A Study on Query Completeness and Answering Time In Knowledgebases and Databases

A paper submitted in partial fulfillment of the project work requirement for the MS degree in Information and Media Technologies

By:
Ramzi RIZK
Matriculation Number: 27365

Supervised by:

Prof. Dr. Ralf MOELLER
Software Systems Department
HAMBURG-HARBURG UNIVERSITY OF TECHNOLOGY

Abstract

This paper discusses various issues related to differences in querying between a knowledgebase reasoner from the description logic family (Racer) and a relational database (MySQL). It covers generating a relational schema to represent structured data, generating data to populate the created database, and finally running several queries on both the database, and the knowledgebase reasoner. Three different database schemas will be created and tested. The paper discusses the advantages and disadvantages of both approaches, with respect to query scope, size, hierarchy and implicit and explicit relationships. Finally, the two approaches are compared in terms of query answering time and completeness.

Declaration

I hereby declare that I am the author of this paper, titled **"A Study on Query Completeness and Answering Time in Knowledgebases and Databases"**. All literary or content related quotations are clearly pointed out and no other sources are used.

Ramzi Rizk

Hamburg, May 1, 2005

Acknowledgement

I would like to thank Prof. Dr. Moeller for giving me the opportunity to work on this project, and for his valuable and continuous feedback and help in enhancing the scope of this work. I would also like to thank Mr. Michael Wessel, for being always available, and readily offering support and advice.

Table of Contents

TABLE OF FIGURES **7**

TABLE OF TABLES **8**

1. INTRODUCTION **9**

1.1 Motivation and Scope 9
1.2 Design of Contribution 10
1.3 Tools and Packages Used 10
1.4 Structure of This Paper 11

2. MAPPING FROM OWL TO MYSQL **12**

2.1 Requirements 12
2.2 Typical Techniques 12
2.2.1 Horizontal Database *13*
2.2.2 Vertical Database *13*
2.2.3 Horizontal Class *13*
2.2.4 Table per Property *13*
2.3 Implemented Technique 13
2.3.1 First Schema *16*
2.3.2 Second Schema *17*
2.3.3 Third Schema *18*
2.4 Issues 19
2.5 Room for Improvement 20

3. TEST DATA **21**

3.1 Modifying the Generator and Creating Writers 21
3.2 Loading the Data 22
3.3 Experimental Setup 22
3.4 Racer Querying Modes 23
3.4.1 Mode 0 ..... *23*
3.4.2 Mode 1 ..... *23*
3.4.3 Mode 3 ..... *23*

4. THE QUERIES **24**

4.1 Query 1 24
4.2 Query 2 26
4.3 Query 3 27
4.4 Query 4 28
4.5 Query 5 30
4.6 Query 6 31
4.7 Query 7 32
4.8 Query 8 33
4.9 Query 9 34
4.10 Query 10 35
4.11 Query 11 36
4.12 Query 12 38
4.13 Query 13 39
4.4 Query 14 40

5. CONCLUSION **42**

5.1 Answering Time 42
5.2 Completeness 43

5.3 SUGGESTIONS 44
REFERENCES 45

Table of Figures

FIGURE 1: PART OF THE TBOX GRAPH, HIERARCHIES FOR “PERSON” AND “ORGANIZATION” 15
FIGURE 2: TABLES “GRADUATESTUDENT”, “STUDENT” AND “PERSON” IN DB-1 17
FIGURE 3: TABLE “PERSON” IN DB-2 18
FIGURE 4: TABLES "UNIVERSITY" AND "ORGANIZATION" IN DB-2 18
FIGURE 5: TABLES "UNDERGRADUATESTUDENT" AND "STUDENT" IN DB-3 19

Table of Tables

TABLE 1: LOADING TIME (MS) FOR RACER AND THE DATABASES 22
TABLE 2: QUERY 1 RESULTS FOR 14 DEPARTMENTS 25
TABLE 3: QUERY 1 RESULTS FOR 2 DEPARTMENTS 25
TABLE 4: QUERY 2 RESULTS FOR 14 DEPARTMENTS 26
TABLE 5: QUERY 2 RESULTS FOR 2 DEPARTMENTS 27
TABLE 6: QUERY 3 RESULTS FOR 14 DEPARTMENTS 28
TABLE 7: QUERY 3 RESULTS FOR 2 DEPARTMENTS 28
TABLE 8: QUERY 4 RESULTS FOR 14 DEPARTMENTS 29
TABLE 9: QUERY 4 RESULTS FOR 2 DEPARTMENTS 29
TABLE 10: QUERY 5 RESULTS FOR 14 DEPARTMENTS 30
TABLE 11:QUERY 5 RESULTS FOR 2 DEPARTMENTS 30
TABLE 12: QUERY 6 RESULTS FOR 14 DEPARTMENTS 31
TABLE 13: QUERY 6 RESULTS FOR 2 DEPARTMENTS 31
TABLE 14: QUERY 7 RESULTS FOR 14 DEPARTMENTS 32
TABLE 15: QUERY 7 RESULTS FOR 2 DEPARTMENTS 33
TABLE 16: QUERY 8 RESULTS FOR 14 DEPARTMENTS 34
TABLE 17: QUERY 8 RESULTS FOR 2 DEPARTMENTS 34
TABLE 18: QUERY 9 RESULTS FOR 14 DEPARTMENTS 35
TABLE 19: QUERY 9 RESULTS FOR 2 DEPARTMENTS 35
TABLE 20: QUERY 10 RESULTS FOR 14 DEPARTMENTS 36
TABLE 21: QUERY 10 RESULTS FOR 2 DEPARTMENTS 36
TABLE 22: QUERY 11 RESULTS FOR 14 DEPARTMENTS 37
TABLE 23: QUERY 11 RESULTS FOR 2 DEPARTMENTS 37
TABLE 24: QUERY 12 RESULTS FOR 14 DEPARTMENTS 38
TABLE 25: QUERY 12 RESULTS FOR 2 DEPARTMENTS 39
TABLE 26: QUERY 13 RESULTS FOR 14 DEPARTMENTS 39
TABLE 27: QUERY 13 RESULTS FOR 2 DEPARTMENTS 40
TABLE 28: QUERY 14 RESULTS FOR 14 DEPARTMENTS 41
TABLE 29: QUERY 14 RESULTS FOR 2 DEPARTMENTS 41

1. Introduction

1.1 Motivation and Scope

Databases have, for years, been at the forefront of data storage and querying technologies. Advanced research has led to databases that can resolve queries involving copious amounts of data, and return relevant results within a fraction of a second.

Knowledge representation systems from the description logic family offer an alternative approach to the issue of querying data. In contrast to the closed-world assumptions of a database system, knowledge reasoners have open-world assumptions. This means that what is not known is not necessarily false. Furthermore, the semantic knowledge available to these knowledgebase reasoners allow for queries about the structure of an ontology, in addition to the usual extensional queries associated with typical relational databases. Knowledgebase reasoners can infer, or realize new knowledge by combining available data with knowledge pertaining to relations that exist between the particulars of that data.

Knowledge representation systems from the description logic family, such as Racer [Racer, 05], have both Aboxes, and TBoxes. A TBox can be roughly compared to the schema of a database, while as an Abox could be compared to the actual data that populates the tables of this schema. Racer allows users to query both.

The “Description Logic Handbook” [Baader et al.] explains this clearly and briefly: “In a simplified view, an ABox can be seen as an instance of a relational database with only unary or binary relations. However, contrary to the “closed-world semantics” of classical databases, the semantics of ABoxes is an “open-world semantics”, since normally knowledge representation systems are applied in situations where one cannot assume that the knowledge in the KB is complete. Moreover, the TBox imposes semantic relationships between the concepts and roles in the ABox that do not have counterparts in database semantics.”

Therefore, from the given data, one could already assume that knowledge reasoners would be able to infer new relations between data, based on the semantic structures provided. This is not available in database systems, which are merely storage and retrieval systems in their essence (data-mining and data warehousing aside). So, it seems natural to attempt and compare these differences. That would only be possible by generating the same data, in the same form for both a knowledgebase reasoner, and a relational database, and running some identical queries on both. This paper covers that benchmarking exercise.

Racer was chosen as the knowledgebase representation system on which to carry out this exercise. The database schemas and queries will be created and carried out on MySQL.

1.2 Design of Contribution

To successfully carry out the benchmarking exercise mentioned above, several steps had to be taken. First of all, identical database schemas and knowledge base TBoxes had to be created. This was accomplished using an OWL [W3C, 05] based schema which will be discussed in detail later. Data had to be generated that would be used to populate both the database and the knowledgebase reasoner. Finally, the main point of this project was to run several queries on both systems, and compare them in terms of speed and completeness.

1.3 Tools and Packages Used

Several tools and packages were available for me to use throughout this project.

- Racer: According to "The Racer Manual" [Harslev et al. 04], "The Racer system is a knowledge representation system that implements a highly optimized tableau calculus for a very expressive description logic. It offers reasoning services for multiple TBoxes and for multiple ABoxes as well. The system implements the description logic $ALCQHI_{R+}$ also known as SHIQ. This is the basic logic ALC augmented with qualifying number restrictions, role hierarchies, inverse roles, and transitive roles."

- JRacer: Provides a Java layer for accessing the services of Racer [Harslev et al., 04]. JRacer was used to populate Racer, and to time and run the queries.

- Rice: An interactive client environment for Racer developed by Ronal Cornet from the Academic Medical Center in Amsterdam [Rice, 04].

- Generator: The Lehigh University Benchmark data generator [UBA, 04] is a random data generator. It was modified to generate data that complies with both the database schemas, and the Racer TBoxes. The data is based on the Lehigh University Benchmark [Lehigh] which will also be discussed in details later.

- nRQL: Is the New Racer Query Language. It offers advanced querying capabilities for Racer.

- MySQL: Is a popular, free, open source, relational database [MySQL, 05]. It was used to host the database schemas and the data, and to run the test queries. Java applications were developed that connected to MySQL, and ran and timed the queries.

1.4 Structure of This Paper

This paper proceeds according to the following structure: Chapter 2 covers the mapping of an owl document containing the Lehigh University Benchmark ontology to a relational database. General approaches are discussed, and then the three designs that I used are introduced. Some issues that arose are also mentioned. Chapter 3 discusses the task of generating identical random data to populate the database and Racer. It also describes the experimental setting for this project. Chapter 4 presents the test queries used, lists their results, and provides a query-by-query analysis of these results. Finally, chapter 5 concludes with a general analysis of the observations and results of this project work, as well as some suggestions as to what these results imply.

An addendum accompanies this paper on CD-Rom. It contains the script files for creating the databases, the modified generator which produces the sample data, and the sample data that was used in the testing. The CD-Rom also contains all the queries used in this project work.

2. Mapping from OWL to MySQL

2.1 Requirements

Racer can read, and translate OWL documents. Furthermore, the Lehigh University Benchmark (from hereon LUBM) is available in the form of an OWL document [UnivBench, 04]. Coupled with the fact that a generator already exists that creates OWL data based on this University Benchmark, and that standard queries created by the Lehigh University Benchmark team were a quasi standard for testing this ontology, it seemed natural to want to build on this, by adapting all of the above to work with MySQL.

The LUBM models a university with departments, students, faculty, staff, courses, and research groups. The general schema has five main categories: “Person”, “Schedule”, “Work”, “Publication” and “Organization”. Each of these is expanded in detail. The LUBM provides an OWL ontology document that pertains to that structure. That ontology is the basis of the database schemas developed here.

OWL (Web Ontology Language) is a W3C standard for the semantic web. OWL describes classes, and the properties of those classes. Some properties are explicitly anointed to particular classes, while as other properties are not assigned to a specific class, and can be employed at the user’s discretion. Throughout this paper, “classes” and “properties” refers to OWL classes and properties. Furthermore, OWL allows for relations to exist between the properties. Again, “property relations” or “relations between properties” refer to OWL properties and the relations between them. "inverseOf” and “subPropertyOf” are examples of such relations. These are discussed extensively in the following sections.

The first task involved creating a database schema that would closely follow the structure of the university benchmark. Several techniques exist which can be used to model an OWL ontology in databases. The following section lists some of the more popular methods used.

2.2 Typical Techniques

There are several standard approaches that would have been possible, to some extent, to be applied in mapping the LUBM from OWL to a relational database structure. [Heflin and Pan, 04], [Beckett and Grant, 01], and [Melnik, 01] discuss several techniques and strategies to be employed for mapping, with varying results. Some of these techniques include:

2.2.1 Horizontal Database

This technique proposes using a single table to contain all the data. It is called horizontal because each property has its own column. A tuple is created for every class instance. This leads to a lot of empty spaces in tuples, since not every class uses all the properties. Furthermore, the size of such a table would render loading time very expensive.

2.2.2 Vertical Database

This technique proposes having a tuple for each property and/or class instance. One column would identify the class/property and another would contain the value. Once again, these resulting tables are too big, and too general. However, it is an improvement over the Horizontal design, with fewer columns, less empty space, and possibly less redundancy, especially if class instances contain multiple values for some specific property. Instead of reinserting the entire data for each value (Horizontal database), a new tuple, containing only that particular value, would be added.

2.2.3 Horizontal Class

Similar to the Horizontal database approach, this technique differs in that a table is created for each class type in the ontology. The properties that relate to the particular class make up the columns of the respective database table. Already this design is an improvement over the previous two since each table relates to a distinct class. Less loss of space and smaller tables naturally ensue. However, there remains the issue of properties that are not specified as belonging to any particular class. Including these in each and every table would be an expensive necessity. This approach formed the basis of the design ultimately employed in this project.

2.2.4 Table per Property

In this case, a table for each property is created, with class/value pairs in that table. This solves the issue of having lots of empty spaces due to unused columns. It would involve more joins, necessarily, due to the larger number of tables. Furthermore, retrieving all the properties of a single class instance now becomes a rather tedious process.

2.3 Implemented Technique

Such techniques as discussed above are usually employed to map OWL documents to relational databases. However, they are typically used as generic solutions.

In the case of this project work, automation was out of the question, because the university benchmark OWL ontology doesn't provide full and infallible data. Rather, a lot of the structure is inherently implied, or implemented in the generator. Furthermore, I was working with a specific ontology, and therefore was in possession of the specific classes and properties. That allowed me to design a specific database schema (or several) that represent the university benchmark exclusively. My hybrid design would naturally apply some of the general concepts mentioned in section 2.2, however, at the same time, it would be more specialized, and not so abstract, since it is representing a particular ontology. Hopefully, such a design would preserve, and provide, the logical relationships between the classes (class hierarchy) that came with that ontology.

The basis of this design was the 43 classes of the LUBM ontology. Each of these classes was represented in a table. So far, this design could be considered similar to the "horizontal class" approach. Since the design is based on data from the LUBM ontology as well as the TBox taxonomy of Racer's interpretation of that document (which was produced using Rice), more knowledge becomes available to me. The graph in Figure 1 provides an overview of the general structure, including some aspects of class hierarchy that were vague in the ontology file itself, or could not have been deduced, but which the reasoner realized.

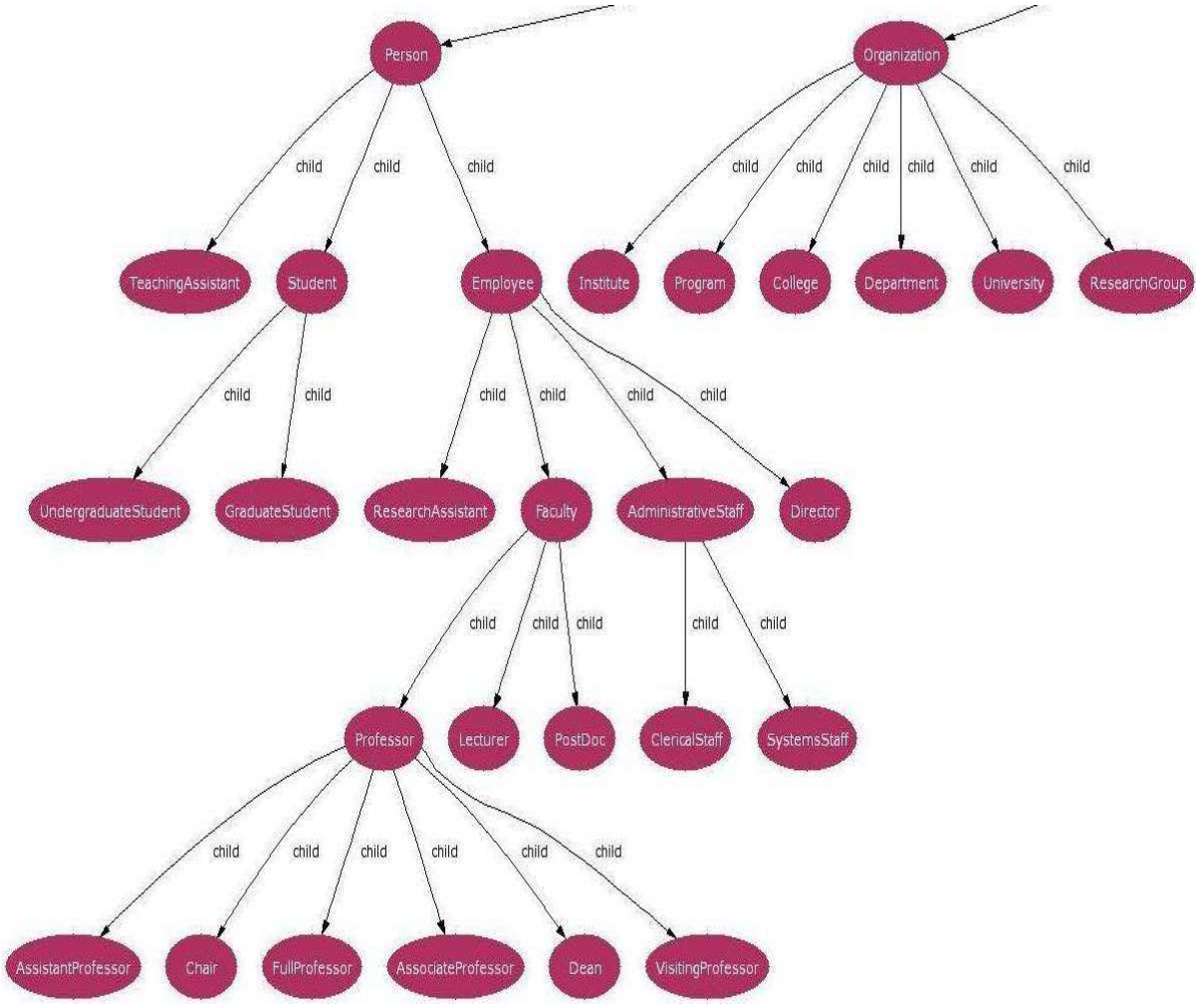

Figure 1: part of the TBox graph, hierarchies for "Person" and "Organization"

From the OWL ontology, I derived the 43 classes that would make up the 43 tables. That was one of the basic constraints on the database design: every class of the ontology would be represented by one and only one table. The tables in the database would be named after their respective classes in the ontology. Furthermore, those tables would have to contain the properties that were explicitly attached to them in the ontology. The reason behind this decision is quite simple. By limiting the design to 43 classes, the SQL queries would generally be similar to the LUBM queries in terms of scope and complexity. Furthermore, the syntax of these queries would not need much modification.

The TBox taxonomy provided the class hierarchy, parts of which were explicitly stated in the ontology file, while as the rest was implicit, and inferred by Racer. An example of such an inference is the 'child' relation between "GraduateStudent" and "Student". The ontology does not mention this relation, but it is implied from their structure. This provided an improvement over the design in the benchmark [Guo et al., 03], since class hierarchy would be represented in my database schema.

To implement this, I took advantage of each instance's "about" attribute, which was in the form of a unique URL. In the database schemas, this URL is to be found in the "globalName" column. Therefore, for example, every entry in table "Student" was also represented in table "Person" ("globalName", "name", "age",, with the two referencing each other using the "globalName" column. The same applies for instances of "University" and "Organization", "Course" and "Work", etc.

As mentioned previously, properties were attributed to classes according to explicit allocations in the owl ontology. This meant that any property listed as belonging to a class would definitely be found as a column in the respective table. However, some of these properties (property "memberOf" is an example) would have to be listed again in other tables. This leads to a varying case of redundancy which will be discussed presently.

As for properties that were not attributed to specific classes, I took the liberty of creating columns that would represent them in the relevant class (classes). This was based, by and far, on the data from the synthetic data generator.

This laid the ground rules for creating the database schema. However, the above constraints and specifications were not enough. Accordingly, I designed and used three different database schemas. All three were similar in essence, but had various small differences.

For brevity, the detailed structure of all the tables is not listed in this paper. Only tables that are used in examples are shown in figures, for clarification purposes. The CD-Rom accompanying this paper contains the entire SQL code required to create the following three schemas.

2.3.1 First Schema

This first schema (from hereon DB-1) was designed with the intention of having minimal redundancy. Only class hierarchy is inherent in this design, while as the relations between properties, which were mentioned above, are ignored. Referring to Fig.1 for an example again: in the case of wanting to create an undergraduate student, the process is iterative. First of all, the "globalname" is inserted into the table "UndergraduateStudent" (which only has that one column), then, information that is general to a student, such as the courses he takes, university he attends, and name of his advisor are inserted into "Student", with "globalname" as a unique identifier, and as a means to access it from "UndergraduateStudent". Then, a tuple is created in "Person", again with "globalname" and any information that is general to a member of table "Person". The process is similar for other classes, and is discussed in more details in the data generation section of this paper. Figure 2 shows the structure of the three tables discussed here.

GraduateStudent	Student	Person
globalName	globalName	globalName
	memberOf	name
	takesCourse	Age
	advisor	undegraduateDegreeFrom
		graduateDegreeFrom
		doctoralDegreeFrom
		officeNumber
		researchInterest
		telephone
		title
		emailAddress

Figure 2: Tables "GraduateStudent", "Student" and "Person" in DB-1

This way, there is minimal repetition of data, on the one hand, and a clear structure on the other. Granted, a user must still know about these relations to exploit them.

This design will produce the smallest database possible while adhering to the basic rules mentioned in the previous section. DB-1 is minimally redundant, but the design could still be improved. However, the main target remains to have as accurate a recreation of the knowledgebase taxonomy as possible, even if that means making a few design missteps.

2.3.2 Second Schema

The second schema (DB-2) builds upon the first. However, it adds information detailing the relations between the properties in the OWL ontology. DB-2 will represent the "inverseOf" relationship between properties "memberOf" and "member" as well as that between "degreefrom" and "has Alumnus". It will also represent the "subPropertyOf" relationship between "memberOf" (parent) and "worksFor" and "headOf" (sub-properties), as well as the one between "degreeFrom" and "undergraduateDegreeFrom", "graduateDegreeFrom" and "doctoralDegreeFrom".

Particularly, the "degreeFrom" property won't be represented by a column, nevertheless, this implementation allows its' sub-properties to also have an "inverseOf" relation with "hasAlumnus". This latter property is added to the table "University".

"MemberOf" is added to table "Person" and contains the data found in its' children. The children will also share an "inverseOf" relation with "member", which is added to table "Organization".

Person
globalName
name
Age
undegraduateDegreeFrom
graduateDegreeFrom
doctoralDegreeFrom
officeNumber
researchInterest
telephone
title
emailAddress
memberOf

Figure 3: Table "Person" in DB-2

As stated before, this design does not alter the general schema (by adding/removing tables). It merely adds the "inverseOf" properties, and sub-property relations, and allows us to generate data for a property based on its' inverse. A reasoner would deduce the inverse from the ontology, but in the case of a database, it has to be explicitly added. As for the sub-properties, their addition also allows for easier data access and manipulation by simulating a property hierarchy.

University	Organization
globalName	globalName
has Alumnus	name
	affiliatedOrganizationOf
	subOrganizationOf
	orgPublication
	Member

Figure 4: Tables "University" and "Organization" in DB-2

The downside to this design lies in the fact that several tuples become redundant. All the data that is added can be deduced with the proper knowledge of the schema, but specifying it explicitly simplifies the process, and keeps in line with having syntactically similar queries.

2.3.3 Third Schema

This schema (DB-3) again builds on the first and second. It is built in a way very similar to the horizontal class technique. It can be considered to have an incremental, expanding structure; however, it is fully redundant. This means that instead of simply adding new columns to a child, with a link to the parent, the child itself would contain all the data found in the parent, as well as any new data relevant to it. The benefits of such a

design would be that we no longer need to query several tables of the class hierarchy. For example, the “Person” table would have the same columns as in DB-2, however all the data found in table “Person” will now be also available in table “Student”, in addition to any extra tuples that relate particularly to “Student”. “UndergraduateStudent” for example will now contain the “takesCourse” column, so we no longer need to query the “Student” table for that information. This design also implements property hierarchy and relations (inverseOf, subPropertyOf). “UndergraduateStudent” and “Student” may seem identical, but the latter contains data of both graduate and undergraduate students, while as the former, as its’ name implies, only contains undergraduate students.

UndergraduateStudent	Student
globalName	globalName
memberOf	memberOf
takesCourse	takesCourse
advisor	advisor
name	name
Age	Age
undegraduateDegreeFrom	undegraduateDegreeFrom
graduateDegreeFrom	graduateDegreeFrom
doctoralDegreeFrom	doctoralDegreeFrom
officeNumber	officeNumber
researchInterest	researchInterest
telephone	telephone
title	title
emailAddress	emailAddress

Figure 5: Tables "UndergraduateStudent" and "Student" in DB-3

Although, from a design point of view, the redundancy in this schema is unacceptable, it should prove to be beneficial by limiting the number of joins (which is advantageous in large databases), and allowing queries to focus on specific tables. It is also interesting, from a theoretical point of view, to compare querying this database in particular to querying Racer, since the queries require less joins.

2.4 Issues

There are several points worth mentioning here. First of all, the fact that I was aware of the general shape and structure of both the schema, and the data that would populate it, allowed me to manipulate the design in a way that would simulate a little bit of the semantic knowledge that would be available to Racer. On the other hand, one could assume that all databases are designed with an eye on their use.

An example of these liberties would lie in the class “ResearchAssistant”. I was aware that all research assistants were graduate students (from the data generator). This allowed me to avoid duplicity, by not adding instances of “ResearchAssistant” to table

“Person”. Granted, the generator didn’t generate any additional data for it, but it would have been possible to just add the “globalname” to “Person”, after each insert on “ResearchAssistant”. This assumption made use of explicit knowledge pertaining to the structure of the automatically generated data.

Another point to be noted is that the databases were not optimized (by means of indexing, or otherwise). This meant that the answering time could only be improved by applying various tweaks and updates.

Also, due to the redundant nature of the generated data, it wasn’t possible to enforce referential integrity on some of the tables, however, the generator already ensures that no duplicate data would be generated so that is not a crucial problem.

Therefore, although the design is not absolutely generic, it fulfills the requirements of the TBox taxonomy, OWL ontology, and the generated data.

2.5 Room for Improvement

If it weren’t for the constraints set by trying to follow the TBox taxonomy as closely as possible, it would have been feasible to design a database that represents the ontology with no redundancy, and optimize it to provide results quickly. For example, the fact that each student may take more than one course leads to redundancy, because the same student information has to be introduced again for each extra course. It would have been simple to extract the “takesCourse” column to another table, and reference it from the “Student” table. However, that would have meant creating a table which wasn’t included in the original ontology, breaking the constraint of representing 43 classes in 43 tables.

Two other arguments against such a change should be noted. Firstly, the design would break from the structure of the original OWL files, and would force us to greatly modify the test queries, and secondly it would not really lead to improvement in terms of completeness. This will be discussed in detail later on.

On the other hand, by using the first, or second, schema in an object-relational database, the issue of redundancy due to certain columns - “takesCourse” in table “Student” of the previous example applies - could be avoided by using nested tables.

3. Test Data and Experiment Settings

The LUBM synthetic data generator is a tool developed by the Lehigh University to generate data for the university benchmark ontology. It can produce the data in OWL, which Racer can read. In order to carry out a successful comparison, it was essential to produce identical data for the databases and Racer. I modified the LUBM generator for that purpose.

3.1 Modifying the Generator and Creating Writers

The generator is a java program, consisting briefly of a Writer object and a Generator object. Upon instantiation, a specific type of Writer (producing a specific type of file) is chosen and registered with the Generator. The generator then proceeds with creating the data, and writing it to the registered Writer object.

The first part of the modification consisted of changing the generator so that it would produce the data four times instead of one. The generator typically sent the data to a specific writer object chosen by the user, in order to output the data in a particular format. In this case, however, each copy needed to be sent to four different writers.

Four writer instances exist: the first produces OWL files, while as the remaining three produce the SQL inserts for each of the three databases. These three new writers extended the abstract Writer class, and functioned similarly to the OWL writer. The major difference was that, in these writers, different methods were called based upon which instance was being generated.

For example, assume that the generator is creating an instance of a graduate student. The OWL writer would simply add the appropriate XML tags, and write it to the stream. In the case of the SQL writers, a method *createGradStud()* would be called. This method would write the appropriate insert statements (for either DB-1, DB-2 or DB-3), and then call *createStudent()*. The same process occurs until an insert is made into one of the main five tables representing the schema's five main categories. The inserts follow the class hierarchy already mentioned in the previous chapter.

Furthermore, inserts that fulfill requirements such as property relations and property hierarchies (DB-2, DB-3) are also added to the stream along the way.

Finally, the generator was also modified to immediately create the data in the four required formats. The user, therefore, no longer needs to specify the file types. The data intended for DB-1 has the file extension ".sql1", likewise, DB-2 data has ".sql2" and DB-3 data has ".sql3". A distinct file is produced for every set of individuals of one department of a university.

3.2 Loading the Data

Small java applications were used to load the ontology and the data onto the databases and Racer, and to time the loading process. Table 1 lists the loading times.

	Racer	DB1	DB2	DB3
Ontology	732	4105	4350	4712
Data	20625	14320	19182	41754

Table 1: Loading time (ms) for Racer and the databases

Two data sets were queried. The first data set constituted of one university containing 14 departments. The second data set had only 2 departments of the same university. Table 2 lists the loading times for the first data set. The loading time for the data was distributed evenly among the departments, so a single department took around 1.5 seconds to load into Racer, 1 second to load into DB-1, 1.4 seconds for DB-2 and almost 3 seconds for DB-3. The difference in load time for the three databases is attributed to the increased number of inserts.

The first data set was made up of 87020 class instances and 353720 property instances. The second set had 13188 class instances and 54856 property instances. It should also be noted that due to redundancy, the data that populated the databases was significantly larger than the one that populated Racer.

3.3 Experimental Setup

The experiment was run on a Pentium M, with 1.4 GHz, 512 MB of Ram and 20 GB of free space. The system ran under a Windows XP professional edition operating system.

Each query was run three times (once for each mode) in racer, and once for each of the three database schemas. The queries were also run for each of the two datasets.

Two distinct settings were used to run the queries. In the first setting, each query was run separately so as to prevent caching and other optimization techniques from influencing the results. The database, or racer, was restarted before running each query. By closing the connection to MySQL between queries, the database makes no use of cache. Furthermore, by restarting Racer, each query would have to prepare the entire index structure before running.

The second setting has two aspects. First of all, the queries are all run sequentially, and without restarting the systems. This allows both the database and Racer queries to take advantage of caching in some instances. Furthermore, in Racer, a query is run, before running the benchmark queries, to allow Racer to prepare the index structure, and check for ABox coherence. This rids the benchmark queries of any overhead, and

allows us to calculate the pure answering time required for each query. The following table lists the time needed to run this preparation query, and therefore, the time needed for racer to prepare the index structure.

	Racer Mode 0	Racer Mode 1	Racer Mode 3
2 Departments	6042	7010	223400
14 Departments	76850	71300	

Table 2: Preparation time for Racer

The query answering time was calculated by starting a counter right before sending the request (query) and stopping it right after the query resolved.

3.4 Racer Querying Modes

Racer offers various modes, with varying degrees of completeness, for query answering. For evaluating the queries in Racer, three query modes were chosen, modes 0, 1 and 3. The RacerPro User's Guide [Racer Systems, 05] defines these three modes as follows:

3.4.1 Mode 0

Told information querying: "In this setting, to answer a nRQL query, only the syntactic, told information from and ABox will be used, which is given by the syntactic format of the ABox axioms." This means that no information that is not explicitly stated in the ABox will be retrieved. However, transitive roles and inverse-of roles are resolved, and this fact will play a factor in the queries below.

3.4.2 Mode 1

Told information querying plus exploited TBox information for concept names: "This is like the previous setting, but now the TBox information is taken into account for concept membership assertions of the form *(instance i C)*, where *C* is an atomic concept / concept name. If C is a concept name, nRQL will use the classified TBox to compute the set of *atomic-concept-synonyms*, as well as the set of *atomic-concept-ancestors*, and put all these implied concept name membership assertions into the ABox as well."

3.4.3 Mode 3

Complete ABox querying. This querying mode returns complete results. However, it is much more expensive, time-wise than the previous two. Furthermore, there are limits to the amount of data that can be queried in this mode.

4. The Queries

Fourteen queries were used in this experiment. These were the benchmark queries from the LUBM [LehighQueries, 04], and are used predominantly when working with the university benchmark. The queries test the data repositories in aspects of scope and selectivity.

I translated the queries from their original (KIF-like) structure to nRQL and SQL, with slight modifications to account for small structural differences. Care was taken not to change the core of the queries. In that sense, the properties chosen remain the same, and the query complexity is generally unchanged. In the database a query was allowed to span several tables, and perform multiple joins, as long as it didn't use knowledge of properties that wasn't specifically stated in the original queries.

Each query was run on all three databases, as well as on Racer, operating in the three querying modes. Due to its' size, however, I was not able to run mode 3 for the larger repository.

Listed below are all 14 queries. The queries are first defined (definitions from [Guo et al., 03]), and then presented in the format in which they were executed. Two tables follow, one for the results of querying 14 departments, and the second for querying 2 departments. Some remarks conclude each section.

Two different answering times are listed per query. The first, "Full Time" is the result of running each query by itself, without any optimization. The second, "Opt. Time" is the result of running the queries after preparing the index structure (in the case of Racer), and taking advantage of caching to improve response time. It should be noted that the following query: *(retrieve (?x) (?x top))* , was run in each of the three Racer modes before running the actual queries, in order to prepare the index structure.

Note: For nRQL queries, the concept or role names were shortened for brevity. The actual name should be of the form: "|http://www.lehigh.edu/%7Ezhp2/2004/0401 /univ-bench.owl#*(name here)*|" . In the queries below, the name was used without the preceding URL.

4.1 Query 1

All the graduate students who take GraduateCourse0 at Department0 of University0

nRQL query:

(retrieve (?x) (and (?x GraduateStudent) (?x GraduateCourse0 takesCourse)))

DB-1 query:

select globalname from GraduateStudent where globalname in
(select globalname from student where takescourse=
"http://www.Department0.University0.edu/GraduateCourse0")

DB-2 query: (same as DB-1)
DB-3 query:

select globalname from GraduateStudent where takescourse=
"http://www.Department0.University0.edu/GraduateCourse0"

Expected: 2	14 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	2	100	45866	161
Racer Mode 1	2	100	52926	400
Racer Mode 3				
DB1	2	100	68208	66996
DB2	2	100	71323	66666
DB3	2	100	440	461

Table 3: Query 1 results for 14 departments

Expected: 2	2 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	2	100	5408	10
Racer Mode 1	2	100	7291	10
Racer Mode 3	2	100	68949	40
DB1	2	100	1612	1583
DB2	2	100	1641	2204
DB3	2	100	90	150

Table 4: Query 1 results for 2 departments

In DB-1 and DB-2, a join is needed with table "Student" since that is where the "takesCourse" column is located. Due to that, the queries take much longer to resolve than DB-3, and also more time than Racer in modes 0 and 1, in the larger repository. In the smaller repository, the database returns the results fairly quickly, and faster than Racer. The optimized time takes away most of the time cost for the racer queries. This is the same for all the ensuing queries, and is due to the fact that the index structure, checking for ABox coherence, etc, have all been already carried out.

4.2 Query 2

All the graduate students who are now studying at the university from which they obtained their bachelor's degrees

nRQL query:

```
(retrieve (?x) (and (?x GraduateStudent)
                    (?y University)
                    (?z Department)
                    (?x ?z memberOf)
                    (?z ?y subOrganizationOf)
                    (?x ?y undergraduateDegreeFrom)))
```

DB-1 query:

```
select t.globalname from student t where memberof in
    (select globalname from organization where suborganizationof in
    (select undergraduatedegreefrom from person where globalname in
    (select globalname from graduatestudent where
    globalname=t.globalname)))
```

DB-2 query:

```
select globalname from person where (globalname in
    (select globalname from graduatestudent)
    and undergraduatedegreefrom in
    (select suborganizationof from organization where
    globalname=memberof))
```

DB-3 query:

```
select globalname from graduatestudent where undergraduatedegreefrom in
(select suborganizationof from department where globalname=memberof)
```

Expected: 0	14 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	0	100	48711	663
Racer Mode 1	0	100	62537	962
Racer Mode 3				
DB1				
DB2	0	100	29742	31165
DB3	0	100	54682	54759

Table 5: Query 2 results for 14 departments

Expected: 0	2 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	0	100	5698	90
Racer Mode 1	0	100	5498	321
Racer Mode 3	0	100	126622	13179
DB1	0	100	48630	56422
DB2	0	100	761	1151
DB3	0	100	1422	1612

Table 6: Query 2 results for 2 departments

DB-1 fails to return a result in the larger repository that is due to the large amounts of data involved in the joins. The triangular pattern of this query makes it particularly costly. In the second, modes 0 and 1 return results much faster than DB-1 as well. DB-2 and DB-3 return the results fairly quickly in the second, but not so quickly in the first. This is the first case where queries could have been simplified by using other approaches to achieve the same result. Again, a dramatic decrease in answering time results from preparing Racer before running the queries.

4.3 Query 3

All the publications of AssistantProfessor0 at Department0 of University 0

nRQL query:

(retrieve (?x) (and (?x Publication) (?x AssistantProfessor0 publicationAuthor)))

DB-1 query:

select globalname from publication where publicationauthor=
"http://www.Department0.University0.edu/AssistantProfessor0"

DB-2 query: (same as DB-1)
DB-3 query: (same as DB-1 and DB-2)

Expected: 6	14 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	6	100	34370	20
Racer Mode 1	6	100	44093	30
Racer Mode 3				
DB1	6	100	360	1202
DB2	6	100	431	350
DB3	6	100	460	671

Table 7: Query 3 results for 14 departments

Expected: 6	2 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	6	100	6659	421
Racer Mode 1	6	100	4546	20
Racer Mode 3	6	100	53026	90
DB1	6	100	80	460
DB2	6	100	100	70
DB3	6	100	110	110

Table 8: Query 3 results for 2 departments

We observe for the first time a great difference in response time between Racer and the databases; this is simply due to the fact that we are only querying one table. This is a direct result of designing the schemas with class hierarchy in mind. Racer would, on the other hand, check each subclass of "Publication", taking more time to return a result. This time difference is, as always, almost completely erased in the optimized time. Note that there isn't a very significant change in response time for the database queries with and without caches.

4.4 Query 4

All the professors at Department0 of University0 and their email addresses and telephone numbers

nRQL query:

```
(retrieve ((datatype-fillers (name ?x))
           (datatype-fillers (emailAddress ?x))
           (datatype-fillers (telephone ?x)))
           (and (?x Professor)
           (?x http://www.Department0.University0.edu worksFor)))
```

DB-1 query:

select name, emailaddress, telephone from person where globalname in
(select globalname from employee where globalname in
(select globalname from professor) and
worksfor="http://www.Department0.University0.edu")

DB-2 query:

select name, emailaddress, telephone from person where globalname in
(select globalname from professor) and
memberof="http://www.Department0.University0.edu"

DB-3 query:

select distinct(name), emailaddress, telephone from professor where
worksfor="http://www.Department0.University0.edu"

Expected: 34	14 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	0	0	30644	20
Racer Mode 1	34	100	43332	581
Racer Mode 3				
DB1	34	100	60417	61709
DB2	34	100	811	170
DB3	34	100	360	290

Table 9: Query 4 results for 14 departments

Expected: 34	2 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	0	0	3125	10
Racer Mode 1	34	100	3005	551
Racer Mode 3	34	100	59095	731
DB1	34	100	1973	1833
DB2	34	100	340	80
DB3	34	100	70	90

Table 10: Query 4 results for 2 departments

Racer fails in mode 0, because the search requires results from subclasses of "Professor". The implementation of sub-properties in DB-2 renders it significantly faster than DB-1 due to lesser number of joins required. DB-3 needs no joins whatsoever, and returns results very quickly. However, care must be taken to select distinct results needed

in db3 due to the redundancy factor. Datatype fillers are an advantage in nRQL, since that same data requires an extra join in DB-1 and DB-2.

4.5 Query 5

All the members of Department0 at University0

nRQL query:

(retrieve (?x)(and (?x Person)
(?x http://www.Department0.University0.edu memberOf)))

DB-1 query: (no query)
DB-2 query:

select globalname from person where
memberof="http://www.Department0.University0.edu"

DB-3 query: (same as DB-2)

Expected: 719	14 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	0	0	21551	10
Racer Mode 1	719	100	104100	621
Racer Mode 3				
DB1				
DB2	719	100	561	20
DB3	719	100	911	611

Table 11: Query 5 results for 14 departments

Expected: 719	2 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	0	0	2524	40
Racer Mode 1	719	100	42451	673
Racer Mode 3	719	100	325618	2847
DB1				
DB2	719	100	271	10
DB3	719	100	332	231

Table 12: Query 5 results for 2 departments

The first syntactic weakness in DB-1 is exposed in this query. Since “memberOf” is in fact the union of two sub-properties: “memberOf” in “Student” and “worksFor” in

"Employee", we are not able to query directly for it. On the other hand, DB-2 and DB-3 resolve much faster than racer since no reasoning is required, rather, merely a select statement on one table.

4.6 Query 6

All the students

nRQL query:

(retrieve (?x) (?x Student))

DB-1 query:

select distinct(globalname) from student

DB-2 query: (same as DB-1)
DB-3 query: (same as DB-1 and DB-2)

Expected: 8492	14 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	0	0	43823	20
Racer Mode 1	8492	100	56421	2732
Racer Mode 3				
DB1	8492	100	691	240
DB2	8492	100	631	221
DB3	8492	100	821	1062

Table 13: Query 6 results for 14 departments

Expected: 1351	2 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	0	0	2574	60
Racer Mode 1	1351	100	266603	721
Racer Mode 3	1351	100	631818	962
DB1	1351	100	270	80
DB2	1351	100	360	40
DB3	1351	100	390	350

Table 14: Query 6 results for 2 departments

While mode 0 doesn't return (no reasoning), modes 1 and 2 take a relatively long time. Class hierarchy is inherent in the database schemas, and as such, all students are found in table "student". The queries resolve quickly. Again, there is a need to request

distinct results since there are multiple entries/student, due to the fact that each student takes several courses.

4.7 Query 7

All the students who take the courses of AssociateProfessor0 at Department0 of University0

nRQL query:

```
(retrieve (?x) (and (?x Student)
        (?y Course)
        (http://www.Department0.University0.edu/AssociateProfessor0 ?y
        teacherOf)
       (?x ?y takesCourse)))
```

DB-1 query:

```
select distinct(globalname) from student where takescourse in
        (select teacherof from faculty where globalname=
        "http://www.Department0.University0.edu/AssociateProfessor0")
```

DB-2 query: (same as DB-1)
DB-3 query: (same as DB-2)

Expected: 41	14 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	0	0	16425	510
Racer Mode 1	41	100	13832	40
Racer Mode 3				
DB1	41	100	51103	51394
DB2	41	100	51815	51073
DB3	41	100	52476	51464

Table 15: Query 7 results for 14 departments

Expected: 41	2 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	0	0	2354	10
Racer Mode 1	41	100	2544	50
Racer Mode 3	41	100	281285	331
DB1	41	100	1222	1282
DB2	41	100	1251	1292
DB3	41	100	1221	1512

Table 16: Query 7 results for 2 departments

Racer modes 0 and 1 resolve much faster than the database in the first repository. In the second, with less data, the databases resolve faster, although modes 0 and 1 return in comparable times.

4.8 Query 8

All the students of University0 and their email addresses

nRQL query:

```
(retrieve (?x (datatype-fillers (emailAddress ?x)))
          (and (?x Student)
          (?y Department)
          (?x ?y memberOf)
          (?y http://www.University0.edu subOrganizationOf)))
```

DB-1 query:

```
select globalname, emailaddress from person where globalname in
       (select globalname from student where memberof in
       (select globalname from department where globalname in
       (select globalname from organization where
       suborganizationof="http://www.University0.edu")))
```

DB-2 query:

```
select globalname,emailaddress from person where globalname in
       (select globalname from student) and memberof in
       (select globalname from department where globalname in
       (select globalname from organization where
       suborganizationof="http://www.University0.edu"))
```

DB-3 query:

select distinct(globalname), emailaddress from student where memberof in
(select globalname from department where
suborganizationof="http://www.University0.edu")

Expected: 8492	14 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	0	0	57650	120
Racer Mode 1	8492	100	84120	11247
Racer Mode 3				
DB1	8492	100	755236	754665
DB2	8492	100	194430	189342
DB3	8492	100	128675	127513

Table 17: Query 8 results for 14 departments

Expected: 1351	2 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	0	0	1733	50
Racer Mode 1	1351	100	9572	2068
Racer Mode 3	1351	100	87790	1600
DB1	1351	100	19268	19528
DB2	1351	100	4266	4266
DB3	1351	100	1853	1803

Table 18: Query 8 results for 2 departments

This query spans two of the largest classes, and therefore takes a lot of time in DB-1. No results are returned in mode 0, naturally, however, mode 1 returns much faster than the database in the case of the large repository. One interesting note is that the database queries resolve because we explicitly state that the departments have "suborganizationOf" relation with University0, otherwise, only Racer mode 1 and 3 would have returned results.

4.9 Query 9

All the students who take the courses of their advisors

nRQL query:

(retrieve (?x)(and (?x Student)
(?y Faculty)
(?z Course)
(?x ?y advisor)

(?x ?z takesCourse)
(?y ?z teacherOf)))

DB-1 query:

select distinct(globalname) from student where takescourse in
(select teacherof from faculty where globalname=advisor)

DB-2 query: (same as DB-1)
DB-3 query: (same as DB-1 and DB-2)

Expected: 213	14 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	0	0	59463	510
Racer Mode 1	213	100	62551	1722
Racer Mode 3				
DB1	213	100	57783	57483
DB2	213	100	58074	57533
DB3	213	100	59375	57793

Table 19: Query 9 results for 14 departments

Expected: 25	2 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	0	0	4947	631
Racer Mode 1	25	100	6720	120
Racer Mode 3	25	100	92664	711
DB1	25	100	1603	1372
DB2	25	100	1452	1372
DB3	25	100	1612	1382

Table 20: Query 9 results for 2 departments

The databases all take similar times to return results. Racer queries take longer however, due to the reasoning involved with "student" and its' subclasses.

4.10 Query 10

All the students who take GraduateCourse0 at Department0 of University0

nRQL query:

(retrieve (?x) (and (?x Student)
(?x http://www.Department0.University0.edu/GraduateCourse0
takesCourse)))

DB-1 query:

select globalname from student where
takescourse="http://www.Department0.University0.edu/GraduateCourse0"

DB-2 query: (same as DB-1)
DB-3 query: (same as DB-1 and DB-2)

Expected: 2	14 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	0	0	39905	10
Racer Mode 1	2	100	38401	10
Racer Mode 3				
DB1	2	100	371	30
DB2	2	100	711	30
DB3	2	100	290	40

Table 21: Query 10 results for 14 departments

Expected: 2	2 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	0	0	3244	90
Racer Mode 1	2	100	4216	10
Racer Mode 3	2	100	69800	20
DB1	2	100	140	10
DB2	2	100	170	10
DB3	2	100	130	5

Table 22: Query 10 results for 2 departments

There is a significant difference in querying time between Racer and the databases. It is interesting to compare the results to those of query 1. The differences can be attributed purely to reasoning about the sub-classes, which is implicit in the databases, but necessary in Racer (mode 0 fails because of that).

4.11 Query 11

All the research groups at University0

nRQL query:

```
(retrieve (?x) (and (?x ResearchGroup)
                    (?x http://www.University0.edu subOrganizationOf)))
```

DB-1 query:

```
select globalname from researchGroup where globalname in
    (select globalname from organization where
    suborganizationof="http://www.University0.edu")
```

DB-2 query: (same as DB-1)
DB-3 query: (same as DB-1 and DB-2)

Expected: 217	14 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	217	100	42050	802
Racer Mode 1	217	100	65225	771
Racer Mode 3				
DB1	0	0	111	440
DB2	0	0	3114	3165
DB3	0	0	3204	3455

Table 23: Query 11 results for 14 departments

Expected: 28	2 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	28	100	4637	40
Racer Mode 1	28	100	4496	20
Racer Mode 3	28	100	102387	30
DB1	0	0	30	40
DB2	0	0	130	101
DB3	0	0	60	220

Table 24: Query 11 results for 2 departments

This query plays to racer's advantage. Even mode 0 returns a result, since transitive roles do resolve even at that degree of completeness. The transitive relation between organizations is not implemented explicitly; therefore the databases can not return a result.

4.12 Query 12

All the department chairs of University0

nRQL query:

```
(retrieve (?x) (and (?x Chair)
                    (?y Department)
                    (?x ?y memberOf)
                    (?y http://www.University0.edu subOrganizationOf)))
```

DB-1 query:

```
select globalname from chair where globalname in
      (select globalname from employee where worksfor in
      (select globalname from organization where
      suborganizationof="http://www.University0.edu"))
```

DB-2 query: (same as DB-1)
DB-3 query:

```
select globalname from chair where worksfor in
      (select globalname from organization where
      suborganizationof="http://www.University0.edu")
```

Expected: 15	14 Departments			
	Answers	Completeness	Time	Time
Racer Mode 0	0	0	47537	390
Racer Mode 1	0	0	41236	391
Racer Mode 3				
DB1	0	0	40	30
DB2	0	0	30	90
DB3	0	0	30	30

Table 25: Query 12 results for 14 departments

Expected: 2	2 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	0	0	7701	80
Racer Mode 1	0	0	6890	70
Racer Mode 3	2	100	90540	40819
DB1	0	0	30	40
DB2	0	0	91	60
DB3	0	0	20	50

Table 26: Query 12 results for 2 departments

Only mode 3, with complete querying returns a result. Table “Chair”, as noted in the benchmark, contains no data. Mode 3 therefore refers to the TBox to realize this query.

4.13 Query 13

All the alumni of University0

nRQL query:

(retrieve (?x) (and (?x Person)
(http://www.University0.edu ?x hasAlumnus)))

DB-1 query: (no query)
DB-2 query:

select globalname from person where globalname in
(select hasalumnus from university where
globalname="http://www.University0.edu")

DB-3 query: (same as DB-2)

Expected: 2	14 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	0	0	43850	10
Racer Mode 1	2	100	45962	40
Racer Mode 3				
DB1				
DB2	2	100	45616	44914
DB3	2	100	47243	44594

Table 27: Query 13 results for 14 departments

Expected: 0	2 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	0	0	3184	50
Racer Mode 1	0	100	3075	10
Racer Mode 3	0	100	87816	170
DB1				
DB2	0	100	1151	1101
DB3	0	100	1081	1082

Table 28: Query 13 results for 2 departments

DB-2 and DB-3 implement the "hasalumnus" property in their schemas, therefore, they do return results. No appropriate query can be used for DB-1, without including properties that weren't mentioned in the original query definition. Note that, in the second case, even though there are no results, mode 0 has 0% completeness, this is due to the fact that mode 0 fails in this query (as shown in querying the first repository). Even though mode 0 does resolve inverse-of roles, the problem lies in the fact that it doesn't resolve that a "FullProfessor" is a "Person". Query times are also comparable (except for mode 3). The reasoning entailed in the racer queries is offset by the joins entailed in the database queries.

4.4 Query 14

All the undergraduate students of University0

nRQL query:

(retrieve (?x) (?x UndergraduateStudent))

DB-1 query:

select globalname from undergraduatestudent

DB-2 query: (same as DB-1)
DB-3 query:

select distinct(globalname) from undergraduatestudent

Expected: 6487	14 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	6487	100	46394	1620
Racer Mode 1	6487	100	58992	2340
Racer Mode 3				
DB1	6487	100	260	231
DB2	6487	100	291	681
DB3	6487	100	301	1292

Table 29: Query 14 results for 14 departments

Expected: 1048	2 Departments			
	Answers	Completeness	Full Time	Opt. Time
Racer Mode 0	1048	100	5720	112
Racer Mode 1	1048	100	3347	783
Racer Mode 3	1048	100	73231	703
DB1	1048	100	140	70
DB2	1048	100	160	60
DB3	1048	100	153	140

Table 30: Query 14 results for 2 departments

This is a straightforward query. The time, however, is worth noting. Since we are only querying a single table for a single tuple, the database queries all resolve very quickly. Extra reasoning leads to a delay in the case of racer.

5. Conclusion

Some interesting facts can be discerned from the queries above. In concluding this project work, I will divide the conclusion into the two main categories that marked the benchmarking: speed and completeness, and discuss accordingly.

5.1 Answering Time

Speed was sacrificed in many aspects of this benchmarking. First and foremost, the databases could have been optimized in several ways. I mentioned before that the design followed some strict rules, in order to stay in line with the LUBM ontology, with a few alterations, those schemas could have been rendered much more efficient. Also, indexing, increasing cache size, and other optimization techniques could have been used. However, in creating the schemas, a major goal was to have them simulate the ontology, and keep them on equal footing with Racer, so as to get the fairest and most balanced results.

A quick look at the queries showed that the database ones are generally faster than Racer (in nRQL modes 0, 1 or 3). Some special cases resulted in long query times. Again, with a change of the database schema, those queries could be allowed to run much faster. It's all a matter of design. As for Racer, by preparing the index structure beforehand, the queries also resolve at very high speeds and become comparable to (and even faster than) the database (index preparation time notwithstanding). Allowing caches in databases had some effect, in some queries, since the cache would only speed up certain queries, and wouldn't have the overall effect that preparing the index structure in Racer has on those queries.

Also noticeable, is the difference between the three racer modes themselves, and the three database modes themselves. Generally, racer mode 0 had the fastest results (if not always complete), followed by mode 1, then mode 3. This was to be expected. With respect to the databases, the speed varied depending on the queries. Being directly proportional to number and size of joins, DB-1 suffered when faced with long queries (that would be solved in DB-2 or DB-3 using a simple select). One should note that DB-2 and DB-3 take longer to load initially, since they contain more insert statements, and that the databases are redundant, and significantly larger than DB-1. The speed difference, however, seems to make up for that.

Finally, speed depends by and large on the processing capabilities of machines, and therefore, can only improve with time.

5.2 Completeness

Speed can always be improved. Completeness, on the other hand, is paramount. The databases returned complete results, when they actually could return a result. Otherwise, it was complete failure. This is natural, since databases are not reasoners, they can only deduce what has already been input to them. This is where Racer showed its fangs. In dealing with transitive relations, the databases couldn't return any result. On the other hand, even mode 0 was able to return complete results for that query. Again, in Query 12, the databases failed completely, in this case, however, only racer mode 3 succeeded in getting a result. Although expensive, time wise, mode 3 nonetheless proves its worth by solving the queries that others can't.

One point to be mentioned is that we can get the queries to do anything we want, with enough knowledge. Reasoning isn't making up new data, it's simply realizing new information from the coupling of the existent data and the rules the data follows. For racer, even in mode 0, one could easily use nRQL rules to syntactically augment the ABox and get the desired results.

In case of Query 12, one could also immediately search for the property "headOf" in "Professor", and get full, complete results (in mode 1, at least). Likewise in the databases, in Query 11, we could have pointed out, in the query that the "ResearchGroups" are sub-organizations of "Department" which itself is a sub-organization of "University". It is again important to note that Racer (in all modes) recognizes that relation, where as a database needs explicit informing.

A major advantage for Racer then, is that the user needs only to know the classes and properties involved. For the same query on a database, more knowledge is needed, such as tables where the specific property is found. That makes Racer queries much more natural.

The bottom line is, a database schema can be designed any which way the user wants. It could be geared towards solving particularly tough queries using some explicit setup. Triggers could be used, stored procedures too. After all, it is only fair to assume that the designer would have an inkling of how the system he's designing is going to be used. It would be reasonable, however, to assume that a certain level of abstractness would be preferable. This is where Racer comes in; only a high level view of the ontology is needed. So, instead of knowing that "takesCourse" is specific to "Student", in Racer, the user could simply ask for any person taking a particular course.

To put it in terms of specific databases and modes, DB-1 allows for simple traversing of the class hierarchy downwards, to traverse it upwards, more knowledge is needed. This is an advantage over Racer in mode 0, but is available in modes 1 and 3. DB-2 attempts to simulate the "inverseOf" and "subPropertyOf" relations. Again, this is available in all three Racer modes. However, Racer has the advantage of recognizing transitive relations, the equivalent of which cannot be simply simulated in a database. DB-3 serves only to reduce the query complexity, at the added expense of redundancy

and larger database size. Therefore, it is clear that a simple database structure cannot compete with Racer, and each specific relation needs to be implemented additionally.

5.3 Suggestions

In summation, this benchmark showed that database querying is still more practical in simple cases, and somewhat more efficient. However, a lot of limitations lay in the database path, due to the high dependence on explicit knowledge. Knowledgebase reasoners, and Racer in particular, excelled in solving those queries that could not be solved by any of the others databases.

I would suggest using a database in most general cases. Of the three database schemas designed, I believe that the second strikes the balance between size, speed, and query time. With a few design tweaks (separate table for “takesCourse”, “member”, etc), it can solve a multitude of queries. For those that don’t resolve, using Racer in mode 3 returns complete results.

An argument could also be made for using a knowledgebase reasoner of the description logic family all the time, since it allows a higher level of querying and relies less on the need to know all the minute details of a schema.

So, it seems, as always, that one must tread lightly in the middle of the road, and keep an open eye on both sides.

References

- [Racer, 05] Ralf Moeller Home page. http://www.sts.tu-harburg.de/~r.f.moeller/racer/

- [Baader et al.] Franz Baader, et al. "The Description Logic Handbook."

- [W3C, 05] World Wide Web Consortium. Web Ontology Language (OWL). Feb. 2005 http://www.w3.org/2004/OWL/

- [Harslev et al. 04] Volker Haarslev, Ralf Moeller, and Michael Wessel. "RACER User's Guide and Reference Manual Version 1.7.19." April 26, 2004 http://www.sts.tu-harburg.de/~r.f.moeller/racer/racer-manual-1-7-19.pdf

- [Rice, 04] Ronal Cornet Home page. Racer Interactive Client Environment. 2004 http://www.b1g-systems.com/ronald/rice/

- [UBA, 04] The Lehigh University Benchmark data generator (v 1.6). Dept. of Computer Science and Engineering, Lehigh University. 2004 http://swat.cse.lehigh.edu/projects/lubm/uba1.6.zip

- [Lehigh] The Lehigh University Benchmark (LUBM). Dept. of Computer Science and Engineering, Lehigh University. http://swat.cse.lehigh.edu/projects/lubm/index.htm

- [MySQL, 05] MySQL Home page. 2005 http://www.mysql.com/

- [Heflin and Pan, 04] J. Heflin and Z. Pan. "DLDB: Extending Relational Databases to Support Semantic Web Queries." Technical Report LU-CSE-04-006, Dept. of Computer Science and Engineering, Lehigh University. 2004 http://www3.lehigh.edu/images/userImages/jgs2/Page_3813/LU-CSE-04-006.pdf

- [Beckett and Grant, 01] D. Beckett and J. Grant, "Mapping Semantic Web Data with RDBMSes." 2001 http://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report/

- [Melnik, 01] S. Melnik, Storing RDF in a relation database. Online posting. Dec. 2001, http://www-db.stanford.edu/~melnik/rdf/db.html

- [Guo et al., 03] Y. Guo, J. Heflin, and Z. Pan. "Benchmarking DAML+OIL Repositories." Second International Semantic Web Conference, ISWC 2003, LNCS 2870. Springer (c), 2003, pp. 613-627. http://www.cse.lehigh.edu/~heflin/pubs/iswc2003.pdf

- [Racer Systems, 05] Racer Systems GmbH & Co. KG. Home page. "RacerPro User's Guide Version 1.8." April 10,2005 http://www.racer-systems.com

- [UnivBench, 04] The Lehigh University Benchmark OWL ontology. Dept. of Computer Science and Engineering, Lehigh University. 2004 http://www.lehigh.edu/%7Ezhp2/2004/0401/univ-bench.owl