Type Inference Methods and Performance for Data in an RDBMS

Peter McBrien
Imperial College London
180 Queens Gate
London, England
p.mcbrien@imperial.ac.uk

Nikos Rizopoulos
Imperial College London
180 Queens Gate
London, England
nr600@imperial.ac.uk

Andrew Charles Smith
Imperial College London
180 Queens Gate
London, England
acs203@imperial.ac.uk

ABSTRACT

In this paper we survey and measure the performance of methods for reasoning using OWL-DL rules over data stored in an RDBMS. OWL-DL Reasoning may be broken down into two processes of **classification** and **type inference**. In the context of databases, classification is the process of deriving additional schema constructs from existing schema constructs in a database, while **type inference** is the process of inferring values for tables/columns from values in other tables/columns. Thus it is the process of type inference that is the focus of this paper, since as data values are inserted into a database, there is the need to use the inserted data to derive new facts.

The contribution of this paper is that we place the existing methods for type inference over relational data into a new general framework, and classify the methods into three different types: Application Based Reasoning uses reasoners outside of the DBMS to perform type inference, View Based Reasoning uses DBMS views to perform type inference, and Trigger Based Reasoning uses DBMS active rules to perform type inference. We discuss the advantages of each of the three methods, and identify a list of properties that each method might be expected to meet. One key property we identify is transactional reasoning, where the result of reasoning is made available within a database transaction, and we show that most reasoners today fail to have this property. We also present the results of experimental analysis of representative implementations of each of the three methods, and use the results of the experiments to justify conclusions as to when each of the methods discussed is best deployed for particular classes of application.

1. INTRODUCTION

There is currently a growing interest in the development of systems that store and process large amounts of Semantic Web knowledge [11, 18, 23, 15]. A common approach is to represent such knowledge in OWL-DL [2]. When large quantities of **individuals** in a OWL-DL ontology need to be processed efficiently, it is natural to consider that the individuals are held in a relational database management system (RDBMS), in which case we refer to the individuals as data. Hence, the question arises of how knowledge expressed in OWL-DL can be deployed in a relational database context, and take advantage of the RDBMS platforms in use today to process data in an ontology. This paper surveys and benchmarks methods that have been developed to allow RDBMS platforms to store such ontology data, and contributes a new classification of these methods, and identifies properties of these methods. One important property newly identified by this paper is that of **transactional reasoning**, where the results of reasoning are made available within the transaction where data used as a basis for the reasoning is first inserted into the database.

To illustrate the issues we address in this paper, consider a fragment from the **terminology box** (**TBox**) of the **University Ontology Benchmark** (**UOBM**) [14] expressed in DL:

$$\mathsf{ScienceStudent} \equiv \mathsf{Student} \sqcap \exists \mathsf{hasMajor}.\mathsf{Science} \quad (1)$$
$$\mathsf{Student} \equiv \mathsf{Person} \sqcap \exists \mathsf{isStudentOf}.\mathsf{Organization} \quad (2)$$
$$\mathsf{Person} \equiv \mathsf{Man} \sqcup \mathsf{Woman} \quad (3)$$
$$\mathsf{Man} \sqcap \mathsf{Woman} \sqsubseteq \bot \quad (4)$$
$$\mathsf{UndergraduateStudent} \sqsubseteq \mathsf{Student} \quad (5)$$
$$\mathsf{GraduateStudent} \equiv \geq 1.\mathsf{takesCourse} \sqcap \forall \mathsf{takesCourse}.\mathsf{GraduateCourse} \quad (6)$$
$$\mathsf{GraduateCourse} \sqsubseteq \mathsf{Course} \quad (7)$$
$$\mathsf{Science} \sqsubseteq \mathsf{AcademicSubject} \quad (8)$$
$$\mathsf{Computer\_ScienceClass} \sqsubseteq \mathsf{Science} \quad (9)$$
$$\mathsf{worksFor} \sqsubseteq \mathsf{isMemberOf} \quad (10)$$
$$\mathsf{isHeadOf} \sqsubseteq \mathsf{worksFor} \quad (11)$$
$$\mathsf{isTaughtBy} \equiv \mathsf{teacherOf}^- \quad (12)$$
$$\top \sqsubseteq \forall \mathsf{hasMajor}.\mathsf{AcademicSubject} \quad (13)$$
$$\top \sqsubseteq \forall \mathsf{isStudentOf}.\mathsf{Organization} \quad (14)$$
$$\top \sqsubseteq \forall \mathsf{isStudentOf}^-.\mathsf{Student} \quad (15)$$
$$\top \sqsubseteq \forall \mathsf{enrollin}.\mathsf{Department} \quad (16)$$
$$\top \sqsubseteq \forall \mathsf{takesCourse}^-.\mathsf{Student} \quad (17)$$

A full tutorial on description logics can be found in [4], but the rules have an intuitive logical reading. For example, (1) states that set of things that are a member of class ScienceStudent is equivalent to the set of things that are a member of class Student intersected with the those things that are related by property hasMajor to things that are a member of class Science. Note that in the TBox, to differentiate between classes and properties, classes start with an upper case letter, *e.g.* GraduateStudent, and properties start with a lower case letter, *e.g.* takesCourse.

For the purposes of this paper, we take the meaning of **classification** to be the process of infering new TBox rules from existing TBox rules. We give three example rules that may be derived in classification of the UOBM TBox:

1. from rule (3) we know that every instance of both Man and Woman must also be an instance of Person, and hence may infer two new rules:

   $\mathsf{Man} \sqsubseteq \mathsf{Person}$
   $\mathsf{Woman} \sqsubseteq \mathsf{Person}$

SWIM 2012, May 20, 2012, Scottsdale, Arizona, USA.

i.e. that each Man is also a Person, and so is each Woman a Person.

2. from (17), we know the domain of takesCourse is always a Student, and (6) states that every GraduateStudent appears in the domain of takesCourse, so we can infer:

$$\text{GraduateStudent} \sqsubseteq \text{Student} \quad (18)$$

We can say that classification of a TBox is **complete** when we have derived all possible rules that can be derived.

Data in DL is represented by naming **individuals** (which conventionally start with a capital letter) as members of classes or properties, such data in DL being called the **assertion box** (**ABox**)). An example of an ABox that could be associated with the UOBM TBox is:

$$\text{GraduateStudent}(\text{Peter}) \quad (19)$$
$$\text{Computer\_ScienceClass}(\text{Computer\_Science}) \quad (20)$$
$$\text{hasMajor}(\text{Peter}, \text{Computer\_Science}) \quad (21)$$
$$\text{takesCourse}(\text{Peter}, \text{C526}) \quad (22)$$
$$\text{takesCourse}(\text{Peter}, \text{C531}) \quad (23)$$
$$\text{enrollIn}(\text{Peter}, \text{ICDoC}) \quad (24)$$
$$\text{undergraduateDegreeFrom}(\text{Peter}, \text{Cambridge}) \quad (25)$$
$$\text{supervisedBy}(\text{Peter}, \text{Susan}) \quad (26)$$
$$\text{isHeadOf}(\text{Susan}, \text{ICDoC}) \quad (27)$$
$$\text{isTaughtBy}(\text{C526}, \text{John}) \quad (28)$$
$$\text{teacherOf}(\text{Susan}, \text{C531}) \quad (29)$$
$$\text{worksFor}(\text{John}, \text{ICDoC}) \quad (30)$$

The process of **type inference** will infer membership of individuals in classes or properties from membership of individuals in other classes or properties. Note, it is well known [4] that there is an important distinction here between the standard closed world semantics of a relation database, and the open world semantics of OWL-DL. For example, from (19) and (18) we infer using OWL-DL:

$$\text{Student}(\text{Peter}) \quad (31)$$

By contrast, in a DBMS one might think it correct to represent the two DL classes as two tables, with a foreign key as follows:

Student(id)
GraduateStudent(id)
GraduateStudent.id → Student.id

However, with this structure, an attempt to insert value 'Peter' into GraduateStudent would fail with a foreign key violation unless value 'Peter' were already in Student. But as we have shown, in the open world of OWL-DL, this should infer that Peter is an instance of Student, unless it can be proved that Peter cannot be an instance of class Student. Since this paper is reviewing techniques of performing open world reasoning for data in a DBMS, methods to support type inference are unable to use foreign keys as illustrated above.

To further illustrate the process that needs to be supported, in OWL-DL the following type inferences in the ABox can all be performed:

1. Perhaps the most simple and common type of inference is to infer membership of a superclass in a hierarchy from membership of a subclass in the hierarchy (which we will term reasoning up the hierarchy). For example, for someone that isHeadOf something, we can infer by rule (11) that someone is also worksFor that same thing, and by rule (10) isMemberOf that same thing.

   Hence from ABox (27) we can infer:

$$\text{worksFor}(\text{Susan}, \text{ICDoC}) \quad (32)$$
$$\text{isMemberOf}(\text{Susan}, \text{ICDoC}) \quad (33)$$

2. A similar type inference up the hierarchy may be made on classes. From (20) and (9) we can infer:

$$\text{Science}(\text{Computer\_Science}) \quad (34)$$

3. Once this inference has been performed, we may use (34) with (1) and (31) to infer that ScienceStudent(Peter). Note that here we derive a member of a subclass from information associated with the superclass, which we will term reasoning down the hierarchy.

4. From the ABox fact (19), and the TBox rule (6), we know that the range of any instance of takesCourse that has range Peter must be an instance of GraduateCourse. Hence form (22) and (23) we infer GraduateCourse(C526) and GraduateCourse(C531).

We say that type inference is **complete** when all possible facts (data) have been derived from the supplied ABox.

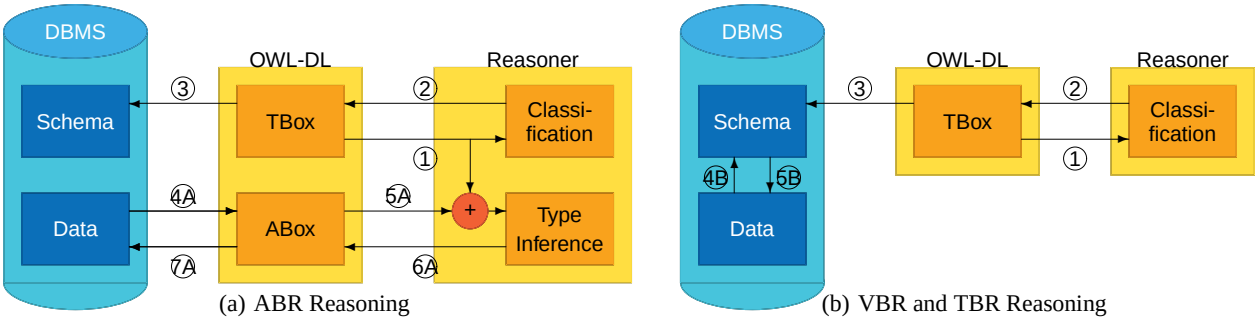
From this short example over a small fragment of a TBox and ABox, we see that there may be a considerable amount of reasoning to perform and data generated. Hence there is considerable scope for different techniques to be deployed in the process of type inference over relational data, and as we will conclude, those techniques may have advantages for certain classes of application. It is this problem that we focus on in this paper by (1) contributing a new general framework for the process, (2) identifying key requirements of the process and identifying how we expect different techniques to meet those requirements, and (3) giving the results of experimental evaluation of one existing implementation of each of the different techniques.

The remainder of this paper is structured as follows. Section 2 gives an outline of how the various techniques for reasoning developed in research or commercial systems can be regarded as fitting into new general framework for type inference over relational data. We then experimentally evaluate the techniques in Section 3. We give a more detailed description of related work in Section 4, and give our summary and conclusions in Section 5.

2. A FRAMEWORK FOR RDBMS TYPE INFERENCE

We now propose a general framework for performing type inference in relational databases, into which we believe all existing techniques for performing type inference on relational data may be represented.

Regardless of the schema used to represent data, there are at least three possible techniques for performing type inference on data held in an RDBMS. In what we term **view based reasoning** (**VBR**), the type inference is performed by writing views that derive type information using database queries. In what we term **trigger based reasoning** (**TBR**), the type inference is performed by writing triggers on relations that derive type information as data is inserted or updated in the database. Finally, in what we term **application based reasoning** (**ABR**), although data is held in the RDBMS, it is extracted by some application (either run by the DBMS engine, or in a completely separate process), and reasoning performed over the data held in the data structures within the application, rather than with the DBMS data structures.



(a) ABR Reasoning (b) VBR and TBR Reasoning

Figure 1: Framework for performing type inference reasoning with a DBMS

All three options (summarised in Figure 1) have the same first three steps:

① The TBox is loaded into a reasoner (such as Pellet [19], or FACT++ [22]), and classification is performed. Since this process is performed infrequently (*i.e.* only after the TBox is changed), the emphasis in this step should be on producing a complete result, rather than a fast result.

② The complete version of the TBox is exported from the reasoner.

③ The TBox is used to create a relational database capable of holding any ABox data associated with the classes and properties in the TBox. For VBR and TBR there will be additional constructs in the schema that perform the type interference within the DBMS engine.

For the ABR type inference, any ABox (*i.e.* data) is first inserted into the database, and when it is desired to perform reasoning the following steps are performed:

4A Any data created in the DBMS is extracted as an OWL ABox.

5A The ABox can be combined with the TBox from ③ and loaded into a reasoner to perform type inference over the data.

6A The result of the type inference is exported from the OWL-Reasoner as a new ABox.

7A Optionally, the ABox is loaded as data back into the database.

By contrast, for the VBR or TBR type inference:

4B Data inserted into the DBMS is immediately available to views or triggers in the schema

5B The type inference is performed by execution of those views or triggers. Due to limitations on the definition of SQL views and triggers, the completeness of the reasoning performed will be limited.

In order to analyse the behaviour of each of the three methods, we will now identify requirements for type inference in a RDBMS context:

- **completeness**: are all possible inferences what can be drawn from the data using the TBox made available in the ABox? Note that for large ABoxes, the emphasis of the reasoner may be on speed rather than providing a complete result, since the most complete reasoners are known to be also slow at handling large ABoxes.

- **transactional reasoning**: are the results of reasoning over newly inserted data available upon completion of the transaction in which that data was inserted?

  The opposite is a **non-transactional** system where data may be inserted into the DBMS, and queries made by other transactions read that data before the results of reasoning are made available.

  The consequence of not using transactional reasoning is that the ACID properties [9] of the DBMS are broken, and that a user may see an inconsistent view of the data. For example, consider if the ABox data (27) was inserted into the DBMS, and the transaction completed before the inference of (32) and (33) were made. If a user views the database just after the transaction completion, then the user will see that Susan is a head of ICDoC without working for ICDoC nor being a member of ICDoC, despite that being a constraint on the database implied by the OWL-DL rules.

  A database textbook approach would normally assume we must maintain the ACID properties of transactions, and hence would regard the non-transaction method as unacceptable. Commercial practice is that some violation of ACID properties is sometimes permitted, via the use of SQL isolation levels. Also, the No-SQL [7] community often promotes that ACID properties be abandoned in favour of fast data access. Hence we regard a non-transactional method as still of interest for certain applications: namely those that do not require the ACID properties of transactions.

- **fast data insertion**: does the reasoning system increase the time taken for inserting data into the DBMS?

  A corollary of fast data insertion is that the approach will not be incremental in generating the results of reasoning after updates to data. If reasoning was performed after each data insert, then there would be an increase in the time associated with inserting data into the DBMS.

- **fast query answering**: are the results of reasoning available for execution times similar to those of querying a normal database table containing the same data?

  This implies in general that the extent of reasoning will be cached in some way, and the only technique available in current implementations is the TBR approach, materialising the reasoning at insert time. Hence in Table 1, fast query answering is only available where fast data insertion is not, and *vice versa*.

- **data compactness**: is the size of the database substantially increased due to holding inferred data?

Data compactness implies not materialising the results of reasoning, but instead generating the results of reasoning at query time.

- **no need for UNA**: can the system not make the UNA?

  The **unique name assumption** (**UNA**) means that each distinct identifier value may be assumed to a distinct real world entity. By contrast, the non-UNA means that more than one identifier value may be used to represent the same real world entity.

We will now discuss each of the approaches in more detail, and review how well we expect each approach to handle these requirements. A summary of this comparison is found in Table 1.

	VBR	TBR	ABR
example system	DLBD2	SQOWL	SOR
completeness	poor	medium	good
transactional	✓	✓	✗
fast data insertion	✓	✗	✓
fast querying	✗	✓	✗
data compactness	✓	✗	✓
no need for UNA	✗	✗	✓

Table 1: Summary of requirements met by VBR, TBR and ABR

2.1 View Based Reasoning

In VBR, as exemplified by DLDB2 [18], it is necessary (due to the definition of SQL) to draw a distinction between intentional and extensional data, and store for any relation R the intentional data in R_I, and write a view definition to derive the extensional data R_E. Then the contents of R are simply the union of R_I and R_E.

For example, in the UOBM, worksFor is both asserted and inferred. In particular, from (11) we know that any members of isHeadOf also are members of worksFor, but we also have ABox assertions for worksFor such as (30). Hence, writing the view statements in Datalog we have:

```
worksFor(X, Y) :- worksForE(X, Y)
worksFor(X, Y) :- worksForI(X, Y)
worksForI(X, Y) :- isHeadOf(X, Y)
isHeadOf(X, Y) :- isHeadOfE(X, Y)
isHeadOf(X, Y) :- isHeadOfI(X, Y)
isHeadOfE(Susan, ICDoC)
worksForE(John, ICDoC)
```

Similarly, rule (1) requires that ScienceStudent be implemented in Datalog as the union of the extensional data in ScienceStudentE and the view defined by ScienceStudentI.

```
ScienceStudent(X) :- ScienceStudentE(X)
ScienceStudent(X) :- ScienceStudentI(X)
ScienceStudentI(X) :-
  Student(X), hasMajor(X, Y), Science(Y).
```

and the same rule implies that part of the implementation in Datalog of Student contains:

```
Student(X) :- StudentE(X)
Student(X) :- StudentI(X)
StudentI(X) :- ScienceStudent(X).
```

The fact that current SQL implementations do not permit such mutual recursion means that current implementations of VBR must choose to not implement either reasoning up the hierarchy (*e.g.* in this case deriving Student entries from ScienceStudent entries), or not to implement reasoning down the hierarchy (*e.g.* in this case deriving ScienceStudent entries from Student entries). Current implementations of SQL do support a limited form a recursion, so for example, transitive properties should be fully supported, though DLDB2 which we used for our experiments does not support compilation to recursive SQL.

We make the observation that reasoning down the hierarchy is more likely to be of interest (by which we mean no more than it makes less obvious inferences), since reasoning up the hierarchy is often limited to maintaining subsumption. However this does not negate the fact that this technical problem severely limits the completeness of reasoning using the VBR approach.

Since the result of updates to a table used within a view definition results in an immediate change in the view extent, the VBR approach will be transactional, unless the views are materialised and not updated on each insertion into the database. We are not aware of any VBR system that uses materialised views, and do not consider it an approach likely to yield benefits. Firstly, if materialised views are not maintained after each update, VBR is no longer supporting transactional reasoning. Then the fact that our experiments show VBR to be slower at query answering than ABR mean that without VBRs advantage over ABR of supporting transactional reasoning, there seems little point in adopting the VBR approach over ABR. Secondly, if materialised views are maintained after each update, then VBR in effect becomes the TBR approach we discuss in the next subsection, except that the views are not kept updated.

Since in VBR nothing is executed as the result of a data insertion or deletion, there are no overheads in inserting data into the database when using VBR. However, the result of reasoning is not cached in any manner, and therefore we expect relatively poor query answering times. Finally, since the reasoning process is conducted within the DBMS, the relational concept of sets of values representing sets of entities means that implicit use of the UNA must be made. For example, we cannot assume that data values 'Peter' and 'Susan' in a table Student could represent the same real world entity and not distinct entities.

2.2 Trigger Based Reasoning

In TBR, as exemplified by SQOWL [15], the TBox is compiled into a set of DBMS triggers statements, that implement a rule based reasoner within the DBMS engine. In particular, the result of reasoning is held in extensional form, with each update to the database resulting in the immediate derivation of new facts to insert into the database.

For example, consider again the worksFor case discussed above. Due to (11), it is the case that any insertion into the isHeadOf property will imply that the same data must be present in the worksFor property. This may be implemented as a trigger statement of the form:

```
when isHeadOf(X,Y)
if   ¬ worksFor(X,Y)
then worksFor(X,Y)
```

Note that the if condition is necessary to ensure that no insertion of values already present in worksFor occur, and this removes the problem we had with VBR in dealing with cyclic definitions. Specifically, dealing with implementing rule (1), we have the triggers:

```
when ScienceStudent(X)
if   ¬ Student(X)
then Student(X)
```

and

```
when Student(X)
if   hasMajor(X,Y),Science(Y),¬ ScienceStudent(X)
then ScienceStudent(X)
```

Note that there is no restriction on having such a cycle using SQL triggers, and the problem of a value being inserted causing an infinite loop is prevented by the negation in the condition part of

the trigger. Thus TBR is able to provide more complete reasoning than VBR approaches. Also, since triggers are executed as part of a transaction execution, the results of reasoning are available before the transaction completes, and hence TBR is transactional.

Inserts into the database will often cause multiple triggers to fire, and because of the checks to ensure that neither duplicates inserts nor cycles of triggers are occurring, there is a significant overhead to performing inserts, and thus we expect TBR to be slower than VBR at handling inserts. Since the reasoning is being materialised in tables, there will also be an overhead in data storage used, but query processing will be relatively fast since the result of reasoning is available in tables, ready for the DBMS query optimiser to execute queries efficiently over.

Finally, like VBR, the fact that reasoning is performed within the DBMS means that the UNA must be made.

2.3 Application Based Reasoning

In ABR, as exemplified by SOR [12], the type inference is performed by an application run with data loaded from the database. Note that in principle any OWL-DL reasoner could be used in such an approach (such as a Prolog application), and the reasoner can even be supplied as part of the database system, such as in Oracle 11g [1], or written as user defined functions in the DBMS. The key feature is that reasoning is not performed as part of transaction execution, but rather executed as an application taking as input the committed results of a transaction execution.

Since we are not restricted in which reasoner is used, the ABR approach will always have the possibility of providing the most complete results of reasoning. Also, the use of external reasoners free from relational concepts of sets of entities means there is no requirement to use the UNA. However, the fact that an application is being used means that all of the ABox data (or if possible to determine, just the data required fpr reasoning) must be read into the reasoner in a **load** phase, before reasoning can be performed. For large ABoxes, this load time will preclude use of reasoning with a transaction (since it means that every transaction must be delayed by the time taken for a ABox load). Hence we summarise ABR as being non-transactional.

Unless the result of reasoning is written back into the database, query processing must also be performed by the external application, and not within the DBMS. We would thus expect that for complex queries (*i.e.* those involving several classes and properties in some form of join), the DBMS querying would be more efficient, and the ABR approach have less performance than the VBR and TBR approaches.

3. EXPERIMENTAL EVALUATION

Having analysed in Section 2 what performance we expect from VBR, TBR and ABR, we now present an experimental evaluation of three representative implementations of the approaches. The next subsection reviews the three benchmarks we have used in these experiments, and the following three Subsections 3.2, 3.3 and 3.4 then present the results of running the three benchmarks. Finally, Subsection 3.5 compares the approaches, and gives a simple method for determining which method to select for a given database and type of query load.

3.1 OWL-DL Reasoner Benchmarks

We have chosen three ontologies for testing the three reasoning approaches. The **Lehigh University Ontology Benchmark** (**LUBM**) [8], the UOBM [14] that has been used so far as a running example, and the W3C **Wine ontology** [3]. These three ontologies pose different type inference challenges, though it is recognised that no benchmark fully tests OWL-DL reasoning systems [21]. LUBM requires only simple inference, while UOBM is a development of LUBM to include more complex reasoning. Both come with a data generator that can be used to create ontologies with large numbers of individuals. Thus we could use them to test the scalability of the three reasoning approaches on simple and more complex type inference problems. In contrast, the Wine ontology is supplied with a small set of example individuals, has no data generator, but has the most complex TBox of the three examples. We used this ontology to test the inference capabilities of different approaches. Other ontologies that have been used to benchmark DL reasoning systems [17] test their classification capabilities. As our framework always uses a reasoner outside of the DBMS to perform classification, these benchmarks are not relevant to the problem of type inference we address in this paper.

LUBM and UOBM come with a set of queries, which we used to test the soundness of our approach. To test approaches against the Wine ontology, we compared the type inference results with those of Pellet [19], the well known tableaux reasoner for $\mathcal{SHOIN}(\mathbf{D})$, and used its results as the reference for what instances should be inferred. Hence all three benchmarks could be used to demonstrate the soundness and completeness of the three implementations under trial. All approaches proved to be sound, and thus we only report on the results of completeness in this paper.

The tests were carried out on a Windows Vista platform with an Intel Centrino 2 processor running at 2.6 GHz and 4GB of RAM. The Java VM was allocated 800M bytes. The implementation of DLDB2 used was part of Hawk version 3, and used MySQL as the RDBMS (we used version 5.1 for benchmarks). The implementation of SQOWL used Postgres as a DBMS (we used version 8.3 for benchmarks). The implementation of SOR uses DB2 as a DBMS (we used DB2 Express version 9.7 for the benchmarks). Obviously the fact that a different DBMS engine is used for each of the approaches reduces the degree to which the results are directly comparable. However, we argue that for the relatively small sizes of data being used in the experiments the differences in DBMS performance will not be significant enough to prevent the results being useful in validating the analysis of the reasoning techniques made in Section 2.

3.2 LUBM

The LUBM ontology [8] classifies individuals in fictitious universities. These individuals may be members of staff, publications, departments, research groups *etc*. Each university contains about 100000 individuals and property tuples. LUBM(1) contains one university, LUBM(2) two universities, and so on. There are 14 example queries provided with the ontology which we here number L1–L14. In our experiments we measured the execution time and completeness of each query. All queries are listed in [8], but for ease of reference we include and discuss two queries:

L1 below takes a large input but is very selective, and also does not need to perform inference to gain a complete result set (*i.e.* inference adds no data to that explicitly classified under GraduateStudent and takesCourse).

```
SELECT ?X
WHERE {
        ?X rdf:type ub:GraduateStudent .
        ?X takesCourse
        http://www.Department0.University0.edu/GraduateCourse0
}
```

The performance of L1 gives a comparison of the performance of each approach where no inference is necessary to give a complete result to a query (though inference might be attempted to verify this).

L6 below only queries the Student class (which has no directly asserted instances), but by inference using rule (5) from the introduction can infer that UndergraduateStudent instances are also Student instances, and using (18) that GraduateStudent instances are also Student instances.

```
SELECT ?X
WHERE {?X rdf:type ub:Student}
```

L6 is not selective, and hence gives a indication of the relative performance where inference is necessary to produce the result of the query.

SQOWL and SOR were able to answer all the LUBM queries. DLDB2 failed on Q11 returning no results. Tables 2 show the query response times for the fourteen LUBM queries on LUBM(1) and LUBM(3) respectively (omitting LUBM(3) for DLDB2 since the execution times became so long). As expected SQOWL and SOR exhibit much faster query response times than DLDB2 because they materialise their inference results at load time into the DBMS, or upon loading the DBMS data into the reasoner. Since SQOWL executes its queries in the DBMS engine (with its good query optimisation capabilities), rather than in the data structures of the SOR, we expect a performance increase for SQOWL over SOR. The percentage increase is greater on LUBM(3) than LUBM(1), reflecting the fact that the benefits of DBMS query optimisation are particularly felt for larger datasets.

3.3 UOBM

The UOBM [14] is an extension of the LUBM with only slightly higher number of individuals, but requiring more complex reasoning, since it uses more of the OWL-DL constructs. There are two versions of the UOBM, an OWL-DL version (used here) and an OWL-Lite version.

We used the 13 queries provided with UOBM in [14] to test the approaches, which we here number U1–U13. All results for systems tested were sound, but DLDB2 was only able to answer three of the queries completely, as it is not able to perform as many type inference tasks as SQOWL or SOR. The completeness results are shown in Table 3. Query U1 is very similar to query L1, except it queries the UndergraduateStudent rather than the PostgraduateStudent class, and all approaches can infer all answers in times comparable with L1. However many of the UOBM queries need much more complex reasoning than LUBM, and DLDB2 fails to provide complete answers. Taking U12 listed below as an example:

```
SELECT DISTINCT ?x
WHERE {
        ?x rdf : type ub:Student .
        ?x ub:takesCourse ?y.
        ?y ub:isTaughtBy
        http://www.Department0.University0.edu/FullProfessor0
}
```

The instances of isTaughtBy are mostly inferred from instances such as (29) of the inverse property teacherOf in rule (12), and DLBP2 does not handle such inverse properties.

In the case of U6, U8, U10, and U11 SQOWL misses one individual due to SQOWL using the UNA. SQOWL provides a more complete answer to U13 than SOR, since SOR does not handle minimum cardinality fully [23], contradicting our expectation that ABR should always be able to infer more that TBR. However, since SOR uses a rule based reasoner, it is more comparable to capabilities of a TBR based approach.

The query times for the thirteen UOBM queries are shown in Table 4. We have not included query times for DLDB2 as it provided complete answers for so few of the queries. Once again SQOWL query times are faster than SOR. The complexity of the inference required in UOBM is somewhere between that of LUBM and Wine ontology so the difference in times here is in line with expectations.

Name	DL reasoning	Rules engine
O-DEVICE	$\mathcal{SHIQ}$ + hasValue	CLIPS rule engine
Dlog	$\mathcal{SHIQ}$	Prolog
KOAN2	$\mathcal{SHIQ}$	Bespoke Datalog engine
Oracle11g	OWLPrime	Application
OWLIM	RDFS	Bespoke + Sesame
DLDB2	$\mathcal{SHOIN}(D)$ classification, DLP type inference	RDBMS (Views)
SOR	$\mathcal{SHOIN}(D)$ classification DLP type inference	Java application
SQOWL	$\mathcal{SHOIN}(D)$	RDBMS (Triggers)

Table 5: Rule based data reasoners

	W1		W2	
SOR	50%	16.0ms	100%	15.0ms
SQOWL	100%	0.5ms	100%	0.5ms

Table 6: Wine Completeness and Execution Times

3.4 The Wine Ontology

We chose the Wine ontology [3] because it is the well known example OWL-DL ontology provided by the W3C. Also, since it includes an example of each type of OWL-DL construct, type inference requires the processing of all the OWL-DL constructs. There is no benchmark set of queries used to test type inference for the Wine ontology, so we propose two benchmark queries. For these reasons we test our results against the type inference calculated by the tableaux based reasoner Pellet, embedded in Protege 3.4.1 [20]. The queries were designed to require a reasonably complicated inference to be performed on the ontology. W1 uses inference both up and down the hierarchy to infer values of WhiteLoire:

```
SELECT DISTINCT ?X
WHERE{ ?X rdf : type wine:WhiteLoire}
```

W2 infers instanced of AmericanWine:

```
SELECT DISTINCT ?X
WHERE{ ?X rdf : type wine:AmericanWine}
```

The number of results for each query compared to the results obtained from Pellet, as well as the time taken, are shown in Table 6 (omitting DBLP2 since it failed to provide any answers). SQOWL was able to correctly infer all the WhiteLoire and AmericanWine instances, but SOR could not correctly infer one of the two WhiteLoire instances. The difference in query result times is greater here than in LUBM because of the greater complexity of the inference required to answer the queries. There are several joins for each restriction on a derived class that appears in a query.

3.5 Comparison of Performance

In any database application, there will be a mixture of updates and queries to process, though the balance between the quantity of each may vary considerably. Thus we need to consider the trade-off there is between taking the TBR approach which actively materialises the data derived by reasoning (and hence will be expected to be slower during loading of the data into the database), and taking the VBR or ABR approach where reasoning is performed on demand for answering a particular query (and hence will be expected to be slower during query execution).

System	Size	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12	L13	L14
DLDB2	LUBM(1)	706	841	539	912	788	18649	19652	128096	20246	19877	–	110	538	67
SOR	LUBM(1)	4	47	3	30	9	16	7	114	58	3	4	7	3	15
	LUBM(3)	112	3650	150	2205	219	120	210	874	2730	140	168	359	187	93
SQOWL	LUBM(1)	8	12	4	14	6	2	11	30	33	5	2	4	1	2
	LUBM(3)	35	46	23	53	20	6	36	67	150	24	2	3	5	6

Table 2: LUBM Query Answer Times (ms)

	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	U11	U12	U13
DLDB2	100	82	100	100	0	0	0	0	0	83	0	20	56
SOR	100	100	100	100	100	100	100	100	100	100	100	100	64
SQOWL	100	100	100	100	100	98	100	99	100	88	97	100	99

Table 3: UOBM(1) Answer Completeness (%)

	LUBM(1)	LUBM(2)	LUBM(3)	UOBM(1)
DLDB2	4	8	14	9
SOR	3	9	16	7
SQOWL	36	165	346	287

Table 7: Database Loading Times (minutes)

First we consider the time taken to insert data into the database. Table 7 shows that unsurprising result that our two examples from the VBR and ABR approaches are roughly 10 to 30 times faster at inserting data. Offset against this, is the fact that querying data is much faster in the TBR approach. Again this is unsurprising, since TBR materialises the result of reasoning, whilst VBR and ABR compute the result when required for the answering of a query. What is surprising from Table 2 is that VBR is consistently much slower than ABR. However, VBR is still of interest as a method since it provides transactional reasoning which ABR does not.

Comparing the query processing times of TBR and ABR on LUBM(3) in Table 2; we see on average TBR is 3 to 100 times faster at query answering than is ABR.

Hence, the choice between TBR and ABR approaches will depend of the size of the database, the ratio of queries to updates, and the particular types of query being performed. The calculation is straight forward: taking Q6 running LUBM(1) as an example, where there are 100,000 individuals and properties supplied in the ABox (and incidentally, 50,000 infered individuals and properties), for SQOWL we calculate the average tuple insert time from data in Table 7 is $36 \times 60000/100000 = 21ms$, and from Table 2 the average query time is $2ms$. Taking the ratio of queries to updates as q, then we calculate the cost of SQOWL as $21 + 2q$. For SOR, the tuple insert time from data in Table 7 is $3 \times 60000/100000 = 2ms$ and the query time from Table 2 is $16ms$, then we calculate the cost of SOL as $2 + 16q$. Thus SQOWL will be more efficient than SOL when $21 + 2q < 2 + 16q$, so when $q > 1.4$, when on average at least 1.4 queries of the complexity of Q6 are being asked after each update to the database.

4. RELATED WORK

DL reasoners come in a number of forms [4], those based on tableaux algorithms are being the most common. Tableaux based reasoners like Racer, FacT++ and Pellet are very efficient at computing classification hierarchies and checking the consistency of a knowledge base. However, the tableaux based approach is not suited to the task of processing large datasets (*i.e.* large ABoxes) [5], since the tableaux algorithm uses a refutation procedure rather than a query answering algorithm [10].

Rule based reasoners provide an alternative to the tableaux based approach that is more promising for handling large datasets. Some of the best known reasoners are summarised in Table 5. Thus they are more suited to the problem we deal with in this paper, and in general rules can be (at least partially) implemented in any of the VBR, TBR and ABR approaches. However, not all the reasoners in Table 5 even allow storage of data in a DBMS, and only DBLP2 and SQOWL use a non-ABR technique.

O-DEVICE [16] translates OWL rules into an in-memory representation using the CLIPS production rules system and the COOL OO language. It can process all of OWL-Lite, and in addition the OWL-DL constructs `hasValue` and class disjointedness. It does not support `oneOf`, `complementOf` or data ranges. The fact that the system is memory based provides fast load and query times, but means that it does not scale beyond tens of thousands of individuals. OWLIM [11] also does its reasoning in memory. It takes rules already defined for RDFS inference in the SAIL (Storage And Inference Layer) of Sesame [6], and adds support for a small subset of OWL-DL constructs, up to the expressiveness of Horn Logic. In common with O-DEVICE its reasoning is fast, but it cannot reason of large numbers of individuals.

Now turning to database oriented approaches. KAON2 [17] does reasoning by means of theorem proving. The TBox is translated into first-order clauses, which are executed on a disjunctive Datalog engine of their own design to compute the inferred closure. KAON2 displays impressive load and query times [5] but is unable to handle **nominals** (*i.e.* `hasValue` and `oneOf`, labelled as O in DL). DLog [13] adopts a similar approach, but uses Prolog to answer queries on individuals that are stored in an RDBMS. Its performance characteristics are similar to those of KAON2.

SQOWL [15] and DLDB2 [18], are the only two systems that use an RDBMS as their rule engine. DLDB2 stores the rules inside the database as views and does not materialise the inferred closure of the ontology at load time. Tables are created for each atomic role and class which are then populated with the individuals from the ontology. A separate DL reasoner is used to classify the ontology. The resulting TBox axioms are translated into non-recursive Datalog rules that are translated in SQL view create statements. DLDB2 enjoys very fast load times because the inferred closure of the database is not calculated at load time but its querying is slow. An advantage of the system is that because the closure is only calculated when queries are posed on the system, updates and deletes can be performed on the system without any overheads compared to a standard database application.

SQOWL compiles TBox rules into DBMS 'before' and 'after' trigger statements, and from our benchmark analysis provides a reasonably complete implementation of type inference.

	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	U11	U12	U13
SOR	600	280	1400	500	327	1216	608	265	5756	328	11248	561	1248
SQOWL	88	67	215	322	88	129	188	130	57	200	105	60	57

Table 4: UOBM(1) Query Answer Times (ms)

SOR [23, 12] (previously called Minerva) uses a standard tableaux based DL reasoner to first classify the ontology. It then generates rules to perform type inference, but differs from DLDB2 in that rules are kept outside the database, and executed at load time to materialise the results of inference. This makes loading slower but query processing faster. The subset of OWL supported is slightly larger than that of Oracle's OWLPrime, and hence we selected it as the representative of the ABR technique over Oracle 11g.

SOR both uses a meta database schema that is not related to the ontology it is processing, but rather to the OWL-DL constructors the system is modelling. For example, there are tables called hasValue and someValuesFrom. Each derived class whose restrictions include one of these constructors has an entry in the relevant table. At query time joins are created over these tables to provide the necessary reasoning capability.

5. SUMMARY AND CONCLUSIONS

We have proposed a general framework in which current techniques for reasoning on data held in a DBMS (equivalent to type inference in an OWL-DL ontology) can be compared. We identified a set of requirements for type inference over relational data, and summarised the satisfaction of different methods for such type inference in our framework (q.v. Table 1). We identified the concept of **transactional reasoning**, and discussed how this implies the use of DBMS views in **view based reasoning** (**VBR**) or DBMS triggers in **trigger based reasoning** (**TBR**) in order to have the result of reasoning generated as part of a DBMS transaction. However using VBR or TBR limits the completeness of the reasoning. By contrast **application based reasoning** (**ABR**) can achieve completeness by loading the data into a separate reasoner application, but due to the entire dataset having to be loaded into the reasoner this precludes transactional reasoning. Query answering is fastest when the result of reasoning is materialised within the database in the TBR approach, and this approach also efficiently supports incremental updates. By contrast, VBR and ABR provide a more compact storage of data.

In essence, applications which involve frequent updates to data, and require the result of reasoning to available immediately after the updates can the use of VBR method. Applications which require fast query processing should use TBR or ABR, with the fastest provided by TBR. Applications which require more complete reasoning should use ABR. However it should noted that a tableaux based reasoner should be used for the most complete answers, which in turn means that size of the ABox handled will be limited.

Future work in this area might consider investigating how the advantages of each approach can be used within one application. For example, by dividing up the ontology into parts that might handled by different type inference approaches.

6. REFERENCES

[1] Oracle database semantic technologies developer's guide 11g, 2009.

[2] Y. al Safadi *et al*. OWL Web Ontology Language Overview, 2004. http://www.w3.org/TR/owl-features/.

[3] Y. al Safadi *et al*. The wine ontology, 2004. www.w3.org/TR/owl-guide/wine.rdf.

[4] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. CUP, 2nd edition edition, 2007.

[5] J. Bock, P. Haase, Q. Ji, and R. Volz. Benchmarking OWL reasoners. In *ARea2008*, 2008.

[6] 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 54–68, 2002.

[7] R. Cattell. Scalable SQL and NoSQL data stores. *SIGMOD Record*, 39:12–27, 2010.

[8] Y. Guo, Z. Pan, , and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3:158–182, 2005.

[9] T. Härder and A. Reuter. Principles of transaction-oriented database recovery. *ACM Computing Survers*, 15(4):287–317, December 1983.

[10] U. Hustadt and B. Motik. Description logics and disjunctive datalog the story so far. In *Description Logics*, 2005.

[11] A. Kiryakov, D. Ognyanov, and D. Manov. Owlim - a pragmatic semantic repository for OWL. In *WISE Workshops*, pages 182–192, 2005.

[12] J. Lu, L. Ma, L. Zhang, J.-S. Brunner, C. Wang, Y. Pan, and Y. Yu. Sor: A practical system for ontology storage, reasoning and search. In *VLDB*, pages 1402–1405, 2007.

[13] G. Lukácsy and P. Szeredi. Efficient description logic reasoning in Prolog: the DLog system. *Theory and Practice of Logic Programming*, 09(03):343–414, May 2009.

[14] L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu. Towards a complete OWL ontology benchmark. In *ESWC*, pages 125–139, 2006.

[15] P. McBrien, N. Rizopoulos, and A. Smith. SQOWL: Type Inference in an RDBMS. In *Proc. 29th ER*, LNCS, 2010.

[16] G. Meditskos and N. Bassiliades. A rule-based object-oriented OWL reasoner. *IEEE Trans. Knowl. Data Eng.*, 20(3):397–410, 2008.

[17] B. Motik and U. Sattler. A comparison of reasoning techniques for querying large description logic aboxes. In *LPAR*, pages 227–241, 2006.

[18] Z. Pan, X. Zhang, and J. Heflin. DLDB2: A scalable multi-perspective semantic web repository. In *Web Intelligence*, pages 489–495, 2008.

[19] Pellet. http://clarkparsia.com/pellet/.

[20] Protege. http://protege.stanford.edu/.

[21] G. Stoilos, B. Grau, and I. Horrocks. How incomplete is your semantic web reasoner? In *Proc. AAAI*, 2010.

[22] D. Tsarkov and I.Horrocks. Fact++ description logic reasoner: System description. In *Automated Reasoning*, volume 4130 of *LNCS*, pages 292–297. Springer, 2006.

[23] J. Zhou, L. Ma, Q. Liu, L. Zhang, Y. Yu, and Y. Pan. Minerva: A scalable OWL ontology storage and inference system. In *ASWC*, pages 429–443, 2006.