# From Relational Databases to Ontology-Based Databases

Hamaz Kamal[1] and Benchikha Fouzia[1,2]
[1]*LIRE Laboratory, University Mentouri of Constantine 2, Constantine, Algeria*
[2]*Computer Science Department, University of 20, Aout 1955, Skikda, Algeria*

Keywords: Relational Databases, Reverse Engineering, Semantic Discovery, Ontology-Based Databases.

Abstract: Nowadays, the volume of data used in an information system grows rapidly. Additionally, enterprise information systems are more open to distributed environments and platforms. Thus, the need for interoperability between the different underlying data sources increases considerably. Therefore, data storage should consider storing data as well as the semantic of it in a single database. To overcome this problem, Ontology-Based Databases seem to be a good choice to replace legacy databases. In this sense, this paper proposes a reverse engineering approach which transforms a relational database to an ontology. The extracted ontology is enriched with more semantics by mean of external domain ontology. Finally the ontology and data are stored in one of the existing specific architectures for Ontology-Based Databases.



## 1 INTRODUCTION

The success of enterprises clearly depends on how services are being improved and upgraded with time. From this perspective many enterprises are faced with the problem of having to optimize their information systems. This would lead to high quality services and better interactions. For every information system, databases play an important role for data storage, retrieval and manipulation. Therefore, the development of databases leads also to the development of any enterprise information system. Despite the fact that the relational model is the most used model for data storage, other database models have appeared. In particular, XML and object-oriented databases have appeared with the aim to give additional capabilities, which doesn't exist in relational databases. More specifically, the semantic representation of the stored data is the most required capability. This capability is essential especially for enterprises that deal with heterogeneous environments, in which it is needed to handle interoperability between different underlying data sources.

In order to maintain semantics of the stored data, the use of a conceptual model within a database is seen to be necessary. Many conceptual models give the ability to represent semantics of any given domain. As an example, UML model or ontologies are able to represent semantics of any domain. Nowadays the use of ontologies increases in many fields. Ontologies provide a rigorous and a formal manner for the formulation of conceptual schemes. Therefore works to combine ontologies and databases have emerged (Pierra, 2005); (Alexaki, 2001); (Broekstra, 2002). Once the ontology is stored with data in a single database, it is possible to retrieve the definition or the meaning of the requested elements. Hence providing semantics of the stored data is considered. Such databases are called Ontology-Based Databases (OBDB).

In this paper we propose a reverse engineering approach that transforms a relational database into an Ontology-Based Database. The literature shows how a reverse engineering process could provide abundant benefits in discovering semantics in legacy databases (Hainaut, 2002). Therefore, in this paper we propose a set of rules to extract an ontology from a relational database. The resulted ontology is enriched with more semantics by mean of external domain ontology. In the last step the ontology and data are stored in an OBDB architecture. Here OntoDB (Dehainsala, 2007) is the architecture that is being used for storage. The motivation behind choosing OntoDB is shown in section five.

The rest of the paper is organized as following: section two and three talk about database development and reverse engineering respectively. In section four our proposed approach for reverse engineering relational databases to ontologies is

Kamal H. and Fouzia B..
From Relational Databases to Ontology-Based Databases.
DOI: 10.5220/0004454802890297
In *Proceedings of the 15th International Conference on Enterprise Information Systems* (ICEIS-2013), pages 289-297
ISBN: 978-989-8565-59-4

detailed. Section five gives some aspects of the OntoDB architecture. A case study is conducted in section six followed by conclusion and future work.

# 2 DATABASE DEVELOPMENT

The development of database models and DBMS had been always an interesting area of research. The progress of information systems regards directly and indirectly the progress of databases. Researchers have employed different models and techniques to improve databases. In the last years, a significant attention is dedicated to reuse existing information resources, and provide access to them at the semantic level. In the following we define two of the major database models that were for big influence in database development.

## 2.1 Object-oriented Databases

Object-oriented databases are databases that store data as objects. Also, objects can be interpreted by methods defined in their related classes. Additionally, interesting relationships between objects such as inheritance are preserved. However, object-oriented database systems are closely dependent to a specific language such as C++. They have been seen for a while as a rival to replace relational databases (Ramanathan, 1997). Though the relational model has continued being broadly used, due of its maturity acquired over the years. In the other side, the inconveniences of the object-oriented model (lack of standard, complex databases…) have long persisted. Meanwhile, ontologies appeared and didn't stop to be increasingly interesting.

## 2.2 Ontology-Based Databases

### 2.2.1 Ontologies

An ontology is defined as a specification of a conceptualization (Thomas 1993). Ontologies define hierarchies based on structured vocabularies, grouping together concepts/classes and their relationships and instances of classes.

### 2.2.2 Advantages of Ontologies

Ontologies provide many advantages in comparison with the relational and object models. They provide rigorous formulation of the conceptual schemas. They make it possible for systems to use data semantics. And other advantages as inferences and use of online vocabularies like WordNet. As a consequence, many works on using ontologies within databases have started to appear (Pierra, 2004); (Broekstra, 2002) assuming that using ontologies will give born to more semantically efficient databases.

### 2.2.3 Approaches for Ontology Storage

The concept of OBDB is widely defined and explained in (Pierra, 2005). Such databases use ontologies as one of the database layers. They store in a same database data and semantics that define data by the mean of an ontology.

There exist different approaches with different ways to store ontologies.

- *Vertical Representation Approach:* In this approach the storage is simple where classes and relationships/properties are stored as triples: 'Subject, Predicate, Object'. Jena (Wilkinson, 2003) is an example of such an approach.
- *Specific Representation Approach:* In this approach, the storage is different from an implementation to another. However the most general strategy stores separately the ontology and data, where each data should have a reference to its conceptual element in the ontology. IBM SOR (Jing, 2007) follows this approach for storing ontology and data. There exist other approaches that add a new part called 'Meta-Schema' as with the OntoDB architecture (Dehainsala, 2007) (See Figure1).

Different languages for exploiting and querying OBDB exist as SPARQL (Seaborne,), OntoQL (Jean, 2006), etc.

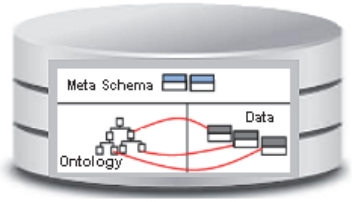


Figure1: Ontology-Based Database architecture.

The advantage of OBDB approaches is the fact that it is possible to query either the stored data, the ontology that defines the data or both. According to the diverse advantages of such approaches and of ontologies in general, works on relational database reverse engineering to ontologies have appeared. In the following we define the reverse engineering process and its early and late targeted models.

# 3 RELATIONAL DATABASE REVERSE ENGINEERING

The reverse engineering process is a process that analyses a given model or a system in order to discover information concerning its components and their relationships. Therefore it is possible to discover semantics, especially for legacy/relational databases. Also the main motivation behind applying a reverse engineering process over legacy databases is that these last contain an important mass of data, as well as useful information to reuse. Many models were defined to be the result of this process. First ones were toward the conceptual schema, as the work in (Johansson) which extracts a conceptual schema from a relational database. Some other works considered extracting semantics under an object oriented model as the work in (Ramanathan, 1997). Hence, this would provide better semantics under an object-based representation of the relevant entities. Lately ontologies are being the most used model to be the source-to-target of relational database reverse engineering. The related work can be classified as follows.

## 3.1 Exploiting Relational Schemes

Although some works on relational database reverse engineering consider using external sources to end up with a better ontology, most of the existing approaches exploit only the relational schema/database. As an example the work in (Astrova, 2004) proposes an approach that analyses the SQL-DDL code of the database and the correlations between keys, attributes and data in order to discover semantics. The process divides the relational schemes into base relations, composite relations and dependant relations. An extension of this work has succeeded in (Astrova, 2006) and represents the resulted ontology in OWL. It takes also into account the representation with OWL of some database constraints as the constraint CHECK. Another recent work in (Zdenka, 2010) focuses on conversion of constructs from relational database to OWL constructs, in addition of transferring the relational data to ontology instances by application of their rules.

## 3.2 Exploiting an External Source

In this category of approaches an ontology is extracted from a relational schema/database and then enriched by help of an external source. Some approaches exploit the semantics that could be found in the related HTML pages/forms used to interact with the database. Such as the work in (Astrova, 2005) and (Benslimane, 2008). This last uses HTML forms to analyze them in order to discover more semantics. The discovered new entities and organized in a way that permits to discover relevant constraints and dependencies between data. Another work in (Kashyap, 1999) exploits user queries to add additional semantics to the ontology extracted from the database. This last shows that user queries can indeed reveal semantics which could be translated to new classes or properties.

In the next section we present our approach for ontology extraction from a relational database and its enrichment.

# 4 THE PROPOSED APPROACH

The proposed approach supposes that the relational database is in the 3NF, and do not use surrogate keys. Three main steps constitute our approach that fulfils respectively the following tasks:

- ✓ Ontology extraction from a relational database by application of a set of rules.
- ✓ Enrichment of the extracted ontology by help of external domain ontology.
- ✓ Ontology and data storage in OntoDB.

## 4.1 Preliminaries

- ***Basic Relational Concepts:*** A relational schema can be seen as a finite set of attributes that we denote $\partial$. Attributes are denoted using the capital letters of the alphabet from beginning *A, B, C,* $A_1$ …, a set of attributes is denoted by the last capital letters of the alphabet *X, Y, Z*. We use *R* to represent a relation schema, where *R* is a subset of $\partial$ and $(R_1 \cup R_2 \cup \ldots R_i) = \partial$. If we assume that *R(X)* represents a relation schema with *n* attributes, we write $X_i = A_1, A_2, A_n$. Also each attribute has a set of values *D*. Tuples are denoted by *t* such that for a given relation $R(A_1, A_2, \ldots A_n)$ we have $t_i(R) \subset D_i(A_1), {}^{X} D_i(A_2) {}^{X} \ldots {}^{X} D_i(A_n)$, where $t_i$ of *R* represents values of a specific tuple.
- ***Keys and Dependencies:*** A primary key *PK* is a candidate key *CK* chosen to identify attributes of *R,* we write *PK(R)* to denote a *PK* of a relation, and *PK(X)* to denote *X* as the attributes used for the *PK*. A foreign Key *FK* in a relation $R_1$ is a *PK* in another relation $R_2$ (sometimes in the same

relation), where values of the *FK* reference values of the *PK*. We use *S* to denote the set of all attributes that represent foreign keys in ∂. Inclusion dependencies IncD exist between two relation schemes if the following form holds: $R_1(Y) \subseteq R_2(Z) \mid Y = Z$ (same sequence of attributes), IncD takes the following form $R_1(Y) \rightarrow R_2(Y)$.

***Ontologies:*** We describe an ontology by a 4-tuple $O=(C_C, A_C, R_C, H_C)$, Where:

- $C_C$ is a finite set of classes $(C_1, C_2, C_3 \ldots C_n)$.
- $A_C$ is a collection that represents the sets of attributes belonging to the classes. A(c) denote attributes that belong to one class.
- $R_C$ is a finite set of roles/relationships which exist between classes. $C_1 R C_2$ denotes a role between two concepts | $C_1$ (Domain), $C_2$ (Range).
- $H_C$ represents the hierarchy or taxonomy between concepts. We write $H_C\ C_1 \subseteq C_2$ to denote that $C_1$ is the subclass of $C_2$.

In the following, our transformation process based on rules is presented and explained.

## 4.2 Transformation Process

The transformation process is composed of a set of rules to create ontological classes and hierarchies between classes. Then it creates roles/relationships between classes and finally the attributes of classes.

**Rule#1:**
**For every:** *PK(R) ∩ S = ∅ (When such a case holds we call R an atomic relation)*
**Do:** *Create C*

**Rule#2:**
**For every:** *Two atomic relations $R_1$, $R_2$ with: $PK(R_1) \subseteq PK\{/CK\}$ ($R_2$)*
**Do:** *Create $H_C\ C_1 \subseteq C_2 \mid C_1, C_2$ correspond respectively to $R_1, R_2$*

**Rule#3:**
**For every:** *Atomic relation $R_1$ with: $FK(R_1) \subseteq PK(R_1)$*
**Do:** *Create C | C Class created for attribute FK*
*Create $H_C\ C \subseteq C_1 \mid C_1$ Class created for $R_1$*

**Rule#4:**
**For every:** *Two atomic relations $R_1, R_2$ with: IncD: $R_1(Y) \rightarrow R_2(Z)$ exists such as: $FK(Y) \subseteq PK(Z)$*
**Do:** *Create $C_1 R C_2 \mid C_1$ (domain), $C_2$ (range)*

**Rule#5:**
**For every:** *$PK(R_1)$ such tha PK(W,Y) represent the attributes used for the PK, with: IncD: $R_1(Y) \rightarrow R_2(Y)$ $PK(R_1) \cap PK(R_2) = Y$*
**Do:** *Create $C_2 R C_1 \mid C_2$ (domain), $C_1$ (range)*

**Rule#6:**
**For every:** *R(Z) where: $Z \in S$ and $Z = n$ number of attribute with $n \geqslant 2$.*
**Do:** *$\forall\ i,j \leqslant n \wedge i \neq j \wedge i < j$*
*Create $C_i R C_j \mid C_i$ (domain), $C_j$ (range)*
*Create Inverse Role $C_j R C_i$*

The values of *i* and *j* represent the relations that are being referenced by the *FK*'s of *R(Z)*.

**Rule#7:**
**For every:** *R(Z,W) such that $Z \in S$.*
*If: $W \cap S = \emptyset$*
**Do:** *Create C*
*Create roles the same way as in Rule#6*

**Rule#8:**
**For every:** *R having X a set of its attributes, such that foreign keys are not considered.*
**Do:** A(c) = *{$(X_1, X_2 .. X_i)$}*

We studied these rules to cover the different possible relational forms to capture as much semantics as possible. After the creation of classes in Rule1 hierarchies between classes are created in rule2. Rule3 treats reflexive cases when a relation references itself, thus a new class is interesting to be created to represent an additional semantic link. In the other hand, roles are created in Rule4, Rule5 and Rule6. More specifically Rule 5 treats the case of weak entities translated to relations, and Rule6 treats binary and n-ary relationships where we used variables *i, j* to treat the different possible cases. Rule7 is a special case of Rule6, when an additional attribute(s) exist and not referencing any other relation. A class is created for this case to not lose the semantics of the possible additional attributes (See attribute grade in table 2). And last, Rule8 creates attributes of classes.

Once the ontological model is extracted it will be enriched under the enrichment process.

## 4.3 Enrichment Process

The enrichment process is an effective way to add more semantics to the extracted ontology. More often the ontology extracted from a relational database is closer to an object model. However, by adding more semantics, the ontology will be more adequate. The enrichment is based on an algorithm and some additional cases for adding more roles.

Table 1: Algorithm of enrichment.

| **Main Function** |
|---|
| ***Description:*** *The main function supposes that we already have two ontologies (our extracted ontology RO and the domain ontology DO).* |
| **Prior Step**. Organize classes of RO from top to down in the array r1. Organize classes of DO from bottom to top in the array r2<br>c1 and c2 represent respectively each class from r1 and r2<br>foreach c1 in r1<br>{ k= Stemming (c1); //receiving the root word in variable k<br>r3= WordNet(k); //get possible synonyms in the array r3<br>foreach c2 in r2<br>foreach q in r3<br>{b=StringCompare(q,c2); //b is boolean false by default<br>if (b); //if b=true<br>//once there's similarities we call these two functions.<br>DegreeOfSimilarities (c1,c2);<br>Enrichment(c1,c2);<br>}<br>} |
| **Degree of Similarities** |
| ***Description:*** *Degree of similarities is calculated between each two classes. With help of the degree calculations done between classes, an expert can decide which classes to keep or to deleted* |
| P1= an array that will have all properties of c1;<br>P2= an array that will have all properties of c2;<br>x1= max (P1.length,P2.length);<br>x2= number of similar properties between c1 and c2;<br>Degree= (x1/x2); // Degree is a float variable |
| **Enrichment** |
| ***Description:*** *The enrichment function adds subclasses/superclasses and their properties once two classes show similarities. Also, it makes sure not to add existing classes. Once all enrichment finished roles that point only to one class will be deleted.* |
| co1Sub= classes subsumed by c1;<br>co1Sup= the super class of c1;<br>co2Sub= classes subsumed by c2 (if they exist);<br>if co1Sup = null //no super classes<br>// ***Case 1*** {Write in an output file the class c1 with all the chain of the super classes found in DO, and their properties; }<br>foreach s in co1Sub<br>for each *f* in co2Sub<br>{<br>if ((s && WordNet(s)) are all different of *f*) // test to not add classes that already exist<br>// ***Case 2*** {Write always in the output file the class s with f as its new subclass with its properties;}<br>if (f.subclass = = (any other s in co1 \|\| Wordnet (s)) // we call d any s that satisfies this condition<br>// ***Case 3*** 'Once a class added to c1' {Write on the same output file that d is now subsumed by *f* and not anymore by c1 (delete subsumption between (c1, d)); }<br>} |

### 4.3.1 Algorithm of Enrichment

This proposed algorithm attempts to add new classes/properties. We inspired some points for our algorithm from the work in (Rene, 2010). The main steps of the algorithm are shown in table 1.

The fact that domain ontologies exist lately for a very large number of fields motivates their use in our algorithm of enrichment. In addition they could cover interesting semantic aspects. In this sense, they could bring many benefits to an extracted ontology from a relational database of the same domain of application.

### 4.3.2 Adding More Roles

Additional roles can be discovered between classes if the following conditions hold: Relations related to the classes are atomic relations. One or more of these atomic relations contain ($\geqslant 2$) foreign keys. In other words let's suppose we have three atomic relation schemes $R_1$, $R_2$ and $R_3$. And $R_1$ has two foreign keys $R_1$ ($X$, $FK_1$, $FK_2$) such as:

$$FK_1(R_1) \subseteq PK(R_2) \text{ and } FK_2(R_1) \subseteq PK(R_3).$$

If such a condition holds it is possible to derive a new role between $R_2$ and $R_3$. However, the nature of the role can be only decided by an expert that knows the semantics of the database.

## 5 PROCESS OF STORAGE

The last step in our approach consists on storing the enriched ontology and data which existed in the relational database. As we mentioned we use the OntoDB architecture for storage. This last offers many advantages: it stores ontology and data separately, it has a meta-schema part, which is flexible and stores different ontology models. And facilitates updates, since the ontology is stored as an instance of the meta-schema, in addition, it enables to store semantic definitions in other languages.

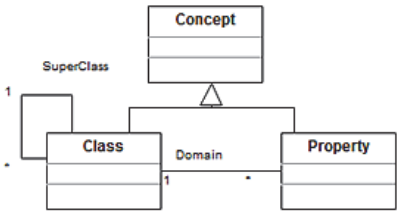


Figure 2: Meta schema part.

OntoDB architecture covers similarities with the

relational model as when using foreign keys to link different parts together. It was implemented under a PostgreSQL environment. OntoDB has a query language called OntoQL, which is very close to SQL. According to these features, an enterprise that uses a relational database would prefer to have a better storage environment, but yet not a totally different from the one existing. Thus we opted for OntoDB because it covers most today's desired features in a database.

The creation of a database based on OntoDB architecture is carried out using OntoQL statements for ontology and data definition. Also it is possible to alter and update the database once it is created.

The general syntax for ontology definition is shown as follows. More details of OntoQL can be found in (Jean (thesis), 2006).

```
⟨class definition⟩     ::= CREATE ⟨entity id⟩ ⟨class id⟩ [ ⟨under clause⟩ ]
                           [ ⟨descriptor clause⟩ ] [ ⟨properties clause list⟩ ]
⟨under clause⟩         ::= UNDER ⟨class id list⟩
⟨descriptor clause⟩    ::= DESCRIPTOR ( ⟨attribute value list⟩ )
⟨attribute value⟩      ::= ⟨attribute id⟩ = ⟨value expression⟩
⟨properties clause⟩    ::= ⟨entity id⟩ ( ⟨property definition list⟩ )
⟨property definition⟩  ::= ⟨prop id⟩ ⟨datatype⟩ [⟨descriptor clause⟩]
```

Once the ontology is defined we proceed by the migration of data from the relational database. These data will be stored in the content part of OntoDB. To populate the content part, data can be directly transferred from each table/relation in the relational database to its related table in OntoDB. Obviously, each class in the ontological part of OntoDB can have a table for storing its instances in the content part. Hence, if a relation $R$ in $\partial$ has a created class $C$ in OntoDB and $C$ has been assigned a table $T$ in the content part. Data from $R$ will be transferred to $T$ with the same sequence of tuples. This applies only for attributes that are common for both $R$ and $C$. Let's suppose that the following form represents a specific tuple in the relation Person:

$t_i\ (Person) \subset D_i(Id\text{-}Person_1),\ ^{X}\ D_i(name_1)\ ^{X}\ D_i(address_1)$

In the content part we use the same values of tuple $t_i$ for populating a tuple in the extent of Person's class, if we suppose that it has the same defined attributes. Tuples in the relation Person will be mapped in the same order to the table Person of the content part. In the other hand, the link between the ontology part and content part is kept (Jean, 2006), because every class is related to the table(s) that define its data. More of the storage details are represented in the next section of the case study.

# 6 CASE STUDY

In this section an example of simple relational database is given to illustrate the application of our approach (See table 2). After this, we present details of the storage process with OntoQL clauses. By the same way, we show the advantages of OntoDB when querying data semantics. We write attribute{number} to distinguish foreign keys, primary keys are underlined.

Table 2: Relational schema.

| |
|---|
| **1) Person** (**Id-Person**, name, address, marriedwith{1}) |
| **2) Professor**(**Id-Prof**{1}, salary, CourseID{6}) |
| **3) Student**(**Id-Stud**{1}, year, DepId{4}) |
| **4)Department**(**DepId**, name, devided-In{9}, located-In{8}) |
| **5) Laboratories**(**IdLab**, interests, budget, DepId{4}) |
| **6) Course**(**CourseId**, title, section) |
| **7) CourseTime**(**CT, CourseId{6},** starttime) |
| **8) City**(**CityId**, name, population) |
| **9) Division**(**NumDiv**, description) |
| **10) Enrolled**(**Id-Stud{3}, CourseId{6}**, grade) |
| **11) Teaches**(**Id-Prof{2}, Id-Stud{3}**) |

Since the enrichment is based on a domain ontology, we use the one found in (Domain-Ontology), a part of it is shown in figure 3.


Figure 3: A part of the domain ontology.

The result of the application of our transformation and enrichment processes is represented in figure 4. Each class labelled with either C or R or both means respectively that it was enriched with more class(es) or role(s).

For the enriched classes they were benefited with more semantics. The class project was added 'SoftwareProject' and 'ResearchProject' as new subclasses. The class Person was added 'Employee' and the class Student 'Graduate' and 'Undergraduate'. In the other hand, roles have been added to Department, Student and Person. Finally a new role (Laboratories '*located-in*' Division) is added with the case of adding more roles of section 4.3.2. Next step consists on ontology and data storage.

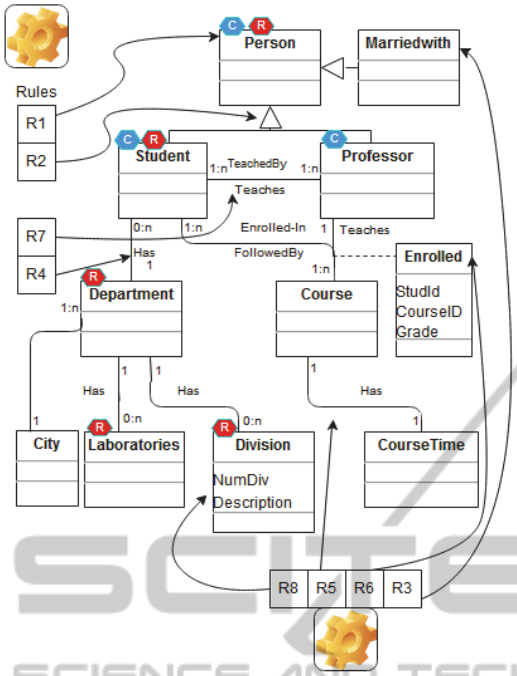


Figure 4: Result after applying our approach.

## 6.1 OntoQL Statements

In order to execute OntoQL statements tools as OntoQL-Plus and PLIB-Editor exist. The clause CREATE CLASS is used to create a new class. The one called DESCRIPTION is used to describe the class and provide it with more semantics. The PROPERTIES clause represents attributes of the class. In the following a part of OntoQL statements is shown for ontology definition in OntoDB.

```
CREATE Class Person
DESCRIPTOR(
#name[fr]='une personne',
#definition='human being')
PROPERTIES (Id-Person INT, name String,
address String)

CREATE Class Professor UNDER Person
DESCRIPTOR(
#name[fr]='Professeur',
#definition='Teacher or supervisor')
PROPERTIES (Id-Prof INT, Salary Float)

CREATE Class Student UNDER Person
DESCRIPTOR(
#name[fr]='Etudiant',
#definition='Person studying')
PROPERTIES (Id-Stud INT, year DATE)

CREATE Class Department
DESCRIPTOR(
#name[fr]='Département',
#definition='Academic buildings')
PROPERTIES (DepId INT, name String)
```

We proceed the same way with the rest of classes. In order to store roles we extend the meta-schema part, the entity Role is created under the existing entity 'Property' in the meta-schema.

```
CREATE #Entity Role UNDER #Property
(#Rolename String,
 #InverseRole String,
 #Domain REF(#Class),
 )
```

The classes related to the roles will be linked to them with internal identifiers. For the content part an extent is created for each class issued of the transformation process.

```
CREATE EXTENT OF Person (Id-Person, name,
address, marriedwith);
```

Such a clause creates a table in the content part. The clause used to insert values is:

```
INSERT INTO (attributes) VALUE (values);
```

It is possible also to create views as an extent of other tables of the content part. For example, we represented 'MarriedWith' as a subclass of Person. We could then create a view for it based on the attributes of person and its data.

```
CREATE VIEW OF MarriedWith AS
(SELECT Id-Person, name, Marriedwith)
FROM Person
WHERE (Marriedwith IS NOT NULL)
```

This clause creates a new table with the attributes of the selection and will eventually represent only married persons.

## 6.2 Querying OntoDB

The OntoQL query language provides different abilities that could be used to exploit the semantics of the stored ontology and data. Here are some examples:

The first query retrieves the code of classes which define a property called 'located-in'.

```
SELECT c.#code
FROM c in #class, p in #properties
WHERE p.#name = 'located-in'
```

It is possible to retrieve the names of classes with the provided other language.

```
SELECT #name[FR] FROM #class
```

The following query returns classes that have 'Etudiant' as their French name.

```
SELECT c.#oid
FROM c in #Class
WHERE c.#name[fr] like 'Etudiant'
```

We could query also on roles

```
SELECT p.#role, p.#inverserole FROM
#PROPERTY AS p WHERE p.#oid=101
```

Furthermore, we could select from ranges of super/sub classes and also properties by using: #superclasses #directsuperclasses #subclasses #scopeProperties #properties #usedproperties. All these are very useful to exploit better the ontological semantics stored in OntoDB. This sub-section highlighted only few of OntoQL abilities, which are more interesting (Jean (thesis), 2006). We show on a future work how we could retrieve more from OntoQL abilities, and from using OntoDB for ontology and data storage.

# 7 CONCLUSIONS

In this paper, we presented a reverse engineering approach that aims to migrate relational databases to OBDB. We proposed first a set of transformation rules to extract an ontology from a relational database. And to optimize the extracted ontology we proposed an enrichment process. This last uses external domain ontology and attempts to add more classes or properties to make the ontology semantically more complete. Then we stored the enriched ontology and data in OntoDB. In a future work we will focus on running more experiments, tests and evaluations on larger databases, and prove how OBDB could bring many benefits for an enterprise. Equally important, we prove the efficiency of our transformation process by testing several transformation criteria. Such criteria are useful to prove that there's no information loss in the transformation process.

# REFERENCES


Alexaki, S. Christophides, V. Karvounarakis, G. Plexousakis, D. Tolle, K., 2001. The ics-forth rdfsuite: Managing voluminous rdf description bases. *In SemWeb.* Hong Kong, pp. 1-13.

Astrova, I., 2004. Reverse Engineering of Relational Databases to Ontologies. *In Proceedings of the 1st European Semantic Web Symposium (ESWS),* Heraklion, Greece, pp 327-341.

Astrova, I., Kalja, A., 2006. Mapping of SQL Relational Schemata to OWL Ontologies. *In Proceedings of the 6th WSEAS International Conference on Applied Informatics and Communications, Elounda, Greece.* pp375-380.

Astrova, I., Stantic, B., 2005. An HTML Forms Driven Approach to Reverse Engineering of Relational Databases to Ontologies. *In Proceedings of the 23rd IASTED International Conference on Databases and Applications (DBA),* Innsbruck, Austria.pp 246 - 251.

Benslimane, S., Malki, M., Rahmouni, M., Rahmoun, A., 2008. Towards Ontology Extraction from Data-IntensiveWeb Sites: An HTML Forms-Based Reverse Engineering Approach. *The International Arab Journal of Information Technology.* pp 34-44.

Broekstra, J., Kampman, A., Van Harmelen, F., 2002. Sesame : A generic architecture for storing and querying rdf and rdf schema. *Proceedings of the First Internation Semantic WebConference, number 2342 in Lecture Notes in Computer Science.* pp 54–68.

Dehainsala, H., Pierra, G., Bellatreche, L., 2007. Ontodb: An ontology-based database for data intensive applications. In *DASFAA*. pp 497-508.

Domain-Ontology http://ontoware.org/swrc/swrc_v0.3.owl

Hainaut, J., 2002. Introduction to database reverse engineering. *Book,* 160 pages.

Thomas R, Gruber., 1993. Formal ontology in conceptual analysis and knowledge representation. Chapter: "Towards principles for the design of ontologies used for knowledge sharing". *Kluwer Academic Publishers.*

Jean,. S., Aït Ameur, Y., Pierra, G., 2006. Querying ontology based databases the ontoql proposal. *In Proceedings of the 18th International Conference on Software Engineering & Knowledge Engineering*, pp 166—171.

Jean, S (thesis).2006., OntoQL, un langage d'exploitation des bases de données à base ontologique. *Mémoire de doctorat.*

Jing, L., Li, M.,Lei, Z., Jean-Sébastien, B., Chen, W., Yue, P., Yong, Y., 2007. SOR: A Practical System for Ontology Storage, Reasoning. *In VLDB 2007, 33rd Very Large Data Bases Conference* ,pp 1402-1405.

Johannesson, P., 1994. A Method for Transforming Relational Schemas into Conceptual Schemas. *In Proceedings of the Tenth International Conference on Data Engineering IEEE Computer Society* Washington, DC, USA, pp 190-201.

Kashyap, V., 1999. Design and creation of ontologies for environmental information retrieval. *In Proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management.* Alberta, Canada. pp. 3–21.

Pierra, G. Dehainsala, H. Aït-Ameur, Y. Bellatreche, L. 2005. Base de données à base ontologique: principes et mise en œuvre. *Ingénierie des Systèmes d'Information,* 10(2):91-115.

Pierra, G., Dehainsala, H. Aït-Ameur, Y., Bellatreche, L. Chochon, J and Mimoune, M. (2004). Base de Données à Base Ontologique: le modèle OntoDB. *Proceedings of Base de Données Avancées 20èmes Journées (BDA'04*), 263–286.

Ramanathan, S., Hodges, J., 1997. Extraction of object-oriented structures from existing relational databases. *SIGMOD Jounral* Vol 26 number 1, pp 59-64.

Rene Robin, C.R., 2010. A Novel Algorithm for Fully Automated Ontology Merging Using Hybrid Strategy. *European Journal of Scientific Research.* Vol.47 No.1, PP. 074-081.

Seaborne, A., Prud'hommeaux, E. Sparql query language for rdf. http: // www. w3. org/ TR/ rdf-sparql-query/

Wilkinson, K., Sayers C., Kuno, H., Reynolds, D. 2003. Efficient RDF storage and Retrtieval in Jena2. *Proceedings of the 1st International Workshop on Semantic Web Database (SWDB'03*). pp. 131–150.

Zdenka, T., 2010. Relational Database as a Source of Ontology Creation. *Proc, of the International Multiconference on Computer Science and Information Technology*. 135.139.