ONTOLOGY DRIVEN QUERY REWRITING USING KNOWLEDGE BASE

A PROJECT REPORT

Submitted by

MUTHUKUMAR G

in partial fulfillment for the requirement of award of the degree of

MASTER OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

Department of Computer Science and Engineering
KUMARAGURU COLLEGE OF TECHNOLOGY,
COIMBATORE 641 049
(An Autonomous Institution Affiliated to Anna University, Chennai)

APRIL 2013

BONAFIDE CERTIFICATE

Certified that this project work titled **"ONTOLOGY DRIVEN QUERY REWRITING USING KNOWLEDGE BASE"** is the bonafide work of Mr. MUTHUKUMAR G. who carried out the research under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other students.

Prof.N.Jayapathi M.Tech.,	Ms.V.P.Sumathi M.E.,
HEAD OF THE DEPARTMENT	SUPERVISOR
Professor	Assistant Professor[SRG]
Computer Science and Engineering	Computer Science and Engineering
Kumaraguru College of Technology	Kumaraguru College of Technology
Coimbatore – 641 049	Coimbatore – 641 049

Submitted for the Project Viva-Voce examination held on _____

---------- ----------

Internal Examiner External Examiner

ABSTRACT

This work investigates the extent to which domain knowledge, expressed in a domain ontology, can assist end-users in formulating relational queries from SPARQL query which is given ontology language that can be executed over a complex relational database. In this regard, an ontology-driven query formulation architectural framework has been devised that implements a three-phased approach – the pre-processing and translation phases and constructing knowledge base. In the preprocessing phase, a new database-to-ontology transformation approach has been synthesized where the domain ontology is populated and enriched with problem domain concepts and semantic relationships specified using OWL-DL. Once the domain ontology has been formulated, end-users can write sophisticated SPARQL query that are then translated, into a relational query using knowledge base, in which knowledge is stored using the triple format. In the translation phase, a SPARQL query will be rewritten into corresponding local relational query statements. The existing SPARQL query syntax cannot support aggregate functions, like count, group by, order by, min, max etc. In this proposed work description is added to SPARQL queries. This description is used while rewriting the SPARQL queries to SQL queries. This project work shows how the SPARQL queries are going to rewrite into SQL queries using the knowledge base. This result shows the recall rate of the query result with rewriting and without rewriting.

ஆய்வுச்சுருக்கம்

இந்த வேலை கொடுக்கப்பட்ட கள உள்ளியத்தில் உள்ள கள அறிவை கொண்டு ஸ்பர்ால் கேள்வியில் இருந்து தொடர்புடைய தரவுதள கேள்விகளாக உருமாற்றம் செய்யப்பட்டு சிக்கலான தொடர்புடைய தரவுதளங்களின் மேல் செயல்படுத்தப்படுவதற்கு இறுதி பயனர்க்கு உதவி புரிகிறது. இது சம்பந்தமாக உள்ளியம் உந்துதல் கேள்வி உருவாக்க கட்டமைப்பானது முன் செயலாக்கம், மொழி பெயர்ப்பு, கட்டுமான அறிவு அடிப்படை ஆகிய மூன்று பகுதிகளை உள்ளடக்கியது. முன் யலாயக்கத்தில் ஒரு புதிய தரவுதள முதல் உள்ளியம் மாற்ற அணுகுமுறை தொகுக்கப்படுகிறது. இதில் கள உள்ளியமானது உள்ளிய வலையமைப்பு மொழியில் கொடுக்கப்பட்டுள்ள களம்பற்றிய கருத்துக்கள் மற்றும் குறிப்பிட்ட சொற்பொருள் உறவுகளை கொண்டும் உருவாக்கப்படுகிறது. கள உள்ளியமானது முறைப்படுத்தப்பட்டுவிட்டால் இறுதி பயனர்கள் அறிவு தளத்தை பயன்படுத்தி ஸ்பர்ால் கேள்விகளாக எளிதில் உருமாற்றலாம். இதில் அறிவு தளமானது மூன்று வடிவத்தில் சேமிக்கப்படும். மொழி பெயர்ப்பு கட்டமைப்பில் கேள்வியானது உள்ளூர் தொடர்புடைய வினவல் அறிக்கையாக மாற்றப்பட வேண்டும். ஸ்பார்ால் கேள்வியானது தொடரியல் எண்ணிக்கை செயல்பாடுகளை ஆதரிக்காது. முன் மொழியப்பட்ட அணுகுமுறையானது ஸ்பார்குல் கேள்விகளுக்கு விளக்கம் சேர்ப்பதன் மூலம் நாம் தொடரியல் எண்ணிக்கை சம்பந்தபட்ட செயல்பாடுகளை செய்யமுடியும். இந்த விளக்கமானது கேள்விகளை உருமாற்றும் போது பயன்படும். இந்த அணுகுமுறையில் ஸ்பார்கூல் கேள்வியானது அறிவு தளத்தை பயன்படுத்தி எவ்வாறு கட்டமைப்பு வினவல் மொழியாக உருமாற்றம் அடைகிறது என்பதை காட்டுகிறது.

ACKNOWLEDGEMENT

First and foremost, I would like to thank the Lord Almighty for enabling me to complete this project.

I express my profound gratitude to **Padmabhusan Arutselvar Dr.N.Mahalingam, B.Sc., F.I.E., Chairman**, **Dr.B.K. Krishnaraj Vanavarayar, B.Com., B.L., Co-Chairman, Mr. M. Balasubramaniam, M.Com., M.B.A., Correspondent, Mr.Sankar Vanavarayar M.B.A., PGDIEM, Joint Correspondent** and **Dr.S.Ramachandran Ph.D., Principal** for providing the necessary facilities to complete my thesis.

I take this opportunity to thank **Prof.N.Jayapathi M.Tech.,** Head of the Department, Department of Computer Science and Engineering, for his support and timely motivation. Special thanks to my Project Coordinator **Dr.V.Vanitha M.E., Ph.D.**, Senior Associate Professor, Department of Computer Science and Engineering, and project committee members for arranging brain storming project review sessions.

I register my sincere thanks to my Guide **Ms.V.P.Sumathi M.E.,** Assistant Professor[SRG], Department of Computer Science and Engineering. I am grateful for her support, encouragement and ideas. I would like to convey my honest thanks to all **Teaching** and **Non Teaching Staff** members of the department and my classmates for their support.

I dedicate this project work to my **Parents** for no reasons but feeling from bottom of my heart that without their love, this work would not be possible.

-MUTHUKUMAR G

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

OWL	Ontology Web Language
RDFS	Resource Description Framework Schema
RDF	Resource Description Framework
UOD	Universe of Discourse
XML	Extensible Markup Language

CHAPTER 1

INTRODUCTION

Ontology formally represents knowledge as a set of concepts within a domain, and the relationships among those concepts. It is used to reason about the entities within that domain and may be used to describe the domain. In recent years, the substantial increase in the use of medical knowledge discovery and decision support applications has required clinical researchers to write complex database queries. The end-users of these applications are normally unaware of the semantic relationships between the concepts stored in their respective databases. This problem grows with an increase in the structural complexity of the underlying data. The objective of this work is to rewrite the OWL into complex relational query and the use of specific knowledge to enable the process of query formulation and the format for structuring such domain knowledge. This work also discusses about the type of mappings required to translate OWL-DL expressions to relational ones.

The work reported here has been applied to the liver function test. The purpose liver function test is aiding in the differential diagnosis of liver disease and injury, and to help monitor response to treatment. The following are two major uses of an ontology based query formulation system in the liver function test project which is proposed by Bendi Venkata Ramana (2011). A user can formulate a any number of queries without specific knowledge of the liver enzyme range information and access mechanisms of the underlying data source clinical researchers can focus their data query requests around a particular genre of information, e.g., clinical observations, genetic information or laboratory test data, and therefore obtain a rather narrow and fragmented view of the individual patient .

1.1 ONTOLOGY

Ontology is a formal explicit representation of shared conceptualization. Conceptualization refers to an abstract model of some phenomenon. Explicit means that the type of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine readable. Shared reflects the notion that an ontology captures consensual knowledge that is, it is not private of some individual but accepted by a group.

Some essential aspects of ontologies:

- Ontologies are used to describe a specific domain.
- The terms and relations are clearly defined in that domain.
- There is a mechanism to organize the terms, (commonly a hierarchical structure is used as well as IS−A or HAS−A relationships).

There is an agreement between users of ontology in such a way the meaning of the terms is used consistently.

1.2. SPARQL QUERY LANGUAGE

SPARQL is an RDF query language, that is a query language for databases, able to retrieve and manipulate data stored in a Resource description framework format. It was made a standard by the RDF Data Access Working Group (DAWG) of the 'World Wide Web' consortium, and considered as one of the key technologies of the semantic web. SPARQL 1.0 became an official W3C recommendation. SPARQL allows for a query to consist of triple patterns, conjunctions, disjunctions, and optional pattern. There exist tools that allow one to connect and semi-automatically construct a SPARQL query for a SPARQL endpoint, for example ViziQuer. In addition, there exist tools that translate SPARQL queries to other query languages, for example to SQL and to XQuery.

Example

An example of the SPARQL user query "Find the albumin value for below the range of 1.5 in liver table":

```
SELECT ?x  ?albumin
WHERE { ?x a :inliver;  :albumin ?albumin .
FILTER (?albumin < 1.5) . }
```

1.3 HETEROGENEOUS SYSTEM

A system consists of different hardware, software, unconventional devices, diverse networks and protocols. Heterogeneity naturally occurs due to the fact that an autonomous development of systems always yields different solutions, for reasons such as different understanding and modelling of the same real-world concepts, the technical environment, the particular requirements of the application, such as high performance for special queries, etc. Bridging heterogeneity is one of the main tasks of integration. They sometimes coincide and sometimes differ. Susanne Busse (1999) has proposed syntactical, data model and logical heterogeneity with several subtypes.

1.3.1 TYPES OF HETEROGENEITY

1.3.1.1 Syntactical Heterogeneity

Syntactical Heterogeneity is classified into two types

Technical Heterogeneity: Technical heterogeneity concerns differences in the technical aspects, like hardware platforms and operating systems (hardware heterogeneity), or access methods, such as Protocol, e.g. HTTP, SQL*Net, ODBC,

CORBA etc., Stateless or state-carrying connection, Security and log-on procedures.

Interface Heterogeneity: Interface heterogeneity exists if different components are accessible through different access languages. The technical aspect is not herein means, such as whether a JDBC or a native interface is used, but refer to restrictions of the possible access methods. This includes:

Language Heterogeneity: different query languages and language restrictions, e.g.no negation, no disjunctive conditions etc.

Query Restrictions: only certain queries are allowed, only certain conditions are expressed, joins can only have up to three relations, etc.

Binding Restrictions: certain attribute values must be specified to form a valid query. Although one could see this as a special form of query restrictions, it is put separately since it requires different techniques.

1.3.1.2 Data Model Heterogeneity

Data model heterogeneity captures the fact that different data models have different semantics for their concepts. For instance, the relational model has no inheritance in contrast to object-oriented models. Although data model heterogeneity is a semantic conflict, it is treated separately because most systems handle it apart from other semantic conflicts, since data models typically contain only a handful of basic concepts. For instance, mediator-based systems require that wrappers hide data model diversity, but not semantic heterogeneity. The classical five layer architecture for federated databases has a separate layer to transform data models (the component schema).

1.3.1.3 Logical Heterogeneity

Semantic heterogeneity

Semantic heterogeneity concerns the semantic of the data and schema. Even schemas formulated in the same model can have different semantics. A schema consists of relations and attributes. These relations and attributes have names and carry an implicit semantic, which is the concept they stand for. However, the schema contains in the first place only the name, but not the concept. The interpretation of names by different people does not necessarily coincide. Therefore, different semantic conflicts can occur: equal names that denote different concepts (homonyms), different names for the same concept (synonyms), diverging understanding of a concept itself, etc.

Attributes can also have the same semantics, but different units (DM or USD for prices). The semantic of data values is defined through the schema element under which they are presented. The schema is the main place for encoding the semantics of a data source. Values do not carry semantic meaning, but are completely described through the schema portion that they belong to (i.e. The attribute and relation). This does not contradict the fact that in existing systems it might be necessary to analyze the values to extract the semantic of the schema

1.4 QUERY REWRITING

Integrating heterogeneous relational databases into a Semantic Web is the essential step to support the precise semantic search and query, where query rewriting from SPARQL to SQL is the key technique. This approach rewriting the SPARQL query to SQL queries based on semantic mappings from relational schemas to a universal shared ontology.
The query rewriting approach is divided into three steps

- Define mappings from relation schemas to the universal ontology by using the editor.
- Pre-process the mappings and the SPARL query for the rewriting .
- Rewrite the SPARQL query to SQL queries based on the pre-process results.

This approach has the following features

- Supports executing the semantic query (SPARQL) over distributed relational databases by rewriting the SPARQL query to SQL queries that are to be executed on individual databases.
- Supports rewriting complex SPARQL queries containing relational join of distributed databases to SQL queries by using the semi-join method in the distributed database community.
- Specially supports keywords of SPARQL (‘optional’, ‘filter’, ‘union’,' count ') in SQL query without losing any constraint information.

1.5 LITERATURE SURVEY

1.5.1 D2R MAP – A DATABASE TO RDF MAPPING LANGUAGE

Christian Bizer (2003) presented a method for database to RDF mapping language. The vision of the Semantic Web is to give data on the web a well defined meaning by representing it in RDF and linking it to commonly accepted ontology. Most formatted data today is stored in relational databases. To be able to use this data on the Semantic Web, One needs a flexible but easy to use mechanism to map relational data into RDF. The poster presents D2R MAP, a declarative language to describe mappings between relational database schema and OWL ontology. The Semantic Web is an extension of the current Web, in which data are given a well-defined meaning by representing it in RDF and linking it to commonly accepted ontology.

This semantic enrichment allows data to be shared, exchanged or integrated from different sources and enables applications to use data in different contexts. Most formatted data today is stored in relational databases. To be able to use this data in a semantic context, it has to be mapped into RDF, the data format of the Semantic Web.In D2R, basic concept mappings are defined using class maps.. The data model behind RDF is a directed labelled graph, which consists of nodes and labelled directed arcs linking pairs of nodes. The level of similarity between both models is very high and mappings are consequently quite direct. To export data from an RDBMS into RDF, the relational database model has to be mapped to the graph-based RDF data model.

1.5.1.2 The D2R Mapping Process

The mapping process performed by the D2R processor has four logical steps as shown in Figure 1.1.

Database to RDF mapping is done in three steps

- For each class or group of similar classes a record set is selected from the database
- The record set is grouped according to the group, by columns of the specific Class Map. Then the class instances are created and assigned a URI or a blank node identifier.
- The instance properties are created using data type and object property bridges.

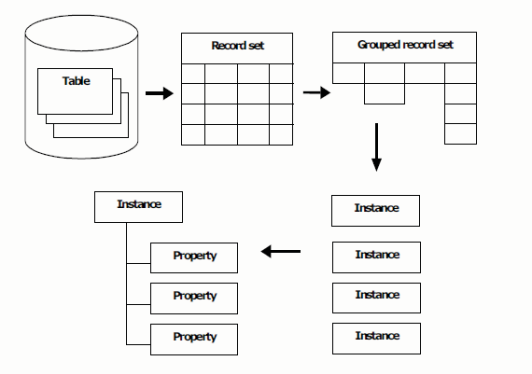

Fig 1.1 D2R mapping process

1.5.1.3 Advantages

- Convert the relational database to RDF language so use the semantic web information retrieval and retrieve more relevant data.
- Flexible and easy mechanism.

1.5.2 FUND FINDER: A CASE STUDY OF DATABASE-TO ONTOLOGY MAPPING

1.5.2.1 Objective

The mapping between databases and ontologies is a basic problem when trying to "upgrades" deep web content to the semantic web. J. Barrasa (2003) proposed this approach suggests the declarative definition of mappings as a way to achieve domain independency and reusability. In a specific language (expressive enough to cover some real world mapping situations like lightly structured databases or not first normal form ones) is defined for this purpose. Along with this mapping description language, the ODE Mapster processor is in charge of carrying out the effective instance data migration.

1.5.2.2 Methodology

This approach is distinguished into two layers. The layers are modelling layer and the implementation layer. At the first one, the ontology conceptual model is there on the Web ODE platform and the E/R model underlying the database. At the implementation layer, the ontology is implemented in several ontology languages (OWL, DAML+OIL, RDF (S)…) using Web ODE translators and the SQL implementation of the database relational model. An instance data sub-layer would contain instance data from the database (records) and instances of the ontology. The ODE mapster processor is the software in charge of the mapping execution according to the directives of the aforementioned mapping document. The execution occurs automatically once the mappings are defined. A database contains the data to be migrated as instances of the ontology. Ontology is to be populated with the data extracted from the database. The ontology is expressed in any ontology implementation language, but instances of the ontology are generated in RDF in the first version of the processor

1.5.2.3 Tool: ODE Mapster processor.

1.5.2.4 Advantage

- Retrieve the semantically equivalent document

1.5.2.5 Disadvantage

- Does not deal with complex situations

1.5.3 VISAVIS: AN APPROACH TO AN INTERMEDIATE LAYER BETWEEN ONTOLOGIES AND RELATIONAL DATABASE CONTENTS

1.5.3.1 Objective

Konstantinou (2006) proposed approach is introduced an approach to mapping relational database contents to ontology. The current effort is motivated by the need of including into the Semantic Web volumes of web data not satisfied by current search engines. A graphical tool is developed in order to ease the mapping procedure and export enhanced ontology linked to database entities. Moreover, queries using semantic web query languages are imposed to the database through its connection to an ontology. Using Protégé, one can to map ontology instances into relational databases and retrieve results by semantic web query languages. The key idea is that, instead of storing instances along with the ontology terminology, one can keep them stored in a database and maintain a link to the dataset. Thus, on one hand one achieves smaller size ontology and on the other hand one can exploit database contents harmonizing the semantic web concept with current wide-spread techniques and popular applications.

1.5.3.2 Methodology

The work implemented includes the functionality of imposing queries by terms of semantic web query languages. Fig 1.2 illustrates how the application handles user's queries. First user queries are parsed and checked syntactically. The query would instantly return with the desired results but here is where the mapping intervenes. Instead of returning the class resources, for each class a check is done if there exists the mapping property. If the related property is found, the query is redirected to the database. The results are then formatted and served to the user.

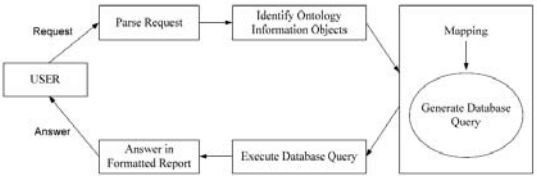

Fig1.2 Database mapping mechanism

The achievement lies in the fact that an extra layer of interoperability is added between the application GUI and the actual data. This hybrid approach leaves the ontology and the actual data intact, only serving as a semantics addition to current systems. The generalization of the current concept to current web technologies is about to offer a semantic layer that will add the desired intelligence without modifying legacy systems.

1.5.3.3 Tool: VisAVis

1.5.3.4 Advantage

- Retrieve the semantic data.
- Relational database information converted into more retrievable format.

1.5.3.5 Disadvantage

- The relational database to ontology implemented only selection from where function.
- In this method SPARQL query does not support the group by, count with, min, max statements.

1.5.4. AN ONTOLOGY BASED VISUAL TOOL FOR QUERY FORMULATION SUPPORT

1.5.4.1 Objective

T. Catarci (2003) proposed approach describe the principles of the design and development of an intelligent query interface, done in the context of the SEWASIE (semantic Webs and Agent in Integrated Economies) European IST project. The SEWASIE project aims at enabling a uniform access to heterogeneous data sources through an integrated ontology. The query interface is meant to support a user in formulating a precise query which best captures her/his information needs even in the case of complete ignorance of the vocabulary of the underlying information system holding the data. The intelligence of the interface is driven by an ontology describing the domain of the data in the information system. The final purpose of the tool is to generate a conjunctive query ready to be executed by some evaluation engine associated with the information system

1.5.4.2 Methodology

Initially the user is presented with a choice of different query scenarios which provide a meaningful starting point for the query construction. The interface guides the user in the construction of a query by means of a diagrammatic interface, which enables the generation of precise and unambiguous query expressions. Query expressions are compositional, and their logical structure is not flat but tree shaped i.e. a node with an arbitrary number of branches connecting to other nodes. This structure corresponds to the natural linguistic concepts of noun phrases with one or more propositional phrases.

The latter can contain nested noun phrases themselves. A query is composed by a list of terms coming from the ontology (classes) e.g. “Supplier” and “Multinational”. Branches are constituted by a property (attributes or associations) with its value restriction, which is a query expression itself e.g. “selling on Italian market”, where “selling on” is an association, and “Italian market” is an ontology term. The focus paradigm is central to the interface user experience manipulation of the query is always restricted to a well defined, and visually delimited, subpart of the whole query (the focus).

The compositional nature of the query language induces a natural navigation mechanism for moving the focus across the query expression (nodes of the corresponding tree). A constant feedback of the focus is provided on the interface by means of the kind of operations which are allowed. The system suggests only the operations which are “compatible” with the current query expression in the sense that do not cause the query to be unsatisfiable. This is verified against the formal model describing the data sources. One of the main requirements for the interface is that it must be accessed from any HTML browser, even in presence of restrictive firewalls.

1.5.4.3 Tool: Sewaise tool.

1.5.4.4 Advantage

- User friendly Query interface.
- Natural language representation of queries.

1.5.4.5 Disadvantage

- The SEWASIE complex concept descriptions have not been considered to support query formulation.

1.5.5 USE OF OWL AND SWRL FOR SEMANTIC RELATIONAL DATABASE TRANSLATION

1.5.5.1 Objective

D.M. Fisher (2008) proposed general purpose query interfaces to relational databases can expose vast amounts of content to the Semantic Web. Automapper is discussed which is a tool that automatically generates data source and mapping ontologies using OWL and SWRL. Auto mapper functionality that exploits some of the expressiveness of OWL to produce more accurate translations. A comparison with related work on Semantic Web access to relational databases is also provided as well as an investigation into the use of OWL 1.1.

1.5.5.2 Methodology

1.5.5.2.1 Architecture

To understand Auto mapper's utility, a description of its subsuming architecture is provided. While that is not the focus of this work, a general understanding is necessary to appreciate Automapper's role in the system as a whole. As shown in figure 1.3. Automapper is part of a larger system for decomposing a SPARQL query, expressed using a domain ontology, over multiple data sources and merging the corresponding data into a single result set. Specifically, Automapper uses the database schema to create an OWL data source ontology and mapping instance data to support two layers of processing the higher-level Semantic Query Decomposition component (SQD) and the lower level SPARQL-to-SQL translation component, also known as a Semantic Bridge for Relational Databases (SBRD).

SQD relies on a set of SWRL data source to domain mapping rules and optional domain-to-domain mapping rules to properly translate inbound queries into data source queries and vice-versa. The use of a domain ontology allows queries to be independent of any particular data source. In addition, different communities can each adopt their own domain ontology while reusing the same data sources.

Mapping rules need only cover the data of interest thereby minimizing integration costs. SBRD uses both Automapper artifacts to correctly map a SPARQL query expressed using the data source ontology into SQL SELECT statements. Database query result sets are mapped back into the data source ontology and returned to SQD. As a final step, SQD recombines the various result sets, maps the data into the domain ontology and returns this data to the user.

Automapper was developed in Java and needs to be run only once against a given relational database to automatically generate both the data source ontology and the mapping data. Using the method java.sql.Connection.getMetaData(), we are able to mine the necessary table definitions such as column names, primary keys, foreign keys, remarks and type declarations to generate the data source ontology. Automapper constructs these artefacts based on a configuration file, which contains instance data based on a simple configuration ontology. This ontology permits additional primary and foreign key information, the option to include or exclude comments in the mapping instance data, the capacity to limit the visibility of specific tables and the ability to override a declared column data type with another XML Schema data type.

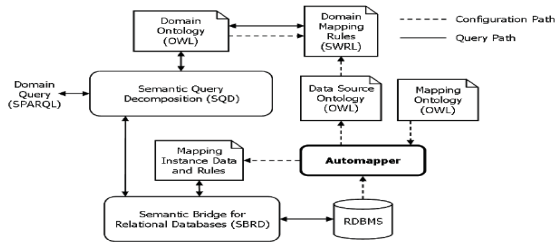

Figure 1.3 Overview of the Semantic Distributed Query Architecture

1.5.5.2.2 Utilizing Automapper Mapping Data

While SQD relies on SWRL rules for mapping, SBRD depends on Automapper's generated mapping data. SBRD employs these mappings for

transforming SPARQL data source queries into SQL queries for data retrieval. Our mapping ontology, based on the mapping ontology used by D2RQ , defines a Class Map OWL class whose instances represent each table in a given schema. A table has a corresponding OWL class in the data source ontology, an owning schema, a name, a url Pattern and one or more data type property bridges and object property bridges. The url Pattern is an OWL data type property used to identify each row in a table (instance) with a unique URI by concatenating its table name with the value of each primary key column in the table. The data type and object property bridges correspond to columns containing literal values and foreign keys, respectively.

1.5.5.2.3 Use of Owl

The following class descriptions, axioms and restrictions are currently generated by Automapper

- Max Cardinality is set to 1 for all null able columns
- Cardinality is set to 1 for all non-null able columns
- All data types and object properties that represent columns are marked as Functional Properties. To ensure global uniqueness and class specificity, these properties are given URIs based on concatenating the table and column names
- All Values From restrictions reflect the data type or class associated with each column

1.5.5.3 Tool: Automapper.

1.5.5.4 Disadvantage

- URIs returned by SBRD are unique but generally not resolvable.
- Decidability is a critical aspect in this architecture.

CHAPTER 2

IMPLEMENTATION

2.1 DRAWBACKS OF EXISTING SYSTEM

In the existing approach SPARQL query supports only Select, from, where clauses. SPARQL query does not support count, order by and group by statements. In this work SPARQL is implemented with description in such a way that it should support count function, group by, and order by .

2.2 PROPOSED SYSTEM

Real world applications that deal with several heterogeneous databases may suffer with a lot of heterogeneity conflicts. Such conflicts may get resolved using this proposed system. The objective of the proposed work is to provide an efficient and effective methodology for query processing. In recent years, the substantial increase in the use of medical knowledge discovery and decision support applications has required clinical researchers to write complex database queries. The end-users of these applications are normally unaware of the semantic relationships between the concepts stored in their respective databases. This problem grows with an increase in the structural complexity of the underlying data. The objective of this work is to rewrite the SPARQL query into complex relational query and the use of specific knowledge to enable the process of query formulation and the format for structuring such domain knowledge. In this work SPARQL is implemented with description in such a way that it should support count function.

The proposed architecture is shown in figure 2.1.

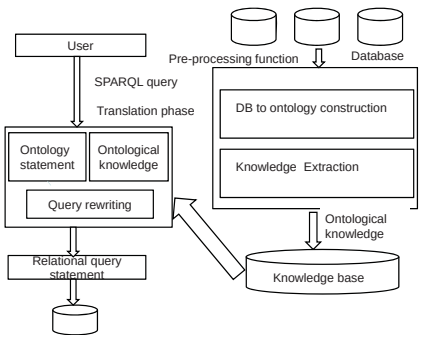

Figure 2.1 Proposed Architecture

Methodology for Proposed System Development

- The work reported here has been applied to the liver function test. The purpose liver function test is aiding in the differential diagnosis of liver disease and injury, and to help monitor response to treatment.
- In this proposed approach, each data source is described by its own schema.
- The schema must be defined by classes, subclasses and their relations and then the ontology is created using a tool protégé.
- Ontology and database are mapped through mapping function.
- The normal SPARQL query result is retrieved through mapping function.
- Count, group by, order by related SPARQL query rewrite to SQL query through translation phase using the knowledge base.

Expected Advantage of Proposed System

- The proposed system supports aggregate function like group by, order by, count related SPARQL queries .
- The proposed system has resolved semantic heterogeneity issues.
- In proposing system user gives the query without knowing the domain knowledge.
- A user can formulate any number of queries without specific knowledge of the liver enzyme range information and access mechanisms of the underlying data source

2.3 PROJECT DESCRIPTION

2.3.1 PROBLEM DEFINITION

In Medical domain several names are having the same meaning. The user does not know the meaning and context. In particularly liver function test several enzyme names are different but the enzyme name represents the same thing the heterogeneity issues are solved by using ontology. SPARQL query supported select, from, where clauses. SPARQL query does not support aggregate function like count, group by, order by clauses. This work will incorporate the component count, order by, group by function with SPARQL .

2.3.2 MODULES

This proposed architecture human involvement is in two stages. Once the user giving queries in natural language and second domain expert convert this query into OWL queries. The remaining part of this work is like rewriting the query in SQL and getting the result from heterogeneous resource using ontological knowledge and domain knowledge is done automatically.

This work is split into three modules

- Preprocessing module.
- Knowledge base construction.
- Translation module.

2.3.2.1.Pre-Processing Module

In the preprocessing phase creating the local ontology and global ontology and store the domain meta data with relationships. The preprocessing phase also maps the relational database to ontology. It can handle 1:1, 1:M, M: N relationship. Ontological knowledge is stored in the server using the RDF triples format. Triple store provides efficient storage and retrieval of ontological information.

2.3.2.1.1 Ontology Creation

One can get the local ontology from the local schema which is provided by UCI machine learning repository. This process contains two main steps: analysis of data source and defining the local ontologies. The first step implies a complete analysis of the data sources, e.g., what data is stored, how it is stored, the meaning of that data (the semantic), etc. It must localize the problems about semantic heterogeneity previously explained.

A complete analysis of the data sources must be made. This analysis is performed independently, that is, without taking into account the other data sources. With this analysis, the second step is performed. In this work took three data sources taken from UCI machine learning repository. The data sources are hepatitis, Indian liver, liver disorder. In this ontology creation semantic heterogeneity issues are solved.

2.3.2.1.2 Mapping Function

Mapping function map the relational database and ontology. It can handle 1:1, M: N relationships. The mapping function is implemented by using the quest 3M. Rodriguez-Muro (2012) Quest allows to link the vocabulary of an ontology to the content of a relational database through mapping axioms. These are used together with the ontology to answer a SPARQL query by means of a single SQL query that is then executed over the database. Quest plug-in offers a powerful mapping language that often compensates for the lack of expressivity of RDFS and OWL 2 QL and that enables use cases that are often tackled with OWL 2 EL, OWL 2 RL or SWRL. Mappings axioms are the means to relate the content of a database to the vocabulary of an ontology we show how using (Mariano Rodr´iguez-Muro, 2012) Quest such scenario is realizable efficiently in an on-the-fly manner using query rewriting and mappings as the following:

<"http://www.semanticweb.org/muthu/ontologies/2013/1/liver1.owl#pid_{$patiend_id}"> rdf :type

[S]

:inliver; :albumin $albumin.

[P] [O]

2.3.2.2 Knowledge Base Construction

The knowledge base is used for extending SPARQL support to aggregate functions. To perform query processing, descriptions are added to SPARQL query. Based on the descriptions appropriate functions are performed for example the user queries to find the number of patients affected by hepatitis. If the patients alamine transminase, asparate aminotransferase exits thirty five the patients is affected by hepatitis. The user query retrieves results from multiple data sources. The records count from each data source is summed up and stored in the knowledge base. The possibility of a patient to have hepatitis is when his alamine transminase,

asparate aminotransferase drops below thirty five such information is stored in the knowledge base. In addition description logic for count, order by, group by are stored in the knowledge base. In this knowledge base contains the normal range of the enzyme values in liver function test. The diseases which are caused due to hypes secretion of enzymes are stored in the knowledge base.

Example for SWRL rules :

The Alamine transminase normal range is below 5 to above 35. The above statement corresponding SWRL rule is

Rule 1

Alamine Transminase(?Al), int[>=5,<=35]→normal range(?Al, true)

The ALKP normal range is below 30 to above 120. The above statement corresponding SWRL rule is

Rule 2

ALKP(?ALK), int[>=30,<=120]→normal range(?ALK, true)

The Albumin normal range is below 3.4 to above 5.4. The above statement corresponding SWRL rule is

Rule 3

Albumin(?Alb), int[>=3.4,<=5.4]→normal range(?Alb, true)

The Sgot normal range is above 35. The above statement corresponding SWRL rule is

Rule 4

Sgot(?Sg), int[>=35]→normal range(?Sg,true)

The range of Alamine Transminase is greater than 35 causes the hepatitis disease. The above statement corresponding SWRL rule is

Rule 5

Alamine Transminase(?Al), int[?Al,?int] swrlb : greater than (?int,35)→ hepatatis(?Al)

The range of ALKP is greater than 120 causes the lung cancer disease.
The above statement corresponding SWRL rule is

Rule 6

ALKP(?ALK), int[?ALK,?int] swrlb: greater than(?int,120)→ lung cancer (?ALK)

2.3.2.3 Translation Module

In the translation phase ontology driven relational query formulation process translates relational algebra query statements into the corresponding SQL query statements. In the translation phase is able to answer unions of conjunctive queries (UCQs) expressed over the ontology alphabet: the class of UCQs is one of the most important classes of query arising in practical cases Acciarri(2005). Answering the query process is performed by query rewriting and strongly separates the intentional level from the extensional one user queries are first reformulated on the basis of the TBox assertions, and then evaluated directly over the database by means of the mappings . Hence, the TBox assertions are "compiled" into the query, so the TBox is disregarded during the query evaluation phase. The translation phase also provides two different mappings handling techniques. The first one exploits the SQL engine of the DBMS managing the data layer of the ontology a view is defined for every general concept, using the SQL query specified in the mapping assertion. The second one unfolds the query by producing an SQL statement that was issued over the source tables. If the count related SPARQL query come into

the translation phase separate Java wrapper method is used to rewrite the SPARQL query into SQL query and retrieve the result. Java wrapper extracts the specific domain keywords from SPARQL query. Based on the extracted information appropriate SQL query is generated.

Example

English Query

How many patient's chances of getting the hepatitis disease.

SPARQL Query

1.SELECT ?x WHERE{ ?x a :hepatitis;)}

Knowledge Base Rule Used for Rewriting

Alamine Transminase(?Al), int[?Al,?int] swrlb : greater than (?int,35)→ hepatatis(?Al)
In the rule tell alamine transminase range is below 35 causes the hepatatis disease.

SQL Query

- select count(patiend_id) as tot from indian_liver where asparate_aminotransferase>35 and alamine_trasaminase>35
- select count(patient_id) as tot from hepatitis where sgot>35
- select count(patient_id) as tot from liver_disorder where asparate_transaminase>35 and alamine_aminotransferase>35

CHAPTER 3

RESULTS

3.1 SYSTEM SPECIFICATION

3.1.1 HARDWARE REQUIREMENTS

Processor	: Pentium IV and above.
Clock speed	: 1.83 GHz.
Hard Disk	: 500GB.
RAM	: 4GB or above.
Cache Memory	: 512KB.
Operating System	: Any Windows version.
Monitor	: Color Monitor.
Keyboard	: 104Keys.
Mouse	: 3Buttons.

3.1.2 SOFTWARE REQUIREMENTS

Operating System	: Windows XP.
Database	: SQL Server, MySQL, Oracle 9i (ORDBMS).
Database drivers	: ODBC driver, Oracle-JDBC driver.
Platform	: J2EE Version 7.
Tool	: protégé with on top framework.
IDE	: Eclipse JUNO, Apache Jena.

3.2 IMPLEMENTATION

The proposed work is implemented using protégé tool. Implementation of this project is done with 3 different databases with several tables. For the evaluation purpose, each table is populated with 1500 records. Local and Global ontology is created with help of protégé tools for all the tables present in the database. The ontology and database had mapped using the on top framework. Knowledge base contains information about enzyme name, ranges, list of diseases. The Knowledge base is implemented using Semantic Web Rule Language rules. If the count related SPARQL query come into the translation phase separate Java wrapper method is used to rewrite the SPARQL query and retrieve the result.

3.3 ANALYSIS

3.3.1 RECALL

Recall in information retrieval is the fraction of the documents that are relevant to the query that are successfully retrieved.

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

3.3.2 RECALL RATE WITHOUT QUERY REWRITE AND WITH QUERY REWRITE

Based on the above formula the recall rate is calculated for with and without query rewrites approach of the OWL. The experiment was conducted by varying the number of records. The following figure 3.1. Illustrates result of the recall rate with query rewriting using knowledge base and without query rewriting concept .

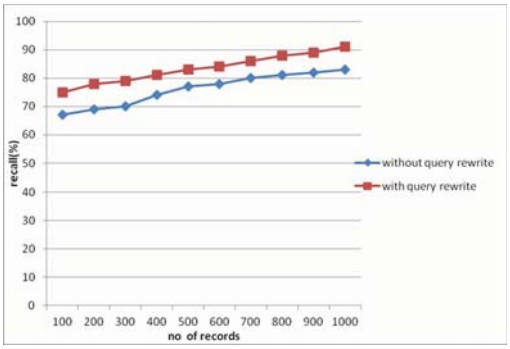

Figure 3.1 Recall rate

3.4 CONCLUSION AND FUTURE OUTLOOK

In this work the SPARQL queries is rewritten into SQL queries. The SQL queries are applied to the data stores present in heterogeneous environments. A mapping mechanism is used to address the heterogeneity problem and automatically inferring joins between relational tables and configures the semi join automatically. The system does not only support rewriting simple SPARQL queries to SQL queries, but also it supports some of the aggregate functions such as a count, group by, order by functions. The effectiveness of the presented approach is demonstrated by using medical database. Particularly, the approach has well developed and in-use in a liver function test query platform.

In the future, functionalities and improvements are still needed to make the system more effective.

APPENDIX

SAMPLE SOURCE CODE

```
import java.applet.Applet;
import java.awt.Event;
import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.TextArea;
import java.awt.TextField;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.io.PrintStream;
import java.io.UnsupportedEncodingException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```



```
import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.Logger;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.OntModelSpec;
import com.hp.hpl.jena.query.Query;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QueryFactory;
import com.hp.hpl.jena.query.ResultSetFormatter;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.util.FileManager;
public class dataint extends Applet {
private static final long serialVersionUID = 1L;
    TextField textField;
  static TextArea textArea;
  String url = null;
    Connection con = null;
  Connection con1=null;
  ResultSet rs1=null;
  String field;
  String s=null;
  String Sql=null;
```

```
@Override
                public void init() {
        textField = new TextField(40);
            textArea = new TextArea(10, 140);
            textArea.setEditable(false);
            textArea.getFont();
            Font newfont = new Font("Monospaced", Font.PLAIN, 12);
            textArea.setFont(newfont);
            // Add Components to the Applet.
            GridBagLayout gridBag = new GridBagLayout();
            setLayout(gridBag);
            GridBagConstraints c = new GridBagConstraints();
            c.gridwidth = GridBagConstraints.REMAINDER;
            c.fill = GridBagConstraints.HORIZONTAL;
            gridBag.setConstraints(textField, c);
            add(textField);
            c.fill = GridBagConstraints.BOTH;
            c.weightx = 1.0;
            c.weighty = 1.0;
            gridBag.setConstraints(textArea, c);
            add(textArea);
            validate();
```



```
        }
    @SuppressWarnings("deprecation")
        @Override
    public boolean action(Event evt, Object arg)
    {
        Logger logger = Logger.getLogger(dataint.class);
                BasicConfigurator.configure();
       logger.info("Entering application.");
        String query1 = textField.getText();
         OntModel model = ModelFactory.createOntologyModel(
        OntModelSpec.OWL_MEM_MICRO_RULE_INF);
                String inputFileName="E:/project/livermuthu1.owl";
                String prefix="PREFIX : <http://www.semanticweb.org/muthu/ontologies/2013/1/liver1.owl#>"+
                "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>"+
                "PREFIX owl: <http://www.w3.org/2002/07/owl#>"+
                "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>"+
                "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>";
                InputStream in = FileManager.get().open( inputFileName );
```



```
                if (in == null)
                {
                   throw new IllegalArgumentException(
                       "File: " + inputFileName + " not found");
                }
                model.read(in, null);
                try {
                        Class.forName("com.mysql.jdbc.Driver");
                } catch (ClassNotFoundException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                Connection con = null;
                try {
                        con = DriverManager.getConnection("jdbc:mysql://localhost:3306/liver","root","muthu");
                } catch (SQLException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                Statement st = null;
                try {
```



```
                        st = con.createStatement();
                } catch (SQLException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
            String split[]=query1.split(" ");
if(split[0].equals("1")&&(query1.contains("asparate_aminotransferase")||query1.contains("alamine_transaminase")))
                {
                    ResultSet rs = null;
                        try {
                                rs = st.executeQuery("select object from knowledge where subject="'+query1+'"");
                        } catch (SQLException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                    try {
                                while(rs.next())
                                {
                                        textArea.appendText("total no of patients affected by hepatits is="+rs.getInt("object"));
                                }
```

```
                } catch (SQLException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }
        else
        {
                String queryString =prefix+
                        query1;
                Query query = QueryFactory.create(queryString);
                QueryExecution qe = QueryExecutionFactory.create(query, model);
                com.hp.hpl.jena.query.ResultSet results = qe.execSelect();
                ByteArrayOutputStream baos = new ByteArrayOutputStream();
                PrintStream ps = new PrintStream(baos);
                ResultSetFormatter.out(ps,results, query);
                try {
                        s = new String(baos.toByteArray(), "UTF-8");
                } catch (UnsupportedEncodingException e) {
                        // TODO Auto-generated catch block
```



```
                        e.printStackTrace();
                }
                textArea.appendText(s);
                qe.close();
        }
        logger.info("Exiting application.");
        return true;
    }
}
```


SCREEN SHOTS

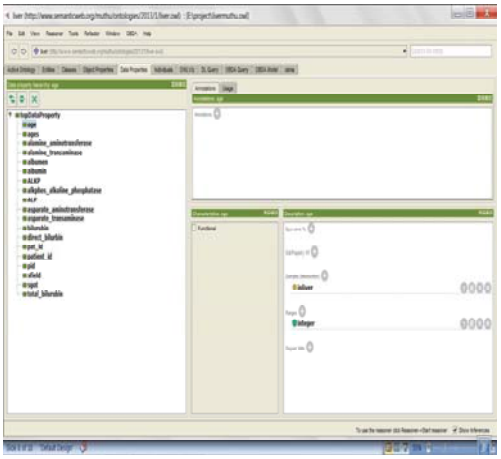

Figure A1 Ontology creation

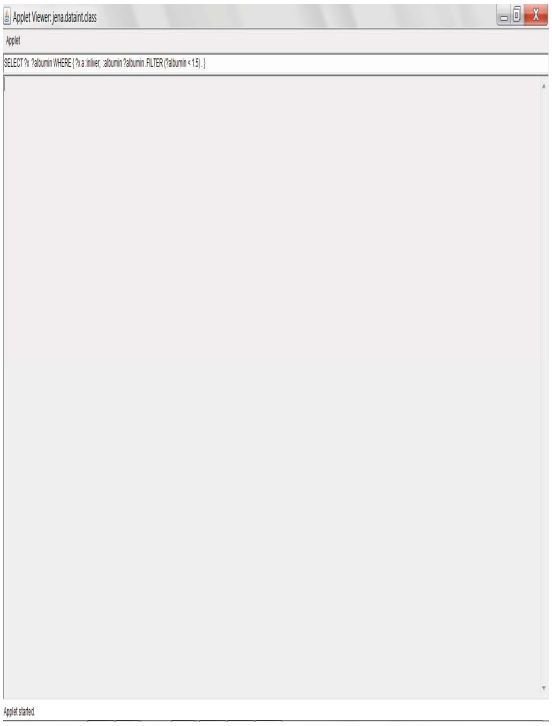

Figure A2 Posting A Query against the database

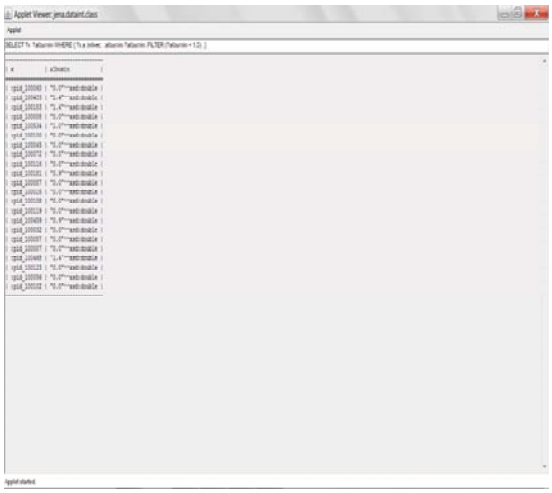

Figure A3 Result Display

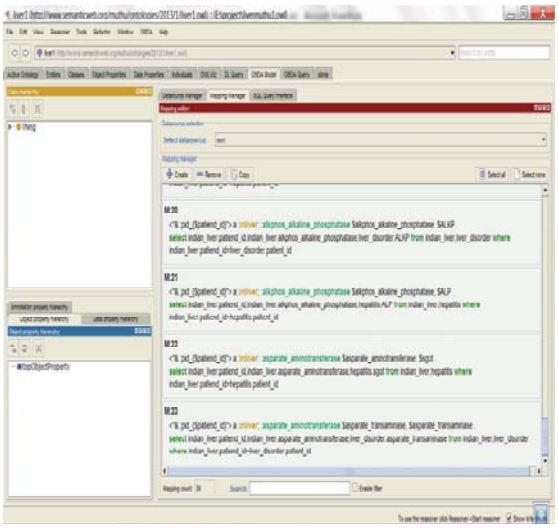

Figure A4 Mapping function

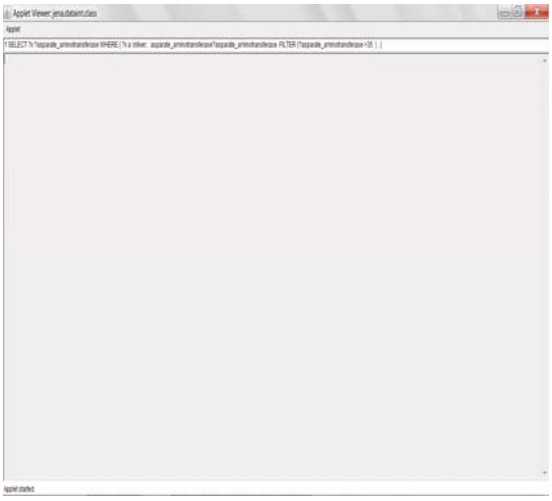

Figure A5 Count related SPARQL Query against the database

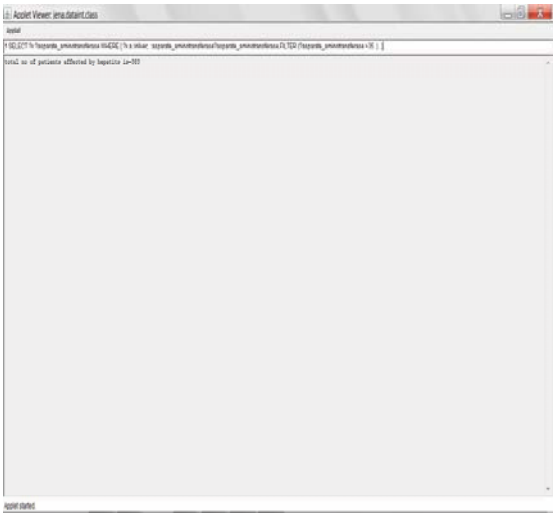

Figure A6 Result Display

TEST QUERY

```
SELECT ?x  ?albumin
WHERE { ?x a :inliver;  :albumin ?albumin .
FILTER (?albumin < 3.4) . }

SELECT ?x  ?albumin
WHERE { ?x a :inliver;  :albumin ?albumin .
FILTER (?albumin > 5.4) . }

SELECT ?x  ?albumin
WHERE { ?x a :inliver;  :albumin ?albumin .
FILTER (?albumin < 3.4 || albumin > 5.4) . }

SELECT ?x  ?asparate_aminotransferase
WHERE { ?x a :inliver;  :asparate_aminotransferase ?asparate_aminotransferase.
FILTER (?asparate_aminotransferase <35) . }

SELECT ?x  ?asparate_aminotransferase
WHERE { ?x a :inliver;  :asparate_aminotransferase ?asparate_aminotransferase
FILTER (?asparate_aminotransferase >35) . }
SELECT ?x  ?alkphos_alkaline_phosphatase
```



```
WHERE { ?x a :inliver; :alkphos_alkaline_phosphatase ?asparate_aminotransferase .
FILTER (?asparate_aminotransferase >35) . }

SELECT ?x ?alamine_transaminase WHERE { ?x a :inliver; :alamine_transaminase ?alamine_transaminase .
FILTER (?alamine_transaminase>5 && ?alamine_transaminase<35) . }

SELECT ?x  ?ALKP WHERE { ?x a:liverdisorder;  :ALKP ?ALKP .
FILTER (?ALKP < 120 ) . }

SELECT ?x  ?ALKP WHERE { ?x a:liverdisorder;  :ALKP ?ALKP .
FILTER (?ALKP < 30 ) . }

SELECT ?x  ?ALKP WHERE { ?x a:liverdisorder;  :ALKP ?ALKP .
FILTER (?ALKP < 30 && ?ALKP < 120  ) . }

SELECT ?x  ?asparate_transaminase
WHERE { ?x a :liverdisorder;  :asparate_transaminase ?asparate_transaminase .
FILTER (?asparate_transaminase <35) . }

SELECT ?x  ?asparate_transaminase
WHERE { ?x a :liverdisorder;  :asparate_transaminase ?asparate_transaminase .
```



```
FILTER (?asparate_transaminase >35) . }

SELECT ?x  ?sgot
WHERE { ?x a :hepatitis;  :sgot ?sgot .
FILTER (?sgot <35) . }

SELECT ?x  ?sgot
WHERE { ?x a :hepatitis;  :sgot ?sgot .
FILTER (?sgot >35) . }

SELECT ?x  ?albumen
WHERE { ?x a :hepatatis;  :albumen ?albumen .
FILTER (?albumen < 3.4) . }

SELECT ?x  ?albumen
WHERE { ?x a :hepatatis;  :albumen ?albumen .
FILTER (?albumen > 3.4) . }

SELECT ?x  ?albumen
WHERE { ?x a :inliver;  :albumen ?albumen .
FILTER (?albumen < 3.4 || albumen > 5.4) . }
```



```
SELECT ?x  ?bilurubin
WHERE { ?x a :inliver;  :albumen ?albumen .
FILTER (?albumen < 3.4 || albumen > 5.4) . }
```

REFERENCES

1. Acciarri, D. Calvanese, G. Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, R.Rosati,"QUONTO: querying ontologies", in: Anon. (Ed.), Proceedings of AAAI, 2005, pp. 1670–1673.

2. Bendi venkata ramana, M.Surendra Prasad Babu, N. B. Venkateswarlu,"A Critical Study of Selected Classification Algorithms for Liver Disease Diagnosis", International Journal of Database Management Systems (IJDMS), Vol.3, No.2, May 2011.

3. J. Barrasa, Ó. Corcho, A. Gómez-Pérez, "A case study of database-to-ontology mapping", in: Anon. (Ed.) Semantic Integration Workshop, ISWC 2003, 2003.

4. Christian Bizer, Freie Universität Berlin, Institut fùr Produktion,"D2R MAP – A Database to RDF Mapping Language", ACM xxx, May 20-24, 2003.

5. T. Catarci, T. Di Mascio, E. Franconi, G. Santucci, S. Tessaris, "An ontology based visual tool for query formulation support", in: On the Move to Meaningful Internet Systems 2003: OTM 2003 Workshops, Springer, Berlin/Heidelberg, 2003, pp. 32–33.

6. D.M. Fisher, M.G. Joiner,"Use of OWL and SWRL for semantic relational database translation", in: Proceedings of the Fourth OWLED Workshop on OWL: Experiences and Direction, 2008.

7. N. Konstantinou, D.-E. Spanos, M. Chalas, E. Solidakis, N. Mitrou,"VisAVis: an approach to an intermediate layer between ontologies and relational database contents", in: anonymous (Ed.), International Workshop on Web Information Systems Modeling (WISM), 2006

8. K. Munir , M. Odeh, R. McClatchey, "Ontology-driven relational query formulation using the semantic and assertional capabilities of OWL-DL", ScienceDirect, Year:2012.

9. Mariano Rodr´ıguez-Muro, Josef Hardi and Diego Calvanese,"Quest: Efficient SPARQL-to-SQL for RDF and OWL", Anon. (Ed.), Proceedings of AAAI, 2012, pp. 1670–1673.

10. Susanne Busse, Ralf-Detlef Kutsche, Ulf Leser, Herbert Weber, "Federated Information Systems: Concepts, Terminology and Architectures", Anon. (Ed.),April 1999,pp. 1760–1768.

LIST OF PUBLICATION

1. Muthukumar.G, Sumathi.V.P, **"** ONTOLOGY DRIVEN QUERY REWRITING USING KNOWLEDGE BASE**",** Fifth International Conference ,Sun College of Engineering and Technology, Erachakulam on 18th February 2013.