Computers, Materials & Continua
DOI:10.32604/cmc.2022.021714
Article

Tech Science Press

An OWL-Based Specification of Database Management Systems

Sabin C. Buraga[1,*], Daniel Amariei[1] and Octavian Dospinescu[2]

[1]Alexandru Ioan Cuza University, Faculty of Computer Science, Iasi, 700706, Romania
[2]Alexandru Ioan Cuza University, Faculty of Economics and Business Administration, Iasi, 700706, Romania
[*]Corresponding Author: Sabin C. Buraga. Email: sabin.buraga@gmail.com
Received: 12 July 2021; Accepted: 17 August 2021

Abstract: In the context of a proliferation of Database Management Systems (DBMSs), we have envisioned and produced an OWL 2 ontology able to provide a high-level machine-processable description of the DBMSs domain. This conceptualization aims to facilitate a proper execution of various software engineering processes and database-focused administration tasks. Also, it can be used to improve the decision-making process for determining/selecting the appropriate DBMS, subject to specific requirements. The proposed model describes the most important features and aspects regarding the DBMS domain, including the support for various paradigms (relational, graph-based, key-value, tree-like, etc.), query languages, platforms (servers), plus running environments (desktop, Web, cloud), specific contexts—i.e., focusing on optimizing queries, redundancy, security, performance, schema *vs*. schema-less approaches, programming languages/paradigms, and others. The process of populating the ontology with significant individuals (actual DBMSs) benefits from the existing knowledge exposed by free and open machine-processable knowledge bases, by using structured data from Wikipedia and related sources. The pragmatic use of our ontology is demonstrated by two educational software solutions based on current practices in Web application development, proving support for learning and experimenting key features of the actual semantic Web technologies and tools. This approach is also an example of using multiple knowledge from database systems, semantic Web technologies, and software engineering areas.

Keywords: Database management systems; knowledge model; web engineering

1 Introduction

Database management systems are permanently classified and ranked by specialized sites around which communities of users and developers are created. As of July 2021, according to the DB-Engines Ranking[1], there are 373 different Database Management Systems (DBMSs) supporting various models such as relational, document, key-value, graph, RDF (Resource Description

[1] DB-Engines Ranking—https://db-engines.com/en/ranking/

Framework) triples, time series, native XML (Extensible Markup Language), etc. This list is constructed scientifically, based on compositions of scores that include aspects such as the number of mentions on Websites, general interest, frequency of technical discussions, number of job offerings in which a DBMS is mentioned, plus relevance in social networks.

Also, as reported by the list of NoSQL Database Management Systems[2], there are 225 DBMSs only from the NoSQL category. Another public source, Database of Databases[3], lists a total of 762 DBMSs.

This proliferation of DBMSs has led to dedicated systems with various trade-offs, suitable for particular use-cases. While this represents a positive aspect, nevertheless, it also increases the difficulty of determining the appropriate DBMS for a particular situation.

An original knowledge model is designed with the goal of providing a high-level description of the DBMSs domain. The proposed ontology is expressed in OWL 2 (Web Ontology Language) [1,2] and can be used to improve the decision-making process for determining/selecting the appropriate DBMS, subject to specific requirements.

Paper contributions and organization: The first and central contribution of the paper relates to the description of our ontology and the development methodology (Section 2). Our knowledge model describes the most important features and aspects regarding the DBMS domain, including the support for various paradigms (relational, graph-based, key-value, tree-like), query languages (SQL, SPARQL, XQuery, and others), platforms (servers) + running environments (desktop, Web, cloud), specific contexts—i.e., focusing on optimizing queries, redundancy, security, performance, schema *vs*. schema-less approaches, programming languages/paradigms, and others. The process of populating the ontology with significant individuals (actual DBMSs) benefits from the existing knowledge exposed by DBpedia [3] and Wikidata [4], free and open machine-processable knowledge bases. Our conceptual model can further be used in order to classify Instance Data from these public knowledge bases. This further step provides the context for an enhanced view of the domain of DBMSs with the prospect of improving the overall support for information integration and search capabilities, by creating a specific knowledge graph.

The second contribution consists of envisioning, designing, and developing two knowledge-based Web applications exposing suitable information about DBMSs according to user needs and preferences. Also, they offer answers for some of the questions formulated in Section 2.1. These original Web systems—described in Sections 3 and 4—are straightforward, yet solid examples of using the actual semantic Web technologies, in general, and our knowledge model, in particular, mainly for educational and pragmatic purposes.

The article continues with related work (Section 5), followed by concluding remarks and future directions of development.

In our opinion, this research is positioned at a confluence of the database systems, knowledge modeling, (semantic) Web technologies, and software engineering areas.

2 A Knowledge Model of Database Management Systems

This section describes the process of specification of an original ontology that models the DBMSs domain. As in [5], "developing an ontology is akin to defining a set of data and their

[2] List of NoSQL Database Management Systems—https://hostingdata.co.uk/nosql-database/

[3] Databases of Databases, maintained by Carnegie Mellon Database Group—https://dbdb.io/

structure for other programs to use", and it is not a goal per se. Rather, it provides the basis for "problem solving-methods, domain-independent applications, and software agents [to] use ontologies and knowledge bases built from ontologies as data".

Subsection 2.1 presents the methodology used for developing the ontology; it also presents some of the artefacts produced at each step. Subsection 2.2 provides details regarding the resulting ontology.

2.1 Ontology Engineering

According to [5], the main stages in the ontology development methodology are: (a) Determine scope; (b) Consider reuse; (c) Enumerate terms; (d) Define taxonomy; (e) Define properties; (f) Define facets; (g) Define instances; (h) Check for anomalies. For details, see also [2].

Determine scope: The domain is focused on DBMSs and related concepts: data models, storage models, storage technologies, transaction support, concurrency aspects, replication mechanisms, licensing, etc. Our purpose is to foster the development of applications that can improve the decision-making process for determining/selecting the appropriate DBMS for a particular scenario, subject to one or more constraints like scalability requirements, partitioning support, suitable data model, etc.

Thus, we should design an ontology providing answers for the following kinds of questions: Which characteristics should we consider when choosing a database system? What is the data model of a particular DBMS? Is MongoDB a suitable database system for creating a Web application? In a big data context, is MariaDB the right choice to store Petabytes of data? What is the best choice of data model for a particular application? Which characteristics of a database system affect its appropriateness for a particular type of application? Under what circumstances we see a change in the performance of a given DBMS?

We identified a set of decisive aspects to be considered for the conceptualization of the DBMS domain:

Data model: relational, hierarchical, graph, key-value, multi-value, object-oriented, object-relational, document, triple-store/quad-store (RDF—Resource Description Framework), etc.

Storage: architecture (disk-oriented, in-memory, hybrid), model (row/record, columnar, hybrid, custom), format (for example, Hadoop Distributed File System), organization (indexed sequential access method, heaps, log-structured and others), data compression.

Query: interface (e.g., command-line/shell, SQL, stored procedures, PL/SQL, HTTP-based, XQuery, custom API, etc.), execution (tuple-at-a-time model, vectorized model, materialized model), having support for indexes, foreign keys, joins, views, concurrent and/or parallel execution, query compilation (stored procedure compilation, code generation, JIT (just-in-time) compilation).

Operations: system architecture (embedded, shared-disk, shared-memory, shared-everything, shared-nothing), logging, support for hardware acceleration, type (academic, commercial, open source), platform (operating system, hardware), license (proprietary, open license), supported programming language(s).

Additional meta-data could be attached in order to specify software versioning and compatibility, the use of other storage sub-systems (for instance, if a DBMS embeds SQLite or uses a cloud-based technology), existing support for developers and/or contributors, public documentation, etc.

These features are used to mainly form the terminology component of the developed model.

Consider reuse: While we were not able to find an ontology that details the DBMS domain to a sufficient extent, we did manage to discover existing concepts and instances related to the domain by accessing knowledge provided by public knowledge bases.

General concepts chosen to be reuse for our purpose are several categories from YAGO[4], *Database*, *Replication_(computing)* and others. Conforming to the semantic Web architecture, each concept is denoted by a Web address—for instance, the *Database* ontological class is identified by *<http://dbpedia.org/ontology/Database>*. By executing designated SPARQL queries, further useful constructs are found—e.g., *Database model* is related to resources like *Data integrity*, *Object database*, *SQL*, and many others–, including links to other components to external knowledge bases/graphs.

These resources (classes, relations, instances) can be incorporated into our model. Though, it can be observed that the naming conventions are not uniform. To maintain a uniform nomenclature, we redefine a subset of the concepts and instances already existing and manually link them (i.e., create mapping sets) with their DBpedia and/or Wikidata equivalent by using RDF Schema and OWL standard predicates.

Enumerate terms: At this step, we need to enumerate relevant terms that are of interest to the DBMSs domain. The objective is "to compare the characteristics of the leading solutions in order to provide guidance to practitioners and researchers to choose the appropriate data store for specific applications" [6]. The identified salient dimensions among which the data stores are compared include: data models, querying capabilities, scaling properties, partitioning, replication, consistency, concurrency control, and security. We considered these dimensions as a starting point in order to determine the relevant terms in our model.

For example, a few of the identified classes related to the *replication* dimension/characteristic are enumerated: *Asynchronous Replication*, *Peer-to-Peer Replication*, *Replication Factor*, *Replication Protocol*, and *Replication Synchronization Strategy*.

Define taxonomy: After the identification of relevant terms, these terms must be organized in a taxonomic hierarchy. Initially, we defined two major stand-alone classes: *DBMS* (*Database Management System*) denoting the knowledge about a DBMS and *Software Application* specifying a target application that need one or more data/knowledge management solutions. Additionally, several concepts like *Concurrency*, *Consistency*, *Data*, *Trigger*, and others are included.

We also define relevant subclasses by using existing OWL restriction-based mechanisms. For example, we specify the *Quad-Store DBMS* concept as a subclass of *Graph DBMS* which is itself a subclass of the *DBMS* class.

Define properties: The classes alone are not able to provide answers to the type of questions that were identified in the *Determine Scope* step. In order to achieve this goal, we need to augment the internal structure of the concepts by specifying properties that link the classes in the hierarchy or primitive types [2,5]. For example, Fig. 1 specifies the constructs about having a concurrency control mechanism.

In addition, some properties and additional vocabularies can be used to link external resources specified by other knowledge graphs.

Define facets: In this phase, we need to specify the following aspects for the identified properties—if applicable: (a) *cardinality*: how many values are allowed or required for a property;

[4] YAGO: A High-Quality Knowledge Base—https://yago-knowledge.org/

(b) *required value(s)*: properties that are required to have a specific value or one or more values from a specific class; (c) *relational characteristics*: symmetric properties, transitive properties, inverse properties, functional properties, etc. conforming to the Description Logics formalism [7] used to specify ontological constructs.

```
:hasConcurrencyControlMechanism rdf:type owl:ObjectProperty ;
    rdfs:subPropertyOf owl:topObjectProperty ;
    rdfs:domain :DatabaseManagementSystem ;
    rdfs:range :ConcurrencyControlMechanism ;
    rdfs:comment "Concurrency control used by a specific DBMS."@en ;
    rdfs:label "Has Concurrency Control Mechanism"@en .
```

Figure 1: Specifying a property regarding concurrency control mechanism

Define instances: At this stage, we need to create the individual instances of classes in the hierarchy. The individuals that populate the ontology forms the assertion-related component of the knowledge base and have two main sources: the (semi-)manual addition of the instances according to the data specified by DBMS-related public information and the extraction of individuals that are present on DBpedia or Wikidata knowledge bases and manual classification according to our ontology.

Check for anomalies: The last step in the ontology engineering process requires the validation of the ontology from syntactic, model-based, and semantic perspectives.

Syntactic validation is concerned with the rules of the specific serialization format—usually, RDF triples in Turtle (Terse RDF Triple Language) format and/or XML markups representing OWL constructs—in which the ontology is written. The ontology is validated with W3C RDF Validation Service[5].

To detect various common pitfalls—e.g., misusing ontology annotations, using recursive definitions, defining untyped classes/properties, namespace hijacking and many others–, we used OOPS! (Ontology Pitfall Scanner!)[6]. We fixed the majority of reported issues shown in Fig. 2 (for example, we defined various common properties such as *label* or *comment* as annotation properties).

Semantic validation implies detecting logical contradictions or inconsistencies in the ontology. As long as the ontology is based on a syntactic subset of Description Logics [7] that is decidable, a reasoner can be used in order to detect inconsistencies. The ontology was inspected for anomalies with Fact++, Hermit, and Pellet OWL reasoners included in the Protégé 5 desktop ontology editing environment[7]. The valid model was processed in 25, 332 and 280 ms, respectively. Using the specified tools, we were able to determine a number of inconsistencies and correct them. We also performed various manual check-ups and reviews. The current version of the ontology is reported as being consistent and coherent.

2.2 Resulting Ontology

The model's important metrics, as provided by the Protégé application, are the following: a total of 815 axioms (264 logical axioms and 204 declaration axioms, and 347 annotation

[5] W3C RDF Validation Service—https://www.w3.org/RDF/Validator/

[6] OOPS! (Ontology Pitfall Scanner!)—http://oops.linkeddata.es/

[7] Protégé—https://protege.stanford.edu

assertions). Our ontology consists of 65 classes, 25 object properties, 13 data properties, plus 93 individuals. A preliminary version of the ontology was published on GitHub to be consulted by interested professionals[8].

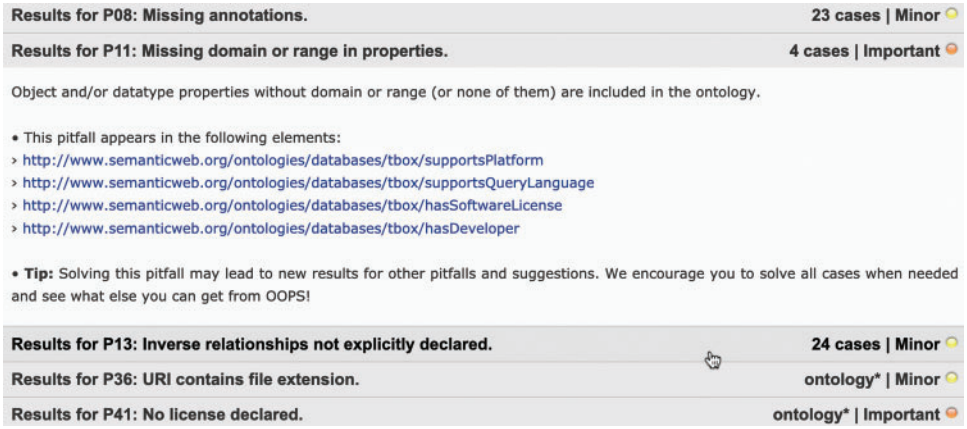

Figure 2: Pitfalls reported by OOPS! web tool

A graphical representation of the main classes is depicted in Fig. 3.

Additionally, several categories of software applications are specified in order to recommend a specific DBMS according to the developer's needs—for example, an application that uses analytics data along to user preferences or location.

We refined the model by integrating concepts specified by *schema.org* [8], in order to adhere to the Linked Data principles [9]. Beneficial classes are *Computer Language*, *Dataset*, *Data Catalog*, *Organization*, *Product*, *Rating*, *Software Application*, etc. As well, various schema.org properties are taken into consideration: *contributor*, *creator*, *description*, *license*, *maintainer*, and many others.

3 Using the Ontology for Testing and Learning Purposes

For testing purposes mainly, we first have developed a Node.js-based application that uses the ontology as a building block. The first purpose of the Web application is to provide assistance for a software developer in the process of determining/selecting the appropriate DBMS, subject to specific requirements (e.g., partitioning support, replication support, data model, operational processes, etc.).

In order to achieve this, the application provides an interactive comparison of several characteristics for multiple databases, also enabling the users to filter and choose the desired characteristics that are included in the comparison. Furthermore, the application takes advantage of the reasoning capabilities that several RDF storage engines (themselves instances of DBMSs) provide in order to augment the information available in the ontology.

[8] Databases Ontology—https://github.com/danielamariei/databases-ontology

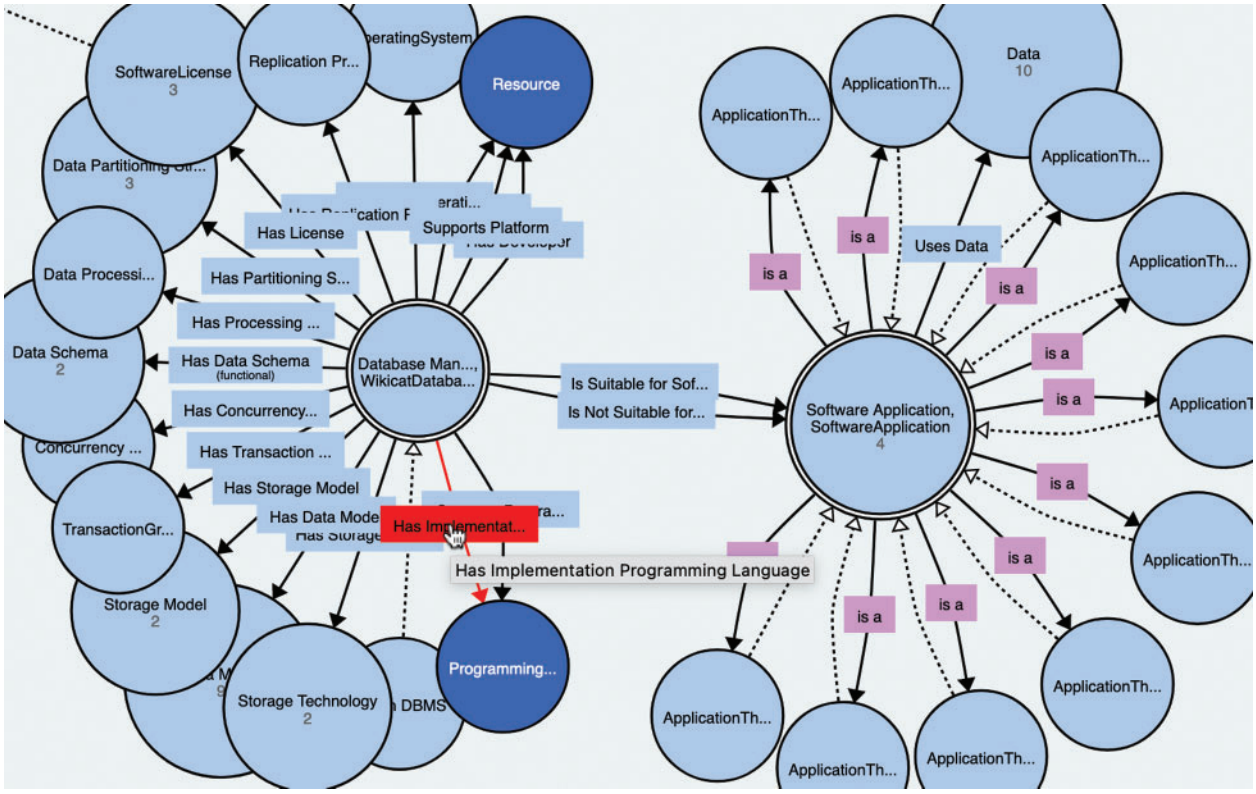

Figure 3: Our ontology visualized online with webVOWL tool[9]

The second purpose of this application is an educational one. The interested students—usually enrolled for (semantic) Web application development university disciplines[10]—could learn concepts and practices regarding knowledge modeling and/or semantic Web technologies in conjunction to other techniques and software tools. Also, we are encouraging them to study, test, modify and/or extend the application's source-code available in a GitHub repository, by conducted online hackathons and proposing new practical projects to be envisioned and implemented in order to successfully pass.

From technological point of view, the main functionalities are implemented by using Express, a widespread Node.js framework, and Stardog[11] as a main storage solution supporting RDF and automated reasoning. Stardog itself is defined by the various RDF assertions included in our ontology (Fig. 4).

Desired features could be selected (e.g., having a partitioning strategy, supporting a query language, a platform or an operating system, etc.), in order to compare the found DBMSs—this process involves various parametrized SPARQL queries [10] to be performed on a set of

[9] WebVOWL—http://vowl.visualdataweb.org/webvowl.html

[10] For example, consult the public Website of Web Application Development discipline—https://profs.info.uaic.ro/~busaco/teach/courses/wade/

[11] Stardog, an enterprise Knowledge Graph platform—https://www.stardog.com/

individuals (actual DBMSs) specified by our knowledge model. For educational purposes, we deliberately simplify these SPARQL statements in order to ease the understanding. For example, to obtain a number of open-source relational DBMSs having certain characteristics, a parametrized query is generated—consult Fig. 5.

```
:Stardog rdf:type owl:NamedIndividual , :DBMS , schema:SoftwareApplication ;
    :hasDataModel :TripleDataModel ; :hasDataSchema :Flexible ;
    :hasLicense :Commercial , :Free ;
    :hasReplicationProtocol :MasterSlaveSynchronousReplication ;
    :supportsIndexing  "true"^^xsd:boolean ;
    :supportsTyping    "true"^^xsd:boolean ;
    :supportsReasoning "true"^^xsd:boolean .
```

Figure 4: Stardog system is an individual modeled by our ontology

```
SELECT ?db
WHERE {
  ?db a :DBMS ; :hasLicense :OpenSource ;
      :hasDataModel :RelationalDataModel .
  OPTIONAL { ?db :supportsIndexing $index . }
} LIMIT $limit
```

Figure 5: A SPARQL parametrized query

The simple, yet flexible user interaction is depicted by Fig. 6, where we can browse the found DBMSs having the desired characteristics.

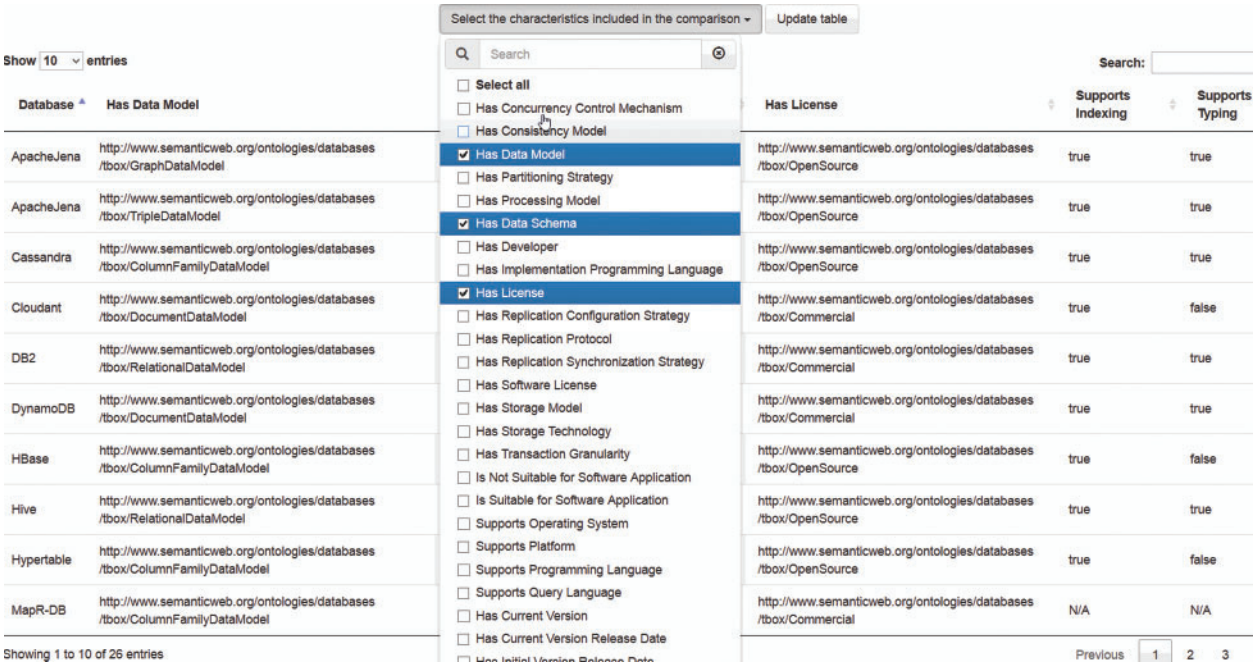

Figure 6: User interface—the list of DBMSs having certain features

For example, the application lists Apache Jena supporting the triple data and graph data models, indexing, and data typing, plus having an open-source license. Using this approach, the user can discover various properties provided by common DBMSs.

4 Recommending DBMSs

4.1 Overview

Next stage of our research was focused on envisioning, implementing, and engineering a more complex Web platform providing support for finding and suggesting the most suitable DBMSs solutions for a specific data-intensive software project to be developed, in order to be up to date with the cutting-edge database technologies. Our secondary objective is to facilitate the procedures through which the users can learn new skills regarding database usage methods, based on their preferences. All of the provided functionalities are presented in a user-friendly manner through a web interface, whilst offering a powerful tool not only for experienced developers, but also for novices. To accomplish this goal, the application provides a basic sand-boxed DBMS playground permitting query execution for a few popular DBMSs, in order to help practitioners to experiment various practical situations.

The content-based recommendations take into consideration the user's past actions—e.g., performing faceted searches on specific criteria such as running environments, types of DBMSs, queries–, plus various preferences stored by the user profile. Additionally, recommendations are influenced by the user engagement. Two aspects are considered: (a) sharing a Web address (URL) of interest regarding a specific DBMS through a social media account; (b) visiting an article from the news feed with the most recent information gathered from a well-known newsletter: DB Weekly[12].

4.2 Formulating Use Cases

We state the main flows that a user can explore while using the application (Fig. 7). The use case routes are color coded and labeled with easy-to-understand symbols to highlight the different ways in which a user can interact with the Web platform: (1) Light red color and "**@**" label correspond to the authentication step by using OAuth 2.0, an open industry-standard for authorization (access delegation); (2) Blue color and "**#**" label correspond to receiving the DBMS-related news feed; (3) Yellow color and "**?**" symbol point to the search for DBMS recommendations; (4) Green color and the "*****" label are used for viewing/editing user preferences; (5) Orange color and "**!**" label denote processes regarding execution queries for a few set of DBMSs (see also Section 4.3).

To suit these needs, our model is extended by integrating necessary entities regarding users (*User* class mapped to *Person* concept from *schema.org* model) and their preferences (*Preference* concept). An instance of *Preference* class points towards an individual from *DBMS* class of our ontology.

4.3 System Architecture and Technological Aspects

The actual architecture is a microservice oriented one, conforming to the well-known *Microservice Architecture* pattern[13]. The general architecture is illustrated in Fig. 8. All functionalities—such as authentication and authorization, user preferences and session

[12] DB Weekly: The Weekly Database Newsletter: https://dbweekly.com/

[13] Microservice Architecture pattern—https://microservices.io/patterns/microservices.html

management, DBMS recommendations, database playground, metrics, etc.—follow the flows explained in Section 4.2.

The *Database Playground* module implements the query execution tasks for three open-source systems: MongoDB (a document-based distributed system), Redis (an in-memory data structure store, used as a distributed, in-memory key-value database, cache and message broker, with optional durability), and PostgreSQL (a popular relational DBMS emphasizing extensibility and SQL compliance). By using a modular approach, the current architecture could accommodate other DBMSs if needed. The DBMSs used by *Database Playground* microservice and additional assets are hosted and accessible in the Heroku, a cloud PaaS (Platform as a Service).

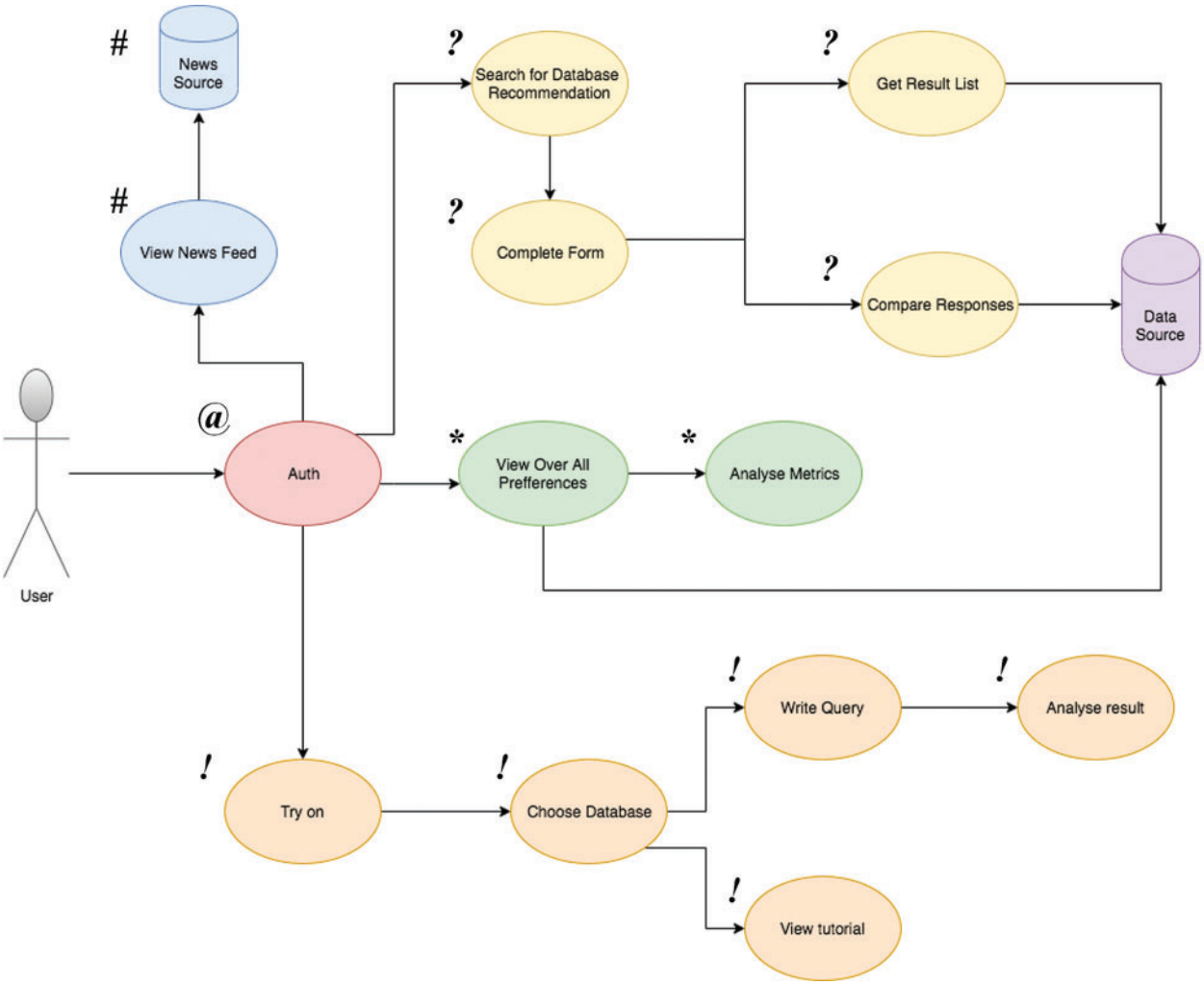

Figure 7: Dataflows and workflows according to considered use cases

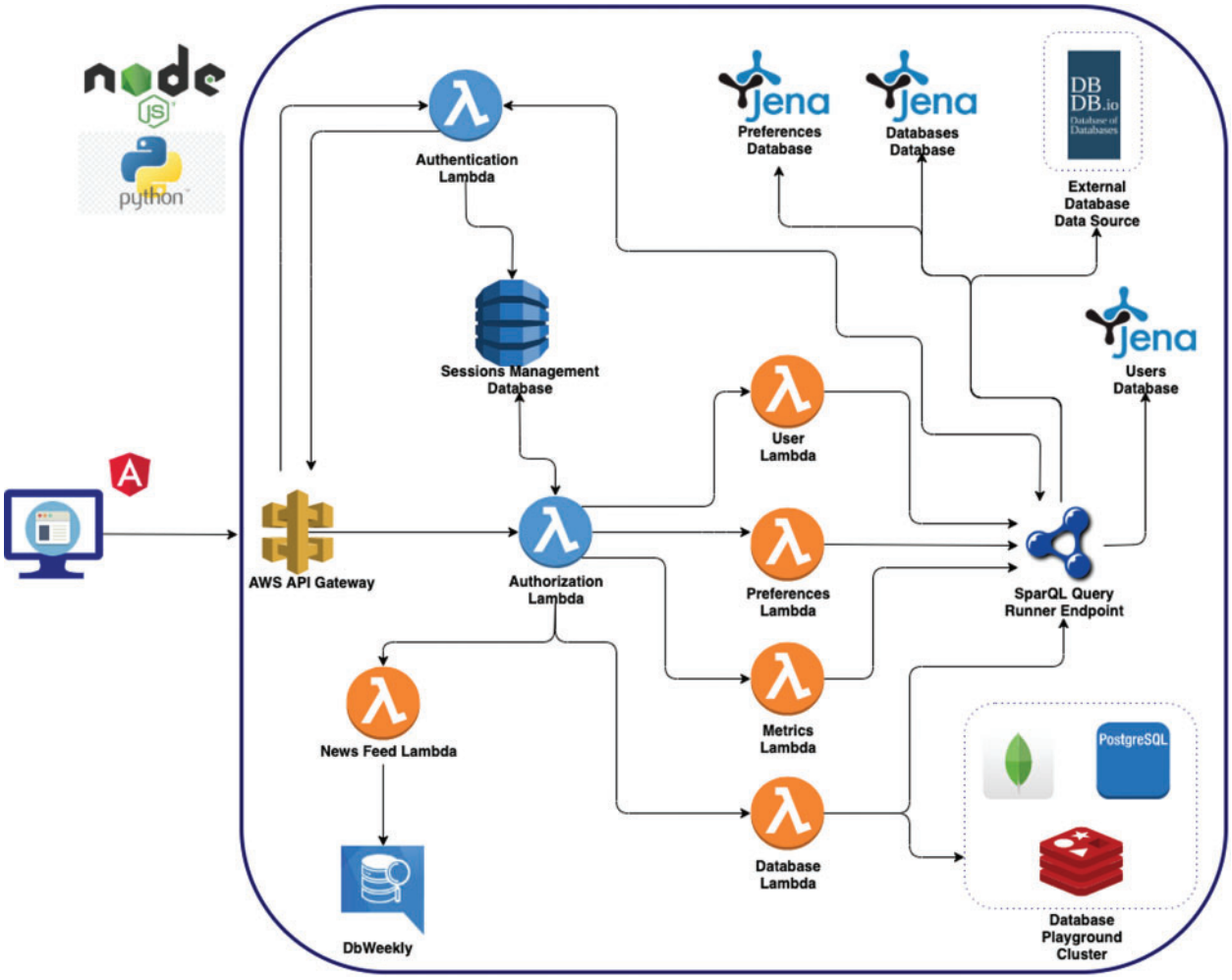

Figure 8: General serverless software architecture of our web platform

The data and knowledge are managed and queried by instances of Apache Jena[14]—a free and open-source Java framework providing support for existing semantic Web technologies (RDF, SPARQL, OWL, plus reasoning).

Two knowledge bases are managed: one modeling users and their preferences, and the other one containing the ontology divided, for logical and computational tasks, into two components: (1) concepts (knowledge regarding DBMS and users) + properties, and (2) assertions about individuals (i.e., instances of *DBMS, User*, and *Preference* classes).

A SPARQL endpoint (seen as a REST API) is exposed with the help of Apache Jena. To easily perform various SPARQL queries [10], the system uses an open-source Python module[15].

Following the "things, not strings" adage (specifying the application domain, not just JSON structures) [11], all data is exchanged in JSON-LD (JavaScript Object Notation for Linked-Data)

[14] Apache Jena—https://jena.apache.org/

[15] SPARQLWrapper—https://rdflib.dev/sparqlwrapper/

format[16]. Furthermore, all functionalities can be queried through this API (without actually needing the client), since we have a decoupled and distributed architecture. Actual Node.js implementation conforms to the *Functions as a Service* paradigm by using so-called "serverless" cloud computing execution model[17]. For deployment, our application is using Amazon Lambda, an event-driven, serverless computing platform provided by Amazon as a part of Amazon Web Services.

The web interaction with the final users is facilitated by angular, a TypeScript-based open-source web application framework

4.4 Practical Usage

The Web application was used to assist the interested students in the decision process of choosing one or more DBMSs suitable for implementing a team-oriented software project to be assessed for Web Application Development discipline.

A project consists of a (micro-)service-based Web application developed by using existing social and semantic Web technologies (minimally, RDF, SPARQL and OWL), with no restrictions regarding the project implementation. Open-source approaches are highly encouraged. For the final assessment, the team must provide the full functionality of the developed solution, according to the specific requirements stated in the project description and discussed during the practical works.

As optional unrewarded tasks, students could study and/or adapt the ontology to get insights about various DBMS, including performing queries and using the reasoners included by Protégé according to several criteria. These activities are also implemented by the application's recommender module. Also, students are encouraged to understand and extend the application's architecture described in Section 4.3.

In the last academic year, 32 persons were enrolled. The interaction was exclusively online via dedicated Website and Discord server. Implemented projects were stored and evaluated on GitHub. More than 65% of enrolled students accessed the ontology and almost 40% of them used the application.

5 Related Work

There are few attempts to conceptualize databases and database management systems. A few lists of these DBMSs and their characteristics are publicly available, but only at the data level. In most cases, these lists offer only HTML and JSON representations, not embracing a conceptual approach. Regarding an ontological perspective, we mention RDBS-O [12], an ontology aiming to identify and represent key concepts of the relational database domain in the architectural scope, but without covering control and execution details. Only 21 classes and 10 properties are defined.

Another proposal [13] describes a recommender system assisting experts to select desired DBMS based on various features (e.g., price, popularity, license, platform, database model, architecture, query language) by using an ontology. The designed ontology is mainly focused on a limited number of DBMSs and their characteristics, all queries being only performed in Protégé. No additional applications are mentioned.

[16] JSON-LD—https://json-ld.org/

[17] Serverless Architectures—https://martinfowler.com/articles/serverless.html

A third approach[18] sketches a conceptual model (glossary) consisting of terms associated with existing DBMSs in order to enable structured descriptions of DBMS products.

These solutions are not general enough to be considered reference ontologies for generic DBMSs and their manifold imaginable features. Our proposal is an apt solution to fill this gap.

6 Conclusion

The paper presented the goals and main stages of creating an original ontology able to properly model the DBMS domain.

The pragmatic use of our conceptual model is demonstrated by two software solutions described in Sections 3 and 4. These Web projects are based on current practices in Web application development and can be considered valuable educational tools for any developer keen to learn and experiment the actual usages of semantic Web technologies and tools: modeling data in RDF, performing queries with SPARQL, defining and managing knowledge via OWL.

We intend to further develop our conceptual model. Subsection 2.1 already identified a partial set of the concepts and instances related to the domain of DBMSs that are already available in other knowledge bases. In order to preserve a uniform nomenclature, we have decided to redefine a subset of the concepts and manually create mapping to their equivalent resources specified by other public knowledge bases. A further improvement would be to link the remaining suitable concepts and/or instances in our ontology to similar ones that are available in other ontologies or knowledge graphs [14]. The designed ontology could be seen as a module to be integrated into a more complex knowledge-driven solution.

Another direction is to extend the proposed knowledge model with various concepts, instances, and axioms, some of them to be automatically learned from existing DBMS-related literature.

The end-purpose is to allow the creation of complex Web applications that use the described ontology in order to intelligently assist human users in the decision-making process for determining the optimal DBMS that should be used for a particular scenario, for a given set of specific requirements.

Acknowledgement: We are grateful to Georgiana Calancea, Octavian Ilie, and Camelia Milut, former students that contributed to the implementation of the DBMS recommender system.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider *et al.,* "OWL 2: The next step for OWL," *Journal of Web Semantics*, vol. 6, no. 4, pp. 309–322, 2008.

[2] D. Allemang and J. Hendler, *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Amsterdam: Elsevier, 2011.

[3] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas *et al.,* "DBpedia-a large-scale, multilingual knowledge base extracted from wikipedia," *Semantic Web*, vol. 6, no. 2, pp. 167–195, 2015.

[18] OpenLink Database Management System Ontology Document—https://github.com/OpenLinkSoftware/glossaries

[4] D. Vrandečić and M. Krötzsch, "Wikidata: A free collaborative knowledgebase," *Communications of the ACM*, vol. 57, no. 10, pp. 78–85, 2014.

[5] N. Noy and D. McGuinness, "Ontology development 101, KSL-01-05, Stanford University," 2001. [Online]. Available: http://www.ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noy-mcguinness.html.

[6] K. Grolinger, W. A. Higashino, A. Tiwari and M. A. Capretz, "Data management in cloud environments: NoSQL and newSQL data stores," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 2, no. 1, pp. 1–24, 2013.

[7] F. Baader, I. Horrocks, C. Lutz and U. Sattler, *An Introduction to Description Logics*. New York: Cambridge University Press, 2017.

[8] R. V. Guha, D. Brickley and S. Macbeth, "Schema.org: Evolution of structured data on the web," *Communications of the ACM*, vol. 59, no. 2, pp. 44–51, 2016.

[9] S. Speicher, J. Arwe and A. Malhotra, *Linked Data Platform 1.0*. W3C Recommendation, Cambridge, MA: W3C/MIT, 2015.

[10] B. DuCharme, *Learning SPARQL*, 2nd ed., Sebastopol, CA: O'Reilly Media, 2013.

[11] M. Lanthaler and C. Gütl, "Model your application domain, not your JSON structures," in *Proc. 22nd Int. Conf. on World Wide Web*, Rio de Janeiro, Brazil: ACM Press, pp. 1415–1420, 2013.

[12] C. Z. de Aguiar, R. de Almeida Falbo and V. E. S. Souza, "Ontological representation of relational databases," in *Proc. ONTOBRAS, CEUR Workshop Proceedings*, São Paulo, Brazil, vol. 2228, pp. 140–151, 2018.

[13] L. Brahimi, L. Bellatreche and Y. Ouhammou, "Coupling multi-criteria decision making and ontologies for recommending DBMS," in *22nd Int. Conf. on Management of Data*, Chennai, India, pp. 8–19, 2017.

[14] H. Zhou, T. Shen, X. Liu, Y. Zhang, P. Guo *et al.,* "Survey of knowledge graph approaches and applications," *Journal on Artificial Intelligence*, vol. 2, no. 2, pp. 89–101, 2020.