# Semantic query in a relational database using a local ontology construction

**Authors:**
Saeed M. Sedighi[1]
Reza Javidan[2]

**Affiliations:**
[1]Department of Computer Engineering, Zanjan Branch, Islamic Azad University, Zanjan, Iran

[2]Computer Engineering Department, Islamic Azad University-Beyza Branch, Fars, Iran

**Correspondence to:**
Saeed Mohamad Sedighi

**Email:**
saeed.mohamadsedighi@gmail.com

**Postal address:**
Fars province, Shiraz, Iran
7146778615

**Dates:**
Received: 29 Jan. 2012
Accepted: 20 June 2012
Published: 31 Oct 2012

**How to cite this article:**
Sedighi SM, Javidan R. Semantic query in a relational database using a local ontology construction. S Afr J Sci. 2012;108(11/12), Art. #1107, 10 pages. http://dx.doi.org/10.4102/sajs.v108i11/12.1107



Semantic Web refers to a Web of linked data in which data can be shared and reused, allowing more uses than the traditional 'Web of documents'. However, most of the information on the Web is stored in relational databases and such databases cannot be used by the Semantic Web. Relational databases can, however, be used to construct an ontology as the core of the Semantic Web. We propose a new approach which enables Semantic Web applications to access data actually stored in relational databases using a corresponding ontology. In our approach, domain ontologies can be used to formulate relational database queries in order to simplify the data access of the underlying data sources. The method we propose involves two main phases: the construction of a local ontology from a relational database and a semantic query in a relational database using relational database query language (RDQL). In the first phase, we construct a Web ontology language ontology from data in a relational database. In the second phase, we propose a technique to automatically extract the semantics of relational databases and transform this information into a representation that can be processed and understood by a machine. The method proposed is simulated and implemented using Jena and the simulation results show the effectiveness of the proposed approach. Therefore, we propose RDQL as a real alternative to the commonly used structured query language access to relational databases.

## Introduction

The use of a structured query formulation language is one way to retrieve information in information management systems. Formulated queries allow the selection of data under particular constraints. In contrast to menu driven or query by example (QBE) information access methods,[1] writing structured queries is a powerful method to access data because it allows end-users to formulate complex database queries. However, this method consequently forces end-users to learn specialised query languages. Therefore, structured query formulation, with the exception of a few visual query generation approaches, is difficult for most end-users. Despite the variety of approaches that exists, three major questions are common when information extraction is requested from available data: (1) what type of request can a specific system handle, (2) how can visual interfaces be provided to generate data requests and (3) how can the user be assisted to formulate queries in order to retrieve more accurate information? Nowadays, information technology has been widely adopted in resolving the first two problems by providing some theoretical and practical solutions using artificial intelligence techniques and graph theories, especially by providing visual tools to generate specific queries. However, in the use of computational techniques, there is inadequate information to provide users with query formulation services using domain ontologies. Laborda and Conrad[2] introduced a representation format for both schema and data information based on Web ontology language (OWL). The advantage of their approach is obvious – relational data is able to be processed for Semantic Web applications using built-in functionality like query languages or reasoning mechanisms. In fact, with their approach, Semantic Web applications no longer need to implement their own in relation to semantic mapping.

In this paper, we propose an approach to learning OWL ontology from data in relational databases. We then investigate the combination of OWL ontology and an exemplary semantic query language – relational database query language (RDQL) – in order to achieve an alternative for ordinary query using structured query language (SQL). In other words, we aimed to determine if combining OWL ontology with such a query language would lead to the same results as a normal relational query would. Because SQL has been extended repeatedly in its expressiveness during the preceding decades, a direct comparison of RDQL and SQL would be unfair. Therefore, the present analysis has been limited to whether the combination of OWL ontology and RDQL is capable of providing the same results as the relational algebra.

## Related work

Managing ontology data alone is not a new topic and several systems have been developed during the past years.[3,4,5] Some of these systems store ontology data in a file system, making querying such data very difficult.[4] Other systems transform the ontology data into RDF form and store the RDF triples (the subject–predicate–object statements) in a relational database. Processing of ontology-related queries in these systems is typically done by an external middle ware (wrapper) layer built on top of a database management system (DBMS) engine. However, DBMS users cannot reference ontology data directly.

Querying relational data together with their semantics encoded in ontology is an emerging topic that has attracted much attention recently. A method to support ontology-based semantic matching in relational database management systems (RDBMS) using SQL directly has been proposed.[6] In this method, ontology data are pre-processed and stored in a set of system-defined tables. Several special operators and a new indexing scheme are introduced so that a database user can reference the ontology data directly using the new operators. The main drawback of this approach is that semantic queries involving the ontology data are usually difficult to write and costly to process (in terms of both processing time and storage overhead) as a result of the graphical structure of the ontology data and the need for reasoning (i.e. transitive closure computation) on the ontology data.

Calvanese et al.[5] proposed virtual view as a way to represent relational data together with their related ontology data in a relational view. However, there are three requirements in the application of virtual view: (1) language extensions to SQL must support the creation and use of virtual view, (2) the DBMS engine must support native XML data (together with relational data) and the processing of the virtual view related operators and (3) the user must understand the complex ontology data and their relationship with the base relational data completely.

Query by example (QBE) is a well-known concept in the database community. It was first proposed by Zloof in the mid 1970s[7,8] as a query language that can be used by database users to define and query a relational database. QBE is quite different from SQL in that it is a graphical query language. Its interface is usually virtual tables where the user can enter commands, examples, etc. Since QBE was presented most of the research on QBE has focused on the enrichment and extension of QBE as a query language and on developing efficient methods for generating and processing the queries defined by the examples.[5]

In commercial database products, QBE is widely used as a graphical front-end for RDBMSs.[1] It is also used as a convenient interface for users to specify queries for image, video and document databases, and various techniques have been studied.[2,5,7,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23] There are two common characteristics in all previous work on QBE: the examples are used to specify a query that will be generated and the generated query is a 'normal' query in that all the query conditions (which may be in the form of similarity measures) are defined on the base attributes in the underlying tables.

The semantic QBE problem addressed by Zloof is very different from the traditional QBE problem. Firstly, as a result of the complexity of the semantic information associated with the data in the base relational tables, the real query associated with the user's intention which is specified by the input examples is difficult (or impossible) to capture by a traditional SQL query. Secondly, in the described problem, the underlying 'query' is defined not only by the base attributes in the relational table, but also by the semantics of the base data encoded in the ontology and the connections between the relational data and the ontology data.

## First phase : Local ontology construction

Databases include conceptual models and information resources that together can be taken as the conceptualisation repository of ontology. Based on analyses of the formal corresponding relationships between relational databases and OWL ontologies: a relational database contains several tables, a table contains several fields and records are the collection of a field's value, whereas an OWL ontology contains several classes, a class contains several properties and instances are the collection of property values. The formal corresponding relationships between tables, fields and records in relational databases and classes, properties and instances in OWL ontologies make it possible to convert one schema to another. The corresponding relationships between relational database components and ontology components are shown in Figure 1.

The use of existing relational databases to generate ontology automatically is the main objective of the proposed approach, in order to reduce the manual tedious work, save developing time and improve the efficiency of ontology. The building of a local ontology architecture from a relational database (Phase 1) is shown in Figure 2.

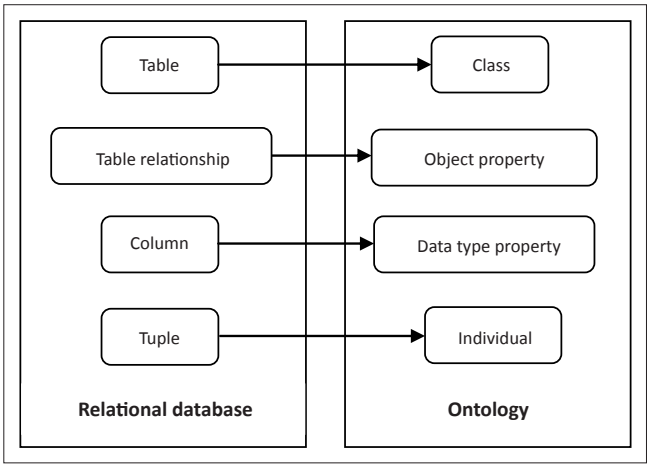
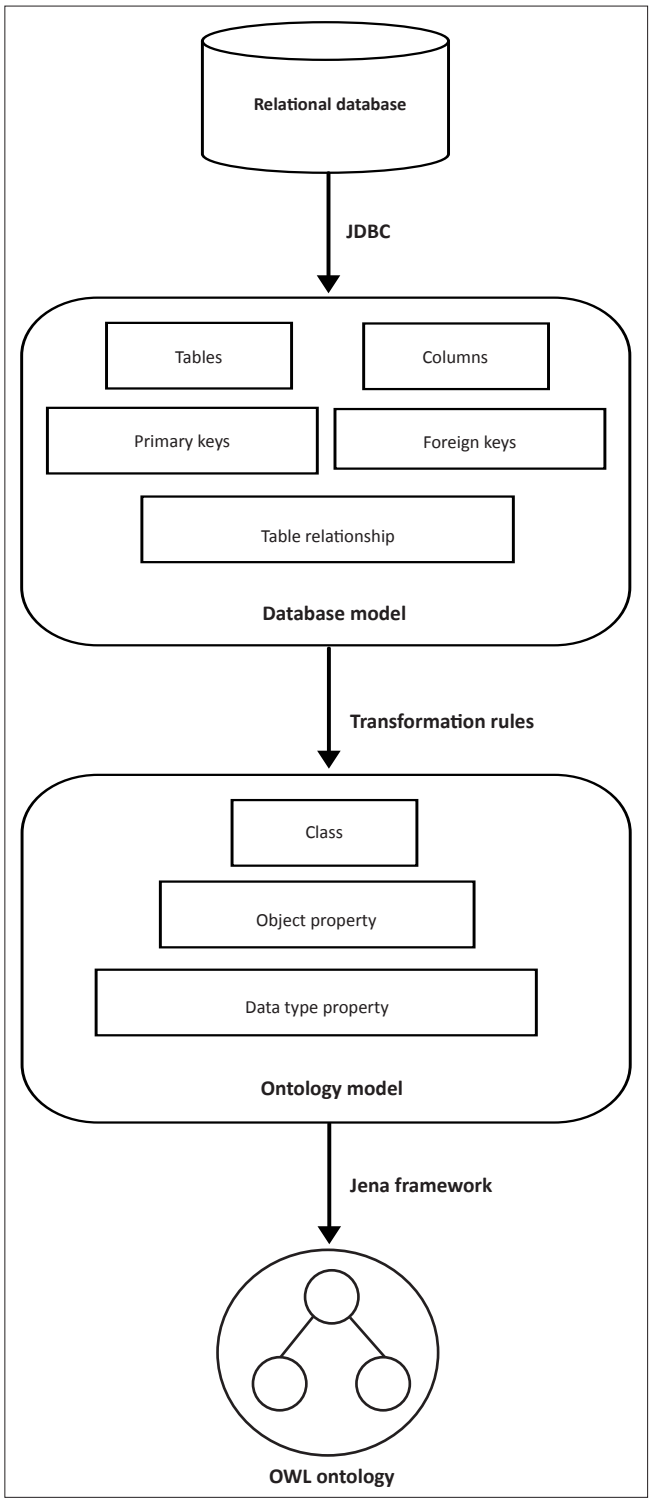



**FIGURE 1:** Relational database components and their corresponding ontology components.




JDBC, Java database connectivity; OWL, Web ontology language.

**FIGURE 2:** The construction of a local ontology from a relational database.

Construction of a local ontology from a relational database includes the following steps:

- Extraction of metadata from the relational database by Java database connectivity components
- Analysis of the metadata from Step 1 and transfer of the database model to the ontology model by transformation rules
- Transfer of the ontology model to the OWL ontology by the Jena framework.

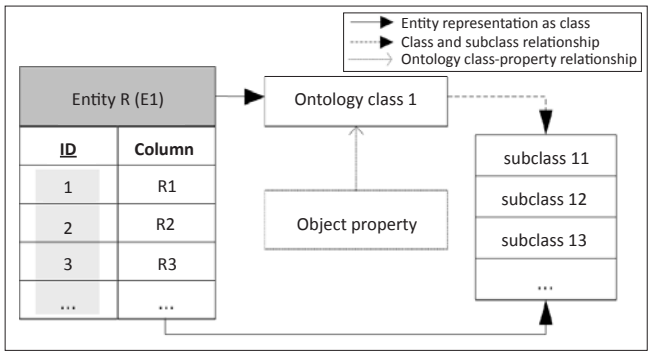



**FIGURE 3:** Ontological representation of an entity type E.

## Ontological representation of an entity containing domain metadata

As shown in Figure 3, an entity type E (containing domain metadata) in a relational schema R(E) is represented as a class in the ontological model. Here, each distinct column value is stored as a subclass of the entity class. An object property is created which points to the class as a property range. This rule is only applicable to a standalone entity and does not apply for the representation of domain metadata based on other cardinal relationships (e.g. one-to-one or one-to-many).

## One-to-one relationships

In one-to-one (1:1) relationships, R is a relationship in a database D that links an entity type E1 in D to the entity type E2 in D, with P1 as the primary key of E1 and P2 as the primary key of E2; and R is a one-to-one relationship type between the entity types E1 and E2. Figure 4 shows entities E1 and E2 in a relational schema R(E). Here, both R(E1) and R(E2) contain domain metadata and there is a one-to-one relationship from R(E1) to R(E2). In such a situation, both R(E1) and R(E2) are required to be represented as ontology classes. For R(E1), an ontology class is created for entity type R(E1) and each entity of the entity type is represented as a subclass of the entity type class. For R(E2), two situations are possible: (1) a 1:1 generalisation (specialisation, 'is-a') relationship from R(E1) to R(E2), as shown in Figure 4a and (2) any other ID-based 1:1 relationship (e.g. 'has-a' or 'part-of') from R(E1) to R(E2) (as shown in Figure 4b).

For an 'is-a' relationship as shown in Figure 4a, each column value (i.e. col-3) stored as a foreign key value is represented as a subclass (i.e. class-111, class-121 etc.) of the parent entity class (i.e. class-11, class-12 etc.). In this way, all of the R(E2) entities are represented under a generalised class (class-1). The parent class (class-1) is defined as a *range* class for the related object property in order to have each *foreign key* value mapped to a common object property. In addition, similar entities could be further defined under one generalised parent class, if needed. An example of such a situation, shown in Figure 4a, is *'Antibiotic drugs'* as the parent class and the drug *'Actinomycin'* as an *(is-a)* antibiotic drug.

For all other types of *ID relationships*, as in Figure 4b, an ontology class (class-2) is created for entity type R(E2) and the column values (i.e. col-3) stored against each *foreign key*

are mapped to a subclass (i.e. class-21, class-22 etc.) of the entity type class (class-2). In order to link class-1 (primary key values) with class-2 (foreign key values), object properties are used. In order to support the relationship between domain entities within the ontology, each subclass of class-2 (i.e. foreign key column values) is linked to an object property (i.e. objprop-21, objprop-22 etc.). All these object properties are defined by a generalised property (object property-2). Here, the individual properties (objprop-21, objprop-22 etc.) link the subclasses of class-1 (i.e. subclass-11, subclass-12 etc.) to the subclasses of class-2 (i.e. subclass-21, subclass-22 etc.) through property *domain* and *range* relationships. This is how the links are established between *primary key* row instances (for which the corresponding class is defined as a *domain* class) and *foreign key* row instances (for which the corresponding class is defined as a *range* class). An example of such a situation, as shown in Figure 4b, is a country '*France*' that has a *(has-a)* capital '*Paris*'.

## One-to-many relationships

In Figure 5, both R(E1) and R(E2) contain domain metadata and there is a one-to-many (1:M) relationship between R(E1) and R(E2). This situation is similar to 1:1 mappings, with the only major difference being the ontological representation of foreign key values for entity R(E2).

In the case of an *is-a* relationship, as shown in Figure 5a, the column values (i.e. col-3) stored against a foreign key are represented as subclasses of the parent entity class (i.e. class-111 and class-112 for class-11 and class-121 for class-12 etc.). For all other types of *ID relationships*, as in Figure 5b, column values (e.g. col-3) stored against one common *foreign key* are represented under a parent class having these values as subclasses (i.e. class-211 and class-212 are defined as subclasses for class-21 etc.). The parent classes (i.e. class-21, class-22 etc.) are represented by one generalised class (class-2) of R(E2). Here, the object properties are created for each distinct *foreign key* value. All of these object properties are defined by a generalised property (object property-2). Individual properties (objprop-21, objprop-22 etc.) link the subclasses of class-1 (i.e. subclass-11, subclass-12 etc.) to the subclasses of class-2 (i.e. subclass-21, subclass-22 etc.) through property *domain* and *range* relationships. This is how links are established between *primary key* row instances (for which the corresponding class is defined as a *domain* class) and *foreign key* row instances (for which the corresponding class is defined as a *range* class). An example of such a situation, as shown in Figure 5b, is a country '*France*' that has cities '*Paris*', '*Lyon*' etc.

## Many-to-many relationships

In many-to-many (M:N) relationships, R is a relationship in a database D that links an entity type E1 in D to the entity type E2 in D, with P1 being the primary key of E1 and P2 being the primary key of E2; R is a many-to-many relationship type mapped to a schema relation denoted by R(R) = P1 U P2.

Figure 6 shows the entities E1 and E2 in a relational schema R(E) with a many-to-many relationship between them. In

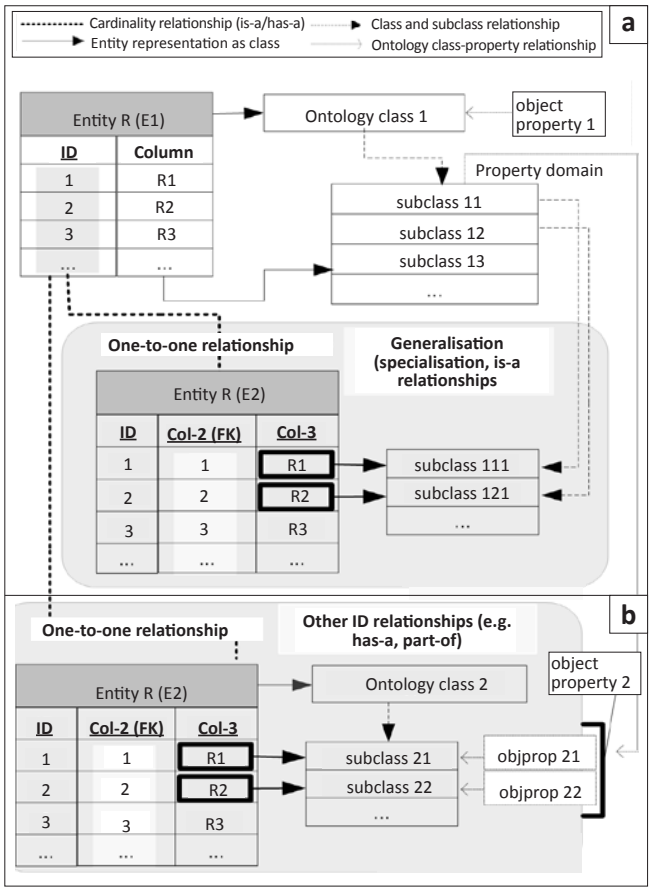



**FIGURE 4:** A schematic showing the one-to-one relationships between entities E1 and E2 for (a) generalisation relationships and (b) other ID relationships.

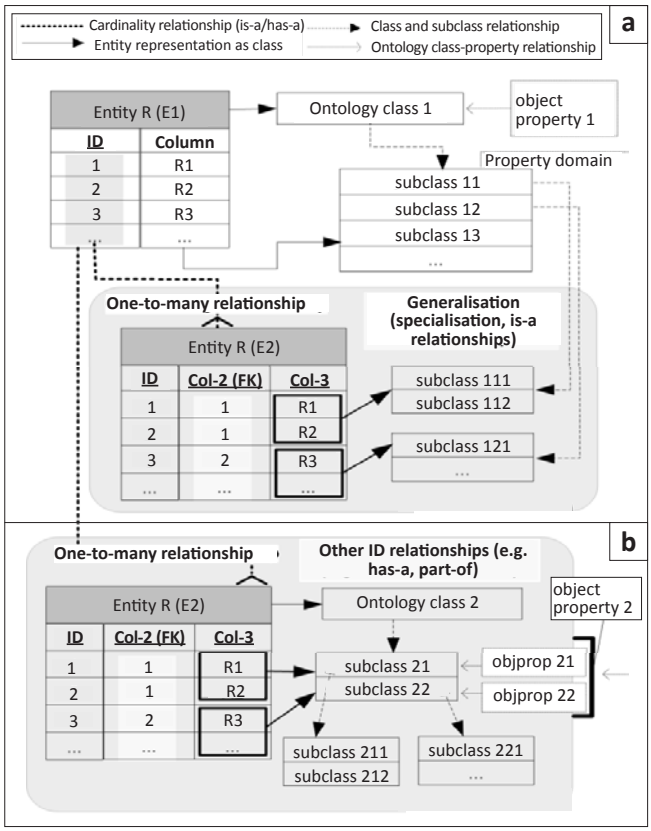



**FIGURE 5:** A schematic showing the one-to-many relationships between entities E1 and E2 for (a) generalisation relationships and (b) other ID relationships.

such a situation, only the entity types R(E1) and R(E2) are represented in the ontology as classes. In practice, the day-to-day transactions are stored with such R(R) relations. Such relationships are not stored in the ontology as classes because we do not store transactional data in the ontology; instead the entity-related domain metadata are used to query such transactional data.

### Database to ontology mapping for data columns

In OWL-DL, property restrictions can be applied to both data type properties (properties for which the value is a data literal) and object properties (properties for which the value is an individual). Here, 'object properties' link individuals to individuals and 'data type properties' link individuals to data values.

In the proposed method, table columns for which values are data literals and do not contain domain metadata or semantics are transformed into data type properties. This transformation is because the table columns that contain domain metadata or semantics are defined by class or subclass relationships or by linking them with object properties. Therefore, in order to specify ontology restrictions on data values, for each of the selected table columns, a data type property is created which links to the related ontology class as *rdfs:domain*. In such a case, the *rdfs:range* data type of the 'data type property' is defined according to the column data type. Examples of such data columns in the medical domain are: '*patient's registration year*', '*patient's disease duration*', '*patient's height*', '*patient's weight*' etc. In these examples, the *rdfs:domain* class is '*patient*'.

Figure 7 shows an entity E in a relational schema R(E) with the attributes 1 2 , , ..., *n Col Col Col* belonging to entity E. Here, R(E) is represented as an ontology class E and columns (i.e. 1 2 , , ..., *n Col Col Col* ) are represented as data type properties (i.e. 1 2 , , ... , *n Data type Property Data type Property – – Data type–Property* ) with class-E defined as *rdfs:domain*. The *rdfs:range* data type for these ontology properties is defined by the columns (1 2 , , ..., *n Col Col Col* ) data type. Such a transformation of data columns can be validated by applying a reverse transformation, that is, by converting ontology data type properties to relational database table columns.[8] Here, all data type properties are parsed in a series. For each parsed data type property, a database table is located similarly to the *rdfs:domain* value for data type property, and a data column is created with the name of that property.

## Second phase: Semantic query

By creating an automatic transformation mechanism from data stored in relational databases into a representation, which can be processed by almost 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 no longer need to query these databases using relational query languages. These applications can use preferred query languages like

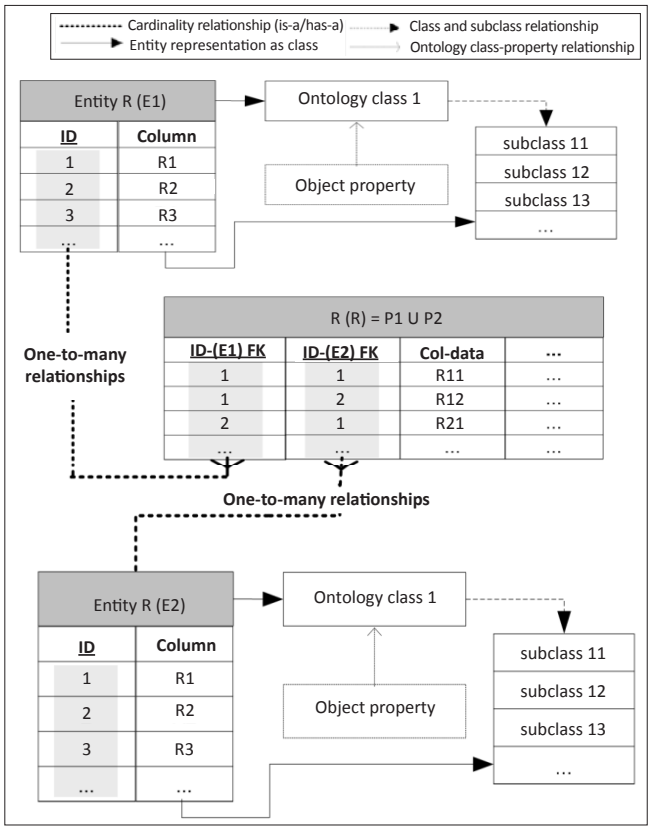



**FIGURE 6:** A schematic showing the many-to-many relationships between entities E1 and E2.

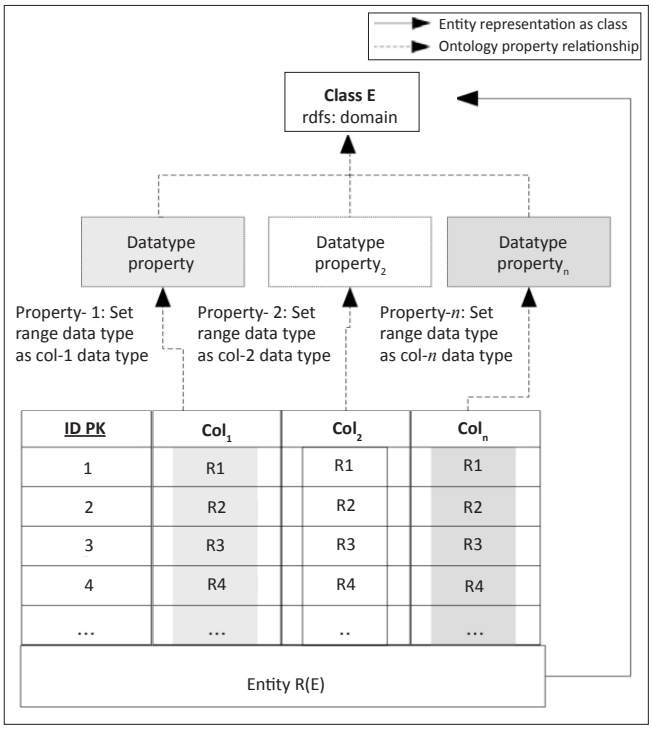



**FIGURE 7:** Ontological representation of data columns.

RQL,[24] RDQL[25] or Xcerpt,[26] as long as the chosen query language provides the required expressiveness. In the present work, RDQL as a representation is analysed for all RDF query languages, because RDQL is supported by the Jena framework[27] and has been submitted to the W3C.[25] All queries presented in this paper have been verified using the

Jena implementation of RDQL. Nevertheless, any RDF query language could be evaluated accordingly. In the current phase, the local ontology built from a previous phase is used, as shown in Figure 8.

Therefore, in this study, we investigated 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 relational calculus to a powerful language for integrating data from across multiple data sources. Hence, we compared the expressiveness of RDQL with only relational algebra[8] and not with SQL (i.e. to determine if RDQL is relational complete). The comparison was based on a simple database containing personal and contact information of, for example, business partners. It consisted of the following two relations: *Address (AddressID, Street, ZIP, City, CountryID) and Country(CountryID, Name).*

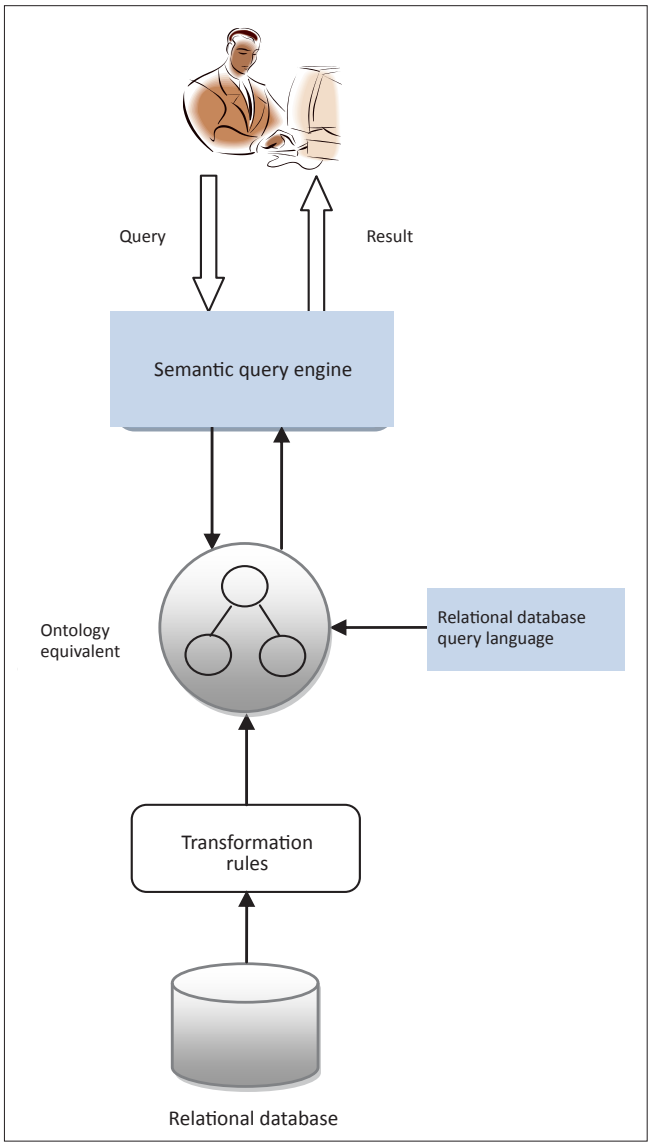



**FIGURE 8:** The architecture of a semantic query.

There are various positions on how to verify the relational completeness of a query language[28]; we followed the method of Choupo et al.[29] regarding the set of relational operations {σ, π, Ụ, −, ×}. We also included the join operation realisation with RDQL because it is one of the most important operations of relational queries. As RDQL is not closed, that is, 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.

## Selection

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

σ Name='Australia'(r(Country))

would therefore select all tuples of the *Country* relation where the attribute *Name* equals *Australia*. Because an *OWL:Class* has been created for each relation in the database, a similar constraint for the objects of this class should be applied 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 – SELECT, WHERE and USING. As RDQL is not closed, the three variables (*?x*, *?y* and *?z*) can be included in the *SELECT* clause from which a valid RDF triple can be created. In the first line of the *WHERE* clause the result set is restricted to contain only objects of the type *dbinst:COUNTRY* having their origin in the *Country* relation of our database. The actual selection 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 that fulfil the conditions described above. Both the *rdf* and the *dbinst* prefixes are defined in the *USING* clause and represent the commonly used prefixes for RDF and the uniform resource identifier (URI) for the schema of the database, respectively. Because the same prefix definitions are used in the remaining RDQL queries, they need not be described again.

## Projection

Selection of the relevant attributes of a relation with the projection operation π is possible. Thus the following expression means that the *Street* and *City* attributes are selected from the *Address* relation:

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

Unlike for SQL, the SELECT clause of RDQL cannot be used for the projection. It must be performed using the AND clause where more complex constraints can be provided.

```
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 [...]
```

As for the query described above, the result set is restricted to objects of the *dbinst:ADDRESS* type in the *WHERE* clause. The actual projection is performed in the AND part of the query, where the properties of the result triples (i.e. *?y*) are required 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.

## Set union

The union Ụ operation unifies two union-compatible relations.[25] The expression

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

therefore unifies all tuples from the *CountryID* attribute in the *Address* with those of the *Country* relation. If the query of the Semantic Web representation of the database using RDQL is required, one first needs to perform the projection within the *AND* clause to restrict 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 therefore returns all *COUNTRYID*s originating in both the *dbinst:COUNTRY* and *dbinst:ADDRESS* objects (i.e. the same result as our expression of the relational algebra).

## Set difference

In order to obtain all the tuples contained in one relation and not in a second relation, the set difference – has been used. Therefore, the expression

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

returns all existing *CountryID*s never used in the *Address* relation. The projection has been introduced only to obtain union compatibility.[30]

Within the corresponding RDQL query, objects of the type *dbinst:COUNTRY* are represented by the variable *?a* and the *dbinst:ADDRESS* objects are represented by *?x*. The set difference constraint is specified in the *AND* clause which refers 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 [...]
```

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

## Cartesian product

The Cartesian product × 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 and all tuples of the second relation. The expression

$$\text{r(Country)} \times \text{r(Address)}$$

therefore 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.[25] The number of values corresponds to ($m * n$), where $m$ is the number of values in the first table and $n$ is the number of values in the second table.

The definition of a Cartesian product within the Semantic Web is more complex than it seems at first glance. Melnik[31], for example, does not mention a Cartesian product of RDF triples or Semantic Web objects within his RDF algebra. 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.[32]

As RDQL is not closed and the objects resulting from an RDQL query cannot be received, the Cartesian product should be expressed differently. There are two main options for expressing 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 which the set union is expressed. The main difference between both queries is the restriction in the union query.

## (Equi-)join

The most important relational operation is indeed the join operation ∞ introduced by Trinkunas and Vasilecas[33]. The θ join of two relations R1 and R2 relating to their attributes B1 and B2 is the concatenation of the attributes of R1 and R2, including their corresponding values, whenever attribute B1 and B2 correlate with the θ condition. If θ is =, the join operation is called equi-join. As the join operation usually is stated in terms of the Cartesian product,[31] the translation of the join operation to RDQL may help to decide which of the two possibilities previously described should be considered the Cartesian product RDQL equivalent.

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

$$r(\text{Address}) \infty_{\text{CountryID=CountryID}} r(\text{Country}).$$

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

Once again, as RDQL is not complete, an exact equivalent query to the expression of the relational algebra just mentioned cannot be found. However, similar constraints can be expressed for both *dbinst:COUNTRY* and *dbinst:ADDRESS* objects in the 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, at first, a free result variable *?a* is defined. The objects of the *dbinst:COUNTRY* class are bound to *?x* and those of the *dbinst:ADDRESS* class are bound to *?r*. The values of the relevant *COUNTRYID* attributes are bound to *?y* and *?s* correspondingly. The remaining relation between these bound and unbound variables is specified in the *AND* clause, where the result set is restricted to either 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.

**TABLE 1:** An example of a vcard table.

| Person name | First name | Last name |
|---|---|---|
| John Smith | John | Smith |
| Matt Jones | Matthew | Jones |
| Sarah Jones | Sarah | Jones |
| Becky Smith | Becky | Smith |

```
<owl:Class rdf:about= NS + "#vcard"/>
<owl:Class rdf:about= NS + "#vcard_pk_class"/>
<owl:InverseFunctionalProperty rdf:about=NS+"#vcard-pkOP">
<rdfs:range rdf:resource=NS+ "#vcard-pk_class"/>
<rdfs:domain rdf:resource=NS+ "#vcard"/>
</owl: InverseFunctionalProperty>
<owl:Restriction rdf:about=NS+ "#vcard-pkMinRes">
<owl:minCardinality
rdf:datatype=http://www.w3.org/2001/XMLSchema#string>
</owl:minCardinality>
<owl:onProperty rdf:resource=NS+ "# vcard-pkOP"/>
</owl:Restriction>
<owl:DatatypeProperty rdf:about=NS+ "#vcard-PersonName">
<rdfs:domain rdf:resource=NS+ "# vcard–pk_Class"/>
<rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about= NS +"#vcard_FirstName">
<rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
<rdfs:domain rdf:resource= NS +"#vcard"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about= NS +"# vcard_LastName">
<rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
<rdfs:domain rdf:resource= NS +"#vcard"/>
</owl:DatatypeProperty>
<NS:vcard>
<NS:vcard-pkOP>
<NS:vcard-pk_Class rdf:about=NS+ "#pk_vcard">
<NS:vcard-PersonName rdf:datatype =
"http://www.w3.org/2001/XMLSchema#string">John Smith
</NS: vcard-PersonName>
</NS: vcard-pk_Class>
</NS:vcard-pkOP>
<NS: vcard_FirstName>John
</NS: vcard_FirstName>
<NS: vcard_LastName>Smith
</NS:vcard_LastName>
....
</NS:vcard>
```

**FIGURE 9:** The Web ontology language (OWL) ontology corresponding to the vcard table in Table 1.

As to which of the queries described above should be considered as the RDQL equivalent of the Cartesian product remains undecided. However, the query defined in this section certainly indicates the second alternative, where a free variable is created. This question is likely to remain unanswered until the queries of RDQL can be referred to as closed.

# Implementation

We propose to implement the method in two phases:

- Phase 1: transform a relational database to OWL ontology
- Phase 2: perform a semantic query on the local ontology built from Phase 1

## Phase 1: Transform a relational database to an OWL file

In order to implement the transformation from a relational database table to an OWL file, we propose a method using RDQL. The proposed method is implemented in Java and is based on the Jena application programming interface (API). Table 1 and Figure 9 together illustrate the use of transformation rules to export data on a vcard table from a database named vcard into OWL ontology. Table 1 shows the vcard table and Figure 9 shows the corresponding OWL ontology.

```
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.util.FileManager;
import com.hp.hpl.jena.vocabulary.*;
import java.io.*;
/** selecting the VCARD resources
public class query extends Object {
static final String inputFileName = "v1.rdf";
public static void main (String args[]) {
// create an empty model
Model model = ModelFactory.createDefaultModel();
// use the FileManager to find the input file
InputStream in = FileManager.get().open(inputFileName);
if (in == null) {
throw new IllegalArgumentException( "File: " +
inputFileName + " not found");
}
// read the RDF/OWL file
model.read( in, "");
// select all the resources with a VCARD.FN property
ResIterator iter = model.listSubjectsWithProperty(VCARD.FN);
if (iter.hasNext()) {
System.out.println("The database contains category for:");
while (iter.hasNext()) {
System.out.println(" " + iter.nextResource()
.getRequiredProperty(VCARD.FN)
47
.getString() );
}
} else {
System.out.println("No vcards were found in the database");
}
}
}
```

**FIGURE 10:** A Jena semantic query on a vcard table using a Web ontology language (OWL) ontology.

## Phase 2: Perform a semantic query on the local ontology

Jena APIs are used for manipulating RDF graphs and OWL files. Jena is a Java class library, and is composed mainly of APIs and SPIs (system programming interfaces). This API provides an interface for the Semantic Web application developer that makes it an ideal programming toolkit when the OWL file should be processed. In the OWL API, the key OWL package for the application developer is com. hp.hpl.jena.ontology.owl. This package contains interfaces for representing models, resources, properties, literals, statements and all the other key concepts of OWL, and a ModelFactory for creating models. Figure 10 shows a semantic query on a vcard table using OWL ontology built from Phase 1.

A comparison of existing methods with the proposed method is shown in Table 2.

# Conclusion and future work

One of the major requirements of ontology-assisted query formulation systems and for performing semantic queries on a relational database is the formulation of a domain ontology which includes a definition of domain metadata, relationships and knowledge of the ontology. In this regard, an ontology modelling approach has been identified which transforms domain metadata and relationships into the ontology schema to assist in the query formulation process. Once the basic structural elements of the domain ontology are defined, they are further enriched with domain knowledge. Moreover, in order to generate relational query statements as per the underlying database schema structure, ontology database mappings are expressed as a set of correspondences that relate the vocabulary of a relational model (table/relation, column etc.) with the ontology model (concept, property etc.). The method proposed here involves two main phases. In the first phase, a local ontology is constructed from a relational database. In the second phase, a semantic query in a relational database is simulated and implemented using RDQL. RDQL can be considered as a real alternative to the commonly used SQL access to relational databases.

There is scope for extending this work by querying distributed relational databases on the Semantic Web using a global ontology. In order to achieve this goal, we propose that

**TABLE 2:** A comparison between existing methods and the method proposed.

| Method | Mode | Relationship | Data transform | Implementation | Semantic query |
|---|---|---|---|---|---|
| Astrova et al.[12] | Auto | 1:1 | No | Yes | No |
| Xu et al.[3] | Semi-auto | 1:1 | No | No | No |
| OGSRD[13] | Semi-auto | 1:1 | Yes | Yes | No |
| OWLFROMDB[18] | Auto | 1:1 | Yes | Yes | Yes |
| Proposed method | Auto | 1:1,1:N,1:M | Yes | Yes | Yes |

a local ontology should first be built from a local relational database and then all local ontologies can be integrated into a global ontology.

# Acknowledgements


## Competing interests

We declare that we have no financial or personal relationships which may have inappropriately influenced us in writing this paper.

## Authors' contributions

R.J. was the project leader and S.M.S. was responsible for the project design and for writing the manuscript.


# References


1. Munir K, Odeh M, McClatchey R. Ontology assisted query reformulation using the semantic and assertion capabilities of OWL-DL ontologies. Paper presented at: IDEAS 2008. Proceedings of the Twelfth International Database Engineering & Applications Symposium; 2008 Sep 10–12; Coimbra, Portugal. ACM International Conference Proceedings Series 299. 2008; p. 81–90.
2. Laborda CR, Conrad S. Relational OWL – A data and schema representation format based on OWL. Paper presented at: APCCM 2005. Proceedings of the Second Asia-Pacific Conference on Conceptual Modelling; 2005 Dec 16–21; Australia. Sydney: Australian Computer Society, Inc; 2005. p. 89–96.
3. Xu Z, Zhang S, Dong Y. Mapping between relational database schema and OWL ontology for deep annotation. Paper presented at: WI 2006. Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence; 2006 Aug 13–18; Washington DC, USA. Washington DC: IEEE Computer Society; 2006. p. 548–552.
4. Munir K, Odeh M, Bloodsworth P, McClatchey R. Using assertion capabilities of an OWL-based ontology for query formulation. Paper presented at: ICTTA 2008. Proceedings of the 3rd International Conference on Information & Communication Technologies: From theory to applications; 2008 Apr 07–11; Damascus, Syria. Bristol: IEEE Xplore; 2008. p. 1–6.
5. Calvanese D, Giacomo GD, Lembo D, Lenzerini M, Poggi A, Rosati R. Linking data to ontologies: The description logic DL-LiteA. Paper presented at: OWLED 2006. Proceedings of the 2nd Workshop on OWL: Experiences and directions; 2006 April 03-08; New York, USA. New York: OWLED, Inc; 2006. p. 1–10.
6. The Information Societies Technology Project: Health-e-Child [homepage on the Internet]. No date [cited 2012 Sep 19]. Available from: www.health-e-child.org
7. Zloof MM. Query-by-example: The invocation and definition of tables and forms. Paper presented at: 1st International Conference on Very Large Data Bases. Proceedings of the 1st International Conference on Very Large Data Bases; 1975 Oct 19–24; New York, USA. New York: ACM, Inc.; 1975. p. 1–24.
8. Zloof MM. Query-by-example: A data base language. IBM Syst J. 1977;16(4):324–343. http://dx.doi.org/10.1147/sj.164.0324
9. Lim L, Wang H, Wang M. Unifying data and domain knowledge using virtual views. Paper presented at: VLDB 2010. Proceedings of the 36th International Conference on Very Large Data Bases; 2010 Sep 23–28; Vienna, Austria . Vienna : ACM, Inc.; 2010. p. 255–266.
10. Freund J. Health-e-Child. An integrated biomedical platform for grid-based pediatric applications. Paper presented at: Health-Grid 2006. Proceedings of Health-Grid 2006; 2006 June 12–17; Valencia, Spain. Valencia: Stud Health Technol Inform; 2006. p. 259–270.
11. Anjum A, Bloodsworth P, Branson A, et al. The requirements for ontologies in medical data integration: A case study. Paper presented at: IDEAS 2007. Proceedings of the Eleventh International Database Engineering & Applications Symposium; 2007 Sep 06–08; Alberta, Canada. Alberta: IEEE Computer Society; 2007. p. 308–314.
12. Astrova I, Korda N, Kalja A. Rule-based transformation of SQL relational databases to OWL ontologies. Paper presented at: the 2nd International Conference on Metadata & Semantic Research. Proceedings of the 2nd International Conference on Metadata & Semantic Research 2007 Oct 11–12; Corfu, Greece. Corfu: MTSR; 2007. p. 1–16.
13. Vysniauskas E, Nemuraite L. Transforming ontology representation from OWL to relational database. Inf Technol Control. 2006; 35(3A):333–343.
14. Belkhatir M, Mulhem P, Chiaramella Y. A conceptual image retrieval architecture combining keyword-based querying with transparent and penetrable query-by-example. Paper presented at: CIVR 2009. Proceedings of the ACM International Conference on Image and Video Retrieval; 2009 Nov 13–18; Berlin, Germany. Berlin: Springer-Verlag; 2009. p. 528–539.
15. Boccignone G, Chianese A, Moscato V, Picariello A. Animate system for query by example in image databases. Paper presented at: EuroIMSA 2010. Proceedings of the European Conference on Internet and Multimedia Systems and Applications; 2010 Feb 21–23; Grindelwald, Switzerland. Grindelwald: EuroIMSA; 2010. p. 451–456.
16. Aleixandre-Benavent R, Aleixandre-Tudo JL, Alcaide GG, Ferrer-Sapena A, Aleixandre JL, Du Toit W. Bibliometric analysis of publications by South African viticulture and oenology research centres. S Afr J Sci. 2012;108(5/6):74–84. http://dx.doi.org/10.4102/sajs.v108i5/6.661
17. Alalwan N, Zedan H, Siewe F. Generating OWL ontology for database integration. Paper presented at: SEMAPRO '09. Proceedings of the Third International Conference on Advances in Semantic Processing; 2009 Oct 11–16; Sliema, Malta. Los Alamitos, CA: IEEE; 2009. p. 22–31. http://dx.doi.org/10.1109/SEMAPRO.2009.21
18. Astrova I, Korda N, Kalja A. Toward the Semantic Web: Extracting OWL ontologies from SQL relational schemata. Paper presented at: IADIS 2006. Proceedings of the International Conference WWW/Internet; 2006 Oct 5–8; Murcia, Spain. Murcia: IADIS Press; 2006. p. 62–66.
19. Chang-rui YU, Hong-wei W, Fu J. Development method of domain ontology based on reverse engineering. Appl Res Comput. 2006;35:1–5.
20. Olajubu EA, Aderounmu GA, Adagunodo ER. Network resources management in a multi-agent system: A simulative approach. S Afr J Sci. 2010;106(9/10):31–36. http://dx.doi.org/10.4102/sajs.v106i9/10.322
21. Fortuna B, Mladeni D, Grobelnik M. Semi-automatic construction of topic ontologies. Paper presented at: Semantics, Web and Mining. Proceedings of Semantics, Web and Mining; 2006 Sep 14–16; Berlin, Germany. Berlin: Springer; 2006. p. 121–131.
22. Sedighi SM, Javidan R. A novel method for improving the efficiency of automatic construction of ontology from a relational database. Int J Phys Sci. 2012;7(13):2085–2092. http://dx.doi.org/10.5897/IJPS12.072
23. Adesina AO, Agbele KK, Februarie R, Abidoye AP, Nyongesa HO. Ensuring the security and privacy of information in mobile health-care communication systems. S Afr J Sci. 2011;107(9/10):26–32. http://dx.doi.org/10.4102/sajs.v107i9/10.508
24. Codd EF. Relational completeness of data base sublanguages. In: Rustin R, editor. Database systems. IBM Research Report RJ 987. San Jose, CA: Prentice Hall, 1972; p. 65–98.
25. Seaborne A. RDQL – A query language for RDF. Bristol: W3C Member Submission ; 2004.
26. Bry FO, Schaffert S. The XML query language Xcerpt: Design principles, examples, and semantics. Lecture Notes in Computer Science. Munich: Springer-Verlag, 2002; p. 295–310.
27. Fahmi I. Jena – A Semantic Web framework for Java. Tenerife: SWHi Blog: ; 2005.
28. Navathe SB, Elmasri RA. Fundamentals of database systems. 3rd ed. Boston: Addison-Wesley Longman Publishing Co. Inc.; 2001.
29. Choupo AK, Berti-Equille L, Morin A. Optimizing progressive query-by-example over pre-clustered large image databases. Paper presented at: the 2nd International Workshop on Computer Vision Meets Databases. Proceedings of the 2nd International Workshop on Computer Vision Meets Databases; 2005 June 13–16; New York, USA. New York: ACM; 2005. p. 13–20.
30. Date CJ. A formal definition of the relational model. ACM SIGMOD Record. 1982;13(1):18–29. http://dx.doi.org/10.1145/984514.984515
31. Melnik S. Algebraic specification for RDF models. California: IEEE Computer Society; 1999.
32. Calvanese D, Giacomo GD, Lembo D, Lenzerini M, Poggi A, Rosati R. Ontology-based database access. Paper presented at: SEBD 2007. Proceedings of the 15th Italian Symposium on Database Systems; 2007 June 17–20; Fasano, Italy. Fasano: SEBD; 2007. p. 324–331.
33. Trinkunas J, Vasilecas O. A graph oriented model for ontology transformation into conceptual data model. Inf Technol Control. 2007;36(1A):126–132.