Ontology Database System and Triggers

Angelina Tzacheva, Tyrone Toland, Peyton Poole, and Daniel Barnes

University of South Carolina Upstate, Department of Informatics
800 University Way, Spartanburg, SC 29303, USA
e-mail: atzacheva@uscupstate.edu

Abstract. An ontology database system is a basic relational database management system that models an ontology plus its instances. To reason over the transitive closure of instances in the subsumption hierarchy, an ontology database can either unfold views at query time or propagate assertions using triggers at load time. In this paper, we present a method to embed ontology knowledge into a relational database through triggers. We demonstrate that by forward computing inferences, we improve query time. We find that: first, ontology database systems scale well for small and medium sized ontologies; and second, ontology database systems are able to answer ontology-based queries deductively; We apply this method to a Glass Identification Ontology, and discuss applications in Neuroscience.

1 Introduction

Researchers are using Semantic Web ontologies extensively in intelligent information systems to annotate their data, to drive decision-support systems, to integrate data, and to perform natural language processing and information extraction. Researchers in biomedicine use ontologies heavily to annotate their data and to drive decision support systems that translate to applications in clinical practice [24] .

An ontology defines terms and specifies relationships among them, forming a logical specification of the semantics of some domain. Most ontologies contain a hierarchy of terms at minimum, but many have more complex relationships to consider. Ontologies provide a means of formally specifying complex descriptions and relationships about information in a way that is expressive yet amenable to automated processing and reasoning. As such, they offer the promise of facilitated information sharing, data fusion and exchange among many, distributed and possibly heterogeneous data sources.

However, the uninitiated often find that applying these technologies to existing data can be challenging and expensive. What makes this work challenging in part is: to have systems that handle basic reasoning over the relationships in an ontology in a simple manner, that scales well to large data sets. In other words, we need the capabilities of an efficient, large-scale knowledge-based system (such as Semantic Web Ontology), but we want a solution as simple as managing a regular relational database system, such as MySQL. We argue that: when used in the proper context, regular databases can behave like efficient, deductive systems.

In this paper, we present an easy method for users to manage an ontology plus its instances with an off-the-shelf database management system like MySQL, through triggers. We call these sorts of databases *ontology databases*.

An *ontology database system* takes a Semantic Web ontology as input, and generates a database schema based on it. When individuals in the ontology are asserted in the input, the database tables are populated with corresponding records. Internally, the database management system processes the data and the ontology in a way that maintains the knowledge model, in the same way as a basic knowledge-base system.

After the database is bootstrapped in this way, users may pose SQL queries to the system declaratively, i.e. based on terms from the ontology, and they get answers in return that incorporate the term hierarchy or other logical features of the ontology. In this way, our proposed system is useful for handling ontology-based queries. That is, users get answers to queries that take the ontological subsumption hierarchy into account.

We find that ontology databases using our trigger-based method scale well.

The proposed method can be extended to perform integration across distributed, heterogeneous ontology databases using an inference-based framework [10].

The rest of the paper is organized as follows: in Section 2 we review related work and provide background information; in Section 3 we discuss the main ideas behind ontology databases and the proposed method; in Section 4 we present case studies; in Section 5 we suggest directions for the future and conclude.

2 Related Work

Knowledge-based Systems. Knowledge-based systems (KBs) use a knowledge representation framework, having an underlying logical formalism (a language), together with inference engines to deductively reason over a given set of knowledge. Users can tell statements to the KB and ask it queries [20], expecting reasonable answers in return. An ontology, different from but related to the philosophical discipline of Ontology, is one such kind of knowledge representation framework [15]. In the Semantic Web [4], description logic (DL) [2] forms the underlying logic for ontologies encoded using the standard Web Ontology Language1 (OWL). One of the major problems with Semantic Web KBs is they do not scale to very large data sets [18].

Reasoning. Researchers in logic and databases have contributed to the rich theory of deductive database systems [12], [13]. For example, Datalog [25] famously uses views for reasoning in Horn Logic. We already mentioned EKS-V1 [27]. Reasoning over negations and integrity constraints has also been studied in the past [19]. Of particular note, one of the side-remarks in one of Reiters papers [22] formed an early motivation for building our system: Reiter saw a need to balance time and space in deductive systems by separating extensional from intensional processing. However33 years laterspace has become expendable. Other works move beyond Datalog views to incorporate active rules for reasoning. An active rule, like a trigger in a database, is a powerful mechanism using an event-condition-action model to perform certain actions whenever a detected event occurs within a system that satisfies the given condition. Researchers in object-oriented and deductive database systems use active technologies in carefully controlled ways to also manage integrity constraints and other logical features [7], [26]. Researchers are studying how to bring database theory into the Semantic Web [21], but more work is needed in that regard.

Scalability. Since reasoning generally poses scalability concerns, system designers have used the Lehigh University Benchmark (LUBM) [16] in the past, to evaluate and compare knoweldge-based (KB) systems. The Lehigh University authors have also proposed the Description Logic Database (DLDB) [17]. We would charactarize DLDB as an *ontology database* because it is similar to our proposed system. It mimics a KB system by using features of a basic relational database system, and it uses a decomposition storage model [1] and [6] to create the database schema.

Information Integration. Another important motivation for using ontologies is the promise they hold for integrating information. Researchers in biomedical informatics have taken to this idea with some fervor [14]. One system in particular, OntoGrate, offers an inferrential information integration framework using ontologies which integrates data by translating queries across ontologies to get data from target data sources using an inference engine [8]. The same logical framework can be extended to move data across a network of repositories.

3 Ontology Database System

In the following sections, we use a simple running example, the SisitersSiblings example, to illustrate how we implement an ontology database system. We begin with the basic idea, and then explain how we structure the database schema and implement each kind of logical feature using triggers and integrity constraints.

3.1 The Basic Idea

We can perform rudimentary, rule-based reasoning using either views or triggers. For example, suppose we assert the statement (a rule): *All sisters are siblings*. Then we assert the fact: *Mary and Jane are sisters*. Logically, we may deduce using modus ponens (MP) that Mary (M) and Jane (J) are siblings. The notation {x/M, y/J} denotes that the variable x gets substituted with M, y with J, and so on, as part of the unification process.

$$\frac{Sisters(x,y) \rightarrow Siblings(x,y) Sisters(M,J)}{Siblings(M,J)} MP\{x/M, y/J\}$$

If sibling and sister facts are stored in two-column tables (prefixed with *a_* to denote an asserted fact), then we can encode the rule as the following SQL view:

```
CREATE VIEW siblings(x, y) as
SELECT x,y FROM a_siblings
UNION
SELECT x,y FROM sisters
```

In the view-based method, every inferred set of data necessarily includes its asserted data (e.g., siblings contains *a_siblings* and sisters contains *a_sisters*). Note: when the view is executed, the subquery retrieving sisters will unfold to access all asserted sisters data. Recursively, if sisters subsumes any other predicate, it too will be unfolded. Database triggers can implement the same kind of thing:

```
CREATE TRIGGER subproperty_sisters_siblings
ON INSERT (x, y) INTO sisters
FIRST INSERT (x, y) INTO siblings
```

The deduction is reflected in the answer a query such as *Who are the siblings of Jane?* Of course, the answer returned, in both cases, is: *Mary*. We easily formulate the SQL query:

```
SELECT x FROM siblings WHERE y=Jane
```

What differentiates these two methods is that views are goal-driven: the inference is performed at query time by unfolding views. Whereas, triggers forward propagate facts along rules as they are asserted, i.e., at load time. We advocate a trigger-based approach as our preferred method of implementation. If we consider the spacetime tradeoff: essentially, triggers use more space to speed up query performance. The technique has some of the advantages of materialized views but it differs in some important ways, specifically: deletions. Assume we assert the following in order: $A \rightarrow B$, insert $A(a)$, delete $A(a)$. Next, ask the query $B(?x)$. A trigger-based implementation returns $\{x/a\}$. A view-based implementation returns $null$. In addition, triggers can differentiate negation from deletion. Finally, aside from rule-based reasoning, triggers support other logical features we find important, such as domain and range restrictions, and inconsistency detection. We describe the methods for handling each case in the following implementation details.

3.2 Implementation Details

Decomposition Storage Model. We use the decomposition storage model [1], [6] because it scales well and makes expressing queries easy. Arbitrary models result in expensive and complicated query rewriting, so we would not consider them. The two other suitable models in the literature are the horizontal and vertical models. Designers rarely use the horizontal model because it contains excessively many null values and is expensive to restructure: The administrator halts the system to add new columns to service new predicates. The vertical model is quite popular because it avoids those two drawbacks. Also, the vertical model affords fast inserts because records are merely appended to the end of the file. Sesame [5] and other RDF stores use the vertical storage model.

However, the vertical storage model is prone to slow query performance because queries require many joins against a single table, which gets expensive for very tall tables. Furthermore, type-membership queries are somewhat awkward. As a typical workaround, designers first partition the vertical table to better support type-membership queries, then they partition it further along other, selected predicates which optimize certain joins based on an informed heuristic. However, this leads back toward complicated query rewriting because the partitioning choices have to be recorded and unfolded in some way.

We view the decomposition storage model as a fully partitioned vertical storage model, where the single table is completely partitioned along every type and every predicate. In other words: each type and each predicate gets its own table. When taken to

this extreme, query rewriting becomes simple, because each table corresponds directly to a query predicate. In this way, the decomposition storage model keeps the advantages of the vertical model while improving query performance (because of the partitions) without introducing complex query rewriting. Figure 1 illustrates the three different models using the Sisters - Siblings example.

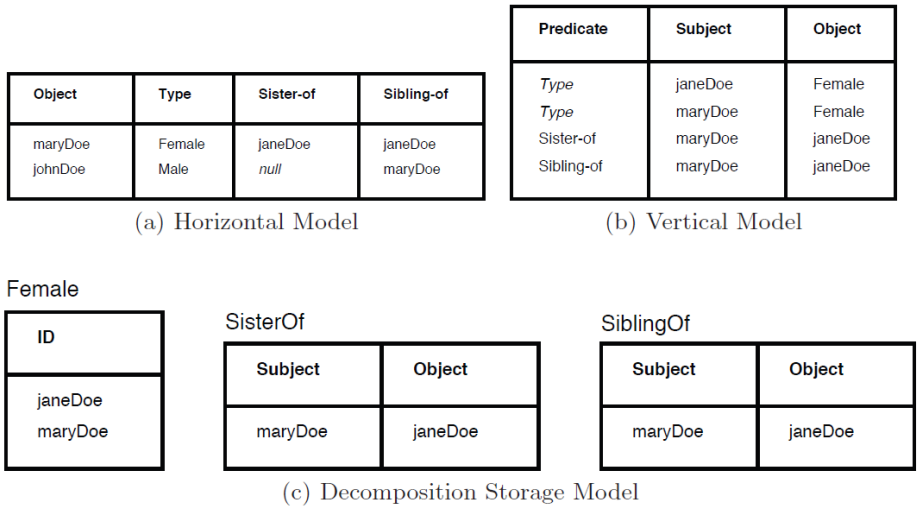

Fig. 1. The SistersSiblings examples using the 1(a) horizontal, 1(b) vertical, and 1(c) decomposition storage model.

Subsumption. Ontology engineers often specify subclass relationships in Semantic Web ontologies, which form a subsumption hierarchy. That constitutes the majority of reasoning for biomedical ontologies [2] as well. As we mentioned earlier (Section 3.1), we handle subclass relationships by using triggers.

The same can be handled by using Views in Datalog. However, Datalog views differ from inclusion axioms in description logic [3]. In other words, the semantics of these two logical formalisms differ:

$$Sisters \rightarrow Siblings \neq Siblings \subseteq Sisters$$

The literature suggests that these differences are formally captured using modal logic [3]. In our proposed method, we ensure that the contrapositive of the rule is enforced as an integrity constraint [23] (and not as a rule): *if Siblings(M,J)* is not true, then *Sisters(M,J)* cannot possibly be true (otherwise, raise an inconsistency error). We, therefore, implement the contrapositive as a foreign-key constraint as follows:

```
CREATE TABLE Siblings(
subject VARCHAR NOT NULL,
object VARCHAR NOT NULL,
CONSTRAINT fk-Sisters-Siblings
FOREIGN KEY subject, object
REFERENCES Sisters(subject, object) ...)
```

Figure 2(a) illustrates the two parts of an inclusion axiom graphically. The trigger rule event is indicated in the figure by the a star-like symbol, denoting that the detected assertion causes a trigger to fire. In this example, we enforce the following rule: $All females are person(s), i.e., Female \rightarrow Person$. Therefore, asserting Female(Mary) causes the trigger to actively assert Person(Mary).

Finally, the contrapositive is checked using the foreign key. Note: consistency requires that forward-propagations occur *before* integrity checking, which explains using the keywords *before* or *first* in our trigger definitions.

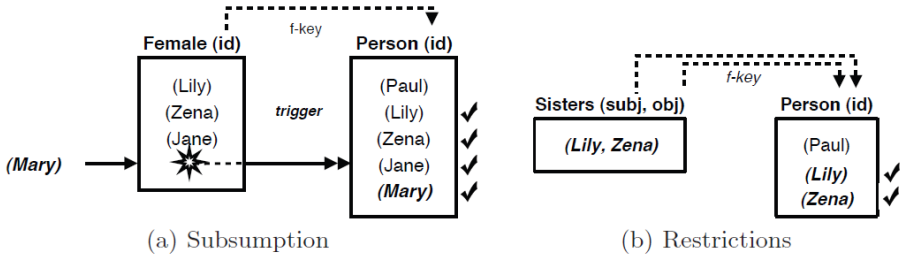

Fig. 2. The star-like symbol denotes an event fires a trigger rule. The checkmark symbol denotes an integrity check occurs. (a) Subsumption is implemented using a combination of triggers and integrity constraints. (b) Domain and range restrictions are implemented using foreign-key (f-key) constraints.

Domain and Range Restrictions. Another important feature of Semantic Web ontologies are domain and range restrictions. These restrict the possible set of instances that participate in property assertions. For example, *Only person(s) may participate in the sisters relationship*. Restrictions are formalized using modal logic. They correspond to integrity constraints. In our proposed method, we implement them as foreign key constraints on the subject or object (i.e. domain and range) of the property.

```
CREATE TABLE Sisters(
subject VARCHAR NOT NULL,
object VARCHAR NOT NULL,
CONSTRAINT fk-Sisters-Subject-Person
FOREIGN KEY subject
```

```
REFERENCES Person(id) ...)
```

4 Experiment

We apply our proposed trigger-based ontology database system method to a Glass Identification Ontology. Next, we discuss its application for a Neural ElectroMagnetic Ontology.

Glass Identification. We apply our proposed method to a Glass Identification Ontology - shown on Figure 3. The Glass Identification Dataset [11] is donated by the Central Research Establishment, Home Office Forensic Science Service, Reading, Berkshire, England. The study of classification of types of glass was originally motivated by criminological investigation. At the scene of the crime, the glass left can be used as evidence, if it is correctly identified.

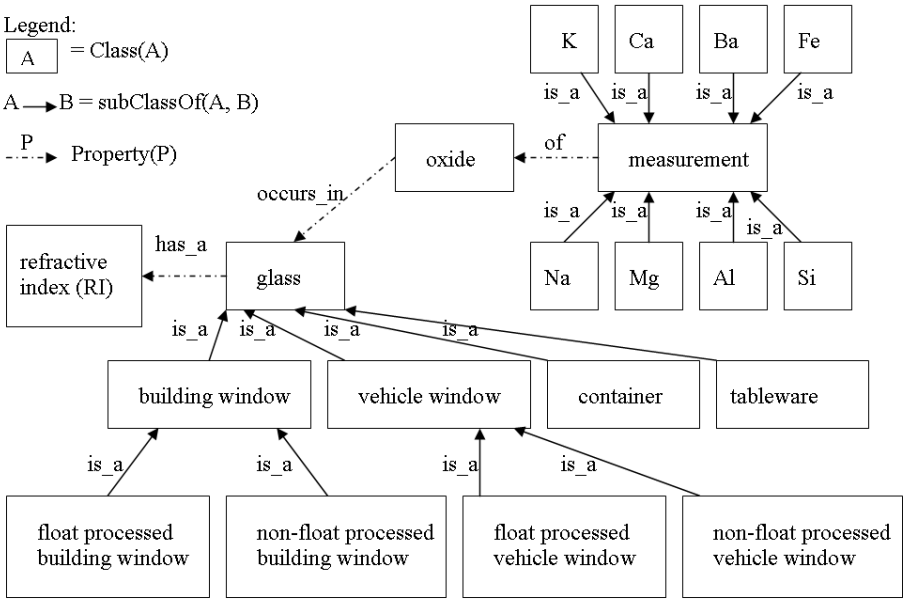

Fig. 3. Glass Identification Ontology

Applying the decomposition storage model [1], [6], we obtained a table for each glass type, i.e. FloatProcessedBuildingWindow, BuildingWindow, Glass, Container, and so on. We implement the subclass relationships in Glass Ontology, which form a subsumption hierarchy, through triggers. For instance:

```
CREATE TABLE FloatProcessedBldWin(
subject VARCHAR NOT NULL,
object VARCHAR NOT NULL,
CONSTRAINT fk-FloatProcessedBldWin-BuildingWindow-Glass
FOREIGN KEY subject, object
REFERENCES Sisters(subject, object) ...)
```

We implement domain and range restrictions as foreign key constraints. These restrict the possible set of instances that participate in property assertions. For example, *Only Windows may participate in the Glass - BuildingWindow relationship*. Restrictions correspond to integrity constraints.

We apply Lehigh University Benchmarks (LUBM) [16] to evaluate the performance of this system. It consists of a set of 14 queries for evaluating load time and query time of knowledge bases. A data generator generates assertins, i.e. $datainstances$, which can be saved as a set of Web Ontology Language (OWL) files and loaded into a knowledge base for evaluation. The main idea is to vary the size of the data instances to quantify the scalability of a Semantic Web Knowledge Base. Mesuring both load time and query time with respect to the imput parameters provides an evaluation of the total performance. We confirmed that by using triggers to materialize inferences, query performance improves by several orders of magnitude. That comes at the cost of load time. Load time is slower. The proposed trigger method increases the disk space required by roughly three times. In other words, our proposed method improves query speed, by costing more space. The proposed approach scales well to very large datasets, and medium-sized ontologies.

Neural ElectroMagentic Ontology. Our ontology database system can be applied to a Neural ElectroMagentic Ontology (NEMO) [9]. NEMO records experimental measurments from brainwave studies, which classify, label, and annotate event related potentials (ERP) using ontological terms. Brainwave activity is measured when certain event happens - such as a word or a sentence is read or heard. Information about scalp distribution, and neural activity during cognitive and behavioral tasks is included. A partial representation is shown on Figure 4.

We show that our propsoed method is useful for answering queries that take subsumption into account - answering queries deductively. In other words, we are able to answer Ontology-Based Queries. For example, the following query requires taking the submsumption hierarchy into account:

Return all data instances that belong to ERP patter classes,
which have a surface positivity over frontal regions of interst and
are earlier than the N400.

In this query, *frontal region* can be unfolded into constituent parts (ex. right frontal, left frontal) as shown in Figure 4. At higher abstraction level, the *N400* is a pattern class that is also associated with spatial, temporal, and functional properties. The patter class labels can be inferred by applying a set of conjunctive rules. This can be implemented by using Semantic Web Rule Language.

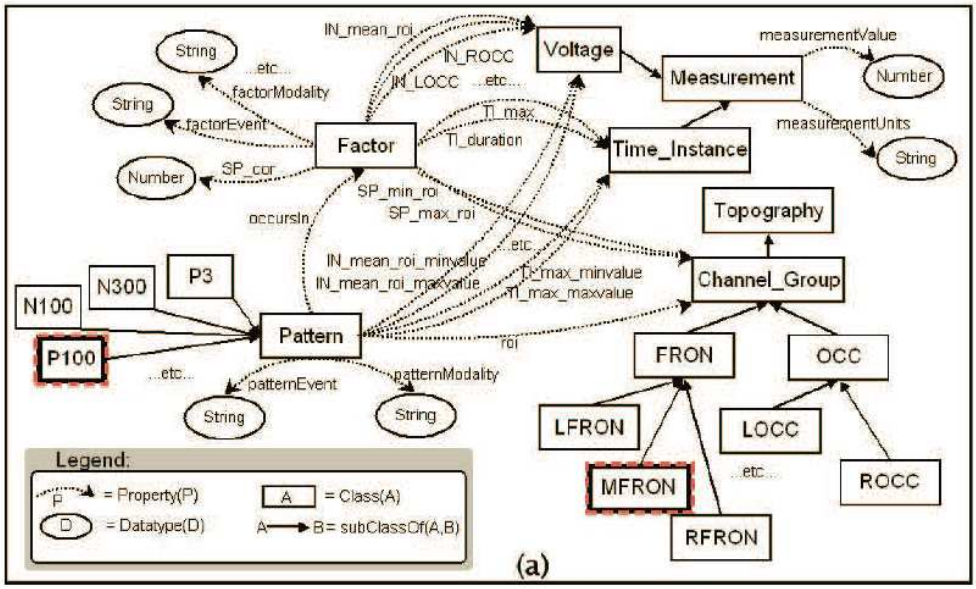

Fig. 4. Neural ElectroMagnetic Ontology (NEMO)

Preliminary results indicate that neurosientists are attracted by the ability to pose queries at the conceptual level, without having to formulate SQL queires; which, would require taking complex logical interactions and reasoning aspects into consideration. Those high-level, logical interactions are modeled only once by specifying the ontology. Other examples of conceptual level queries include:

Which patterns have a region of interest that is left-occipital and manifests between 220 and 300ms?

What is the range of intensity mean for the region of interst for N100?

5 Conclusions and Directions for the Future

We present an ontology database system, a tool for modeling ontologies plus large numbers of instances using off-the-shelf database management systems such as MySQL. An ontology database system is useful for answering ontology-based, scientific queries that require taking the subsumption hierarchy and other constraints into account (answering queries deductively). Furthermore, our trigger implementation method scales well with small and medium-sized ontologies, used with very large datasets.

The proposed method pre-computes inferences for the subsumption hierarchy, so larger and deeper ontologies will incur more costly up-front penalties. However, the query answering time is significantly improved at the end.

Mapping rules between ontologies also can be implemented as trigger-rules, giving us an efficient and scalable way to exchange data among a distributed ontology database systems.

With Ontology Database Systems, it is possible to integrate two knowledge bases. The key idea is to map ontology terms together, then to reason over them as a whole, which comprizes - *a merged ontology*. We can use namespaces to distinguish terms from each knowledge base; next, we can map the ontologies together using *bridging axioms* [10]; and finally, we can reason over the entire merged ontology to achieve integration. As a direction for the future, we can adopt the Dou and LePendu [10] approach for ontology-based integration for relational databases.

Further future steps include studying ontology evolution and concept drift to propagate changes within an ontology database. Changes in the ontology affect the structure, rules and data for an ontology database, which makes efficiently managing the knowledge model a challenging problem.

References

1. D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. SW-Store: A Vertically Partitioned DBMS for Semantic Web Data Management. VLDB Journal, 18(2):385406, April 2009.
2. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider,editors. The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2003.
3. F. Baader and W. Nutt. Basic Description Logics. In Description Logic Handbook, pages 4395, 2003.
4. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. Scientific American, May 2001.
5. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In International Semantic Web Conference, pages 5468, 2002.
6. G. P. Copeland and S. N. Khoshafian. A decomposition storage model. In SIGMOD 85: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 268279, New York, NY, USA, 1985.
7. O. Cure and R. Squelbut. A Database Trigger Strategy to Maintain Knowledge Bases Developed Via Data Migration. In EPIA 05: Proceedings of the 12th Portuguese Conference on Artificial Intelligence, pages 206217, 2005.
8. D. Dou, J. Z. Pan, H. Qin, and P. LePendu. Towards Populating and Querying the Semantic Web. In International workshop on Scalable Semantic Web Knowledge Base Systems (SSWS), pages 129142, 2006. co-located with ISWC 2006.
9. D. Dou, G. Frishkoff, J. Rong, R. Frank, A. Malony, and D. Tucker. Development of Neuro-ElectroMagnetic Ontologies (NEMO): A Framework for Mining Brainwave Ontologies. In Proceedings of the 13th ACM International Conference on Knowledge Discovery and Data Mining (KDD), pp. 270279, 2007.
10. D. Dou and P. LePendu. Ontology-based Integration for Relational Databases. In ACM Symposium on Applied Computing (SAC), pages 461466, 2006.
11. A. Frank and A. Asuncion. UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science, 2010.

12. H. Gallaire, J. Minker, and J.-M. Nicolas. Logic and Data Bases. 1977.
13. H. Gallaire and J.-M. Nicolas. Logic and Databases: An Assessment. In ICDT, pages 177186, 1990.
14. C. Goble and R. Stevens. State of the nation in data integration for bioinformatics. Journal of Biomedical Informatics, February 2008.
15. N. Guarino. Formal Ontology in Information Systems. In International Conference on Formal Ontology in Information Systems.
16. Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. Journal of Web Semantics, 3(2-3):158-182, 2005.
17. Y. Guo, Z. Pan, and J. Heflin. An Evaluation of Knowledge Base Systems for Large OWL Datasets. In ISWC 04: Proceedings of the International Semantic Web Conference, pp. 274-288, 2004.
18. V. Haarslev and R. Moller. High Performance Reasoning with Very Large Knowledge Bases: A Practical Case Study. In IJCAI 01: Proceedings of the International Joint Conferences on Artificial Intelligence, pages 161168, 2001.
19. R. A. Kowalski, F. Sadri, and P. Soper. Integrity Checking in Deductive Databases. In VLDB, pages 6169, 1987.
20. P. LePendu, D. Dou, and D. Howe. Detecting Inconsistencies in the Gene Ontology using Ontology Databases with Not-gadgets. In (to appear) ODBASE 09: Proceedings of the International Conference on Ontologies, Databases and Application of Semantics, 2009.
21. B. Motik, I. Horrocks, and U. Sattler. Bridging the Gap Between OWL and Relational Databases. In WWW 07: Proceedings of the 16th International Conference on World Wide Web, pages 807816, 2007.
22. R. Reiter. Deductive Question-Answering on Relational Data Bases. In Logic and Data Bases, pages 149177, 1977.
23. R. Reiter. What Should a Database Know? Journal of Logic Programming, 14(1):127153, 1992.
24. N. Shah, C. Jonquet, A. Chiang, A. Butte, R. Chen, and M. Musen. Ontology-driven Indexing of Public Datasets for Translational Bioinformatics. BMC Bioinformatics, 10, 2009.
25. J. D. Ullman. Principles of Database and Knowledge-Base Systems, Volume I. Computer Science Press, 1988.
26. O. Vasilecas and D. Bugaite. An algorithm for the automatic transformation of ontology axioms into a rule model. In CompSysTech 07: Proceedings of the International Conference on Computer Systems and Technologies, pages 16, New York, NY, USA, 2007. ACM.
27. L. Vieille, P. Bayer, V. Kuchenhoff, A. Lefebvre, and R. Manthey. The EKS-V1 System. In LPAR 92: Proceedings of the International Conference on Logic Programming and Automated Reasoning, pages 504506, London, UK, 1992. Springer- Verlag.