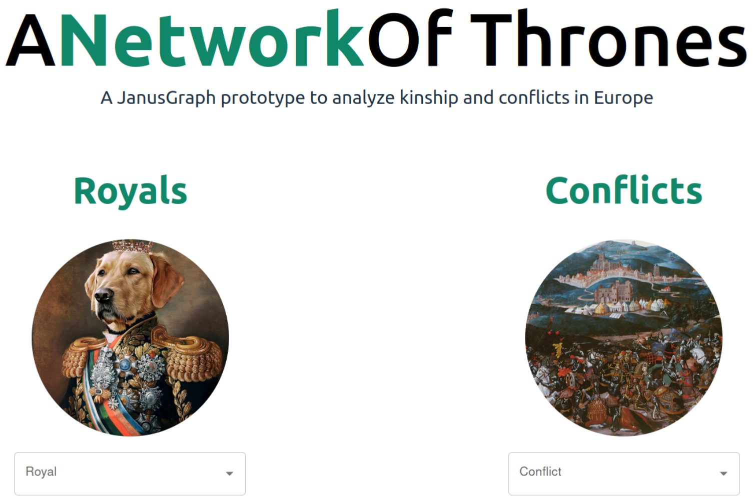
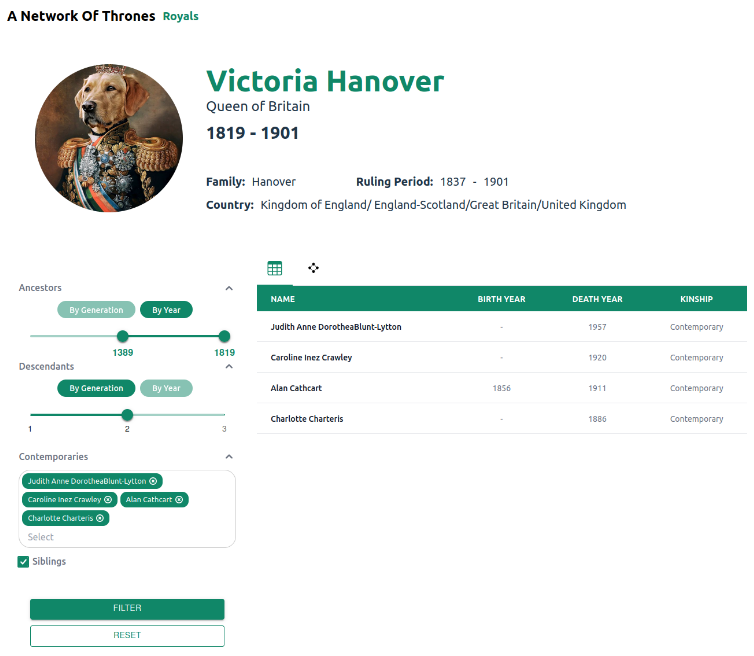

ANetworkOf Thrones
A JanusGraph prototype to analyze kinship and conflicts in Europe
Royals
Conflicts
Royal
Conflict

Fig. 7. Network of Thrones homepage. The prototype offers a way of searching through the available royals and the countries present on the dataset, in order to filter and view their information.

Received 15 May 2023


A Network Of Thrones Royals

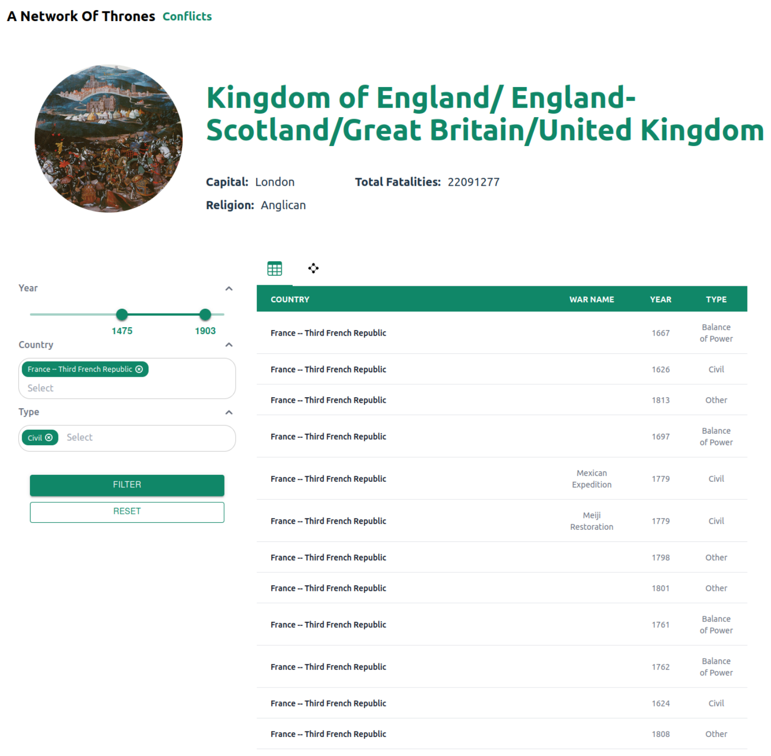
Victoria Hanover
Queen of Britain
1819 - 1901

Family: Hanover Ruling Period: 1837 - 1901
Country: Kingdom of England/ England-Scotland/Great Britain/United Kingdom

NAME	BIRTH YEAR	DEATH YEAR	KINSHIP
Judith Anne DorotheaBlunt-Lytton	-	1957	Contemporary
Caroline Inez Crawley	-	1920	Contemporary
Alan Cathcart	1856	1911	Contemporary
Charlotte Charteris	-	1886	Contemporary

Ancestors: By Year, 1389 – 1819
Descendants: By Generation, 2
Contemporaries: Judith Anne DorotheaBlunt-Lytton, Caroline Inez Crawley, Alan Cathcart, Charlotte Charteris
✓ Siblings

Fig. 8. Royals' Page. Each royal page offers filters that parse the information regarding how royals relate with each other: the ancestors can be filtered by year or generation, as well as the descendants, their contemporaries are highlighted as well as the royals' siblings.



A Network Of Thrones Conflicts

Kingdom of England/ England-Scotland/Great Britain/United Kingdom

Capital: London

Total Fatalities: 22091277

Religion: Anglican

Year

1475 1903

Country

France -- Third French Republic

Select

Type

Civil

Select

COUNTRY	WAR NAME	YEAR	TYPE
France -- Third French Republic		1667	Balance of Power
France -- Third French Republic		1626	Civil
France -- Third French Republic		1813	Other
France -- Third French Republic		1697	Balance of Power
France -- Third French Republic	Mexican Expedition	1779	Civil
France -- Third French Republic	Meiji Restoration	1779	Civil
France -- Third French Republic		1798	Other
France -- Third French Republic		1801	Other
France -- Third French Republic		1761	Balance of Power
France -- Third French Republic		1762	Balance of Power
France -- Third French Republic		1624	Civil
France -- Third French Republic		1808	Other

Fig. 9. Conflicts' Page. Each country page offers filters that parse the information regarding the conflicts countries have had with each other. They can be filtered by year of conflict, by country that participated in the same conflict or by type of conflict.