TRANS: Schema-Aware Mapping of OWL Ontologies into Relational Databases

Saurabh Kejriwal N. S. Narayanaswamy

Indian Institute of Technology, Madras
Department of Computer Science and Engineering
Chennai, India
saurabh@cse.iitm.ac.in swamy@cse.iitm.ac.in

Abstract

An ontology is an explicit specification of shared conceptualization. The Web Ontology Language (OWL) is a family of knowledge representation languages for authoring ontologies, and is endorsed by the World Wide Web Consortium. The main operation on ontologies is to query them and to store them so as to answer queries efficiently. By mapping ontologies to relational databases, we can leverage the power of SQL for ontology reasoning over millions of instances. We address the problem of efficiently mapping and querying OWL ontologies using relational databases. We present an OWL ontology mapping tool, called TRANS which can automatically map OWL ontology to mapped relational schema. Using TRANS, we answer both extensional queries and some intentional queries. We compare our approach with schema-aware OWL-mapping tool, Genea[7].

1 Introduction

Nature of the semantic web is more inclined towards instance specific queries. To deal with massive instance data set is the prerequisite of semantic web. RDBMS stands as a first choice where massive data storage and retrieval are key factors of performance. The ontology is vital for creation of the semantic web. Mapping between the ontologies and relational databases can be a good step towards realization of semantic web.

Mapping OWL ontology into relational databases gives the power of SQL for ontology reasoning task such as subsumption, equivalence, consistency, instantiation. OWL mapping can be classfied into schema oblivious mapping and schema-aware mapping. Schema oblivious mapping is a mapping at the syntax level. It does not capture the underlying semantics of the ontology. On the other side, schema-aware mapping maps the ontology considering the semantics of the ontology. Common strategy in schema-aware mapping of OWL ontologies into relational databases is to create separate relation for each ontology class and property. Separate relation generation leads to large number of relations in the relational databases which subsequently leads to more number of joins for complex queries involving many predicates. We propose a schema-aware representation: Our goal is to map an ontology into a relational database in such a way that no of generated relations for mapping should be less, similar to schema-oblivious mapping and query response time should be good as well.TRANS focuses on extensional queries. TRANS also answers few intentional queries. For example, equivalent classes, least common ancestor, consistency checking for disjoint classes etc.

Our main contributions are

1. We propose a set of mapping rules for mapping OWL[13] ontologies into relational database schema.
2. A transitive chain creation and level normalization algorithm for preprocessing ontology schema.
3. A penta tuple representation for RDF triples and clustering of instances for giving a unique class to each individual.
4. An experimental evaluation of the TRANS as compared to schema-aware mapping tool Genea.
5. A set of intentional queries which TRANS can answer.

1.1 Background

1.1.1 Languages of Semantic Web

Semantic web[6] is not a separate web but an extension of current web in which information is given well defined meaning, better enabling computer and human to work in cooperation.

15th International Conference on Management of Data COMAD 2009, Mysore, India, December 9–12, 2009

RDF Resource Description Framework (RDF)[15] is a family of World Wide Web Consortium (W3C) specifications originally designed as a metadata model, subsequently has come to be used as a general method of modeling information, through a variety of syntax formats.The underlying structure of any expression in RDF is a collection of triples. An RDF triple contains three components:

- the subject, which is an RDF URI reference or a blank node
- the predicate, which is an RDF URI reference
- the object, which is an RDF URI reference, a literal or a blank node

An RDF graph is set of such RDF triples.
Example

```
<rdf:RDF xmlns:rdf="http://www.w3.org/
1999/02/22-rdf-syntax-ns#"
xmlns:s="http://description.org/schema/">
<rdf:Description about="http://www.w3.org/
Home/Anuj">
  <s:Creator>Anuj</s:Creator>
</rdf:Description>
</rdf:RDF>
```

The above RDF construct describes "Anuj is the creator of the resource

```
http://www.w3.org/Home/Anuj".
```

RDFS RDF Schema[14], is a semantic extension of RDF. It provides following additional features.

- Classes and subclasses
    1. rdfs:Class: It allows to declare a resource as a class for other resources.
    2. rdfs:subClassOf: It allows to declare hierarchies of classes.
- Property domain and range
    1. rdfs:domain: rdfs:domain of an rdf:property declares the class of the subject in a triple using this property as predicate.
    2. rdfs:range: rdfs:range of an rdf:property declares the class or datatype of the object in a triple using this property as predicate.

OWL OWL[13] is a superset of RDF Schema and adds following more vocabulary. Additional OWL constructs are sameAs,differentFrom, disjointWith, complementOf, hasValue, equivalentProperty, equivalentClass, inverseOf, FunctionalProperty, InverseFunctionalProperty, TransitiveProperty, SymmetricProperty, someValuesFrom, allValuesFrom, min/max Cardinality, oneOf, unionOf and intersectionOf. Following are the three flavors of OWL.
OWL Lite OWL Lite uses only some of the OWL language features and has more limitations on the use of the features than OWL DL or OWL Full. For example, in OWL Lite classes can only be defined in terms of named superclasses (superclasses can not be arbitrary expressions)
OWL DL (Description logics[22]) OWL DL includes all OWL language constructs with certain restrictions. Example: a class may be a subclass of many classes, a class can not be an instance of another class. OWL DL supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time).OWL DL is so named due to its correspondence with description[22] background , a field of research that has studied the logics that form the formal foundation of OWL.
OWL FULL OWL Full contains all the OWL language constructs and provides free, unconstrained use of RDF constructs.

1.1.2 Ontology Reasoning Services

Ontology reasoning service can be classified into following types.

1. Consistency Checking: It checks for consistency of concepts. For example an instance can not be member of both VegPizza and NonvegPizza class if both VegPizza and NonvegPizza classes are disjoint.

2. Subsumption Checking: Subsumption reasoning checks for the superclass and subclass reasoning questions. For example an instance which is member of Professor class will be member of Human class also if Professor is defined as subclass of Human class.

3. Equivalence Checking: It answers the question whether concept A and concept B are equivalent. Equivalence can be of two types. One is both concept A and B are equivalent extensionally. This implies that they both have the same set of instances. Other one is both A and B are equivalent intentionally.

4. Instantiation Checking: Instantiation asks the questions about instance membership. For example: Does an instance I belongs to a particular class C?

5. Satisfiability Checking: Satisfiability reasoning checks whether any arbitrary concept expression can have an instance or not. In other words Is there any instance that satisfies that concept expression?

1.1.3 Types of Inference Strategy

There are two types of inference strategies for ontology reasoners.

1. Load time inference: In load time inference, we compute and store the closure of a given graph as a basis for querying. Genea[7] is an example of Load time inference reasoner.
2. Query time inference: Query time inference strategy allows the query processor infer new statements as needed per query. DLDB[9] and Seasame[2] use query time inference as their ontology reasoning strategy. TRANS is also a query time inference reasoner.

1.2 Related Work

Related work in ontology mapping can be classified into two types.
Schema-oblivious mapping Seasame[2] is a schema oblivious approach for mapping RDF[15] AND RDF Schema[14]. Seasame design is independent of any specific storage devices. Underlying structure of Seasame schema is of triple format[1, 11]. Seasame inserts inferred triples in the database table at the load time itself. This feature significantly increases the loading time for RDF and RDFS. Seasame schema is ontology independent means the structure of schema is fixed irrespective of ontology.

Schema-aware mapping DLDB[9] is a knowledge based system, proposes a new schema-aware mapping approach for mapping OWL ontologies. It extends RDBMS with additional capability for DAML+OIL[18] inference. DLDB uses FACT description logic reasoner for computing OWL subsumption inference. It uses RDBMS views for storing class and property subsumption results. They have done light weight implementation of their approach using MS-Access and FACT[4] reasoner. DLDB presents the scaling performance of their system with increase in number of instances in the RDBMS and proves that idea of extending relational databases using description logic is feasible.

The most closely related to our work is Genea[7]. Genea[7] is a schema-aware mapping approach for mapping OWL ontologies into relational databases. It maps the basis of set of mapping rules. Genea maps each simple named classes to a separate relation and individuals to rows in (one or more) relations. Similarly for simple data and object property a separate relation is created in relational database. Main differences between Genea and TRANS are as follows: Genea uses materialization technique at load time for OWL subsumption inference, while TRANS uses query time inference for OWL subsumption inference. Genea creates a separate relation for simple named class and property. On the other side, TRANS creates a separate relation for each directed acyclic graph in ontology concept hierarchy.

1.3 The Body of The Paper

The remainder of paper is organized as follows. Section 2 describes the architecture of TRANS. In section 3, we describe TRANS mapped schema for LUBM University ontology fragment. Section 4 shows experimental setup. The results of an experimental evaluation of our mapping rules compared to Genea are presented in the section 5 and 6. Section 7 list of some intentional queries which TRANS can answer. The next section concludes.

2 Architecture of TRANS

Figure 1 depicts the components involved for the complete mapping process. The architecture description is as follows:

Definition 1. Let { D_1, D_2, D_1 ... D_n } is a set of disjoint directed acyclic graph extracted out of concept hierarchy of an ontology O, D_{ir} is a root node of the D_i and D_{il} is a leaf node of the D_i then a **transitive chain** is a path of nodes < D_{ir}, p_1, p_2,... p_n, D_{il} > between D_{ir} and D_{il} including both D_{ir} and D_{il}. Here p_1, p_2,... p_n are intermediate nodes between D_{ir} and D_{il}.

Definition 2. Let T_i is a transitive chain and { N_r, N_{r+1},.... N_l } is the set of nodes in the transitive chain Ti. **Leveling** is the process of numbering the nodes considering the constraint that for each node N_i level Number of N_i should be less than level number of N_{i+1}.

Definition 3. Let { T_1, T_2, T_1 ... T_n } is set of transitive chains of disjoint directed acyclic graph D_i(as mentioned in Definition 1) and D_{in} is a node in the D_i. If D_{in} passes through the r transitive chains < T_j, T_{j+1}, ... T_{j+r} > with levels value < l_j, l_{j+1}, ... l_{j+r} > then **normalization** is the process of giving Max(l_j, l_{j+1}, ... l_{j+r}) to node D_{in}. Here Max represents maximum level value of all level numbers in < l_j, l_{j+1}, ... l_{j+r} >.

2.1 TRANS Relation Generator

TRANS uses the Jena[17] parser for parsing the input ontology file. The TRANS relation generator generates the relations out of ontology concept data. Relations are created in relational databases for the ontology through TRANS rules for mapping. Following are the TRANS schema-aware mapping rules.

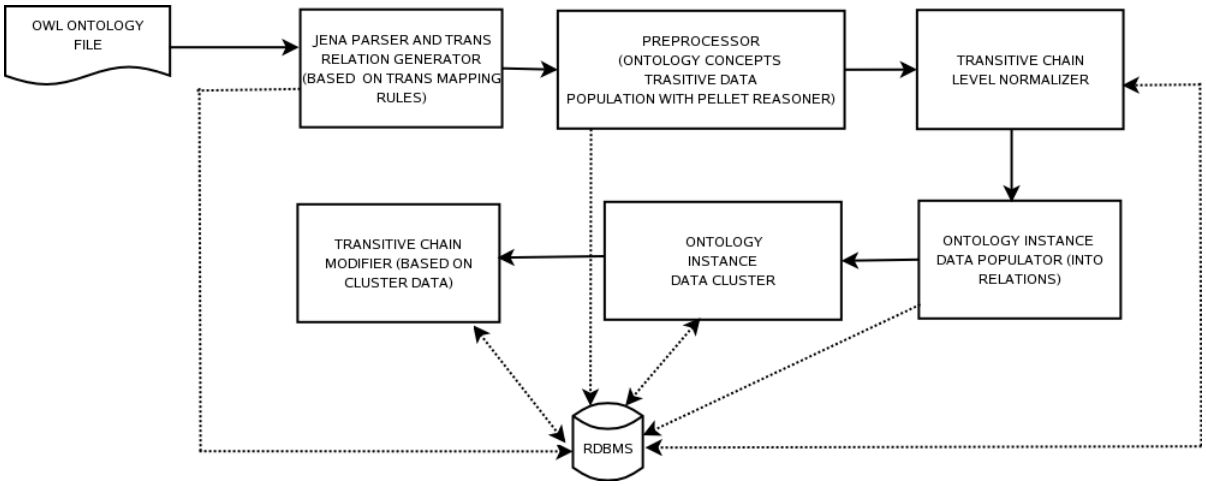

Figure 1: Architecture of TRANS schema aware mapping tool

2.1.1 Schema-Aware Mapping Rules

Overview of schema-aware mappping rules
In TRANS, schema-aware mapping rules are executed by TRANS relation generator component in the TRANS architecture. These rules are not ontology independent. It means number of relations generated changes from ontology to ontology. Here in Rules RM stands for relational model. TRANS supports reasoning at RDFS level(subsumption, instantiation, property domain and range based reasoning). These four kinds of reasoning are supported by schema-aware mapping rules 1, 2, 3 and 5. Apart from this, TRANS also supports some of the OWL intentional queries. Example: class disjointness reasoning. Rule 5 is used for this purpose.

Rule 1:Simple named classes, subsumption

```
RM: Class[CId INT, IndivURI VARCHAR(130)]

PRIMARY KEY [IndivURI]
UNIQUE [CId, IndivURI]

RM: store_level[TransCId INT, CId INT,
    CURI VARCHAR(130), Level INT,
    DisjointId INT]

PRIMARY KEY [TransCID, CId, CURI, Level]
```

TRANS mapps each disjoint directed acyclic graph for ontology class hierarchy into a relation Class. In the Class relation IndivURI is the primary key. Ontology class subsumption information is mapped to `store_level` relation. Here `store_level` relation stores all the transitive chains of the ontology class hierarchy.

Rule 2: Simple object properties and subsumption

```
RM: ObjectProperty [ Domain INT,
    Indiv1URI VARCHAR(130), PropId INT,
    Indiv2URI VARCHAR(100), Range INT ]

 PRIMARY KEY [Indiv1URI, Indiv2URI]
 UNIQUE [Domain, Indiv1URI, PropId,
        Indiv2URI, Range]

RM: store_plevel[ TransPId INT, PId INT,
    PURI VARCHAR(130), Level INT]

PRIMARY KEY [TransPId, PId, PURI, Level]
```

Similar to ontology class hierarchy, TRANS maps each disjoint directed acyclic graph for ontology object property hierarchy into a relation ObjectProperty. In the ObjectProperty relation Indiv1URI and Indiv2URI together make primary key. Ontology object property subsumption information is mapped to `store_plevel` relation. Here `store_plevel` relation stores all the transitive chains of the ontology object property hierarchy.

Rule 3: Simple datatype properties and subsumption

```
RM: DataProperty [Domain INT,
    IndivURI VARCHAR(130), PropId INT,
    Value VARCHAR(160)]

PRIMARY KEY [IndivURI, Value]
UNIQUE [Domain, IndivURI, PropId,
        Value]

RM: store_dplevel[TransDPId INT, PId INT,
    PURI VARCHAR(130), Level INT]

PRIMARY KEY [TransDPId, PId, PURI, Level]
```

Similar to ontology object property hierarchy, TRANS maps each disjoint directed acyclic graph for ontology data property hierarchy into a relation DataProperty. In the DataProperty relation IndivURI and Value together make primary key. Ontology data property subsumption information is mapped to `store_dplevel` relation. Here `store_dplevel` relation stores all the transitive chains of the ontology data property hierarchy.

Rule 4: Class disjointness

```
RM: store_level[TransCId INT, CId INT,
    CURI VARCHAR(130), Level INT,
    DisjointId INT]

UNIQUE [Transcid, Disjointid]
```

Here we check the Disjointness of two classes by DisjointId attributes. If two classes are disjoint then they can not lie in the same transitive chain. Transcid and Disjointid together make primary key.

Rule 5: Property domain and range

```
RM: PropDomain[PId INT, DomainClasses INT]

PRIMARY KEY [PId, DomainClasses]

RM: PropRange[PId INT,  RangeClasses INT]

PRIMARY KEY [PId, RangeClasses]
```

Property Domain and Range information is mapped to PropDomain and PropRange relation respectively. In PropDomain relation, PId and DomainClasses together make primary key. In PropRange relation, PId and RangeClasses together make primary key.

Rule 6: Relations for storing clustering data information

```
RM: Cluser_Class[ClusterCId INT, CId INT]

PRIMARY KEY [ClusterCId INT, CId INT]

RM: Cluster_Property[ClusterPId INT, PId INT]

PRIMARY KEY [ClusterPId INT, PId INT]
```

These relations stores newly created clustered classes and properties out of TRANS cluster operation. In `Cluster_Class` relation, ClusterCId and CId together make primary key. In `Cluster_Property` relation ClusterPId and PId together make primary key.

2.2 Preprocessor

The preprocessor preprocesses the ontology concept hierarchy using the Pellet[10, 12] reasoner. A set of transitive chains is extracted out of ontology concept hierarchy by algorithm 1. The algorithm 1 also assigns a level number to each node in the transitive chain. Here level number of a node in a transitive chain is equal to its distance from the root node + 1. All the extracted transitive chains are populated into the transitive chain relations created by rule 1, 2 and 3. Algorithm 1 describes the transitive chain creation for ontology class hierarchy. Same transitive chain creation is done for the ontology property hierarchy also.

```
input  : O, An OWL ontology file
output: A set of OWL class transitive chains
        and corresponding levels
begin
    for Each root class R ϵ
    RootHierarchyClasses (O) do
        Transid ⟵ 1;
        Transid = Transid + 1;
        Print (Transid);
        Push (R);
        PrintTransChain(R);
end
PrintTransChain (R)
begin
    Leaf ⟵ 0;
    for Each each S ϵ subClassOf (R) do
        Leaf ⟵ 1;
        Push (S);
        PrintTransChain (S);
    if Leaf = 0 then
        PrintStackElements ();
    if StackNotEmpty () then
        Pop ();
end
PrintStackElements ()
begin
    Level ⟵ 1;
    while IterateAllStackElements () = False
    do
        PrintStackElement ();
        Print (Level);
        Level ⟵ Level + 1 ;
end
```

Algorithm 1: OWL Class Trans chain creation algorithm

2.3 Transitive Chain Level Normalizer

The transitive chain level normalizer normalizes the level numbers of nodes in the transitive chains created by algorithm 1. Normalization is the process of giving unique level number to each node. Algorithm 2 describes the transitive chain level normalization pro-

cess for ontology class hierarchy.

```
input  : A set of OWL class transitive chains
output: Normalized set of transitive chain where
        [ClassId(C1) = ClassId(C2)] ⟹
        [level(C1) = level(C2)]
begin
    for Each Class id C ϵ
    OccursMoreThenOneTransChain (C) = True
    do
        UpdateRightClass (C);
end

UpdateRightClass (C)
begin
    LMax ← MaxLevel (C);
    for Each Transitive id T of C do
        Level ← LevelT (T) ;
        Diff ← LMax- Level ;
        for Each Class Id Where GetLevel (C)
        ≥ Level do
            Check ←
            OccursMoreThenOneTransChain (C) ;
            if Check = True then
                UpdateRightClass (C);
            else
                for Each Class Id Where
                GetLevel (C) ≥ Level do
                    Level ← GetLevel (C) + Diff ;
end
```

Algorithm 2: OWL Class Trans Chain Normalization algorithm

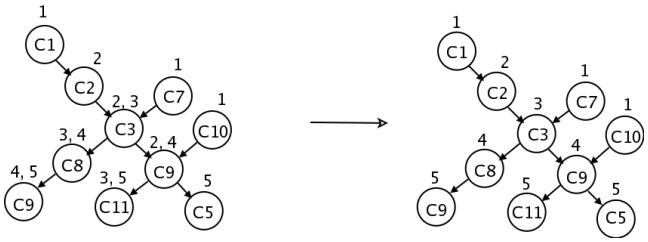

Figure 2: Reflection of Transitive chain creation and normalization algorithm

Figure 2 reflects the transitive chain creation and normalization of a given class hierarchy. Here in figure 2 node labels denote class labels like C1 denotes class 1 and number on top of label denotes level number of the node like class c3 got two level numbers 2 and 3. In figure 2, left side part of the figure depicts the situation of ontology class hierarchy before the normalization operation. In normalization, class labels are numbered considering the constraint that each node should get a unique level number. Right side part of the figure 2 shows the ontology class hierarchy after the normalization process.

2.4 Ontology Instance Data Populator

The ontology instance data populator populates the ontology instance data into corresponding relations. It populates the class instance and property instance data.

2.5 Ontology Instance Data Cluster

Ontology instance data cluster clusters the ontology instance data in such a way that each instance should have a unique class. Similar is done for the triples also. For example if I1 and I2 are two instances connected with property P1 and P2 then these two triples will be clustered into a single triple and a new property id is generated. This way ontology instance data cluster generates new classes and properties due to clustering for both ontology classes and properties. Population of these newly generated classes and properties into the transitive chain relations is dealt by next component.

2.6 Transitive Chain Modifier

Transitive chain modifier is the last component in the TRANS architecture. Transitive chain modifier populates newly generated classes and properties due to clustering into the transitive chain relations.

3 Ontology Mapping Example

Following is the fragment of LUBM University ontology. LUBM is the Lehigh University benchmark ontology for university domain. We add one object property profTrainingFrom in this fragment.

3.1 LUBM University Ontology Fragment

```
<owl:Class rdf:ID="AssistantProfessor">
 <rdfs:label>assistant professor</rdfs:label>
 <rdfs:subClassOf rdf:resource="#Professor"/>
</owl:Class>

<owl:Class rdf:ID="AssociateProfessor">
 <rdfs:label>associate professor</rdfs:label>
 <rdfs:subClassOf rdf:resource="#Professor"/>
</owl:Class>

<owl:Class rdf:ID="Faculty">
 <rdfs:label>faculty member</rdfs:label>
 <rdfs:subClassOf rdf:resource="#Employee"/>
</owl:Class>

<owl:Class rdf:ID="Lecturer">
 <rdfs:label>lecturer</rdfs:label>
 <rdfs:subClassOf rdf:resource="#Faculty"/>
</owl:Class>
```

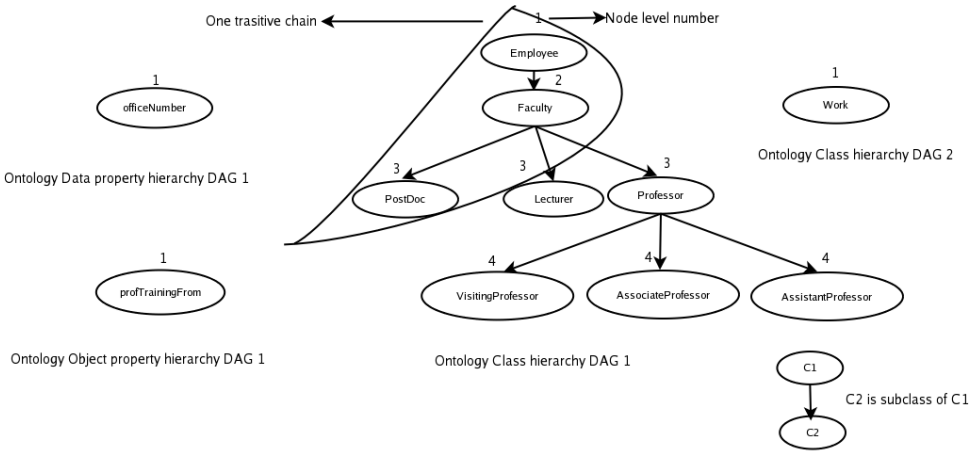

Figure 3: TRANS transitive chains for LUBM University ontology fragment

```
<owl:Class rdf:ID="PostDoc">
 <rdfs:label>post doctorate</rdfs:label>
 <rdfs:subClassOf rdf:resource="#Faculty"/>
</owl:Class>

<owl:Class rdf:ID="Professor">
 <rdfs:label>professor</rdfs:label>
 <rdfs:subClassOf rdf:resource="#Faculty"/>
</owl:Class>

<owl:Class rdf:ID="VisitingProfessor">
 <rdfs:label>visiting professor</rdfs:label>
 <rdfs:subClassOf rdf:resource="#Professor"/>
</owl:Class>

<owl:Class rdf:ID="Work">
  <rdfs:label>Work</rdfs:label>
</owl:Class>

<owl:ObjectProperty rdf:ID="profTrainingFrom">
 <rdfs:label>has an Professional Training from
 </rdfs:label>
 <rdfs:domain rdf:resource="#Lecturer" />
 <rdfs:range rdf:resource="#University"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="officeNumber">
 <rdfs:label>office room No.</rdfs:label>
</owl:DatatypeProperty>
```

3.2 Transitive Chain Creation for LUBM University Ontology Fragment

Figure 3 depicts the transitive chain creation for example LUBM University ontology fragment. Here `{Employee, Faculty, PostDoc}` is one transitive chain with node levels 1, 2 and 3 respectively. Here Employee is the superclass of the class Faculty.

3.3 TRANS Mapped Schema

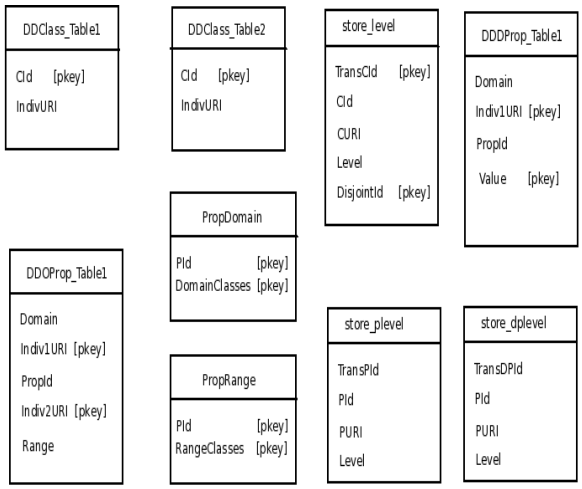

Figure 4: TRANS mapped schema for LUBM University ontology fragment

Following is the description of relations in the figure 4. These relations are generated as per TRANS schema-aware mapping rules.

1. `DDClass_Table1`: It stands for Disjoint Directed acyclic class graph table1. It stores all the asserted individuals of AssistantProfessor, AssociateProfessor, Faculty, Lecturer, Postdoc, Professor and VisitingProfessor class.

2. `DDClass_Table2`: It stands for Disjoint Directed acyclic class graph table2. It stores all the asserted individuals of work class.

3. `DDOProp_Table1:` It stands for Disjoint Directed acyclic object property graph table1. It stores property instances of object property ProfTrainingFrom.

4. `DDDProp_Table1:` It stands for Disjoint Directed acyclic datatype property graph table1. It stores property instances of datatype property officeNumber.

5. `PropDomain:` It stands for domain of properties. It stores domain information of ProfTraningFrom property.

6. `PropRange:` It stands for range of properties. It stores range information of ProfTrainingFrom property.

7. `store_level:` It stores the transitive chains and levels of the university ontology class hierarchy.

8. `store_plevel:` It stores the transitive chains and levels of the university ontology object property hierarchy.

9. `store_dplevel:` It stores the transitive chains and levels of the university ontology data property hierarchy.

4 Experimental Setup

Following is experimental setup.

Processor: 2.6 GHz Intel Celeron Processor
RAM: 1 GByte RAM
Hard disk: 80 GByte
Operating System: Fedora Core 3
Database: PostgreSQL 8.1
JDBC Driver: JDBC driver 3
Java: Sun JDK 1.5.0

TRANS is compared with Genea schema aware mapping tool as Genea performs better than the other available schema aware mapping tools in the literature. Both TRANS and Genea tool are experimented on the above experimental setup. We use the data generator tool UBA version 1.7[16] with seed value 0 to generate LUBM University ontology random individual data. For comparing TRANS with Genea we have built eleven LUBM test sets (LUBM 1, LUBM 3, LUBM 5, LUBM 7, LUBM 10, LUBM 15, LUBM 20, LUBM 25, LUBM 30, LUBM 40, LUBM 50) that correspond to size of 10 to 500 MBytes.

Genea Genea was implemented in MySQL[19] database and TRANS is implemented in PostgreSQL[20]. So to have a fair comparison, we obtained Genea source code from Genea author and customized some part of it to make it work with PostgreSQL. We use the SQL script of Genea specified in their source code for ontology schema creation. Genea is also experimented on the above experimental setup.

Query no.	Input	Selectivity	Description from LUBM Benchmark and Genea paper
1	large	high	Queries about just one class and one property and does not assume any hierarchy information or inference.
3	large	high	Similar to Query 1 but class Publication has a wide hierarchy.
4	small	high	It assumes subClassOf relationship between Professor and its subclasses. Class Professor has a wide hierarchy. Another feature is that it queries about multiple properties of a single class.
6	large	low	This query queries about only one class. But it assumes both the explicit subClassOf relationship between UndergraduateStudent and Student and implicit one between GraduateStudent and Student.
7	large	high	This query is similar to Query 6 in terms of class Student but it increases in the number of classes and properties.
11	small	low to high depending on the used dataset	Inference about the subOrgnizationOf relationship between instances of ResearchGroup and University is required to answer this query.
14	large	low	This query is the simplest in the test set, as it relies only on one triple.

Table 1: Selected LUBM queries for TRANS and Genea

TRANS TRANS uses Pellet[10, 12] reasoner for computing ontology hierarchy and Jena[17] reasoner for ontology data loading as Jena is not able to return some of the asserted conceptual data. Query response time is calculated based on Genea specified SQL queries for LUBM test queries.In TRANS, for having a more fair comparison query response time is calculated without the cache effect. For flushing the cache effect, we restarted our system before every new LUBM[3] test query reading for both TRANS and Genea.

Table 1 lists the set of selected LUBM test queries for both TRANS and Genea. In table 1, Query no. column specifies the selected LUBM query number, Input column specifies the size of ontology input data for a query, Selectivity column depicts the estimated proportion of the class instances involved in the query that satisfy the query criteria and Description column specifies about the nature of each queries. Following are the reasons for query selection. We got Genea SQL queries and code from Genea author. Genea experimented on the set of queries in table 1 so to have a fair comparison TRANS also tested on the same set of LUBM test queries. Another reason is TRANS currently does not directly support SPARQL[23] and RQL[8] ontology languages. LUBM test queries in SPARQL are mapped to corresponding SQL queries. Queries in table 1 operates directly on database and easy to be converted into SQL for fair comparison.

5 Query and Schema Complexity Comparison

Query complexity comparison is done on the basis of number of select, join and union clauses used in the

Query	*Joins*	*Unions*	*Selects*
1	1	0	1
3	0	0	1
4	3	0	1
6	1	0	1
7	2	0	1
11	1	0	1
14	1	0	1

Table 2: Query complexity of TRANS mapped queries for selected LUBM test queries

Query	*Joins*	*Unions*	*Selects*
1	2	0	1
3	2	0	1
4	5	0	1
6	1	0	1
7	4	0	1
11	3	0	1
14	1	0	1

Table 3: Query complexity of Genea mapped queries for selected LUBM test queries

SQL query. Select, Join and Union are important factors for calculation of SQL query complexity. Query complexity table 2 for TRANS and 3 for Genea shows that TRANS needs less number of joins in compare to Genea.

5.1 Comparison of Query Complexity of TRANS and Genea

As specified in the experimental setup, query complexity comparison is done on the queries listed in table 1. Table 1 specifies input size and selectivity factors for query complexity which are also important factors similar to select, join and union factors for query complexities.

Table 2 Query column specifies the LUBM test queries. Apart from that joins, unions and selects columns are used to measure SQL query complexity of TRANS mapped SQL queries for LUBM test queries. These test queries are the same queries on which Genea author evaluated Genea. Similar to TRANS query complexity table, table 3 specifies the query complexity table for Genea. Both TRANS and Genea are evaluated on the same set of LUBM test queries. For Union and select clause both Genea and TRANS have the same query complexity but for joins Genea has more query complexity. TRANS has less query complexity for query 1, 3, 4, 7 and 11. For query 6 and 14 both TRANS and Genea have the same query complexity. Query complexity table 2 for TRANS and 3 for Genea shows that TRANS needs less number of joins in compare to Genea.

Complexity parameters	TRANS	Genea
No of tables	39	76
No of foreign keys	0	109
No of views	0	0
No of indexes	217	141

Table 4: Schema complexity for LUBM ontology

5.2 Comparision of Schema Complexity of TRANS and Genea

Table 4 specifies the schema complexity for TRANS and Genea schema-aware mapping tool for LUBM ontology. As per table 4, TRANS needs less number of tables in compare to Genea. Need of less number of tables makes TRANS more scalable in compare to Genea for larger ontologies. Genea stores ontology semantic constraints in terms of foreign keys so it needs foreign keys listed in table 4.

6 Query Performance Comparison

In the query response curves, x-axis specifies the LUBM benchmark factor(LUBM(N,S)). For example benchmark factor 5 specifies LUBM(5,0) as we uses seed value 0, specified in experimental setup. Y-axis specifies the query response time in milliseconds. We have applied least square linear curve fitting[21] over the query performance curves. A least square linear fit minimizes the square of the distance between every data point and the line of the best fit.

6.1 Query1

Query 1 requests for all Graduate students who takes a given course. It does not assume any hierarchy information or inference. The result of comparison is shown in Figure 5.

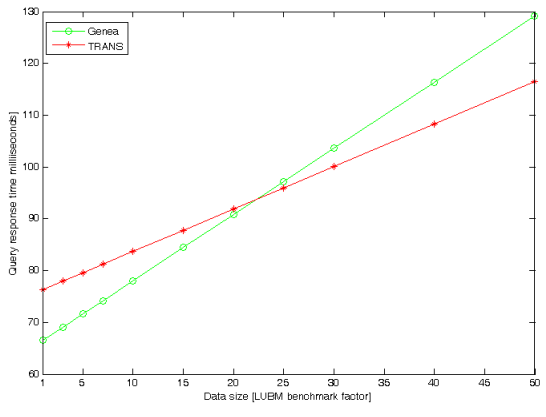

Figure 5: Query response time of LUBM Query 1 with curve fitting

6.2 Query3

Query 3 requests for all the publications of a given professor where the class publication has wide hierarchy. It bears large input and high selectivity. The result of comparison is shown in Figure 6.

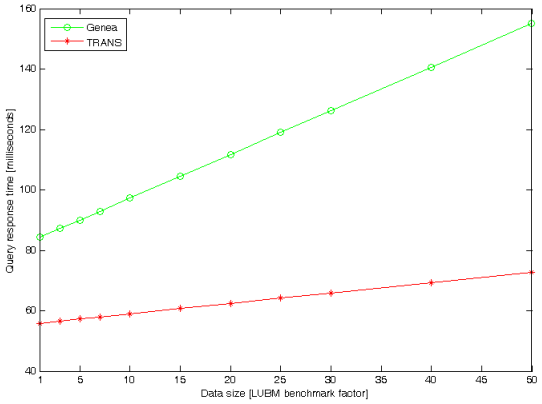

Figure 6: Query response time of LUBM Query 3 with curve fitting

6.3 Query4

Query 4 demands name, emailaddress, telephone of all the professors who work for a given department in University where class professor has wide hierarchy. It increases query complexity by requesting multiple properties of a professor class and has small input and high selectivity. Figure 7 depicts the result of comparison.

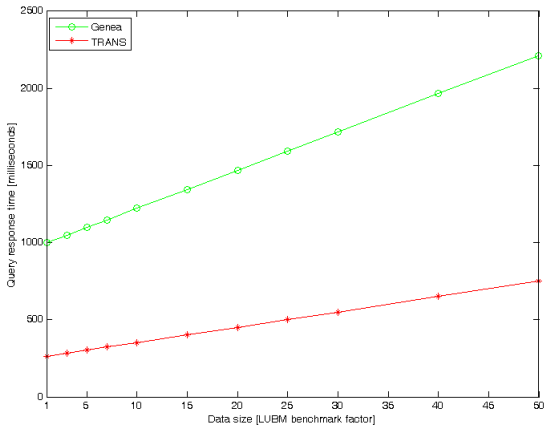

Figure 7: Query response time of LUBM Query 4 with curve fitting

6.4 Query6

Query 6 selects all individuals who are student assuming both implicit and explicit subclassof relationship. It bears large input and low selectivity and produce a big set of results. The result of comparison is shown in Figure 8.

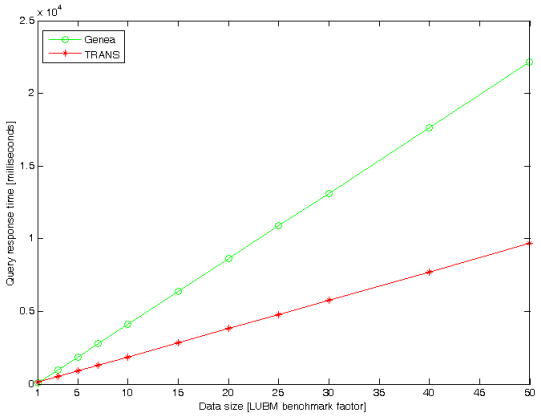

Figure 8: Query response time of LUBM Query 6 with curve fitting

6.5 Query7

Query 7 selects students and courses taken by them, considering that course is offered by a given professor. It is similar to Query 6 in terms of student but with more properties and classes and its selectivity is high. The result of comparison is shown in Figure 9.

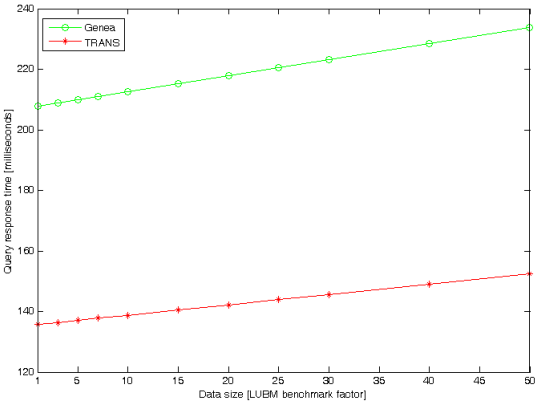

Figure 9: Query response time of LUBM Query 7 with curve fitting

6.6 Query11

Query 11 requests for all research group who are suborganization of given one. The result of comparison is shown in Figure 10.

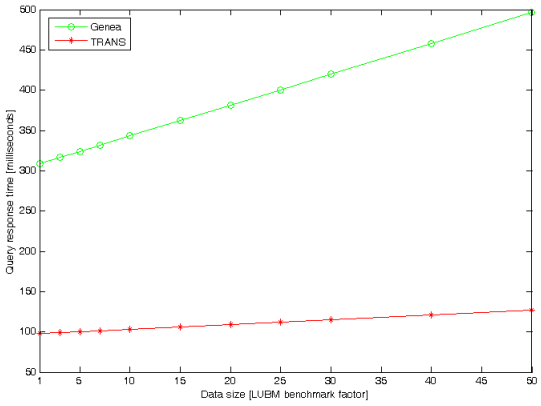

Figure 10: Query response time of LUBM Query 11 with curve fitting

6.7 Query14

Query 14 requests all individuals who are undergraduate student. It bears large input and low selectivity. The result of comparison is shown in Figure 11.

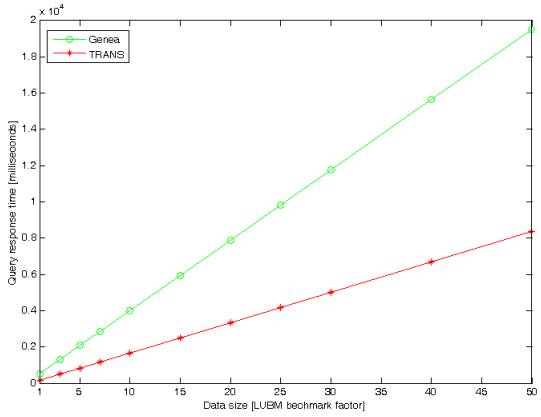

Figure 11: Query response time of LUBM Query 14 with curve fitting

7 Intentional Queries Which TRANS Can Answer

Q1:Subclasses-Superclasses Q1 intensional[5] query demands set of all the subclasses and superclasses of a class. For example in LUBM benchmark University ontology subclasses of class person should give both asserted classes like Student etc and inferred classes like GraduateStudent etc.
Example Give all the subclasses of a class named exampleclass.

```
select CURI from store_level where Level >=
(select Level from store_level where curi =
'exampleclass') and transcid IN (select
transcid from store_level where curi =
'exampleclass')
```

Similarly for superclasses query also, it enumerate the set of all ancestors of a class.
Example Give all the superclasses of a class named exampleclass.

```
select CURI from store_level where Level <=
(select Level from store_level where curi =
'exampleclass') and transcid IN (select
transcid from store_level where curi =
'exampleclass')
```

TRANS answers the successor and ancestor set for object and data properties also.

Q2:Reachability It asks the question "Does the UndergraduateStudent belongs to Person in the LUBM University ontology?". Here the distance between UndergraduateStudent and Person class does not matter. Q2 requests whether there is any superclass of UndergraduateStudent which is labeled as Person.
Example Does the exampleclass1 belongs to exampleclass2.

```
select CURI from store_level where Level <=
(select Level from store_level where curi =
'exampleclass1') and transcid IN (select
transcid from store_level where curi =
'exampleclass1') and CURI='exampleclass2'
```

If above query returns some value then exampleclass1 is reachable to exampleclass2.

Q3:Least common ancestor It queries least common ancestor for set of concepts.
Example What is the least common ancestor of exampleclass1 and exampleclass2.

```
select t.CURI from
(select CURI, Level from store_level where
Level <= (select MIN(Level) from store_level
where curi = 'exampleclass1' or
'exampleclass2')and transcid IN (select
transcid from store_level where curi =
'exampleclass1' or 'exampleclass2') ) t,

(select CURI, Level from store_level where
Level <= (select MIN(Level) from store_level
where curi = 'exampleclass1' or
'exampleclass2') and transcid IN (select
transcid from store_level where curi =
'exampleclass1' or 'exampleclass2') ) t1
where t.Level = Min(t1.Level)
```

8 Conclusions and Future Work

TRANS is a new schema aware approach for mapping OWL ontologies in relational databases. It has succeeded in performing better in both query response time and space used to store the ontology instance data compared to Genea. Unlike the other schema aware approach TRANS does not map each class or property to a separate table which would not lead to exponential increase in number of tables for an ontology of too many classes. Although TRANS does some preprocessing over ontology hierarchy for computing transitive chain but preprocessing should be one time as structure of ontology usually does not change. TRANS currently supports OWL subsumption reasoning, instantiation, some consistency checking and also some intentional queries which is not supported by Genea.

One direction for future work on TRANS would be to extends its reasoning capability to support full consistency checking. Another direction is to provide direct support for SPARQL[23] queries. Currently TRANS provides indirect support for SPARQL queries.

References

[1] S. Alexaki, V. Christophides, G. Karvounarakis, and D. Plexousakis. On storing voluminous RDF descriptions. *In Proceedings of the 4th International Workshop on the Web and Databases (WebDB)*, pages 43–48, 2001.

[2] J. Broekstra, A. Kampman, and F. van Harmelen. Seasame: A generic architecture for storing and querying RDF and RDF schema. *International Semantic Web Conference (ISWC)*, 2342:54–68, 2002.

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

[4] Ian Horrocks. The FACT system. *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 307–312, 1998.

[5] Ian Horrocks. Ontology reasoning:Why and How. *URL:www.comlab.ox.ac.uk/people/ian.horrocks/../DL-ontology-langs.ppt*.

[6] Ian Horrocks Introduction to Semantic Web. *www.cs.man.ac.uk/ horrocks/Teaching/cs646/*.

[7] T. Kraska and U. Rohm. Genea: Schema-aware mapping of ontologies into relational databases. *International Conference on Management of Data (COMAD)*, pages 103–114, 2006.

[8] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis and M. Scholl RQL: A Declarative Query Language for RDF. *Eleventh International World Wide Web Conference (WWW)*, pages 592–603, 2002.

[9] Z. Pan and J. Heflin. DLDB: Extending relational databases to support semantic web queries. *Workshop on Practical and scalable Semantic Systems (PSSS1)*, pages 109–113, 2003.

[10] B. Parsia and E. Sirin. Pellet: An OWL DL reasoner. *Poster in International Semantic Web Conference (ISWC)*, 2004.

[11] A. Reggiori. RDFStore: Perl/C rdf storage and api. *URL:http://rdfstore.sourceforge.net*, 2004.

[12] The Pellet Team. Pellet - an OWL DL reasoner. *URL:http://clarkparsia.com/pellet*

[13] W3C. The web ontology language (OWL). *URL:http://www.w3.org/2004/OWL/*.

[14] W3C. RDF vocabulary description language 1.0: Resource description framework Schema. *URL:http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/*.

[15] W3C. RDF concepts and abstract syntax. *URL:http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/*.

[16] The Semantic Web and Agent Technologies Lab(SWAT). LUBM benchmark. *URL: http://swat.cse.lehigh.edu/projects/lubm/*.

[17] The Jena Team. Jena - a semantic web framework for java. *URL:http://www.jena.sourceforge.net/*.

[18] W3C. DAML+OIL(2001) reference description. *URL:http://www.w3.org/TR/daml+oil+reference*.

[19] The MySQL Team. MySQL - an open source database. *URL:http://dev.mysql.com/doc/*.

[20] The PostgreSQL Team. PostgreSQL database. *URL:http://www.postgresql.org/docs/*.

[21] The Wolfrom Team. Least square linear curve fitting. *URL:http://mathworld.wolfram.com /LeastSquaresFitting.html*.

[22] Description Logics. The Course Slides of DL. *URL:http://www.inf.unibz.it/franconi/dl/course*.

[23] W3C. The SPARQL Query Language for RDF. *URL:www.w3.org/TR/rdf-sparql-query/*.