SoftwareX 32 (2025) 102333

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

OntologyGen: A smart software for automatic ontology generation from MongoDB using Formal Concept Analysis

Elmehdi Elguerraoui [a], Omar Boutkhoum [a], Mohamed Hanine [b], Waeal J. Obidallah [c,*]

[a] LAROSERI Laboratory, Department of Computer Science, University of Chouaib Doukkali, El Jadida, Morocco
[b] LTI Laboratory, National School of Applied Sciences, Chouaib Doukkali University, El Jadida, Morocco
[c] College of Computer and Information Sciences, Imam Mohammad Ibn Saud Islamic University (IMSIU), Riyadh, 11673, Saudi Arabia

ARTICLE INFO

Keywords:
Ontology engineering
MongoDB
Formal concept analysis
OWL ontology
Automated context generation
OntologyGen

ABSTRACT

OntologyGen is a web-based framework that automates OWL ontology generation from MongoDB databases, using Formal Concept Analysis (FCA). Built with Python and Django, It extracts a formal context from NoSQL data, builds concept lattices, and applies rule-based mappings to produce OWL ontologies. OntologyGen offers an interactive graphical interface that requires less user involvement, allows the user to extract semantic structures from schema-flexible data, and then builds OWL ontologies that can be used with other existing tools. By using two publicly available MongoDB datasets of varying complexity, the framework's usability and efficacy were established, with a subsequent assessment of performance metrics including execution time, memory footprint, and ontology size. It was concluded that OntologyGen represents a considerable opportunity to reduce the difficulty of ontology engineering for data scientists and domain experts, while also providing scalability, interoperability, and extensibility beyond the current implementation with other NoSQL systems or possible future ontology learning extensions.

Code metadata

Current code version	1.0.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-25-00318
Permanent link to Reproducible Capsule	https://github.com/Med-Gon/OntologyGen/blob/main/ReadMe.md#video-demonstration
Legal Code License	MIT License
Code versioning system used	Git
Software code languages, tools, and services used	Python, Django, Docker, HTML, CSS, JavaScript
Compilation requirements, operating environments & dependencies	Python 3.11+, Django 4.2+, Docker, MongoDB, Web Browser (Chrome/Firefox/Edge)
If available, link to developer documentation/manual	https://github.com/Med-Gon/OntologyGen/blob/main/ReadMe.md
Support email for questions	elguerraoui.elmehdi@ucd.ac.ma

1. Introduction

As organizations are increasingly looking to NoSQL document-oriented databases, there are new challenges for semantic data modeling and ontology engineering. In contrast to more traditional relational systems, NoSQL databases, such as MongoDB, are schema-less, therefore providing flexibility in your data structure at the cost of formal representation. With the increasing amount of loosely structured data organizations are collecting, there is a need for tools that can automatically convert that data into formal machine-readable ontologies.

As organizations are increasingly adopting NoSQL document-oriented databases there also comes new challenges in terms of semantic data modeling and ontology engineering. NoSQL databases such as MongoDB are schema-less as compared to more traditional relational systems. This schema-less architecture allows users' flexibility with less formal representation of their data. With the growing amount of loosely structured data organizations are collecting, it is important

* Corresponding author.
E-mail address: wobaidallah@imamu.edu.sa (W.J. Obidallah).

https://doi.org/10.1016/j.softx.2025.102333
Received 15 May 2025; Received in revised form 22 August 2025; Accepted 26 August 2025
Available online 18 September 2025

to acquire tools that can automatically convert that data into formal machine-readable ontologies.

OntologyGen has a user interface that collaborates towards simplicity and interpretability. The intermediate stages of the transformation from raw data into semantic knowledge are visualized, guiding the users through the structural changes. Generated OWL files can immediately be used in standard ontology tools, which can enable downstream work including reasoning, querying and integration.

This papers describes the design, and implementation, of OntologyGen, including the core methodology and system architecture. The tool is evaluated on two real-world MongoDB datasets with different complexities to evaluate its performance, expressiveness, and generalizability. Also included is a discussion of the strengths and weaknesses of the tool along with directions for future development.

The rest of this paper consists of the following sections: Section 2 discusses prior related work concerning ontology learning and FCA techniques. Section 3 explains the motivation and importance of developing OntologyGen. Section 4 gives background on Formal Concept Analysis (FCA). Section 5 describes the software architecture and key features. Section 6 describes the sequential workflow of the tool. Section 7 presents an example from a live MongoDB dataset. Section 8 evaluates a number of datasets and evaluation criteria of the system. Section 9 discusses broader implications and positional comparisons about OntologyGen. Finally, Section 10 will summarize conclusions and directions for future work.

2. Related work

Ontology engineering and database mining have long been studied in both relational and NoSQL contexts. Early work reversed relational schemas into ontologies, such as Chiang et al. [1], who recovered Enhanced Entity–Relationship (EER) models, and Ding and Feng [2], who transformed relational structures into OWL via schema matching. These methods work for structured data but are less effective for heterogeneous NoSQL sources.

For MongoDB, Jabbari and Stoffel [3] extracted field types to form context attributes, while Abbes et al. introduced M2Onto [4], mapping document structures to OWL classes and properties. Such systems generated ontologies but lacked reasoning based on Formal Concept Analysis (FCA).

FCA, formalized by Ganter and Wille [5], derives class hierarchies from object–attribute relations. Fu [6] and Baader & Sertkaya [7] applied FCA to ontology generation and description logics, while Sertkaya [8] highlighted its value for DL-based models. Other tools, like CB2Onto [9] for Couchbase and C2Ont [10] for Cassandra, automate ontology learning but rely on fixed schemas. Distributed FCA methods—RD-FCA by Khaund et al. [11] and Spark-based approaches by Baixeries et al. [12]—improve scalability but target large-scale processing rather than interactive prototyping. Hassan et al. [13] enriched FCA-based ontologies with WordNet semantics without extra complexity. FCA-Merge by Stumme & Maedche [14] used a bottom-up strategy to merge ontologies, unlike OntologyGen, which builds new ones from schema-less data.

Extensions of FCA expand its scope, including fuzzy FCA for graded attributes [15], relational context families for complex relational data [16], and fuzzy relational context families [17]. These represent future directions for enhancing OntologyGen.

To the best of our knowledge, OntologyGen is the only lightweight, web-based FCA-driven framework for MongoDB. It integrates context extraction, lattice generation, and mapping rules into a transparent pipeline, making ontology building accessible for education, experimentation, and small- to medium-scale use.

3. Motivation and significance

NoSQL systems such as MongoDB challenge semantic integration due to their semi-structured nature, complicating automatic ontology generation. Few tools address this with a clear, user-friendly solution.

OntologyGen was developed to let non-experts transform MongoDB instances into OWL ontologies through a simple web interface, without scripting or ontology editors. The workflow supports dataset upload, context extraction, and OWL export in a transparent, stepwise process.

OntologyGen was developed to let non-experts transform MongoDB instances into OWL ontologies through a simple web interface, without scripting or ontology editors. The workflow supports dataset upload, context extraction, and OWL export in a transparent, stepwise process.

By abstracting technical complexity, OntologyGen serves a wide user base—from engineers to educators—providing a lightweight framework for experimentation, learning, and semantic modeling in NoSQL contexts.

4. Formal Concept Analysis (FCA) background

Formal Concept Analysis (FCA) is a mathematical framework based on lattice and order theory for data mining and knowledge representation. Introduced by Ganter and Wille (1999), it identifies structured relations between objects and their attributes, supporting the systematic discovery of conceptual hierarchies and dependencies.

A formal context in FCA is a triple (G, M, I), where:

- G is a set of objects.
- M is a set of attributes.
- $I \subseteq G \times M$ indicates which objects have which attributes.

From this formal context, FCA constructs formal concepts, each represented as a pair (A, B), where:

- A is the set of all objects (extent) sharing the attributes in B.
- B is the set of attributes (intent) common to all objects in A.

Concepts are partially ordered by inclusion, forming a concept lattice that inherently captures hierarchical relationships. This structure underpins the derivation of ontological constructs such as classes, subclasses, and properties. Key advantages of FCA include:

- Automatic discovery of latent hierarchies in data.
- Representation of disjointness and subsumption without manual intervention.
- Handling incomplete or sparse attributes, useful for semi-structured databases like MongoDB.

In OntologyGen, FCA transforms MongoDB data into a semantic structure: the tool extracts the formal context from schema and sample data, builds the concept lattice, and maps it into OWL ontology constructs. This ensures a robust, explainable, and consistent ontology generation process.

5. Software description

OntologyGen is developed as a lightweight, web-based system to automate the creation of OWL ontologies from MongoDB databases using Formal Concept Analysis (FCA). The application is intended to be simple, lightweight, and extensible, empowering users, irrespective of their technical abilities, to generate semantic representations as effortlessly as possible. This section describes the overall architecture, implementation stack, and core components of the tool.

5.1. Architecture overview

OntologyGen is based on a three-tiered architecture within a Django web framework:

- **Presentation Layer** : A web front end application that allows the user to import data from MongoDB, visualize the formal context and concept lattices, and export as OWL. Developed in Django means it runs in any web browser, and provides a simple multi-user experience with no installation required.
- **Business Logic Layer** : Implemented in Python, where the FCA reasoning, lattice generation, and rule-based construction occurs. The concepts are derived from Ganter's algorithm [5] and mapped to the OWL classes, properties, and datatypes.
- **Data Access Layer** : Where the MongoDB connectivity happens, schema reading occurs, and formal context is built using PyMongo, exposing the logic layer from the data layer.

The modular approach provides easy maintainability, extensibility, and integration with other databases.

5.2. Functionalities

OntologyGen features the following central functionality in its web interface:

- **Database Integration**: Connects to MongoDB via a URI connection. Collections will be listed for navigation, allowing the user to inspect the collections before generating the ontology.
- **Automated Context Generation**: Extracts binary relations between collections and attributes that can form the formal context table. Collections will form as rows and attributes will be used as columns. Attributes that belongs to a collection will be marked with an "X", which forms at the end a table of binary relations between collections and attributes.
- **Concept Lattice Generation**: After FCA processing to generate the concept lattice from the context, all extents and intents along with their hierarchical relations will be a part of the lattice structure. The concept lattice will be available for clustering and visualization.
- **Ontology Construction Using Mapping Rules**: OntologyGen comes with predefined navigation rules that will transform the concepts from the lattice to OWL classes, properties and datatypes, ensuring consistent mapping.
- **Export Options**: The output ontology can be saved as an OWL file and reused in editors like Protégé [18]. OntologyGen has been tested on datasets of varying size, proving its scalability while preserving consistency.

Together, these features transform raw MongoDB data into a structured lattice and OWL ontology with just a few steps, offering a complete end-to-end workflow.

To assess the scalability of OntologyGen, we gathered measurements of execution time, memory use, and ontology size against two datasets of greater complexity. The outcomes, appearing in the Evaluation section, illustrate the efficiency and scalability of the system.

6. Workflow steps

The OntologyGen tool follows a linear workflow from database import to ontology export, hiding the underlying semantic processing but ensuring each step complies with semantic web engineering conventions. The process translates a large MongoDB instance into an OWL ontology in six key stages, each mirrored in the web interface for structured execution.

1. **Database Connection and Import :** Users connect to MongoDB via a URI string. Once connected, the tool retrieves and visualizes the schema.
2. **Formal Context Generation :** After schema loading, OntologyGen creates a binary context table with entities as rows and attributes as columns, marking presence with an "X."
3. **Concept Lattice Generation and Visualization :** FCA is applied to derive a lattice, where each node groups entities with shared attributes. The lattice organizes concepts hierarchically by generalization and specialization.
4. **Ontology Mapping and Class Hierarchy Generation :** Based on the lattice, predefined FCA rules translate concepts into OWL classes, properties, and hierarchies:

Rule 1: Single-Entity Class Generation

- Condition: If a formal concept C_i has one element in its extensional set ($|C_i| = 1$), OntologyGen creates an OWL class for that entity.
- Example: In a sample database, concepts like ['*Artist*'] and ['*Album*'] are mapped to distinct OWL classes.
- OWL Syntax:

```
<owl:Class rdf:ID="Artist"/>
<owl:Class rdf:ID="Album"/>
```

Rule 2: Direct Object Relationship Binding

- Condition: If C_i contains two elements ($|C_i| = 2$) and is directly linked to the root concept (Thing), an object property is created between the two entities.
- Direct linkage: No intermediate concepts exist between C_i and Thing.
- Example: The concept [Artist, Album] generates an object property $ArtistAlbumLink$ with [*Artist*] as the domain and [*Album*] as the range.
- OWL Syntax:

```
<owl:ObjectProperty rdf:ID="ArtistAlbumLink">
  <rdfs:domain rdf:resource="#Artist"/>
  <rdfs:range rdf:resource="#Album"/>
</owl:ObjectProperty>
```

Rule 3: Intermediate Class Creation

- Condition: If C_i contains two elements $|C_i| = 2$) but is indirectly linked to Thing (via intermediate concepts), OntologyGen creates a new composite class.

Rule 4: Attribute-to-Property Conversion

- Condition: Attributes in the intentional part of a concept C_i (e.g., ArtistId, Name) are mapped to datatype properties of the corresponding ontological class.
- OWL Syntax:

```
<owl:DatatypeProperty rdf:ID="ArtistId">
  <rdfs:domain rdf:resource="#Artist">
  <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>
```

Rule 5: Hierarchical Inheritance Derivation

- Condition: Subclass relationships are inferred from the concept lattice. If C_i is a direct sub-concept of C_j, OntologyGen assigns C_i as a subclass of C_j.
- Example: Artist becomes a subclass of Thing.
- OWL Syntax:

```
<owl:Class rdf="Artist">
  <rdfs:subClassOf rdf:resource="#thing"/>
</owl:Class>
```

Rule 6: Class Disjointness Enforcement

- Condition: All generated classes are inherently disjoint. Formal concepts in FCA ensure no overlap between extensional sets, guaranteeing ontological disjointness.

Rule 7: Bidirectional Property Inference

- Condition: For every object property P_{ij}, OntologyGen automatically generates its inverse P_{ji}
- Example: AlbumToTrack is inferred as the inverse of TrackToAlbum.
- OWL Syntax:

```
<owl:ObjectProperty rdf:ID="AlbumToTrack">
<owl:inverseOf rdf:resource="#TrackToAlbum"/>
</owl:ObjectProperty>
```

Rule 8: Cardinality Constraint Assignment

- Condition: Datatype properties are assigned cardinality bounds based on null value occurrences.
- Example: If ArtistId has no null values, it receives minCardinality=1.
- OWL Syntax:

```
<owl:Restriction>
   <owl:onProperty rdf:resource="#ArtistId"/>
   <owl:minCardinality>1</owl:minCardinality>
   <owl:maxCardinality>1</owl:maxCardinality>
</owl:Restriction>
```

Rule 9: Nested Entity Individualization

- Condition: Attributes with nested object values are mapped to sub-classes or individuals.
- Example: A nested Address document becomes an individual of a class Address linked to its parent entity.

The current rule set supports standard object and datatype property translation, class disjointness, cardinality detection and individuals extractions. Future work includes extending rule coverage for richer semantic constructs and edge cases.

5. **Ontology Graph Visualization :** The tool depicts the ontology as a graph of class hierarchies and relationships. Solid arrows denote subclass links; dashed arrows denote object properties.
6. **OWL Ontology Export :** Finally, the ontology is exported in OWL (RDF/XML) format, with an option to download for reuse in external tools.

OntologyGen thus produces a structured ontology that represents MongoDB data while leveraging FCA reasoning for mapping and hierarchy generation. Workflow robustness was validated on two datasets: Chinook (medium complexity) and Sample Training (larger, diverse). Results are detailed in the Evaluation section.

7. Illustrative example

To assess OntologyGen's practical value, we applied the workflow to the Chinook MongoDB database, a digital music store with data on artists, albums, tracks, customers, invoices, and related entities. This case study illustrates the end-to-end process and resulting ontology.

We began by importing a JSON dump of Chinook. OntologyGen connected and identified 11 collections (Fig. 1). It then generated the formal context table, listing attributes per collection—for example, Artist with ArtistId and Name; Album with AlbumId, Title, ArtistId; and Track with TrackId, Name, AlbumId. Shared fields such as Id, Name, and foreign keys (CustomerId, EmployeeId) are visible in the matrix view (Fig. 2).

The tool next constructed the concept lattice, showing hierarchical relations. Album and Track group under AlbumId, while Customer and Employee share address-related attributes. The most specific concepts appear at the bottom, while the top concept has no attributes common to all collections (Fig. 3).

OntologyGen then mapped FCA findings into ontology constructs: 11 OWL classes plus the root Thing, object properties for detected relations (e.g., AlbumToTrack, CustomerToInvoice), and datatype properties for attributes like Name, Title, and Email. Subclass relations also emerged, such as Genre and MediaType under Track (Fig. 4).

The generated ontology graph confirmed these relationships, with Thing at the top and domain classes beneath; connections such as InvoiceLineToInvoice and AlbumToTrack are clearly represented (Fig. 5).

Finally, the system exported an OWL file containing all classes and properties with correct domains and ranges. Imported into Protégé [18], the ontology rendered properly and passed consistency checks. Listing 1 shows a preview of the file.

Listing 1: Excerpt from the generated OWL file (Chinook dataset).

```
<?xml version="1.0"?>
<!DOCTYPE Ontology [
  <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
]>
<Ontology xmlns="http://www.w3.org/2002/07/owl#"
  xml:base="http://test.org/chinook.owl" >

  <Declaration><ObjectProperty IRI="#TrackToAlbum"/></Declaration>
  <Declaration><DataProperty IRI="#Name"/></Declaration>
  <Declaration><Class IRI="#Artist"/></Declaration>

  <ClassAssertion>
    <Class IRI="#Artist"/>
    <NamedIndividual IRI="#Artist_252"/>
  </ClassAssertion>
  <DataPropertyAssertion>
    <DataProperty IRI="#Name"/>
    <NamedIndividual IRI="#Artist_252"/>
    <Literal datatypeIRI="&xsd;string">Amy Winehouse</Literal>
  </DataPropertyAssertion>

  <!-- ... truncated ... -->
</Ontology>
```

This example demonstrates how OntologyGen converts a multi-collection MongoDB dataset into a structured ontology that captures hierarchies and inter-collection relationships with minimal user interaction.

8. Evaluation

To evaluate the effectiveness, robustness, and scalability of OntologyGen, it was applied to two MongoDB datasets that had different complexity and structure: the Chinook database, and the more complex Sample Training dataset provided by MongoDB Atlas.

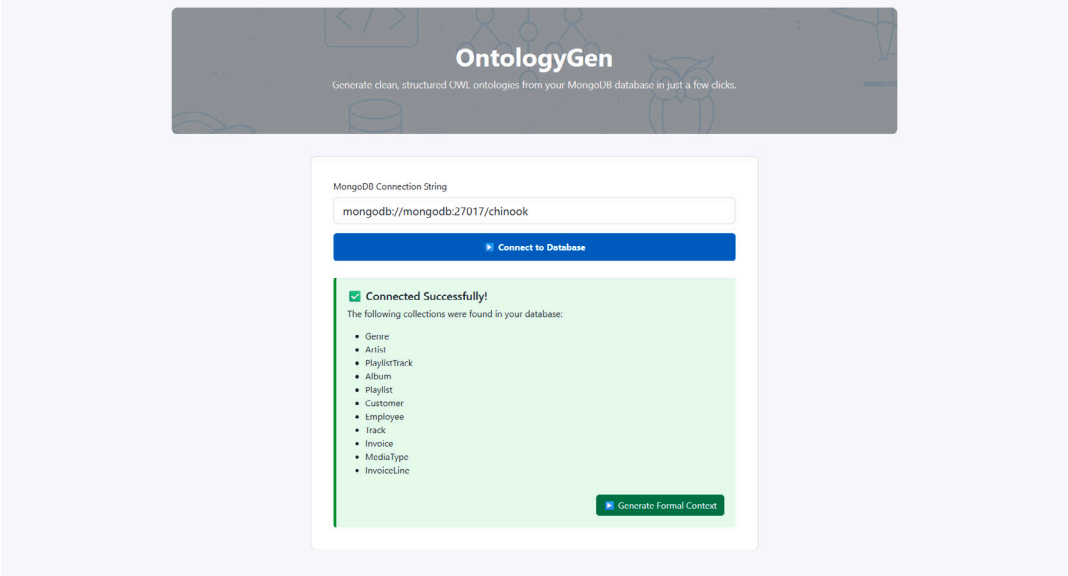
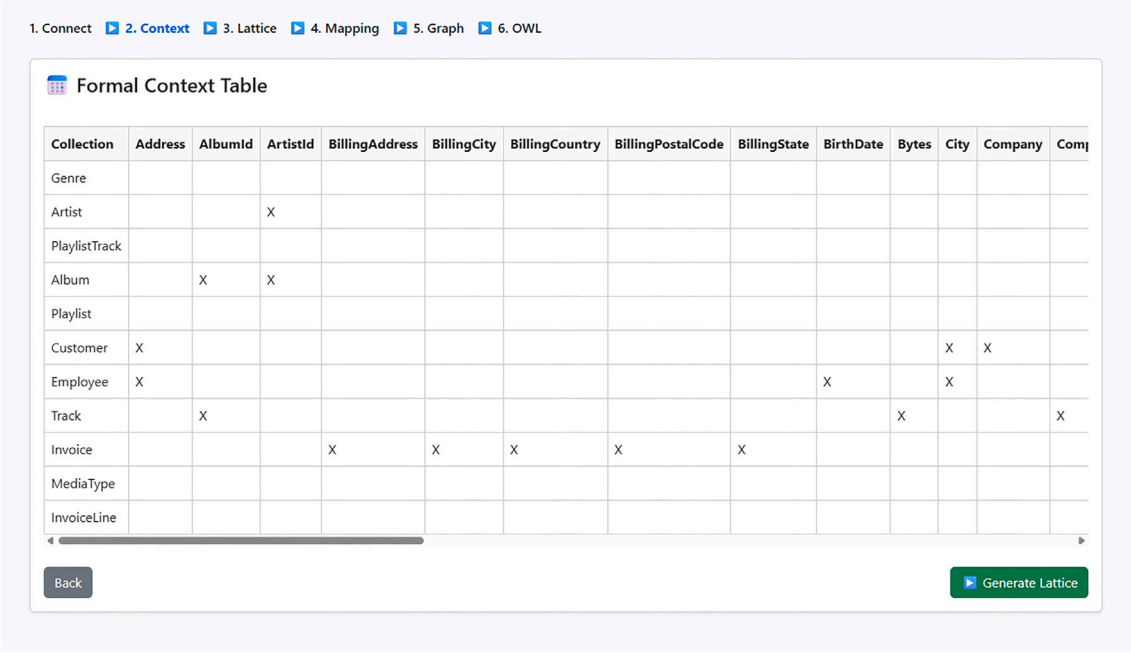


Fig. 1. Display of detected MongoDB collections.



Collection	Address	AlbumId	ArtistId	BillingAddress	BillingCity	BillingCountry	BillingPostalCode	BillingState	BirthDate	Bytes	City	Company	Comp
Genre													
Artist			X										
PlaylistTrack													
Album		X	X										
Playlist													
Customer	X										X	X	
Employee	X								X		X		
Track		X								X			X
Invoice				X	X	X	X	X					
MediaType													
InvoiceLine													

Fig. 2. Generated formal context table.

8.1. Dataset overview

- **Chinook Dataset**: A structured music-related dataset with 11 collections and about 9,000 documents, including schemas for Artists, Albums, and Tracks.
- **Sample Training Dataset**: A larger, semi-structured business dataset with 9 collections (e.g., customers, orders, invoices), over 40,000 documents, and mixed schemas modeling real-world applications.

8.2. Evaluation objectives

The evaluation focuses on the following dimensions:

- **Ontology Accuracy**: measuring the fidelity of the OWL file to the underlying MongoDB schema;
- **Ontology Complexity**: numerically representing the complexity of the OWL meaning the number of classes, individuals, datatype properties, object properties, and disjoint axioms generated;
- **Performance Metrics**: measuring execution time, memory, and the size of the generated OWL file;

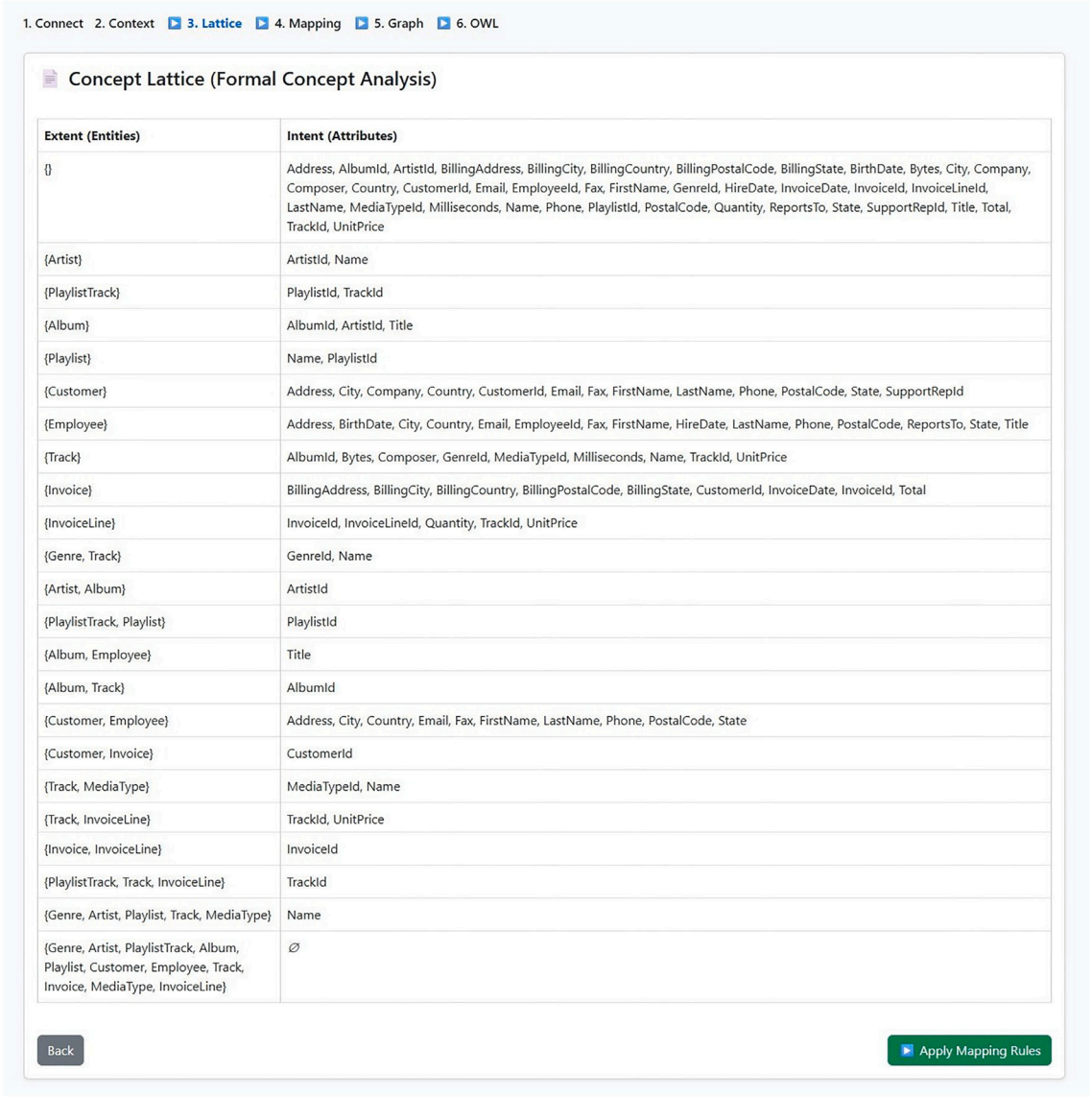


Fig. 3. Generated Concept lattice (extents and intents).

Table 1
Performance comparison of OntologyGen on two MongoDB datasets.

Metric	Chinook dataset	Sample training dataset
Generated Classes	11	9
Individuals Generated	9,312	43,271
Object Properties	14	10
Datatype Properties	39	101
Execution Time	7.27 sec	26.36 sec
Memory Usage	0.29 MB	705.84 MB
CPU Usage	0.0%	0.0%
Ontology File Size	3,956.13 KB	88,260.66 KB

- **Tool Robustness**: the tool's performance in inconsistent schema and in the presence of nested/noisy structures.

8.3. Comparative results

All tests were run on a local machine with 32 GB RAM and Intel i7 processor. Measurements were collected using Python's time and tracemalloc modules integrated into the web application.

8.4. Performance metrics analysis

To evaluate the scalability and performance of OntologyGen, we examined performance on two data sets of varying size and complexity (Chinook database and Sample Training). We examined execution time, memory used, CPU usage, and OWL ontology file size.

As shown in Table 1 and Fig. 6, OntologyGen scales linearly, and it shows efficient performance across data sets of different sizes. For the Chinook dataset the ontology generation process was completed in 7.27 s, using 0.29 MB of memory (marginal overhead), and generating an ontology of 3956.13 KB. Meanwhile, when applied to a much larger data set (Sample Training) the system completed in 26.36 s, consuming 705.84 MB of memory, and generated an ontology of 88260.66 KB. In both cases, OntologyGen recorded 0.0% CPU usage, suggesting it is ultimately memory-bound, and the tool is able to run on standard computing environments.

The performance chart (Fig. 6) demonstrates a clear indication of the proposed system's scalability. The ontology size increase and memory consumption between datasets is linear against the more structural complexity and volume of documents. The growth rates of all aspects of OntologyGen confirms its ability to efficiently handle both light-weight,

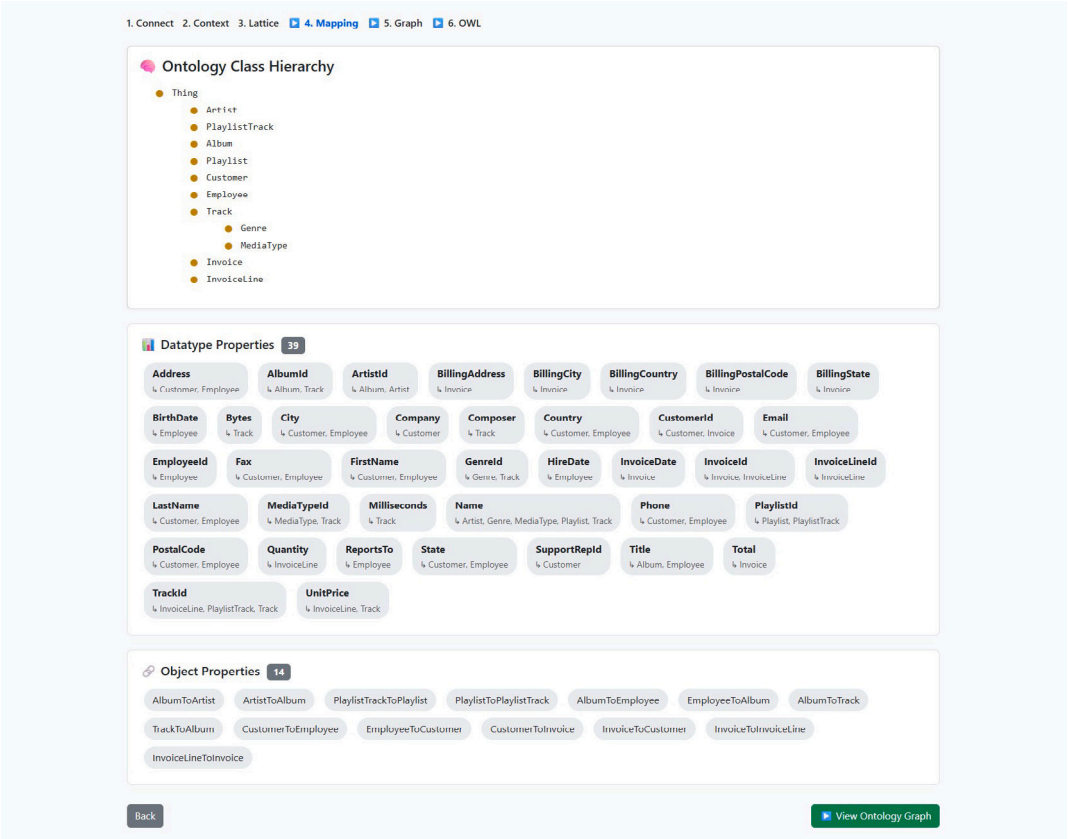


Fig. 4. Generated ontology class hierarchy, object and datatype properties.

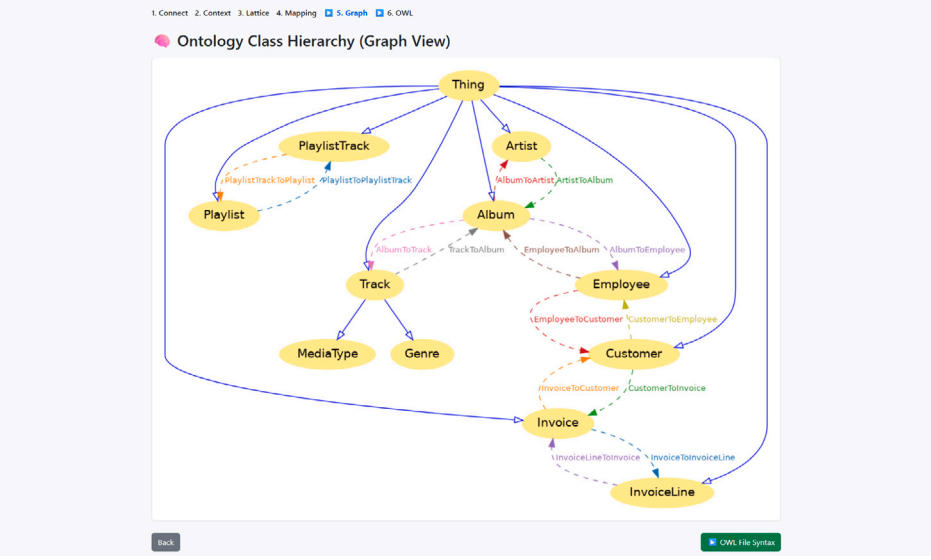


Fig. 5. Ontology graph visualization.

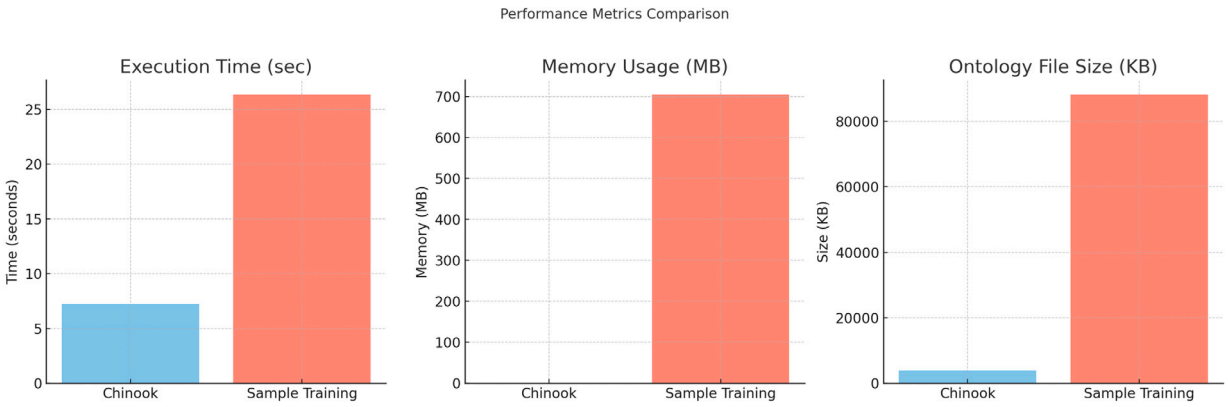


Fig. 6. Performance metrics comparison.

and mid-sized datasets, with no measurable degradation in system responsiveness and stability. The findings further substantiate the tool's feasibility for a practical usage in any scenario where schema-less, semi-structured data is commonplace.

9. Impact and discussion

OntologyGen offers a practical framework to generate OWL ontologies from unstructured or semi-structured MongoDB data. By combining Formal Concept Analysis (FCA) with rule-based mapping, it enables users without ontology engineering expertise to extract semantic structures from schemaless NoSQL databases.

We evaluated OntologyGen on two datasets: the Chinook database and the larger Sample Training dataset (> 40,000 documents in 9 collections). In both cases, OntologyGen produced valid OWL ontologies with class hierarchies, datatype and object properties, relationships, and instances. Performance metrics (memory, execution time, ontology size) confirmed robustness across varying data scales.

Compared to Cb2Onto [9] and C2Ont [10]—platform-specific and schema-dependent—OntologyGen provides a flexible web-based solution for any NoSQL source. FCA's pattern-based inference supports adaptive, data-driven ontology construction. Its interactive "what you see is what you get" design makes results accessible to researchers, professionals, and non-experts with limited Semantic Web knowledge.

Transparency is enhanced: users can trace the process from database import to OWL export, with visibility into intermediate results (formal context, lattice, rules, OWL preview). This benefits both automation and manual refinement.

Ontologies are OWL-compliant and usable with tools such as Protégé [18], that enables the reasoning and semantic checking capabilities to be used. An open-source project (MIT license) allows for customization for research and commercial use and can be augmented by fuzzy FCA [19], distributed computation [20], and relationship mappings.

To conclude, OntologyGen provides the sweet spot for automation, transparency, and extensibility, not for rigid schemas, but rather the ontology can create knowledge from the data itself therefore, the possibility for the transformation can cover a wide selection of domains. OntologyGen provides a flexible pipeline for transforming NoSQL data into structured, semantic knowledge, provided with the opportunity to foster more widespread Semantic Web usage.

10. Conclusion and future work

OntologyGen is a simple and lightweight web-based approach to ontology generation from NoSQL databases using Formal Concept Analysis (FCA). Our approach is an end-to-end workflow process that ranges from context extraction and exported into OWL. We have validated our approach on a small dataset (Chinook) and a larger dataset (Sample Training) with a focus on usability and scalability.

The main contributions of OntologyGen are:

- Automatic ontology learning from MongoDB, an underserved data model in ontology engineering.
- An FCA-driven method to derive concept hierarchies and relations.
- An intuitive web interface lowering barriers for non-experts.
- Standard OWL outputs enabling Semantic Web interoperability.

Future improvements include enhancements to the mapping rules for more flexibility, machine learning that would aid in inferring and/or pruning relations, and support for more databases. Advanced FCA extensions would also improve the overall adaptability of OntologyGen. Examples include fuzzy FCA [15], relational context families [16], and fuzzy relational context families [17].

OntologyGen is connecting schema-less NoSQL data structure with semantic representations, and it is an open and transparent platform that has the potential to become a reference tool assisting in ontology learning/processes in different domains.

CRediT authorship contribution statement

Elmehdi Elguerraoui: Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis, Data curation. **Omar Boutkhoum:** Writing – review & editing, Resources, Methodology, Formal analysis, Conceptualization. **Mohamed Hanine:** Writing – review & editing, Validation, Project administration. **Waeal J. Obidallah:** Writing – review & editing, Project administration, Methodology, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported and funded by the Deanship of Scientific Research at Imam Mohammad Ibn Saud Islamic University (IMSIU) (grant number IMSIU-DDRSP2504).

References

[1] Chiang RHL, Barron TM, Storey VC. Reverse engineering of relational databases: extraction of an EER model from a relational database. Data Knowl Eng 1994;12(2):107–42. http://dx.doi.org/10.1016/0169-023X(94)90011-6.

[2] Ding Ying, Feng Zhiyong. OWL ontology extraction from relational databases via database reverse engineering. In: Proceedings of the 3rd international conference on knowledge science, engineering and management. Berlin, Heidelberg: Springer; 2014, p. 145–55, https://www.jsoftware.us/index.php?m=content&c=index&a=show&catid=97&id=2017.

[3] Jabbari S, Stoffel K. Ontology extraction from mongodb using formal concept analysis. In: Proc. 2nd int. conf. knowledge engineering and applications. 2017, p. 178–82. http://dx.doi.org/10.1109/ICKEA.2017.8169925.
[4] Abbes Hanen, Gargouri Faiez. M2onto: an approach and a tool to learn OWL ontology from mongodb database. In: Proceedings of the international conference on model and data engineering. Cham: Springer; 2017, p. 524–31. http://dx.doi.org/10.1007/978-3-319-53480-0_60.
[5] Ganter Bernhard, Wille Rudolf. Formal concept analysis: mathematical foundations. English ed. Springer; 2012, http://dx.doi.org/10.1007/978-3-642-59830-2.
[6] Fu G. FCA based ontology development for data integration. Inf Process Manage 2016;52(5):765–82. http://dx.doi.org/10.1016/j.ipm.2016.02.003.
[7] Baader F, Sertkaya B. Applying formal concept analysis to description logics. In: Eklund P, editor. Concept lattices, proc. 2nd int. conf. formal concept analysis. Lecture notes in computer science, vol. 2961, Berlin: Springer; 2004, p. 261–86. http://dx.doi.org/10.1007/978-3-540-24651-0_24.
[8] Sertkaya B. A survey on how description logic ontologies benefit from formal concept analysis. 2011, http://dx.doi.org/10.48550/arXiv.1107.2822, arXiv:1107.2822.
[9] Mhammedi S, el Massari H, Gherabi N. Cb2Onto: OWL ontology learning approach from couchbase. In: Advances in intelligent systems and computing. Cham: Springer; 2021, p. 95–110. http://dx.doi.org/10.1007/978-3-030-72588-4_7.
[10] Soe Nang Kham, Yee Tin Tin, Htoon Ei Chaw. C2Ont: An OWL ontology learning approach from apache cassandra. In: 2019 6th international conference on advanced information technology and communication. IEEE; 2019, p. 212–7. http://dx.doi.org/10.1109/AITC.2019.8921025.
[11] Khaund A, et al. RD-FCA: A resilient distributed framework for formal concept analysis. J Parallel Distrib Comput 2023;179:104710. http://dx.doi.org/10.1016/j.jpdc.2023.04.011.
[12] Baixeries J, et al. Scalable formal concept analysis algorithms for large datasets using spark. J Ambient Intell Humaniz Comput 2020;11(3):1105–22. http://dx.doi.org/10.1007/s12652-018-1105-8.
[13] Hassan Saad, Nabi Zubair, Abbas Syed. Towards enriching formal concept analysis-based ontology learning with WordNet. In: Proc. 2023 international conference on intelligent systems. IEEE; 2023, p. 1–6. http://dx.doi.org/10.48550/arXiv.2311.14699.
[14] Stumme G, Maedche A. FCA-merge: Bottom-up merging of ontologies. In: Proc. 17th int. joint conf. on artificial intelligence. 2001, p. 225–30, https://dl.acm.org/doi/10.5555/1642090.1642121.
[15] Xu Xiaoliang, Wu You, Chen Jinkui. Fuzzy FCA based ontology mapping. In: 2010 first international conference on networking and distributed computing. IEEE; 2010, http://dx.doi.org/10.1109/ICNDC.2010.45.
[16] Dolques Xavier, et al. RCAExplore, a FCA-based tool to explore relational data. In: Workshop on applications and tools of formal concept analysis. 2019, Available at CEUR-WS: http://ceur-ws.org/Vol-2378/.
[17] Boffa S. Extracting concepts from fuzzy relational context families. IEEE Trans Fuzzy Syst 2023;31(4):1202–13. http://dx.doi.org/10.1109/TFUZZ.2022.3197826.
[18] Musen MA. The protégé project: a look back and a look forward. AI Matters 2015;1(4):4–12. http://dx.doi.org/10.1145/2757001.2757003.
[19] Formica A. Ontology-based concept similarity in formal concept analysis. Inform Sci 2006;176(18):2624–41. http://dx.doi.org/10.1016/j.ins.2005.11.014.
[20] Cherukuri Aswani Kumar, Usha Devi SSSN. Scalable formal concept analysis algorithms for large datasets using spark. J Ambient Intell Humaniz Comput 2019;10(9):3421–32. http://dx.doi.org/10.1007/s12652-018-1105-8.