Querying Relational Databases with RDQL

Cristian Pérez de Laborda and Stefan Conrad
Institute of Computer Science
Heinrich-Heine-Universität Düsseldorf
D-40225 Düsseldorf, Germany
{perezdel, conrad}@cs.uni-duesseldorf.de

Abstract:
Most Semantic Web applications are still unable to query data stored in relational databases using their own built-in functionality. Hence, needing access to such data, they have to fall back on SQL and the relational model. In this paper we describe Relational.OWL, our technique to automatically extract the semantics of relational databases and transform this information into a machine processable and understandable representation. Since this data can now be queried using semantic query languages, it has to be analyzed whether this combination of Relational.OWL and e.g. RDQL can be a real alternative to the commonly used SQL access to relational data.

1 Introduction

Despite all the efforts to build up a Semantic Web, where each machine can understand and interpret the data it processes, information is usually still stored in ordinary relational databases. Both, the semantic and the relational worlds coexisted for a long time with little effort in merging them. The result were large amounts of information stored in relational databases and inaccessible for any application of the Semantic Web, unless it used SQL. As a consequence, such applications needed to create a corresponding mapping between the relational and the semantic data by themselves. Realizing this situation, some efforts have arisen to straighten out this deplorable situation.

In [PC05] we have presented Relational.OWL, a technique to automatically extract the semantics of relational databases and transform this information into RDF/OWL, processable by a wide majority of Semantic Web applications. The advantage of such an approach is obvious, since it makes relational data processable for Semantic Web applications using their built-in functionality like query languages or reasoning mechanisms. In fact, no Semantic Web application needs to implement its own relational to semantic mapping any more.

In this paper we analyze whether the combination of Relational.OWL and an exemplary semantic query language (RDQL) can be an alternative to an ordinary query with SQL. In other words, we want to check out if combining Relational.OWL and such a query language can lead us to the same results as a normal relational query. Since SQL was extended repeatedly in its expressiveness during the last decades, a direct comparison of RDQL and SQL would be unfair. We thus limit our analysis to the question, whether the combination of Relational.OWL and RDQL is capable to provide the same results like the relational algebra.

The remainder of this paper is organized as follows: In the following section we discuss some related work, followed by a short introduction to Relational.OWL in section 3. The comparison between the operations of the relational algebra and RDQL is made in section 4. Section 5 finishes with a small discussion and some ideas for future work.

2 Related Work

Recently, some efforts arose in bringing together relational databases and the Semantic Web. For instance, An et al. outline an automatic mapping between relational tables and ontologies in [ABM04]. Unlike our approach, it requires a target ontology onto which the relational tables are mapped to. Since a target ontology may not always exist or the agent assigned with the mapping task may not know of its existence, such a mapping may often be impossible. However, having found such a mapping between the relational database and the target ontology, the authors do not mention how they deal with schema evolution in the original database and whether the generated data looses its connection to the originating database. Our approach takes schema and data evolution directly into account and gives the possibility to link the generated data and schema representation to its original source.

Bizer and Seaborne introduced in [BS04] a query mechanism for their mapping relational databases to ontologies approach. Contrary to our technique, it converts legacy data stored in databases into "real" RDF objects, i.e. an address would be represented as a RDF address object. Since the corresponding mappings between the database and RDF have to be created manually, they must be rechecked after each evolution of the schema. Especially in environments with constantly changing schemas, this is not manageable any more.

Haase et al. provide in [HBEV04] a survey describing different RDF query languages, including RDQL on which we want to focus in this paper. The paper starts with the promising challenge to examine, whether the query languages are relational complete or not. Since the paper examines six different query languages it analyzes the languages superficially, omitting the argumentation for the conclusions. In fact, the authors seem to have underestimated the expressiveness of RDQL.

Karvounarakis et al. present in [KCPA01] RQL, an advanced declarative query language for RDF schemas and descriptions. To achieve the best possible query results, the authors store the RDF data in a relational database, where the actual query is performed. Therefore, they make an adequate mapping of all RDF triples to the relational data model. Since this mapping is specific to the needs of querying *normal* RDF triples, stored in a special way in a relational databases, it is hard to apply it to our needs. In fact, we would have to transform data stored in relational databases to RDF/OWL using Relational.OWL, just to be stored once again in a relational database, to be able to query it with RQL.

In [Mel99], Melnik introduces a working draft for an algebraic definition of RDF models. Unfortunately, this proposal is rather rudimentary lacking of operations required for querying the RDF graph. A more sophisticated approach is presented by Frasincar et al.

[FHVB04]. They introduce a complete algebra for RDF, inspired by the relational algebra. Consequentially, the authors do not only introduce the corresponding algebraic data model, but also adequate operators like the projection, the cartesian product, or the selection. Nevertheless, the authors did not make a direct comparison to the relation algebra.

3 Relational.OWL

In [PC05] we introduced Relational.OWL, a data and schema representation technique, which adopts Semantic Web techniques to the data and schema representation process of (relational) databases. Contrary to other approaches where RDF is stored in relational databases and relational data into RDF (e.g. [Mel01]), Relational.OWL aims at bringing together the representation of both, database data and schema components with a common mediated language based on the *Web Ontology Language* (OWL) [Mv04]. In this section we give a short introduction to the Relational.OWL representation technique, since it is essential for a subsequent application to the data querying process with RDQL (cf. Section 4). For a detailed description of Relational.OWL we again refer to [PC05].

3.1 The Relational.OWL Ontology

To represent the upmost relevant metadata of a relational database using Semantic Web techniques, we require an OWL ontology which describes the schema of a relational database in an abstract way. This OWL representation can easily be interpreted by any remote database or application, which is capable to process OWL and has access to the Relational.OWL ontology. As a further step we use this schema representation itself as a novel ontology for creating a representation format, which is suitable for the corresponding data items.

To describe the schema of a relational database with the techniques provided by the Web Ontology Language OWL, we have to define reference OWL classes centrally, to which any document describing such a database can refer to. The abstract representation of classes like `Table` or `Column` become hereby a central part of the knowledge representation process realized within OWL. Additionally we have to specify possible relationships among these classes resulting in an ontology, a relational database can easily be described with. We call this central representation of abstract schema components and relationships *Relational.OWL.*

Similar representations based on RDF or OWL, which may evolve elsewhere, may be linked to Reltional.OWL with corresponding `owl:equivalentClass` or `owl:equivalentProperty` relationships. As a result, database representations using one of the ontologies mapped, can be understood by any application, which usually uses one of these ontologies.

In other words, each component (database) involved in a representation based on one of these ontologies is able to process documents based on any of the interconnected representation formats. We do not even have to adapt the reasoning processes, since it is enough to create a semantic mapping between two or more ontologies to make them exchangeable, as long as they correlate semantically.

3.2 A novel Ontology

After the description of how to represent the schema of a database using OWL and our Relational.OWL ontology, we now focus on how to use the schema representation just created as a novel ontology. With this tailored ontology-based representation of the database schema, we are able to represent the data, which is stored in that specific database. As a result, data stored in a relational database can be represented as instances of its own OWL schema.

According to the possibilities given by OWL and due to the schema representation presented above, we are able to use individuals (i.e. instances) of our Relational.OWL ontology as classes. Thus the schema representation just created belongs to OWL Full, and not to OWL Lite, nor to OWL DL [DS04]. Of course, the fact that we can not restrict the complexity to one of the subclasses does not automatically result in a complex representation of data and schema items. First implementations make us confident of most OWL reasoning tools being able to handle data and schema representations created using Relational.OWL.

In order to realize this kind of data representation process, we have to ensure that all components involved (e.g. exchange partners) are able to process and understand OWL, have access to Relational.OWL (or a semantically equivalent ontology), and to the OWL schema representation of the corresponding database.

Using the schema constructed above as a novel ontology means to represent the data stored in that specific database using a tailored data representation technique. As a result, the data can be handled using common OWL techniques for data backups, data exchanges, or any kind of data processing tasks.

In fact, this interdigitation of schema and data corresponds exactly to the data management in relational database systems, where data items are stored as instances of their schema. As a result, the ontology, the data is described with, changes as soon as the schema of the originating database is altered. Using conventional techniques in data exchange processes would mean to manually adjust the corresponding exchange format. Using knowledge representation techniques, this is done automatically.

A summary of all the classes and relationships among them together with an exemplary database representation can be found in [PC05].

4 Relational.OWL and RDQL

Having created an automatic transformation mechanism from data stored in relational databases into a representation, which is processable by basically any Semantic Web application, all kinds of legacy data stored in relational databases become an integral part of the Semantic Web.

As a result, Semantic Web applications needing access to data stored in relational databases do not have to query these databases using relational query languages any more. They may equally use their preferred query language like RQL [KCPA01], RDQL [Sea04], or Xcerpt [BS02], as long as this query language provides the required expressiveness. In this paper we analyze RDQL as a representative for all RDF query languages, since it is supported by the Jena Framework [Jen04] and submitted to the W3C [Sea04]. All queries presented in this paper have been verified using the Jena implementation of RDQL. Nevertheless, any RDF query language could be evaluated accordingly.

We thus have to analyze whether all the possible queries on the original relational database can be expressed using RDQL on the Relational.OWL representation of that specific database. In fact SQL has developed throughout the years from a simple query language based on the relational calculus to a powerful language for integrating data from across multiple data sources (cf. [MMJ$^+$01]). Hence, we have decided to compare the expressiveness of RDQL only with the relational algebra [Cod72] and not with SQL, i.e. to check if RDQL is relational complete or not.

The comparison is based on a simple database containing personal and contact information of e.g. business partners. It consists of the following two relations:

> `Address(AddressID, Street, ZIP, City, CountryID)` and
> `Country(CountryID, Name)`.

Since there are various positions on how to verify the relational completeness of a query language (c.f. [CH79]), we have decided to follow the perception of Elmasri and Navathe [NE01] regarding the set $\{\sigma, \pi, \cup, -, \times\}$ of relational operations as complete. Additionally we show how to realize the join operation with RDQL, since it is one of the most important operations of relational queries. Due to the fact that RDQL is not closed, i.e. the result of an RDQL query is not an RDF triple but a list of possible variable bindings, a direct comparison to the relational algebra, which itself is closed, may in some cases be slightly imprecise.

4.1 Selection

One of the basic operations of the relational algebra is the selection σ. The expression

$$\sigma_{Name="Australia"}(r(Country)) \quad (1)$$

would thus select all tuples of the `Country` relation where the attribute `Name` equals `Australia`. Since we have created an own `OWL:Class` for each relation in our database, we have to apply a similar constraint for the objects of this class to obtain the corresponding result with the Semantic Web version of our database. A possible RDQL query is

```
SELECT  ?x, ?y, ?z
WHERE   (?x, rdf:type, dbinst:COUNTRY)
        (?x, dbinst:COUNTRY.NAME, "Australia")
        (?x,?y,?z)
USING   dbinst for [...]
        rdf for [...]
```

The RDQL query representing the selection contains three main clauses. Since RDQL is not closed, we have decided to include the three variables into the `SELECT` clause, from which a valid RDF triple could be created. In the first line of the `WHERE` clause we restrict the result set to contain only objects of the type `dbinst:COUNTRY` having their origin in the `Country` relation of our database. The actual selection σ is performed in the next line, where we enforce the value of the property `dbinst:COUNTRY.NAME` of all the objects represented by the $?x$ variable to be `Australia`. The last line of the `WHERE` clause is required to select the entire set of triples describing the classes which fulfil the conditions described above. Both, the `rdf` and the `dbinst` prefixes are defined in the `USING` clause, representing the commonly used prefix for RDF and the URI for the schema of the database respectively. Since we need the same prefix definitions in all remaining RDQL queries, we do not describe them in the following sections once again.

4.2 Projection

We are able to select relevant attributes of a relation with the projection operation π. Hence, the following expression means that the `Street` and `City` attributes are picked out of the `Address` relation:

$$\pi_{Street,City}(r(Address)). \tag{2}$$

Unlike SQL, we cannot use the `SELECT` clause of RDQL for the projection. It has to be done in the `AND` clause where we can provide more complex constraints.

```
SELECT  ?x, ?y, ?z
WHERE   (?x, rdf:type, dbinst:ADDRESS)
        (?x,?y,?z)
AND     ((?y EQ dbinst:ADDRESS.STREET)||
        (?y EQ dbinst:ADDRESS.CITY))
USING   dbinst for [...]
        rdf for [...]
```

Analogous to the query described above, the result set is restricted to objects of the `dbinst:ADDRESS` type in the `WHERE` clause. The actual projection is done in the `AND` part of the query, where we require the properties of the result triples (i.e. `?y`) to be either `dbinst:ADDRESS.STREET` or `dbinst:ADDRESS.CITY`. The result is a list of all the triples containing city or street information within an address object.

4.3 Set Union

The union $\cup$ operation unifies two union-compatible relations (cf. [Dat82]). The expression

$$\pi_{CountryID}(r(Address)) \cup \pi_{CountryID}(r(Country)) \tag{3}$$

thus unifies all tuples from the `CountryID` attribute in the `Address` with those of the `Country` relation. If we want to query the Semantic Web representation of the database using RDQL, we first have to perform the projection within the `AND` clause restricting the `?y` variable to both `COUNTRYID` attributes. The restriction to both classes is done in the remaining two lines of the `AND` clause.

```
SELECT   ?x, ?y, ?z
WHERE    (?x, ?y,?z)
         (?x,rdf:type,?a)
AND      ((?y EQ dbinst:COUNTRY.COUNTRYID)||
          (?y EQ dbinst:ADDRESS.COUNTRYID))&&
         ((?a EQ dbinst:COUNTRY)||
          (?a EQ dbinst:ADDRESS))
USING    dbinst for [...]
         rdf for [...]
```

This RDQL query thus returns all `COUNTRYID`s originated in both, the `dbinst:COUNTRY` and `dbinst:ADDRESS` objects, i.e. the same as our expression of the relational algebra.

4.4 Set Difference

If it is required to obtain all the tuples contained in one relation and not in a second one, we use the set difference $-$. Thus, the expression

$$\pi_{CountryID}(r(Country)) - \pi_{CountryID}(r(Address)) \tag{4}$$

returns all existing `CountryID`s never used in the `Address` relation. The projection has been introduced only to obtain union-compatibility (cf. [Dat82]).

Within the corresponding RDQL query, objects of the type `dbinst:COUNTRY` are represented by the variable `?a` and the `dbinst:ADDRESS` objects by `?x`. The set difference constraint is specified in the `AND` clause where we refer to the values of both `COUNTRYID` properties assigned to the variables in the `WHERE` clause.

```
SELECT  ?b
WHERE   (?a, dbinst:COUNTRY.COUNTRYID, ?b)
        (?a, rdf:type, dbinst:COUNTRY)
        (?x, dbinst:ADDRESS.COUNTRYID, ?y)
        (?x, rdf:type, dbinst:ADDRESS)
AND     !(?b EQ ?y)
USING   dbinst for [...]
        rdf for [...]
```

Similar to the queries presented above, this RDQL query returns exactly the same information as its corresponding relational algebra expression.

4.5 Cartesian Product

The cartesian product $\times$ unifies two relations into a new relation containing the complete set of attributes from the two original relations. The values of this relation are a combination of all tuples of the first relation with all tuples of the second relation. The expression

$$r(Country) \times r(Address) \tag{5}$$

thus corresponds to a relation containing all attributes from the `Country` relation and all those from the `Address` relation. The original attributes are renamed to guarantee their uniqueness (cf. [Dat82]). The amount of values corresponds to $(m * n)$, whereas m is the number of values in the first table and n in the second table respectively.

The definition of a cartesian product within the Semantic Web is more complex than it seems at a first glance. Melnik, for example, does not mention a cartesian product of RDF triples or Semantic Web objects within his RDF algebra [Mel99]. Intuitively, the cartesian product of two sets with m and n objects would be to create $(m*n)$ new objects, containing the properties of two objects, one of each set respectively (cf. [FHVB04]).

Since RDQL is not closed and we cannot receive objects as a result form an RDQL query, we have to express the cartesian product differently. There are two main options on how to express the cartesian product. Both are as close as possible to the cartesian product of the relational model.

The first option returns all possible combinations of two properties, each from a different set of objects, i.e. one property from the `dbinst:COUNTRY` and one from the `dbinst:ADDRESS` objects at a time:

```
SELECT  ?a, ?b, ?c, ?x, ?y, ?z
WHERE   (?a, ?b, ?c)
        (?a, rdf:type, dbinst:COUNTRY)
        (?x, ?y, ?z)
        (?x, rdf:type, dbinst:ADDRESS)
USING   dbinst for [...]
        rdf for [...]
```

The second option returns a list of all properties contained in any object of the `dbinst:COUNTRY` and `dbinst:ADDRESS` classes.

```
SELECT  ?x, ?y, ?z
WHERE   (?x, ?y,?z)
        (?x,rdf:type,?a)
AND     ((?a EQ dbinst:COUNTRY)||
         (?a EQ dbinst:ADDRESS))
USING   dbinst for [...]
        rdf for [...]
```

Intuitively, this query seems to be more adequate than the one mentioned above. However, it is very similar to the RDQL query in section 4.3 where we expressed the set union. The main difference between both queries is only the restriction in the union query.

4.6 (Equi-)Join

The most important relational operation is indeed the join operation $\bowtie$ introduced in [Cod79]. The θ join of two relations R_1 and R_2 relating to their attributes B_1 and B_2 is the concatenation of the attributes of R_1 and R_2, including their corresponding values, whenever attribute B_1 and B_2 correlate with the θ condition. If θ is $=$, the join operation is called *equi-join*. Since the join operation is usually stated in terms of the cartesian product (cf. [NE01]), the translation of the join operation to RDQL may help to decide, which of both possibilities described in section 4.5 should be considered the cartesian product RDQL correspondent.

The two relations `Address` and `Country` can be joined with the expression

$$r(Address) \bowtie_{CountryID=CountryID} r(Country). \tag{6}$$

Contrary to the natural join, the resulting relation contains all the attributes from the first and the second relation including both `CountryID` attributes.

Once again, since RDQL is not complete, we cannot find an exact equivalent query to the expression of the relational algebra just mentioned. However, we can express quite similar constraints and put both `dbinst:COUNTRY` and `dbinst:ADDRESS` objects into a corresponding relation.

```
SELECT  ?a, ?d, ?e
WHERE   (?a, ?d, ?e)
        (?a, rdf:type, ?c)
        (?x, rdf:type, dbinst:COUNTRY)
        (?x, dbinst:COUNTRY.COUNTRYID, ?y)
        (?r, rdf:type, dbinst:ADDRESS)
        (?r, dbinst:ADDRESS.COUNTRYID, ?s)
AND     (((?c EQ dbinst:COUNTRY) || (?c EQ dbinst:ADDRESS)) &&
         (?y EQ ?s) &&
         ((?x EQ ?a) || (?r EQ ?a)))
USING   dbinst for [...]
        rdf for [...]
```

For expressing the required join condition between both classes, we first define a free result variable `?a`. The objects of the `dbinst:COUNTRY` class are bound to `?x` and those of the `dbinst:ADDRESS` class to `?r`. The values of the relevant `CONTRYID` attributes are bound to `?y` and `?s` correspondingly. The remaining relation between these bound and unbound variables is specified in the `AND` clause, where we restrict the result set to either be a `dbinst:COUNTRY` or a `dbinst:ADDRESS` object. The actual equality condition for the values in `?y` and in `?s` from the join condition is given in the next line. The free variable `?a` is finally bound to the result set in the last line of the `AND` clause.

The decision, which of the queries described in section 4.5 should be considered as the RDQL correspondent of the cartesian product remains unanswered. Even though the query defined in this section certainly signs to the second alternative, where we also had a free variable, this question may probably not be answered precisely until the queries of RDQL can be referred to as closed.

5 Discussion and Future Work

In this paper, we described our approach to semantically represent the schema and data stored in relational databases. Contrary to other approaches, we take the complete schema of the database into account creating a database specific ontology. Based on this ontology, we showed how to represent data items stored in that specific database. With this approach, the schema and data items of each relational database are ready for semantic reasoning without the intervention of any expert. Nevertheless, an expert could add additional semantics to such a data, if it is required.

The next step was to analyze whether such a relational data and schema representation could potentially replace existing interfaces for the access of relational data out of the Semantic Web. The advantage of such an approach is obvious: All Semantic Web applications could query data stored in relational databases with their own built-in query languages without having to convert this data in a manual and time-consuming process. A direct comparison of this approach with SQL, the commonly used interface to relational databases, would be unfair, since SQL has developed during the last decades and most semantic query languages are rather rudimental. Hence, we decided to check whether the expressiveness of querying semantic data created with Relational.OWL is similar to that of the basic relational algebra. We chose RDQL as an exemplary query language since it is widely accepted and provides a stable implementation.

During our analysis we observed that the main problem of RDQL (and of most semantic query languages) is the absence of closeness, i.e. the result set of a query is not a valid RDF/OWL expression any more.

As long as this fundamental drawback is not resolved, we probably will not achieve a language capable to compete with SQL. Nevertheless, we showed in this paper that we are able to simulate each of the main relational algebra operations. However, the question of the relational completeness of RDQL in combination with Relationaol.OWL is still to be answered.

In fact, the results of this paper should be verified with a semantic query language which is closed concerning its queries, i.e. where the result set contains valid RDF triples. The results of this paper make us confident that the combination of such a language with Relational.OWL could replace current SQL gateways to relational databases.

References

[ABM04] Yuan An, Alexander Borgida, and John Mylopoulos. Refining Semantic Mappings from Relational Tables to Ontologies. In Christoph Bussler, Val Tannen, and Irini Fundulaki, editors, *Semantic Web and Databases, Second International Workshop, SWDB 2004, Toronto, Canada, August 29-30, 2004, Revised Selected Papers*, volume 3372, pages 84–90. Springer, 2004.

[BS02] François Bry and Sebastian Schaffert. The XML Query Language Xcerpt: Design Principles, Examples, and Semantics. In *Web, Web-Services, and Database Systems, NODe 2002 Web and Database-Related Workshops, Erfurt, Germany*, Lecture Notes in Computer Science, pages 295–310. Springer, 2002.

[BS04] Christian Bizer and Andy Seaborne. D2RQ -Treating Non-RDF Databases as Virtual RDF Graphs. In *The Semantic Web - ISWC 2004: Third International Semantic Web Conference, Hiroshima, Japan*, 2004. poster presentation.

[CH79] Ashok K. Chandra and David Harel. Computable Queries for Relational Data Bases (Preliminary Report). In *STOC '79: Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 309–318. ACM Press, 1979.

[Cod72] Edgar F. Codd. Relational Completeness of Data Base Sublanguages. *R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California*, 1972.

[Cod79] Edgar F. Codd. Extending the Database Relational Model to Capture More Meaning. *ACM Trans. Database Syst.*, 4(4):397–434, 1979.

[Dat82] Christopher J. Date. A Formal Definition of the Relational Model. *SIGMOD Record*, 13(1):18–29, 1982.

[DS04] Mike Dean and Guus Schreiber. OWL Web Ontology Language Reference. http://www.w3.org/TR/2004/REC-owl-ref-20040210/, 2004.

[FHVB04] Flavius Frasincar, Geert-Jan Houben, Richard Vdovjak, and Peter Barna. RAL: An Algebra for Querying RDF. *World Wide Web*, 7(1):83–109, 2004.

[HBEV04] Peter Haase, Jeen Broekstra, Andreas Eberhart, and Raphael Volz. A Comparison of RDF Query Languages. In *The Semantic Web - ISWC 2004: Third International Semantic Web Conference,Hiroshima, Japan*, pages 502–517, 2004.

[Jen04] Jena - A Semantic Web Framework for Java. http://jena.sourceforge.net/, 2005.

[KCPA01] Gregory Karvounarakis, Vassilis Christophides, Dimitris Plexousakis, and Sofia Alexaki. Querying RDF Descriptions for Community Web Portals. In *17èmes Journées Bases de Données Avancées, BDA'2001, Agadir, Maroc*, pages 133–144, 2001.

[Mel99] Sergey Melnik. Algebraic Specification for RDF Models. http://www-diglib.stanford.edu/diglib/ginf/WD/rdf-alg/rdf-alg.pdf, 1999. Working Draft.

[Mel01] Sergey Melnik. Storing RDF in a Relational Database. http://www-db.stanford.edu/∼melnik/rdf/db.html, 2001.

[MMJ$^+$01] Jim Melton, Jan-Eike Michels, Vanja Josifovski, Krishna G. Kulkarni, Peter M. Schwarz, and Kathy Zeidenstein. SQL and Management of External Data. *SIGMOD Record*, 30(1):70–77, 2001.

[Mv04] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. http://www.w3.org/TR/2004/REC-owl-features-20040210/, 2004.

[NE01] Shamkant B. Navathe and Ramez A. Elmasri. *Fundamentals of Database Systems*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[PC05] Cristian Pérez de Laborda and Stefan Conrad. Relational.OWL - A Data and Schema Representation Format Based on OWL. In Sven Hartmann and Markus Stumptner, editors, *Second Asia-Pacific Conference on Conceptual Modelling (APCCM2005)*, volume 43 of *CRPIT*, pages 89–96, Newcastle, Australia, 2005. ACS.

[Sea04] Andy Seaborne. RDQL - A Query Language for RDF. http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/, 2004.