A FEDERATED DATA REPOSITORY QUERY SYSTEM

By

SHUAI ZHENG

A Thesis Submitted to the Graduate Faculty of

WAKE FOREST UNIVERSITY

in Partial Fulfillment of the Requirements

for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

May 2010

Winston-Salem, North Carolina

Approved By:

Stan J. Thomas, Ph.D., Advisor _____

Examining Committee:

V. Paúl Pauca, Ph.D., Chairperson _____

Yaorong Ge, Ph.D. _____

Acknowledgement

I am sincerely grateful to the faculty and staff in the Computer Science department. Not only for this thesis, but also for the help and guidance that I received during my two years study at Wake Forest University.

I owe my deepest gratitude to my advisor, Dr. Stan Thomas, for his kind and patient support throughout this thesis and my study. He has inspired me with his comprehensive knowledge and insightful advice, furthermore, cultivated me with his dedicated spirit of research and strict attitude of science.

I am heartily thankful to Dr. Yaorong Ge, who has leaded me into a research area that I want to strive for in my future life. His precious experience and magnetic personality have enriched my growth as a student and researcher.

I would like to thank Dr. Paul Pauca. The experience and instructions he shared with me has broadened my prospective, enlightened my view and given me the confidence to start the adventure of study in the future.

I would like to show my sincere gratitude to Dr. David John. He is always one of the most constant sources of warm encouragement and considerate assistance. His generous support makes my life at Wake Forest so enjoyable and smooth.

The last but not least, I am greatly indebted to Dr. Errin Fulp. For countless times, I walked into his office and talked with him as friends, but he always stimulated me with his invaluable guidance and instruction like a mentor and tutor.

Let me say “Thank you” to all my teachers and friends. I am not a brilliant student, it is because of you, I get the opportunity to shine.

Table of Contents

List of Figures

List of Tables

Abstract

In both scientific research and data analysis, one of the most important challenges is managing and utilizing large amounts of data. Unfortunately, the information needed may be distributed across multiple databases with inconsistent or conflicting schemas. Such data inconsistencies are, at the very least, an impediment to users. This phenomenon raises the need for a tool that can integrate heterogeneous databases.

This project established an architecture for integrating data from heterogeneous databases into an RDF data repository and then mapping the integrated dataset to concepts in an ontology. Through the concepts represented in the ontology, users can readily retrieve information from the federated data repository system.

Chapter 1: Introduction

In both scientific research and data analysis, one of the most important challenges is managing and utilizing large amounts of data. Unfortunately, the information needed may be distributed across multiple databases with inconsistent or conflicting schemas [3]. Such data inconsistencies are, at the very least, an impediment to users. This phenomenon raises the need for a tool that can be used to integrate heterogeneous databases.

In addition, non-professional users do not have in-depth knowledge about traditional database structures, which makes it difficult for them to utilize the information in heterogeneous databases efficiently [13]. Therefore, a user-friendly interface for accessing large, heterogeneous data repositories would make information more accessible and available to them. One of the popular approaches for solving this kind of data access problem is incorporating ontologies into information management.

An ontology concerns the problem of categorizing and representing concepts or objects and defining relationships among them. As those semantic features are introduced, the logical relationships among concepts can be recognized and interrupted by both human beings and machines. Related work, such as the development of the semantic web and related semantic application development has become a hot topic in recent years.

Currently, most large data sets are stored in traditional relational databases. Utilizing ontologies to access those databases can make full use of already existing data. Therefore, a requirement for establishing a connection between databases and ontologies has been raised.

This project established an architecture that integrates data from heterogeneous databases into an RDF data repository, and then maps the integrated dataset to concepts in an ontology. Through the concepts represented in the ontology, users can readily retrieve information from the federated database system.

There are two major contributions of this project. First, three typical challenges of database integration (data distribution, data duplication and database schema conflict) have been considered and addressed separately. The data distribution and data duplication challenge can be eliminated in an efficient way, which has an advantage compared to common approaches. Secondly, the connection between databases and ontologies has been addressed with flexible mapping tools, instead of only allowing mapping a table to a concept directly in some other application.

In this project we designed an architecture that revealed the advantage of integrating database and mapping information to ontology with RDF. With the results of this project, further development or adaption such as automatic mapping and higher level semantic features can be introduced into the framework.

Chapter 2: Background

2.1. Approaches to Database Integration

There are two major approaches to share and merge existing repositories. One popular solution involves data warehousing (Figure 2.1). The warehouse system extracts, transforms, and loads data (ETL) [33] from several sources into a single schema that can be queried. Architecturally, this offers a tightly coupled approach because the data reside together in a single repository at query-time. Problems with tight coupling can arise with the "freshness" of data, for example when an original data source gets updated, but the warehouse still contains the older data and the ETL process needs to be re-executed [1].

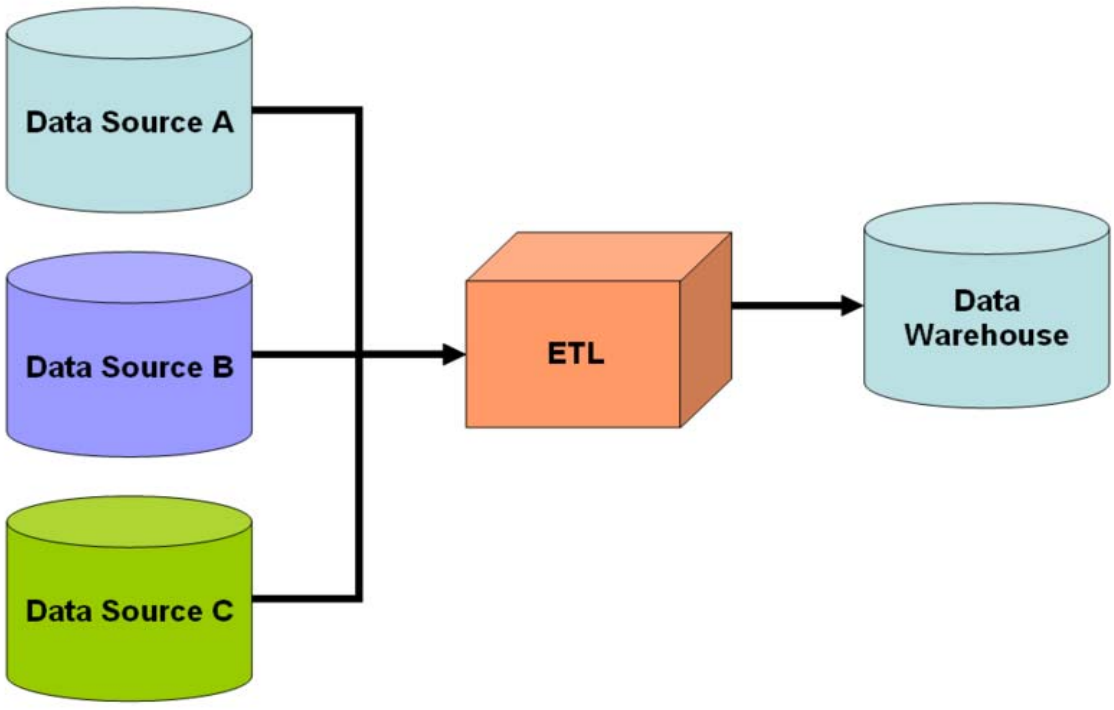

Figure 2.1: Database Integration into a Data Warehouse [1]

A second approach involves providing a uniform query interface over a "mediated" or virtual schema (Figure 2.2). This approach requires transforming end-user queries into specialized queries over the original databases. One can also term this process "view-based query-answering" because each of the data sources functions as a view over the (virtual) mediated schema. This approach is attractive due to the simplicity involved in answering queries over the mediated schema [2]. However, one must rewrite the view for the mediated schema whenever a new source gets integrated and/or an existing source changes its schema [1]. What is more, transforming end-user queries into specialized queries [11] and merging the result set can be very complicated.

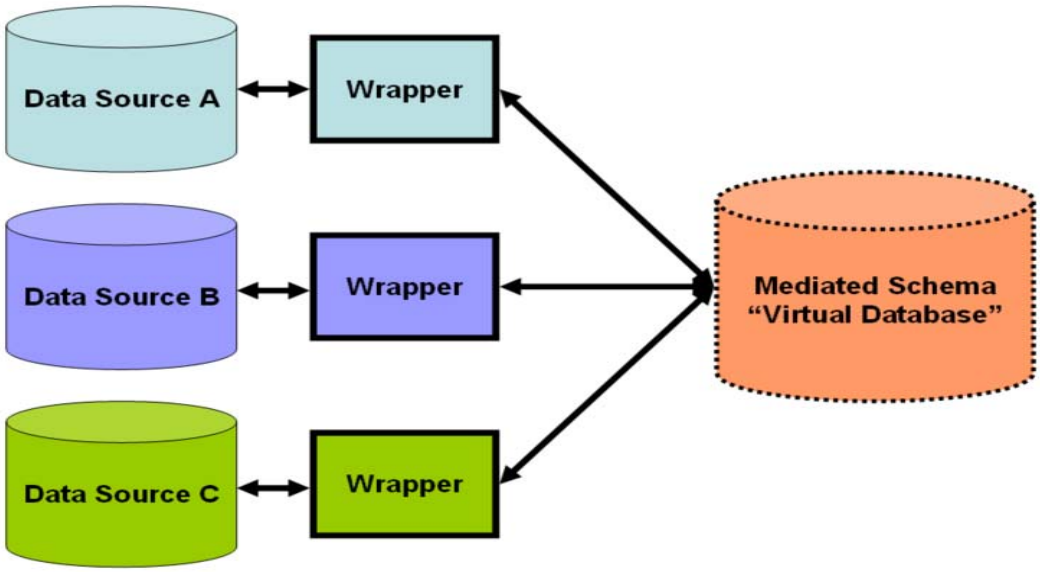

Figure 2.2: Database Integration with Mediated Schema [1]

In this research we implemented a very general yet powerful approach to the ETL methodology. Similar to the first approach mentioned above, this system extracts, transforms, and loads data (ETL) from several sources. The RDF (Resource Description Framework) Triplestore [34] is used as the data repository for deploying the integrated data.

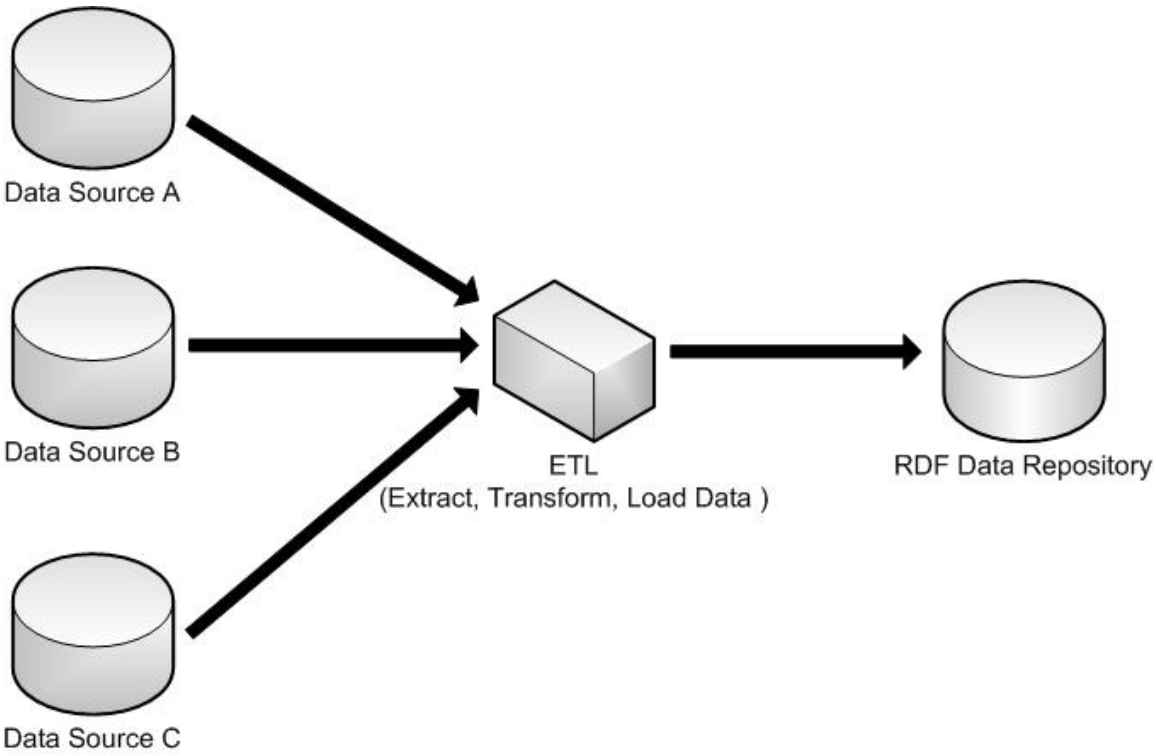

Figure 2.3: Database Integration Approach in This Research

2.2. The RDF Model and SPARQL

Resource Description Format (RDF) is a general purpose metadata standard developed by the W3 Consortium. It is aimed at enabling machines to understand information on the Web [4]. There are two basic elements in the RDF model [6] (Figure 2.4). The first one is the resource node, which has a unique description to differentiate the node from other nodes. The second one is the property, which has value that describes the resource. Each pair of relationships between a resource and property, or between two resources, can be expressed by a "triple". The triple is a statement that has three parts: the subject, the predicate and the object. With multiple "triples", a user can describe a complete RDF model. The repository that stores the triples is called the "triple store" [5].

RDF is a very flexible tool that can describe data containing semantic features. In this research, we use the Jena triple store (TDB) to implement the data repository. The reason we select the RDF model to realize the data repository lies in its power to eliminate duplication and distribution among heterogeneous databases, which will be addressed in detail in chapter 5.

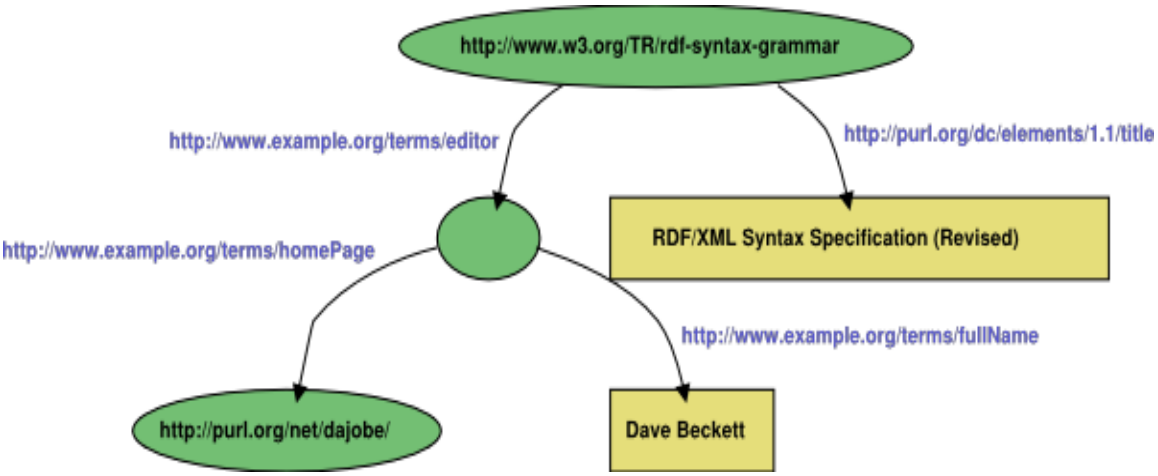

Figure 2.4: RDF Model Example [4]

SPARQL is an RDF query language. It was standardized by the RDF Data Access Working Group (DAWG) of the World Wide Web Consortium [7], and is considered a key semantic web technology. On 15 January 2008, SPARQL became an official W3C Recommendation [8]. Using SPARQL, the information stored in an RDF model can be searched efficiently.

The RDF model and SPARQL are used to realize the data repository and its metadata, as well as the mapping between the data repository and domain ontology.

2.3. Domain Ontology

In computer science and information science, an ontology [20] is a formal representation of a set of concepts within a domain as well as the relationships among those concepts. It is used to reason about the properties of that domain and may also be used to define the domain [35]. An ontology provides a

shared vocabulary which can be used to model a domain—that is, the type of objects and/or concepts that exist, and their properties and relations [14] [15] [16].

A domain ontology (or domain-specific ontology) [17] [18] models a specific domain, or part of the world. It represents the particular meaning of terms as they apply to that domain [14].

Figure 2.5 is an example of a medical domain ontology.

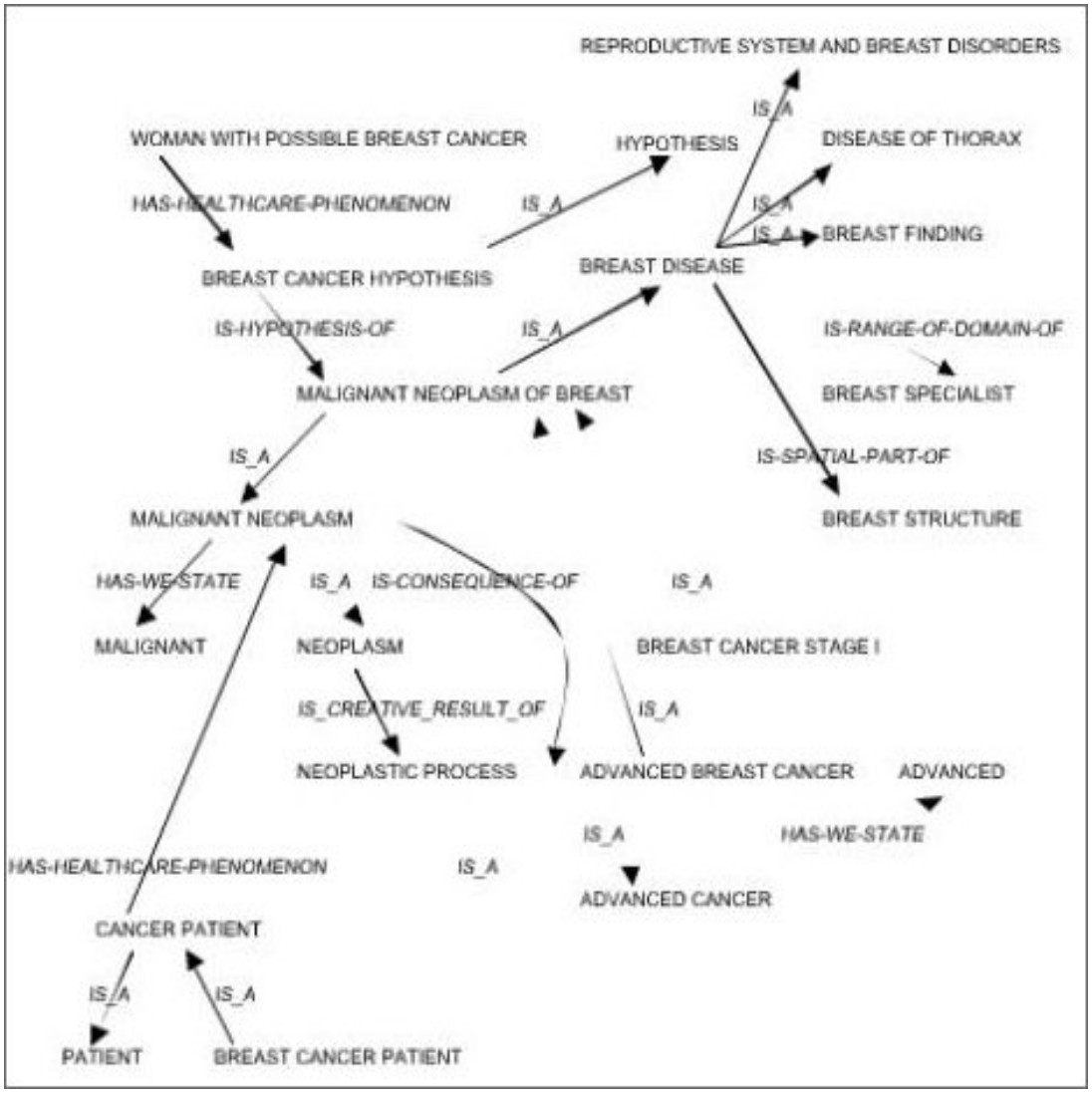

Figure 2.5: Domain Ontology Example [21]

An ontology also defines a vocabulary that describes the relationship among concepts. If ontologies can be fully utilized in the information retrieval field, the quality of searching can be improved significantly because the underlying logic and semantics will be taken into consideration. However, most current databases are relational in structure. How to effectively map the data in relational databases to ontologies is an interesting and important topic. In addition, the RDF model is also used to map the data in data repositories to ontologies in order to establish a connection between the database and the ontology.

Chapter 3: System Architecture Overview

3.1 System Deployment

Our software system is implemented using a standard Client-Server Architecture [22] (Figure 3.1). The end-user interacts with the system via a browser. A server that supports Java Servlets handles HTTP requests, and after retrieving the requested data from backend servers, it sends the results to the user.

On the client side, the user interacts with the system through dynamic web pages under the control of JavaScript. The web page identifies and parses a user's commands and sends them to a server that supports Java Servlets. While waiting for results, the asynchronous feature of Ajax allows the user to initiate other tasks on the web page, instead of just waiting. Once the result of a request is returned to the client browser in XML format, the web page parses the response and displays it on the user's browser screen dynamically.

The middleware server software was developed using Java Servlet technology to handle requests sent from a user's browser. After indentifying the type of request, the system initializes the appropriate module such as data dumping, ontology mapping, querying and so on, to handle different kinds of request. Once the module completes the required action, the result set is parsed into XML and sent back to the end-user's web interface.

The RDF Triple Store (TDB of Jena) is the data repository system that stores all the data that comes from the heterogeneous databases. Under control of the Triple Store server, the data is migrated into the form of RDF triples and dumped into the Triple Store. In this process, the distribution, duplication and conflict between heterogeneous databases is eliminated. The system will retrieve information from the data repository directly upon request.

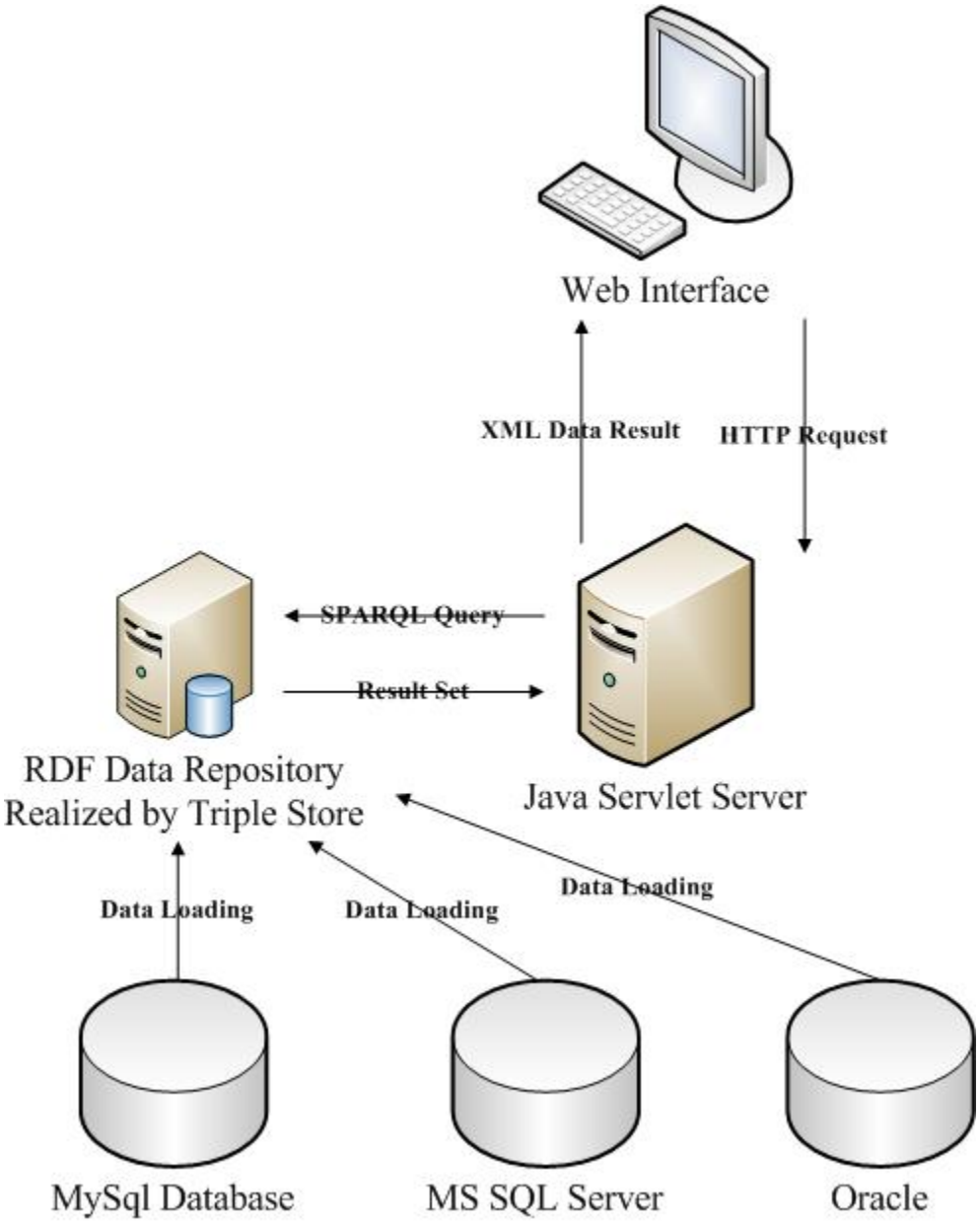

Figure 3.1: System Deployment

3.2 System Layers Model

From the perspective of system layers, our system supports the following functions: integrating databases into the RDF Triple Store scheme; mapping the data repository's content to a concept in an ontology; and processing end-user queries based on the terms defined in the ontology. Each function

is implemented with a different module to handle special tasks. Therefore, the system can be divided into 4 layers based on the task its supports and the data it generates. In sum, the system architecture can be represented by the four layers shown in Table 3.1.

System Layers		
Layer	**Components and Modules**	**Data**
Layer 4 Information Retrieval Layer	1. Ontology Tree 2. Query Construction Table	Result set generated according to the relation between ontology concepts
Layer 3 Ontology Definition Layer	1. Ontology Tree 2. Data Repository Metadata Tree 3. Data Repository to Ontology Mapping Tools	Ontology concept defined by the data stored in the data repository
Layer 2 Data Repository Layer	1. Data Repository 2. Metadata Tree 3. Data Dumping Tools	Integrated data
Layer 1 Heterogeneous Database Layer	1. Heterogeneous Databases	Distributed and duplicated heterogeneous data

Table 3.1: System Layers

3.2.1 Description and analysis of system layers

Layer 1: This is the distributed data layer. In this layer, data is distributed among heterogeneous databases; therefore, duplication and data conflicts may exist.

Layer 2: This is the data repository layer. In this layer, the data repository has been realized as an RDF Triple Store and the metadata of the data repository will be managed by the RDF model. This data repository interfaces with the RDF model and processes SPARQL queries in order to simulate the

basic operations of a read-only relational database. Therefore the system provides a repository to integrate data from heterogeneous databases. During the ETL data dumping process that creates the Triple Store repository, inconsistencies and data conflicts are eliminated using dumping tools. One of the most difficult problems in database integration, the duplication of data, can be easily solved by the RDF model. Therefore integrated data will be realized in this layer. Furthermore, the relationship between the data repository's tables can be defined by the user, a property that is useful for constructing the query strategy for higher level layers.

Layer 3: This is the ontology concept definition layer. Based on the theory of domain ontologies, terms in an ontology tree can be defined by picking attributes from the data repository's table and setting constrains for them. In this way, the concepts in an ontology tree can be represented by the values in the data repository.

Layer 4: This is the information retrieval layer. In this layer, the end-user web interface will send the terms selected by the user to the middleware Servlet server. The server checks the underlying relationship between the terms by comparing the relationships of columns associated with them. It then constructs the query based on the relationship information between columns and constrains. After the query has been executed, the result set is returned to the web interface and displayed.

3.3 System Interface Introduction

3.3.1 Data integration interface

See figure 3.2, a user with administrative privileges can load the metadata information of heterogeneous databases into the Data Server Tree (1).

During the data extraction process, distribution issues are eliminated by Database to Data Repository Mapping Table (2) and conflicts are cleared by Database Structure Adaptation Tools (3).

The integrated data is deployed in RDF triple store, and the metadata is represented by the Data Repository Metadata Tree (4).

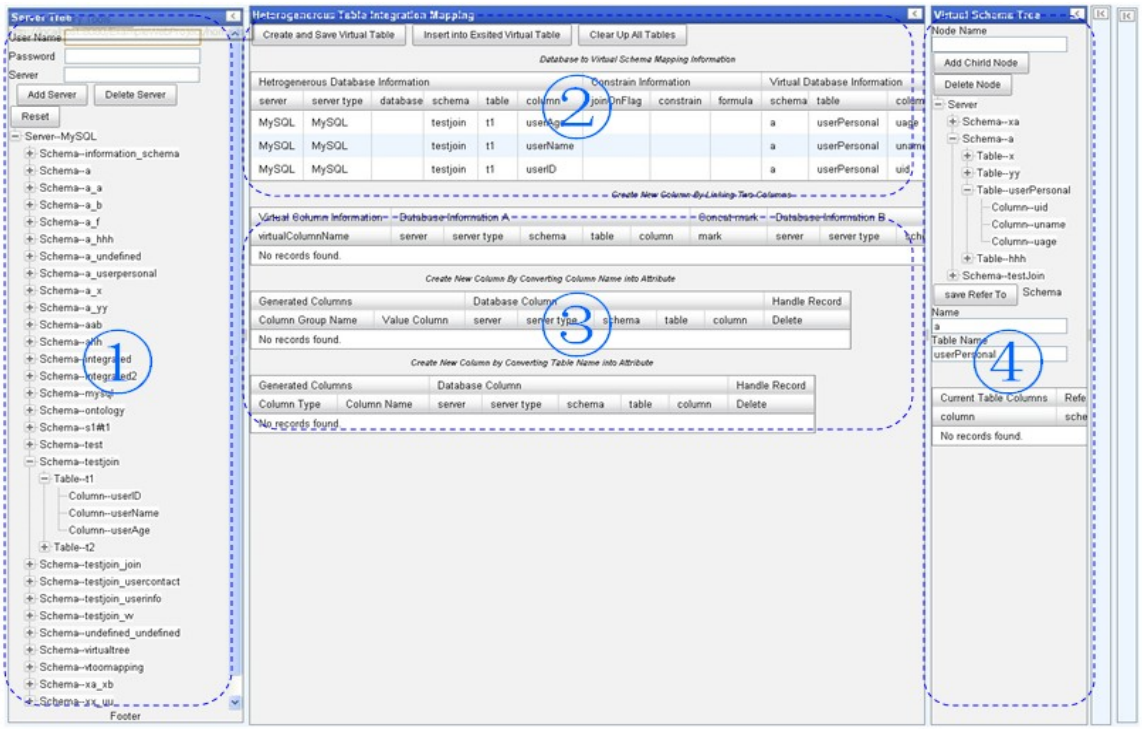

Figure 3.2: Data Integration Interface

3.3.2 Ontology concept definition interface

As illustrated in Figure 3.3, the administrator can a pick attributes from the Data Repository Metadata Tree (2) to define term in the Ontology Tree (1), set constraints for them on the Data Repository to Ontology Mapping Table (3). In this way, the concepts in the ontology tree are mapped to values in the data repository.

3.3.3 Query tool interface

As represented in Figure 3.4, users can pick terms from the Ontology Tree (1) and search for the associated information via the Query Table (2). The server checks the underlying relationship between the terms to construct the query.

After the query has been executed, the Result Set (3) is sent to the web interface and displayed in the proper way.

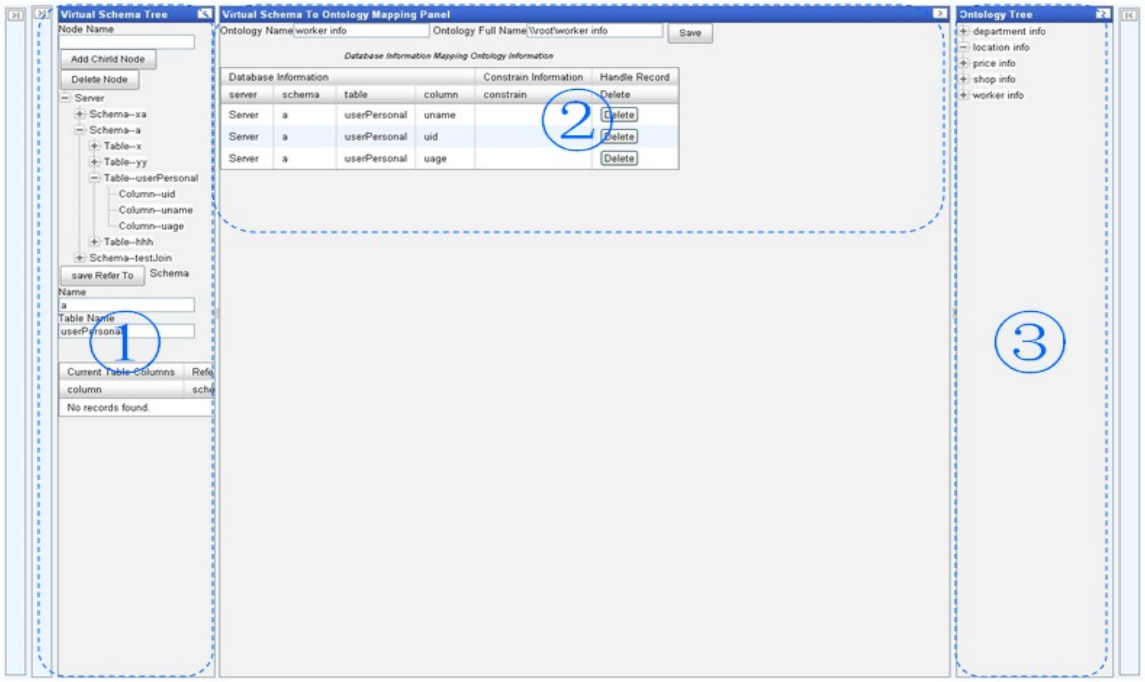

Figure 3.3: Ontology Concept Definition Interface

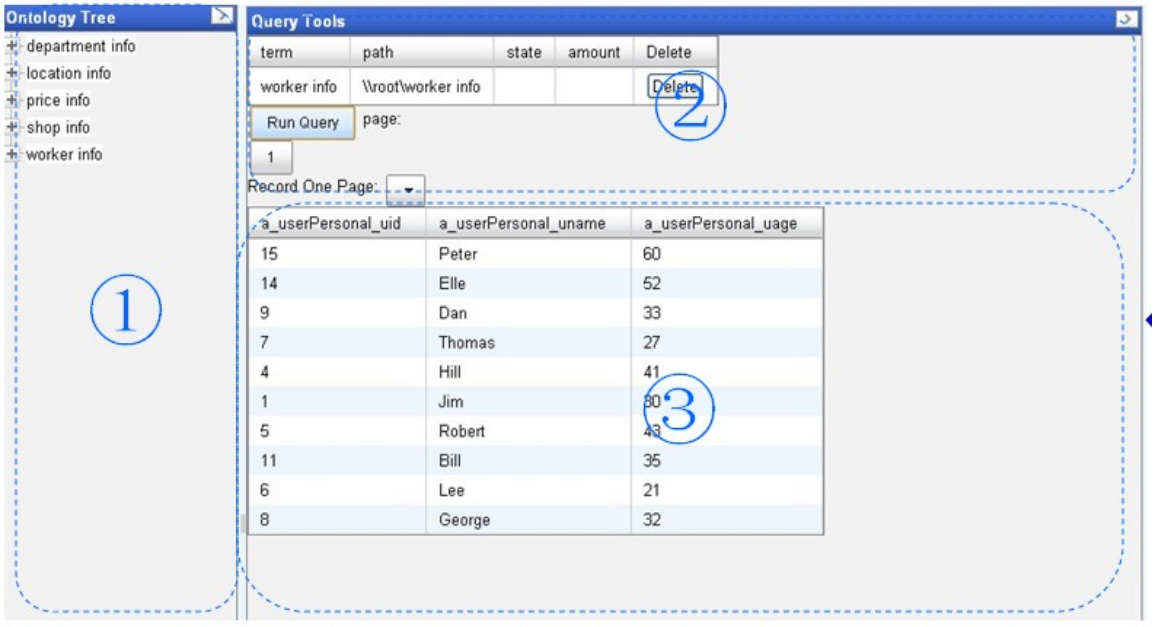

Figure 3.4: Query Tool Interface

Chapter 4: Data Repository and Metadata Model

4.1 Design of the Data Repository Model

4.1.1 Design considerations

In Relational Databases (RDB), one row in a table is a single instance of a record, and each column is an attribute of this instance. Similarly, a table is a group of instances that describe a certain concept. The structure or schema of several tables forms a database schema, which usually represents the information in a project or certain application domain. This model is simple, generalizes well, and is well understood by database developers. We utilize the relational model for our data repository.

For each table in the data repository, a unique RDF model is created. The schema that the table belongs to is marked in the name of the model. When the system needs to do inter-table search, the needed tables are located by name, and then unioned together. By dividing the data repository into multiple independent models, the efficiency of search can be improved, since storing all data in a single model would take more time to search. Another advantage is that it provides an automatic solution to eliminating the duplication of records during data integration, which will be addressed in detail in the following chapter.

4.1.2 Implementation of data repository model

To represent the data resources and their properties in the data repository model (Figure 4.1), the triple store triples shown in Table 4.2 are used.

Subject	**Predicate**	**Object**	**Description**
Server Node	hasSchemas	Schema Node	Schemas contained in the server
Schema Node	schemaName	*Name of schema*	The name of this schema node
Schema Node	hasTables	Table Node	Tables contained in the schema

Table Node	tableName	*Name of table*	The name of this table node
Table Node	hasColumn	Column Node	Columns contained in the table
Column Node	column(*Name*)	*Value of Column*	The name of this column node

Table 4.1: Triplestore Triples in the Virtual Database

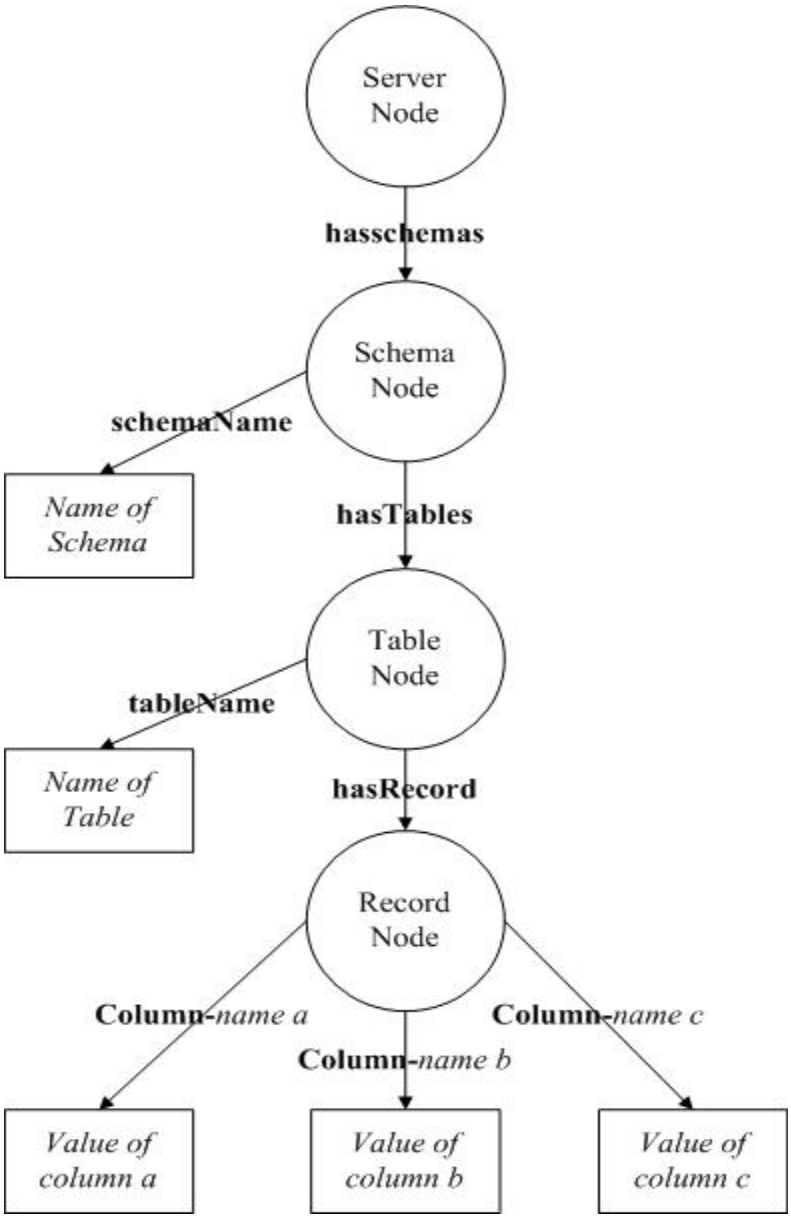

Figure 4.1: Data Repository Model

Moreover, to prepare for table union operations during inter-table search, the conflict between node descriptions should be avoided. Because different tables in different schemas may have the same name, when two models merge together, the system may not differentiate them, which will result in messed up records. Therefore, it is necessary to mark the table name uniquely in the model. The same problem exists in the situation that different tables have different columns that have the same name. To solve such potential misinterpretations, a good choice is to extend the description of nodes by including the schema name and table name to give each node a unique name. In this way, the node description in the model becomes that shown in Table 4.2.

Node type	Description
Schema Node	*Schema Name*
Table Node	*Schema Name / Table Name*
Column Node	*Schema Name / Table Name / Column Name*
Record Node	*Schema Name / Table Name / Primary Key*

Table 4.2: Node Description in Model

4.2 Design of the Data Repository Metadata Model

4.2.1 Design considerations

To integrate the data of heterogeneous databases, a global schema is needed to eliminate inconsistencies between metadata. There are at least two requirements for designing the data repository metadata model. First, the metadata should be organized systematically. A good choice would be to represent the metadata in a tree structure. Users are familiar with this kind of structure. Second, when the system needs to retrieve metadata information, the data must be obtained efficiently. For example, the “join on” operation is important when a user selects columns from different tables to

generate the result. Therefore, defining and checking relationships between columns must be done efficiently.

In summary, the following metadata information has to be represented: Schemas, Tables, Columns, Column Relationships.

4.2.2 Implementation of the data repository metadata model

To represent the resource and properties in the data repository metadata model (Figure 4.2), the triple store triples shown in Table 4.3 need to be introduced.

Subject	Predicate	Object	Description
Server Node	hasSchemas	Schema Node	Schemas contained in the server
Schema Node	schemaName	Name of schema	The name of this schema node
Schema Node	hasTables	Table Node	Tables contained in the schema
Table Node	tableName	Name of table	The name of this table node
Table Node	hasColumn	Column Node	Columns contained in the table
Column Node	columnName	Name of Column	The name of this column node
Column Node	refersTo	Column Node	Column A refers to column B, indicating a potential join

Table 4.3: Triple Store Triples Representing Metadata

In the RDF model, two nodes can have multiple predicates between them. This feature makes multiple “refersTo” relationships between nodes possible. For the reason previously described, the description of table and column nodes is represented by Schema_name.Table_name and Schemaname.Tablename.Columnname in the RDF model to avoid the duplicated node description.

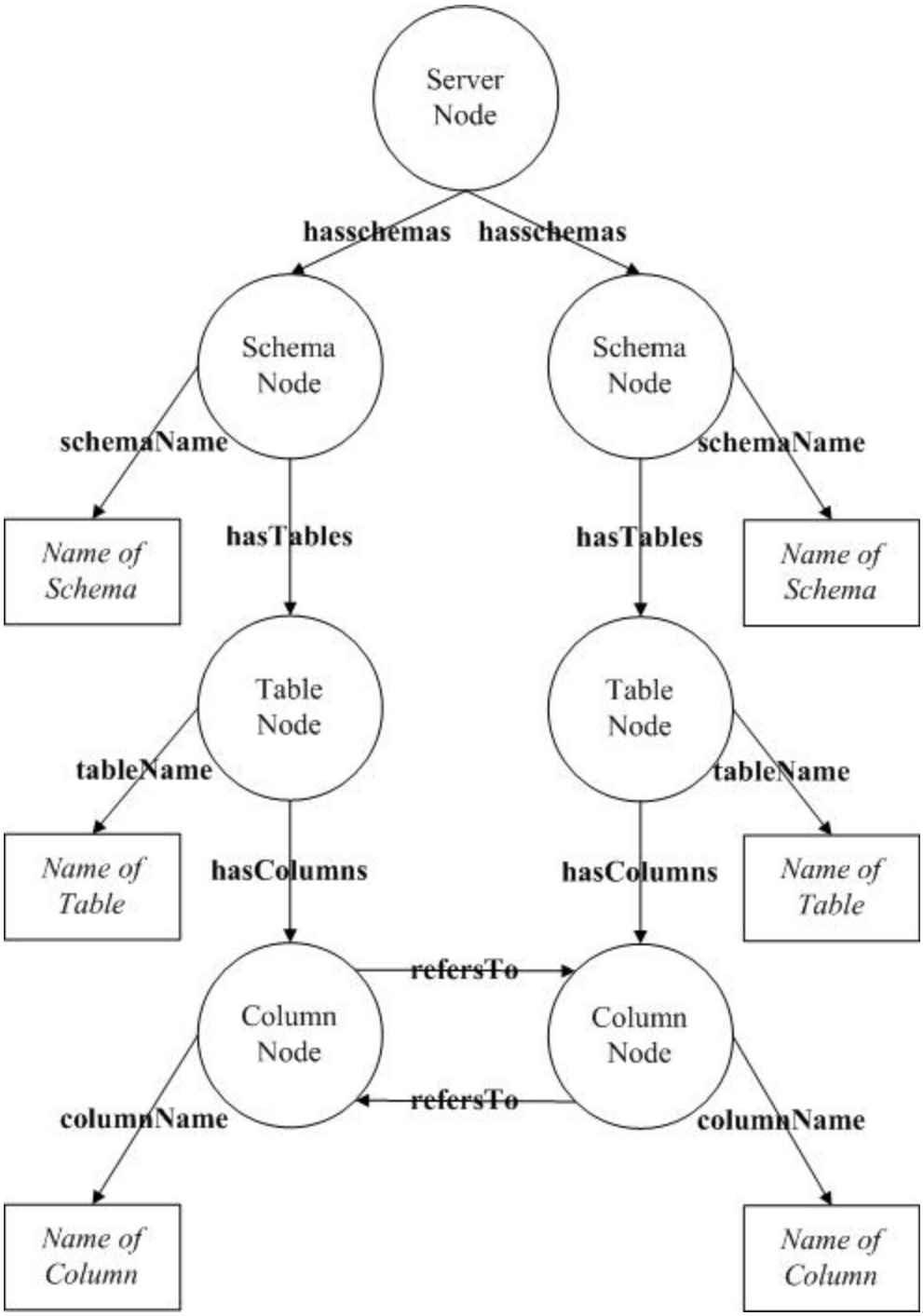

Figure 4.2: Data Repository Metadata Model

4.3. Operations Supported through SPARQL

There are two situations when the system needs to use metadata. The first one is when the metadata of the data repository needs to be loaded for browsing by users. The second is when, in the information retrieval layer, a query needs to union multiple tables and it is necessary to check relationships between tables.

4.3.1 Exploring the metadata

When creating or loading the data repository's schema in the form of a tree structure, the metadata should be requested and displayed dynamically. The following SPARQL queries can load the metadata information at the level required:

a. Get schemas in a server

When loading the schema names in a server, need the name of the server

```
PREFIX j.0: <http:// metadata#>
SELECT ?schemaName
WHERE
{
        ?serverNode j.0:serverName "Name of the server".
        ?serverNode j.0:hasSchemas ?schemaNode.
        ?schemaNode j.0:schemaName ?schemaName.
}
```

b. Get tables in a schema

When loading the table names in a schema, need the name of the server and schema

```
PREFIX j.0: <http:// metadata#>
SELECT ?tableName
WHERE
{
        ?serverNode j.0:serverName "Name of the server".
        ?serverNode j.0:hasSchemas ?schemaNode.
        ?schemaNode j.0:schemaName "Name of the schema".
        ?schemaNode j.0:hasTables ?tableNode.
        ?tableNode j.0:tableName ?tableName.
}
```

c. Get columns in a table

When loading the column names in a table, need name of the server, schema and table

```
PREFIX j.0: <http:// metadata#>
SELECT ?tableName
WHERE
{
        ?serverNode j.0:serverName "Name of the server".
        ?serverNode j.0: hasSchemas ?schemaNode.
        ?schemaNode j.0:schemaName "Name of the schema".
        ?schemaNode j.0:hasTables ?tableNode.
        ?tableNode j.0:tableName "Name of the table".
        ? tableNode j.0:hasColumns ?columnNode .
        ? columnNode j.0:columnName ?columnName.
}
```

4.3.2 Check relationships between columns

When constructing a SPARQL query, after obtaining the list of selected tables, the system has to check the underlying relationship between them.

a. Check to see if there are any relationships between tables

```
PREFIX j.0: <http:// metadata#>
SELECT ?columnName1 ?columnName2
WHERE
{
        ?schemaNode1 j.0:schemaName "Name of the schema1".
        ?schemaNode2 j.0:schemaName "Name of the schema2".
        ?schemaNode1 j.0:hasTables ?tableNode1.
        ?schemaNode2 j.0:hasTables ?tableNode2.
        ?tableNode1 j.0:tableName "Name of the table1".
        ?tableNode2 j.0:tableName "Name of the table2".
        ?tableNode1 j.0:hasColumns ?columnNode1.
        ?tableNode2 j.0:hasColumns ?columnNode2.
        ?columnNode1 j.0:referTo  ?columnNode2.
        ?columnNode1 j.0:columnName ?columnName1.
        ?columnNode2 j.0:columnName ?columnName2.
}
```

b. Get the referred-to column's location information

It will take two steps to get the referred-to column's location information.

Step1: Get the description of the referred column's node.

```
PREFIX j.0: <http:// metadata#>
SELECT ?columnName ?referToURL
WHERE
```

```
{
        ?serverNode j.0:serverName “Name of the server”.
        ?serverNode j.0:hasSchemas ?schemaNode."+
        ?schemaNode j.0:schemaName “Name of the schema”.
        ?schemaNode j.0:hasTables ?tableNode."+
        ?tableNode j.0:tableName “Name of the table”.
        ?tableNode j.0:hasColumns ?columnNode ."+
        ?columnNode j.0:columnName ?columnName."+
        OPTIONAL {?columnNode j.0:referTo ?referToURL.}"+
}
```

Step2: Get the location information via the description of referred column node.

```
PREFIX j.0: <http://metadata#>
SELECT ?serverName ?schemaName ?tableName ?columnName
WHERE
{
        ?serverNode j.0:serverName ?serverName .
        ?serverNode j.0:hasSchemas ?schemaNode.
        ?schemaNode j.0:schemaName ?schemaName.
        ?schemaNode j.0:hasTables ?tableNode.
        ?tableNode j.0:tableName ?tableName.
        URL j.0:columnName ?columnName.
}
```

4.3.3 Retrieval information from data repository

When retrieving data from the data repository, there are different situations. The data may come from a single table or multiple tables, and the constraints on attributes may also be applied:

a. From single table

For example, select columns X,Y,Z from a table

```
PREFIX j.0: <http://virtualDB#>
SELECT ?X ?Y ?Z
WHERE
{
        ?serverNode j.0:serverName “Name of the server”.
        ?serverNode j.0:hasSchemas ?schemaNode.
        ?schemaNode j.0:schemaName “Name of the schema”.
        ?schemaNode j.0:hasTables ?tableNode.
        ?tableNode j.0:tableName “Name of the table”.
        ?tableNode j.0:hasRecord ?recordNode .
        ?recordNode j.0:column-X ?X.
        ?recordNode j.0:column-Y ?Y.
        ?recordNode j.0:column-Z ?Z.
}
```

b. From multiple tables

For example, select from table1 and table2, and join on the key table1:A, table2:X

```
PREFIX j.0: <http://virtualDB#>
SELECT ?A ?B ?C ?Y ?Z
WHERE
{
        ?serverNode1 j.0:serverName "Name of the server".
        ?serverNode1 j.0:hasSchemas ?schemaNode1.
        ?schemaNode1 j.0:schemaName "Name of the schema".
        ?schemaNode1 j.0:hasTables ?tableNode1.
        ?tableNode1 j.0:tableName "Name of the table".
        ?tableNode1 j.0:hasRecord ?recordNode1.
        ?serverNode2 j.0:serverName "Name of the server".
        ?serverNode2 j.0:hasSchemas ?schemaNode2.
        ?schemaNode2 j.0:schemaName "Name of the schema".
        ?schemaNode2 j.0:hasTables ?tableNode2.
        ?tableNode2 j.0:tableName "Name of the table".
        ?tableNode2 j.0:hasRecord ?recordNode2.
        ?recordNode1 j.0:column-A ?A.
        ?recordNode1 j.0:column-B ?B.
        ?recordNode1 j.0:column-C ?C.
        ?recordNode2 j.0:column-X ?A.
        ?recordNode2 j.0:column-Y ?Y.
        ?recordNode2 j.0:column-Z ?Z.
}
```

c. Query with constraints

For example, a constraint on column "X" indicates that the value should be bigger than 0.

```
PREFIX j.0: <http://virtualDB#>
SELECT ?X ?Y ?Z
WHERE
{
        ?serverNode j.0:serverName "Name of the server".
        ?serverNode j.0:hasSchemas ?schemaNode.
        ?schemaNode j.0:schemaName "Name of the schema".
        ?schemaNode j.0:hasTables ?tableNode.
        ?tableNode j.0:tableName "Name of the table".
        ?tableNode j.0:hasRecord ?recordNode .
        ?recordNode j.0:column-X ?X.
        ?recordNode j.0:column-Y ?Y.
        ?recordNode j.0:column-Z ?Z.
}
FILTER (xsd:double ( ?X)>0).
```

4.4. Summary

As described above, the metadata model and data repository model are flexible enough to simulate the basic operations of relational databases. At the same time, the data can be stored without losing important relational database logic. With these two models, the metadata and data content of integrated data can be organized and regulated systematically.

Chapter 5. Database Integration

Database integration is the process of merging data from heterogeneous databases into a single data repository. Due, in part, to differences among the schema of the databases to be merged, simply merging the data will result in a loss of information and can result in several types of inconsistencies. These inconsistency problems can be classified into data distribution, data duplication, and database structure conflict. Utilizing the RDF model and the following integration strategies, the potential inconsistencies can be eliminated.

5.1 Data Distribution

The existence of distributed data is common in the database world. The data that describes an entity or concept may be stored across multiple databases. Moreover, those databases may be of different types, for example: MySQL, MS SQL Server, Oracle, etc. To utilize distributed data, a method is needed that can access multiple data sources simultaneously. This project takes the approach of loading data from different databases into a single data repository with a single global schema [9].

5.1.1 Distribution situations

Even though the distribution of data relating to a single entity can be very complex in the real world, the possibilities can be categorized into two basic ones: row distribution and column distribution. Table 5.1 illustrates the primary aspects of these possibilities.

a. Row distribution

In the example shown in Table 5.1, three tables describe the same entity, a patient information dataset. However, compare the table from Database 1 to the table from Database 2. They contain columns representing the same information, but the content of the table, patient records, has been stored in two different databases. This is an example of data inconsistency related to row distribution issues.

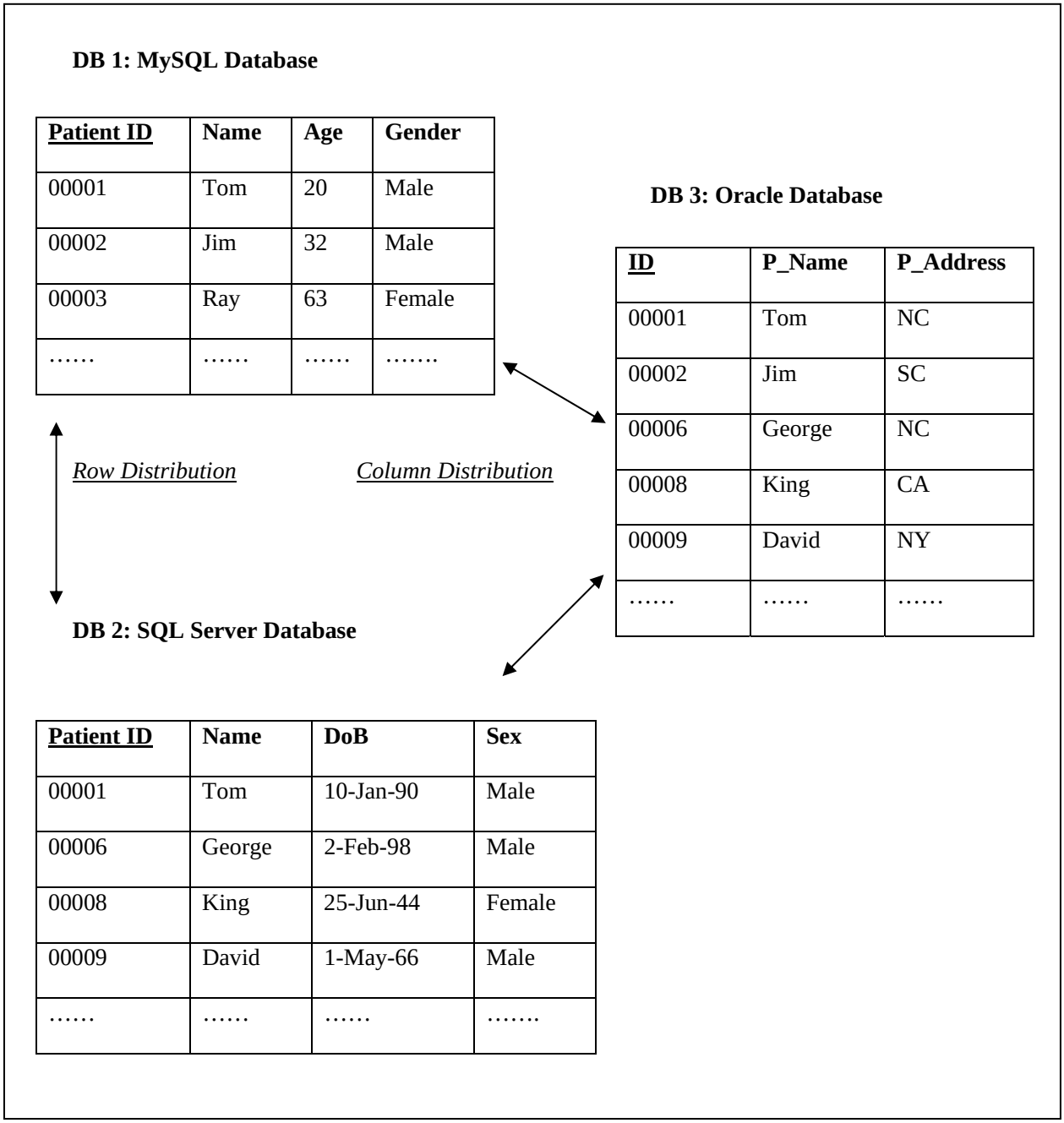

Table 5.1: Data Distribution Example

b. Column distribution

On the other hand, compare the tables from Database 1, Database 2, and Database 3. Many records from the tables contain the same primary key values (as indicated by an underscore), indicating that

those attributes belong to the same instance; however, some attributes of the same entity have been distributed to different databases. For example, if the tables from Database 1 and Database 3 were to be viewed as components of a distributed database, when users search for the record for Patient_ID = 00001, they will get two rows that relate to the same entity as shown in Table 5.2.

Source	**ID**	**Name**	**Age**	**Gender**	**Address**
DB 1	00001	Tom	20	Male	NULL
DB 3	00001	Tom	NULL	NULL	NC

Table 5.2: Example of Row Duplication in Search Results

It is usually the case in relational databases that each row represents an instance of a record describing a single entity. Queries from heterogeneous databases can lead to problems in data management. This is an example of a problem related to the column distribution issue.

To use information from these related, yet distributed, tables, a logically merged dataset is expected, which means both the “columns” and “records” should be a “union” of different tables. Ideally, the virtual table should have the table structure and content illustrated in Table 5.3.

Patient ID	**Name**	**Age**	**Gender**	**Address**
00001	Tom	20	Male	NC
00002	Jim	32	Male	SC
00003	Ray	63	Female	
00006	George	18	Male	NC
00008	King	29	Female	CA
00009	David	25	Male	NY
……	……	……	……	……

Table 5.3: Integrated Table of Data Distribution

5.1.2 Solutions for distribution problems

To eliminate the potential issues associated with data distributions, data that comes from heterogeneous database can be loaded into an RDF data repository, where row distribution and column distribution are then solved with the following approach:

a. Row Distribution

Since all the data will be stored in a single data repository, loading data from heterogeneous databases into data repository solves this problem directly. Each record is still a unique record in the RDF table.

b. Column Distribution

In the RDF model, record nodes that belong to the same instance have the same description. Therefore, when records that come from different database can be loaded into the same model, the record nodes with the same description will merge into a single one. As a result, the attributes that belong to different record nodes will also be linked to the same record node. In this way, the column set that is the union set of different databases' columns will be generated in the data repository. Therefore, when the data is loaded from heterogeneous databases into the same RDF model, the information in different tables will merge automatically.

Using Table 5.1 as an example, the following SPARQL query can be used to simulate the "SELECT * FROM Table1 where ID = "00001"" query of SQL:

```
SELECT ?A ?B ?C ?D
FROM{
        ?x T1/Patient_ID "00001".
        ?x T1/Gender ?A
         ?x T1/Name ?B.
         ?x T1/Age ?C.
         ?x T1/Address ?D.
}
```

Instead of returning a result set like that shown in Table 5.2, which has multiple distributed rows of the same record, the self-merged RDF model will have a result set that guarantees the information belonging to the associated records are merged into a single row as shown in Table 5.4.

Record	ID	Name	Age	Gender	address
	00001	Tom	20	Male	NC

Table 5.4: Result from RDF Triple Store

In this way, the attributes distributed among multiple databases but relating to the same entity can be merged together.

5.2 Data Value Duplication

In the data integration process, the data from different databases may contain redundant and duplicate records. This kind of redundancy can result in inaccuracies in calculations or statistics. Therefore, data duplication in an integrated data set should be eliminated.

5.2.1 Duplication situations.

Duplication situations can be classified into row duplication and column duplication. It can also be a combination of the two.

a. Row Duplication

In row duplication, two rows that come from different databases may be exactly the same, which means they are the same logical record. Consider Table 5.1 as an example. If the records of Database 1 and Database 2 are combined together directly, there are two rows describing the same record with a primary key value of ‘00001’.

b. Column Duplication

Column duplication also happens in this example from Table 5.1. Both Database 1 and Database 2 have the columns named ‘Patient_ID’ and ‘Name’. This is a situation where different tables have redundant columns referring to the same attribute.

5.2.2 Solutions for duplications.

To solve the two duplication problems, the RDF model provides an efficient solution for both row duplication and column duplication.

a. Row Duplication

Instead of checking rows in pairs of tables one by one, which results in an M*N complexity, for table lengths M and N, merging data into the same RDF model in the data repository can solve this problem directly. In the RDF model, every resource node is unique, which means that as long as the descriptions of several nodes are the same, the node will merge into a single one. Therefore, even if repeated rows have been added into the RDF model, only a single row will be stored. Therefore, the row duplication problem is easily solved.

b. Column Duplication

From the example above, we can see that the column duplication has also been eliminated via the RDF model. Because the value of each column is an RDF property node, as described in the previous paragraph, repeated nodes will not existed in the RDF model. Therefore, as long as the column has been assigned the same description and predicate, duplicated columns will also be merged into the same one. In this way, the column duplication issue has also been solved.

In summary, we can see that both the row distribution and column distribution issues can be solved. Therefore, using the RDF model can eliminate the duplication problem efficiently and safely, which is hard to achieve by other approaches.

5.3. Conflicts in Database Integration

Conflicts between database structures can be classified into six basic types according to the theory developed by Chiang Lee, Chia-Jung Chen and Hongjun Lu [10]. All of them can be solved in the data loading process. To deal with these different situations, our system allows users to constructs SQL queries to select data from the original source and generates a result set table having the same

structure as the table structure uses need. Different situations can be classified into the following categories. The corresponding solution for each has been provided.

a. Value-to-Value Conflict

Situation:

Columns in different tables are the same attribute, but the representations are different. For example, the different units (kilometer and mile), the different scale (4 point and 100 point grade), and the different definitions (prices, before or after tax).

Solution Example:

This kind of conflict comes from different representations of the same attribute. As long as the relation between the original attribute and the target attribute can be expressed by a computable functions, formulas can be applied to convert it.

For example, kilometer = 0.621*mile, 4 point scale = (100 point scale – 60) / 10, and price before tax = (price after tax) / (1-tax rate).

In the data loading process, conversion functions can be inserted into the column name field and the value can be converted, for example:

```
SELECT  ID, Name, (Grade(100 scale) – 60)/10 as Grade
FROM  Grades Table
```

In our implementation, the system allows the user to specify the conversion functions between original value and object value.

b. Attribute-to-Attribute Conflict

There are two typical sub-situations in attribute-to attribute conflicts:

Situation 1.

The same attribute may have different names in different tables, for example, Patient_ID and ID. Alternately, different attributes may have same name in different tables.

Solution Example 1:

The data from different databases will be loaded into the same RDF model; therefore, the consistency will be achieved by using the same RDF model's metadata.

Situation 2:

The value stored in a single column is spread over two or more columns in another table. For example, "Name" in objective table, "First_Name" and "Second_Name" in original Table.

Solution 2:

The solution here is to use "CONCAT"(MySQL) or "+"(SQL Server) mark to link values.

```
SELECT (First_Name+" "+Second_Name) as Name
FROM Name Table
```

c. Table-to-Table Conflict

Situation:

Different tables have different names, and their attributes may have duplication or distribution problems.

Solution Example:

In the previous section the elimination of data distribution and data duplication by using a global RDF schema was shown to eliminate this conflict in the data loading process.

Indy

Sales

Store	Dept	AvgSales
PineSt	Wmn	62500
WestRd	Wmn	75000
AndAve	Wmn	81500
PineSt	Men	50000
AndAve	Men	73500
PineSt	Toddler	41250
WestRd	Toddler	55000
AndAve	Toddler	68500

Chicago

AvgSales

Store	Wmn	Men	Boy	Girl
CedarRd	48500	35000	25500	⊥
CtrSq	55500	50000	32000	52500
WashSt	63500	58500	42250	58500
IllSt	78000	63250	50000	65500

Milw

LincAveSales

Dept	AvgSales	LCode
Girl	45000	1
Boy	55000	2
Men	65000	3
Wmn	80000	4

FreePkSales

Dept	AvgSales	FCode
Girl	35000	A
Men	48500	B
Wmn	55000	C

WashStSales

Dept	AvgSales	WCode
Boy	28500	B38
Men	46500	C18
Wmn	60000	X27

Table 5.5: Example of Conflicts in Database Integration [12]

The three tables shown in Figure 5.5 (Developed by Catharine M. Wyss and Edward L. Robertson [12]) are examples of the following three different conflicts. To illustrate the difference of the database structure at an abstract level, we take Table 5.6 as an example. The DB1, DB2 and DB3 are corresponds to “Indy”, “Chicago” and “Milw” separately.

DB1

A	B	C
a1	x	c1
a1	y	c2
a1	z	c3
a2	x	c4
a2	y	c5
a2	z	c6
a3	x	c7
a3	y	c8
a3	z	c9

DB2

A	X	Y	Z
a1	c1	c2	c3
a2	c4	c5	c6
a3	c7	c8	c9

DB3

Table a1

B	C
x	c1
y	c2
z	c3

Table a2

B	C
x	c4
y	c5
z	c6

Table a3

B	C
x	c7
y	c8
z	c9

Table 5.6: Abstract Examples of Conflicts in Database Integration

d. Value-to-attribute conflict

Situation:

In the example of Table 5.5, there is a value-to-attribute conflict between Table “Indy” and Table “Chicago”. In table Indy, the column “Dept” has values Wmn and Men. However, in table “Chicago”, each department has a column of its own. Therefore, value-to-attribute conflict can be described as: the values of an attribute in a table have been expressed as different columns in another table.

Solution Example:

In table 5.6, table DB1 (has the same structure as "Indy" in Table 5.5) has column A, B, C; table DB2 (has the same structure as "Chicago" in Table 5.5) has column A, X, Y, Z; DB1.B has value x, y, z which becomes the column X, Y, Z in DB2.

1. From attributes to values (DB2 to DB1):

```
SELECT a1, "DB2.X" as DB1.B, DB2.X as DB1.C from DB2
UNION
SELECT a1, "DB2.Y" as DB1.B, DB2.Y as DB1.C from DB2
UNION
SELECT a1, "DB2.Z" as DB1.B, DB2.Z as DB1.C from DB2
```

2. From values to attributes(DB1 to DB2)

```
SELECT DISTINCT DB1.A,
   (SELECT DB1.C from DB1 where DB1.A = T.A and DB1.B = 'DB2.X') as "DB2.X",
   (SELECT DB1.C from DB1 where DB1.A = T.A and DB1.B = 'DB2.Y') as "DB2.Y",
   (SELECT DB1.C from DB1 where DB1.A = T.A and DB1.B = 'DB2.Z') as "DB2.Z"
FROM DB1 as T
```

e. Value-to-table conflict

Situation:

In the example from Table 5.5, there is a value-to-table conflict between Table "Indy" and Table "Milw". In Table "Indy", the "Store" column has different store location values. In table group "Milw", each store has its own table. Therefore, value-to-table conflict is the situation in which values in one attribute are expressed as different tables in another schema.

Solution Example:

In table 5.6, table DB1 (has the same structure as "Indy" in Table 5.5) has columns A, B, C. And the column DB1.A has value a1, a2, a3, which is the name of tables in Schema DB3. In Schema DB3 (a group of tables, has the same structure as "Milw" in Table 5.5), table a1 has columns B and C, which is the same attribute as B, C in DB1. It is the same cases for table a2 and a3. The conversion between them can be described as follows.

1. From table to value (DB3 to DB1)

SELECT “*a1*” as A, DB3.a1.B as “*DB1.B*”, DB3.a1.C as “*DB1.C*”……from DB3.a1
UNION
SELECT “*a2*” as A, DB3.a2.B as “*DB1.B*”, DB3.a1.C as “*DB1.C*”……from DB3.a1
UNION
SELECT “*a3*” as A, DB3.a3.B as “*DB1.B*”, DB3.a1.C as “*DB1.C*”……from DB3.a1

2. From value to table (DB1 to DB3)

Write different SQL for each table to separate them into different tables

SELECT DB1.A, DB1.C from DB1where DB1.A = “*a1*”
SELECT DB1.A, DB1.C from DB1where DB1.A = “*a2*”
SELECT DB1.A, DB1.C from DB1where DB1.A = “*a3*”

f. Attribute-to-table conflict

Situation:

In the example of Table 5.5, this kind of conflict exists between Table “Chicago” and Table “Milw”. In this situation, attributes in one table have been expressed as different tables in another schema.

Solution Example:

In table 5.6, table DB2 (has the same structure as “Chicago” in Table 5.5) has column A, X, Y, Z, in which the X, Y, Z are the value of column B in Schema DB3 (a group of tables, has the same structure as “Milw” in Table 5.5). Moreover, in DB2, the column A’s value is the name of tables in DB3.

1. From attributes to tables (DB2 to DB3)

Write different SQL to distill the information for table DB3.a1.

SELECT “*DB2.X* “as B, DB2.X as C from DB2 where DB2.A = “*a1*”
UNION
SELECT “*DB2.Y* “as B, DB2.Y as C from DB2 where DB2.A = “*a1*”
UNION
SELECT “*DB2.Z*” as B, DB2.Z as C from DB2 where DB2.A = “*a1*”

The SQL query for construct DB3.a2 and DB3.a3 is similar to the query above.

2. From tables to attributes (DB3 to DB2)

Write different SQL for each table separately and union the result. For example, for table a1:

SELECT DISTINCT "*a1*" as DB2.A,
(SELECT DB3.a1.C from DB3.a1 where DB3.a1.B = “*DB2.B*”) as “*T1.B*”,
(SELECT DB3.a1.C from DB3.a1 where DB3.a1.B = “*DB2.C*”) as “*T1.C*”,
(SELECT DB3.a1.C from DB3.a1 where DB3.a1.B = “*DB2.D*”) as “*T1.D*”

```
FROM DB3.a1
```

By putting the “union” clause between the SQL queries that write for each table, the merged result set can be obtained.

5.4 Summary

In this chapter, several problems in database integration, which include distribution, duplication and conflicts, have been solved with solutions that can be applied during the data loading process. Therefore, as long as those adaptation options can be represented properly in the integration management interface, data that comes from heterogeneous databases can be integrated.

Even though the methods of integration between database structures can be very flexible, giving recommendation about the global table structure is necessary. Since the integrated data will be used by upper layers, the structure of “a class represented by a table” and “a property equals to a column” can make the mapping between data repository and ontology’s concept easier.

In summary, as the classification of problems and corresponding solutions described above, in this layer, the data can be integrated properly. What is more, SPARQL provides a reliable query service to upper layers. This kind of federated dataset and query service will satisfy the system’s upper layer searching and mapping requirements.

Chapter 6: Mapping a Data Repository to an Ontology

6.1. Domain Relations between Ontology and Database

In a domain ontology, every concept is regarded as a area that contains certain attributes and instances. Due to the domain difference between a database table and a concept in an ontology, in most cases, a table cannot be mapped to a concept directly. Even though many mapping strategies assume that a table will have exactly the same domain as a concept [24] [25] [26], if the ontology is to be applied to an already existing database, such an ideal mapping relationship will not always exist. Therefore, taking different mapping situations into consideration is necessary.

According to the theoretical analysis developed by Jesús Barrasa, Óscar Corcho and Asunción Gómez-Pérez, the domain differences between tables and concepts can be categorized into the following types (Figure 6.1) [23]:

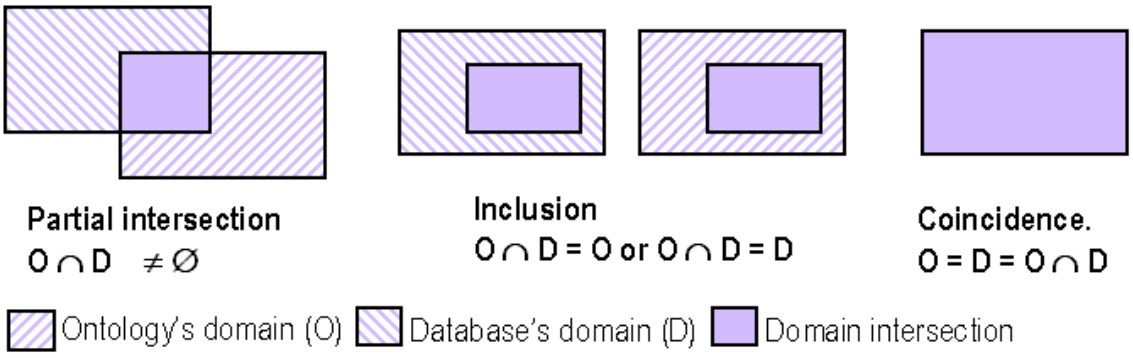

Figure 6.1: Relation between Database Domain and Ontology Concept Domain [23]

- In a "partial intersection", a database table and related ontology concept have some overlapping fields, however, both of them also have unique fields. Therefore, when mapping the database's information to the ontology, it is necessary to specify the overlapping fields, and then identify the unique fields of the concept, which may belong to other tables. As a result, it is necessary to union all the required database tables to complete the description of the concept.

- In an “inclusion”, there are two scenarios. Either the ontology concept’s domain is included completely in the database’s domain or a database table’s domain is included in the ontology’s concepts. In the first scenario, to define an ontology concept from the database table, the user can specify constraints. In the second scenario, the user needs to figure out the region of the ontology concept, and then union the associated database tables.
- In a “coincidence”, the overlapping domain between an ontology concept and database table is ideal, therefore, the user just has to map them directly. However, this situation seldom happens in the real world.

6.2. Applying Domain Relations to a Database

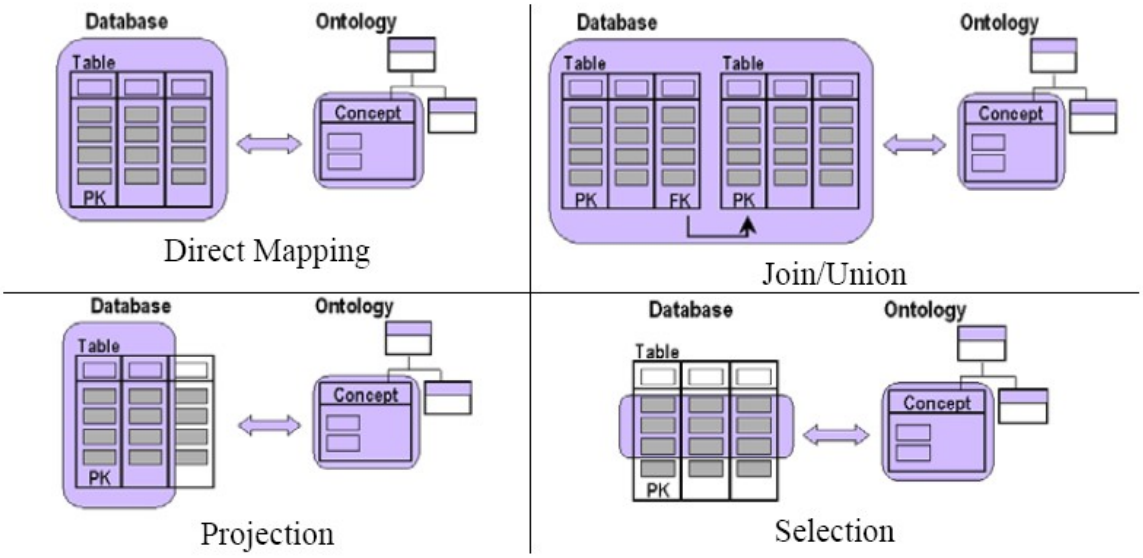

Figure 6.2: Operations can be applied in mapping [23]

Jesús Barrasa, Óscar Corcho and Asunción Gómez-Pérez also categorized mapping situations into four cases (see Figure 6.2) [23]. Each database to ontology mapping situation mentioned above can be supported by the following database operations:

- In the “Direct Mapping” situation, the defining concept domain only needs to store the information from selected database columns.

- In the “Join/Union” situation, the concept’s domain needs to be defined with columns from multiple tables. There are two different strategies that to be applied according to different situations. In the first situation, multiple tables are independent, therefore, when they union together, a cartesian-product operation can be applied to list all the records matching pairs directly. In the second situation, there are relationships between tables. One table may have a column that refers to another table’s key column, so “join on” operations can be applied with those columns. This information can be defined and saved in the Data Repository Metadata Model.
- In the “Projection” situation, the domain of a concept needs to be described with a subset of a table’s attributes. Therefore, the projection operation can be used to specify the needed columns. The projected columns will represent the attribute set of a concept.
- In the “Selection” situation, the ontology concept has to be defined with a subset of a table’s records. To obtain the correct subset, constraints on certain attributes can be specified in order to select the needed records from database table. The selected records will represent the instance set of a concept.

In practice, the actual case can be a combination of all the situations mentioned above. In summary, an ontology’s concept can be represented with a table set (one table or multiple tables) with selected columns and defined constraints.

6.3. The Ontology Concept Definition Model

6.3.1 Information needed

To store mapping information in a proper way, the RDF model can be used to maintain the column and constraint information, because it has an object-based structure that is easily understood by a humans and interpreted by a machine.

There are some requirements of the design of such a model that must be considered. The model must contain all the database location information such as the schema and the table and column the

attribute belongs to. Furthermore, constraints on the needed columns has also have to be stored in the model.

6.3.2 The mapping model

The database-to-ontology mapping model can be designed as shown in Figure 6.3:

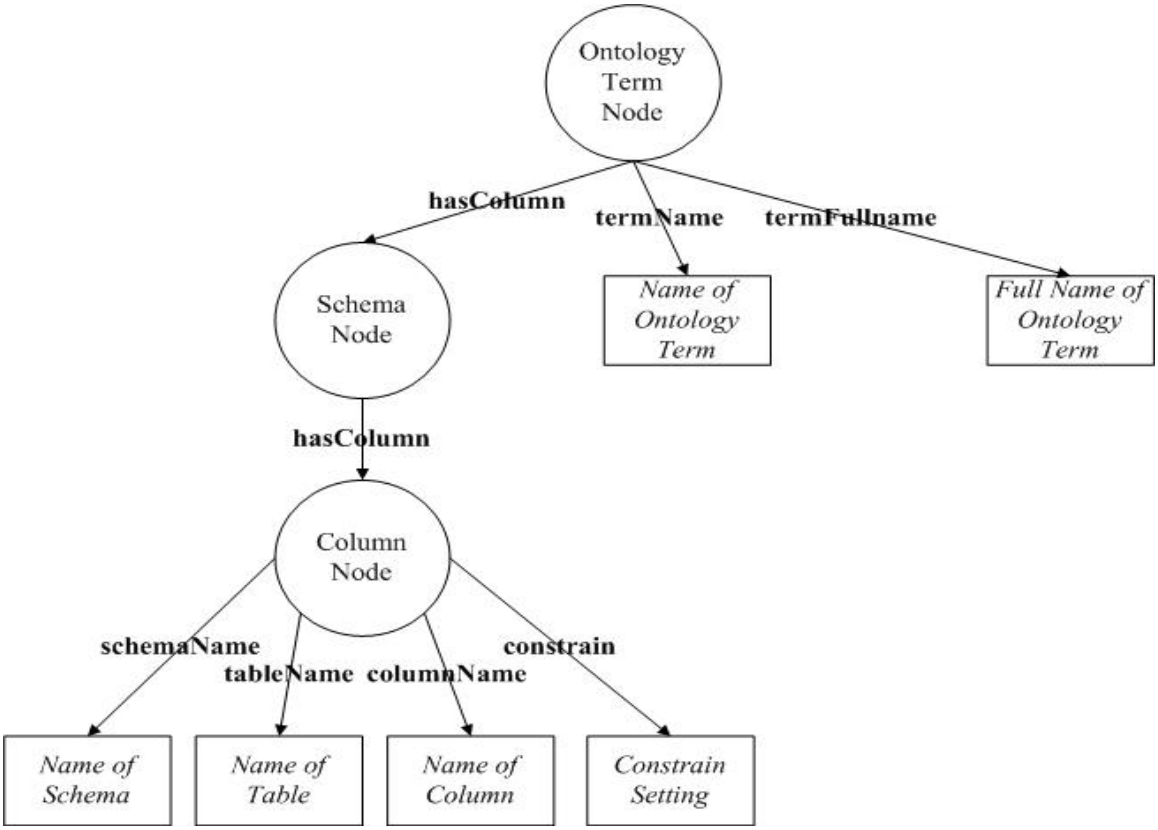

Figure 6.3: Ontology Mapping Model

Since every node in a triple store model must be unique, the description of a column node has to use the form "Concept_Name:Schema_Name/Table_Name/Column_Name". For the same reason, the system will use the full name of the ontology's concept as the ontology node's description.

In this model, different mapping cases have been realized with the following methods:

- **Projection**: An attribute is represented as a column node that has the location information, such as "name of schema", "name of table" and "name of column" properties.

- **Selection**: Records will be selected during the query process with the "Constraint Setting" property.
- **Union**: When columns that define the same concept come from different tables, the system will check the underlying logic between tables by scanning the metadata model.

6.3.3 The triple store model

To represent the mapping information of the model, the triple store statement pairs shown in Table 6.1 are used:

Subject	Predicate	Object	Description
Ontology Term Node	hasColumns	Column Node	The columns associated with the term
Ontology Term Node	termName	Name of the term	The name of this term
Ontology Term Node	termFullName	Full name of the term	The full name of this term
Column Node	schemaName	Name of schema this column belongs to	Describes the source schema of this column
Column Node	tableName	Name of table this column belongs to	Describes the source table of this column
Column Node	columnName	Name of table this column belongs to	Describes the source column of this column
Column Node	constraint	Constraint of the value in this column	The constraint formula

Table 6.1: Store Statement Pairs of Ontology Mapping

6.3.4 SPARQL query operations for the model

With the following SPARQL query, the system can retrieve mapping information from the database-to-ontology model mentioned above:

```
PREFIX j.0: <http://ontology#>
SELECT ?serverName ?schemaName ?tableName ?columnName ?constrain
WHERE {
        ?ontologyTerm j.0:ontoName "Ontology Name"
        ?ontologyTerm j.0:ontoFullName "Ontology Full Name"
        ?ontologyTerm j.0:dbMappingInfo ?columnNode.
        ?columnNode j.0:serverName ?serverName.
        ?columnNode j.0:schemaName ?schemaName.
        ?columnNode j.0:tableName ?tableName.
        ?columnNode j.0:columnName ?columnName.
```

```
        ?columnNode j.0:constrain ?constrain.
}
```

6.4. Summary

This model helps the user map the data in a database to the ontology's concepts. During the mapping definition process, the basic semantic feature, a domain, has been introduced. The connection between an ontology and a database has been established. Utilizing the defined ontology concepts, the upper layers of the system can use them to construct a query strategy.

Chapter 7: Information Retrieval

7.1. Query Strategy

7.1.1. Single concept

When a user picks exactly one concept from an ontology tree, the strategy is straightforward. There are seven steps the system has to follow:

- Check the ontology mapping model, find the associated columns and constraints of selected term.
- Identify the tables the associated columns belong to.
- Check whether or not there any “refersTo” relationships existing between those tables by inspecting the virtual database metadata model. If the related columns have been found, record that information.
- Prepare the dataset that will be used during search by merging the selected table’s data repository models together.
- Construct a SPARQL query based on the collected information, which includes the columns’ location, columns’ constraints and the “join on” keys. Paging and sorting information may also have to be considered.
- Run the generated SPARQL query on the prepared dataset.
- Retrieve the result and parse it into XML, then send it back to the user’s web interface.

7.1.2. Multiple concepts

The search strategy is much more complicated when the user picks multiple concepts. In addition to the steps mentioned in the previous section, the system has to consider the domain relationship between concepts. The possibilities of how those domains interact with each other can be categorized into the types illustrated in Figures 7.1 through 7.4:

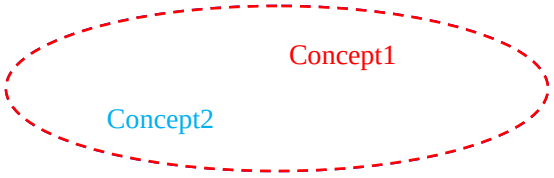

Figure 7.1: Domain Relation between Ontology's Concept: Coincidence

In the "Coincidence" case, two concepts have the same domain. When a user searches both of them at the same time, the system just has to list out the information of either one.

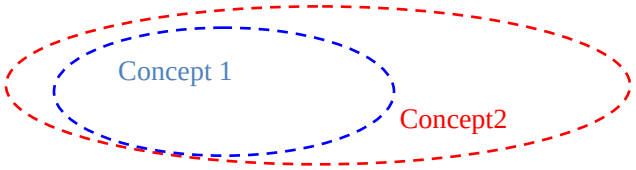

Figure 7.2: Domain Relation between Ontology's Concept: Inclusion

In this "Inclusion" case, one concept is the parent class of the other one, which means its domain embodies the other's. When the user selects both, the system will display the information of the smaller domain that belongs to the more specific concept.

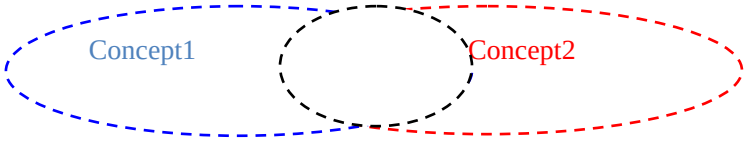

Figure 7.3: Domain Relation between Ontology's Concept: Intersection

In the "Intersection" case, two concepts overlap in some fields, therefore, when a user picks both of them, the system will do a union operation to find the records in the fields in the intersection. The attribute in the result set would be the columns that come from both concepts.

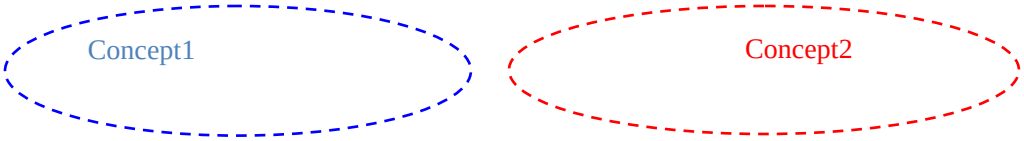

Figure 7.4: Domain Relation between Ontology's Concept: Independent

In the “Independent” case, the domain of two concepts does not have any overlap, therefore, when a user picks both of them, the system does not have to consider the relation between concepts, instead, a Cartesian-Product operation will be used to list all the possible instance combinations of the pair of concepts.

In summary, based on the different relation situations between concepts, the system will follow different strategies to collect information and construct the correct SPARQL query.

7.2. Paging and Sorting on the Server side

As a Client-Server architecture system that deals with large datasets, the cost of data transmission needs to be taken into consideration. Instead of sending all of the result at one time, handling paging and sorting on the server side can greatly improve efficiency, because a smaller result set will have to be transmitted.

In SPARQL, the fields of ORDER BY, OFFSET and LIMIT provides a solution for server side paging and sorting. The example SPARQL query is:

```
PREFIX j.0: <http://virtualDB#>
SELECT ?X ?Y ?Z
WHERE
{
        ?serverNode j.0:serverName “Name of the server”.
        ?serverNode j.0:hasSchemas ?schemaNode.
        ?schemaNode j.0:schemaName “Name of the schema”.
        ?schemaNode j.0:hasTables ?tableNode.
        ?tableNode j.0:tableName “Name of the table”.
        ?tableNode j.0:hasRecord ?recordNode .
        ?recordNode j.0:column-X ?X.
        ?recordNode j.0:column-Y ?Y.
        ?recordNode j.0:column-Z ?Z.
}
ORDER BY ASC(?X) offset starting_index limit amount_of_record
```

In this SPARQL query, The ORDER BY clauses indicates the sorted column and order, the offset means the starting index, and the limit indicates the number of result records.

7.3. Summary

In the information retrieval level, users can query the data repository based only on their knowledge about the concepts belonging to ontologies. The system administrator works on the underlying logic between the ontology's concepts and deals with data source location, which is totally invisible to common users. Therefore, this kind of architecture makes the searching process very easy and convenient for those not familiar with the details of the databases being searched.

Chapter 8: Testing

8.1. Preliminaries

Compared to the size of an RDF data repository model, the sizes of a metadata model and an ontology mapping model are much smaller. For this reason, the runtime of this system will be affected mostly by the efficiency of the data repository. Therefore, the following test only focuses on determining the efficiency of our RDF data repository model.

The runtimes of loading data and retrieving data will be tested separately. The data loading test simulates the process of transferring data from a MySQL database into the data repository (Deployed in Jena's Triplestore, TDB). The information retrieval test estimates the runtime of several SPARQL queries which simulate common search operations on relational databases.

8.1.1 The test environment

The following is a description of the system used for testing

- Hardware:

  Processor: Intel (R) Pentium (R) D CPU 3.73 GHz

  Memory: 8.00 GB

- Software:

  Operation System: 64-bit Windows 7 Professional

  Java Version: jre-6u19-windows-x64 (64 bit version)

  Database Version: MySQL Server 5.1

  IDE: Eclipse-SDK-3.5-win32-x86_64

  TDB Version: TDB 0.8.5

8.1.2 Test dataset metadata

There are two dataset has been used in this test. The following is a description of their metadata.

Dataset1: Name: Table 1. Columns: x, y, z.

Dataset2: Name: Table 2. Columns: a, b, c.

8.1.3 Test approach

In the data loading test, the loading process was tested 5 times for each of 10 data set sizes. In the information retrieval test, each query was executed 10 times for each data set size. The results reported here are the averages of the testing samples. The final results will be displayed in the form of graphs.

8.2. Test Result

8.2.1 Data loading time

Figure 8.1 shows the test results for loading data into the triple store repository from a MySQL database. The run times are essentially linear.

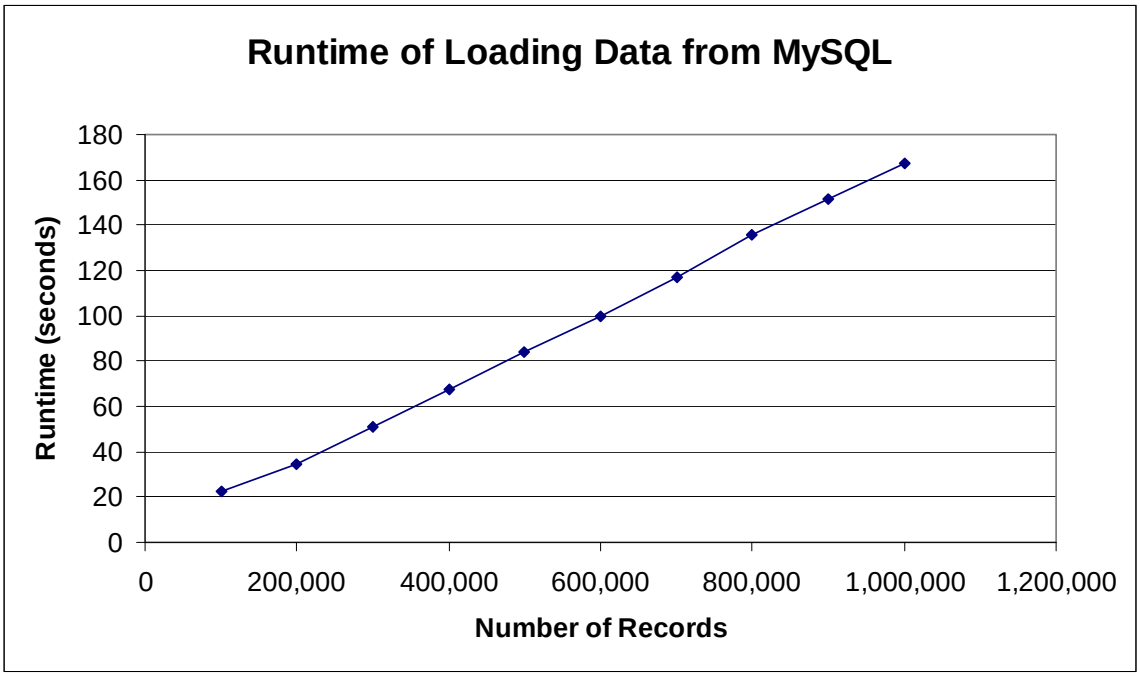

Figure 8.1: Test Results for Loading Data

8.2.2 Information retrieval time

The following are testing results for 6 typical queries used in this system. Each query has simulated one operation of SQL query in relational database.

Query 1	
Function	Find the value of columns x, y, z in Table 1.

SPARQL Query	SELECT ?x ?y ?z
	WHERE
	{
	?record1 j.0:C-x ?x.
	?record1 j.0:C-y ?y.
	?record1 j.0:C-z ?z.
	}

Equivalent SQL Query	SELECT x, y, z FROM Table1

Table 8.1: Information of Query 1

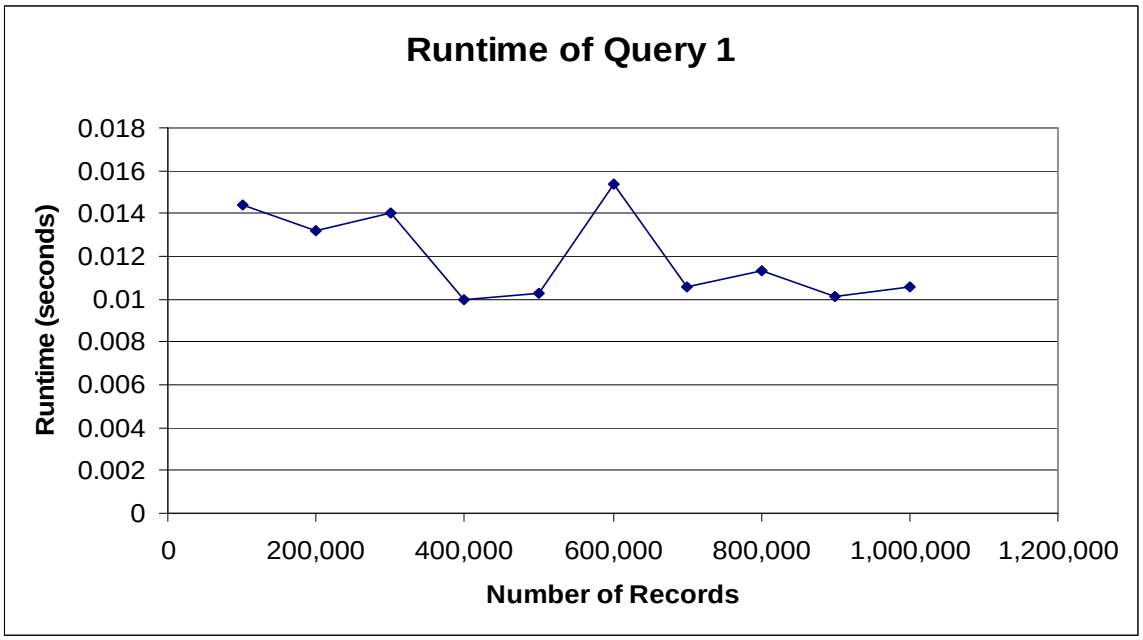

Figure 8.2: Test Results for Query 1

Query 2	
Function	Find the value of columns x, y, z in Table 1. Result set is ordered by column x.

SPARQL Query	SELECT ?x ?y ?z
	WHERE
	{
	?record1 j.0:C-x ?x.
	?record1 j.0:C-y ?y.
	?record1 j.0:C-z ?z.
	}
	order by xsd:double (?x)

Equivalent SQL Query	SELECT x, y, z FROM Table1 ORDER BY x

Table 8.2: Information of Query 2

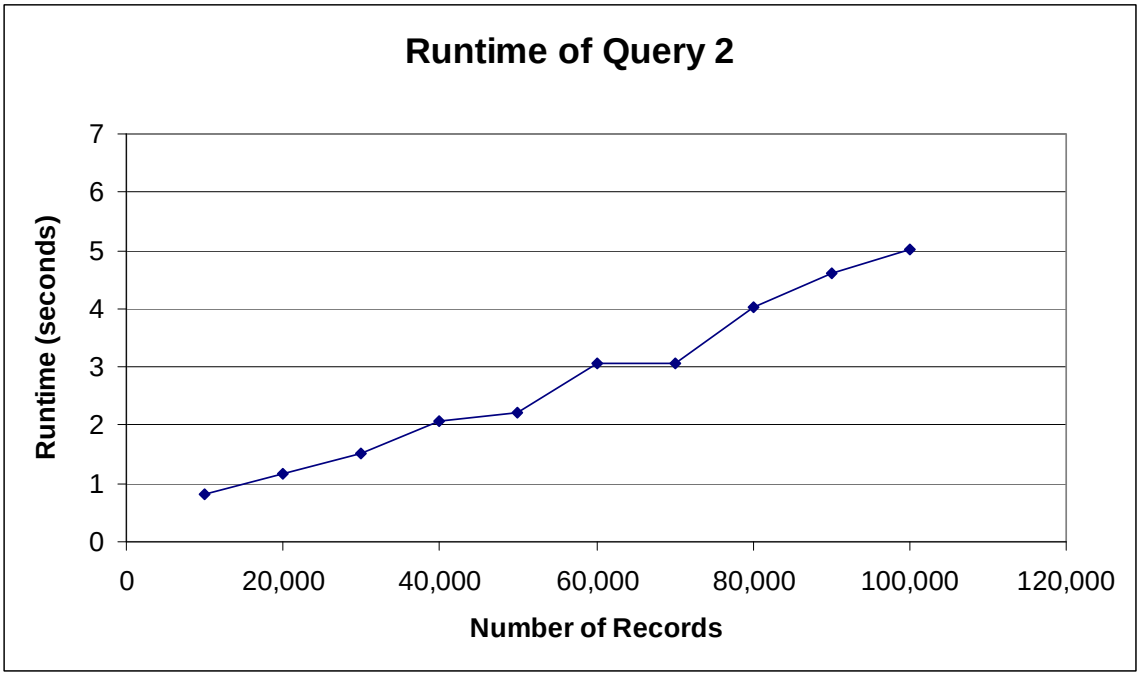

Figure 8.3: Test Results for Query 2

Query 3	
Function	Find the value of columns x, y, z in Table 1, but only if the value in column x is larger than a given constant.

SPARQL Query	SELECT ?x ?y ?z
	WHERE
	{
	?record1 j.0:C-x ?x.
	?record1 j.0:C-y ?y.
	?record1 j.0:C-z ?z.
	FILTER (xsd:double (?x)>*number*).
	}

Equivalent SQL Query	SELECT x, y, z FROM Table1 WHERE x > *number*

Table 8.3: Information of Query 3

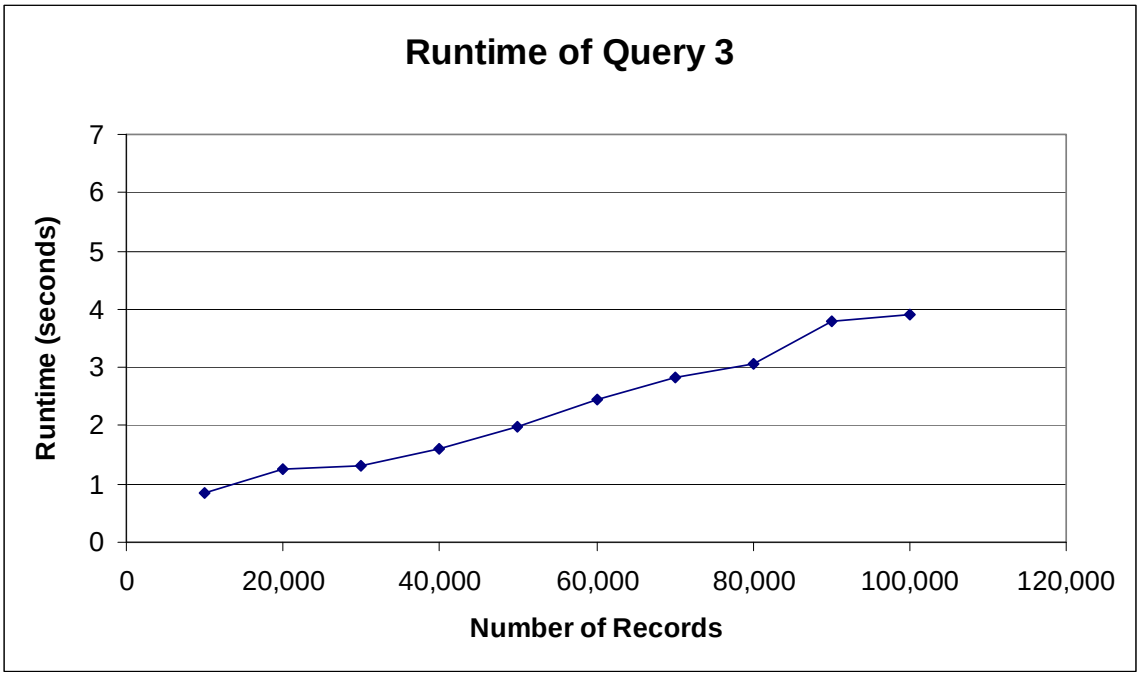

Figure 8.4: Test Results for Query 3

Query 4	
Function	Find the value of columns x, y, z in Table 1, but only if the value in column x is larger than a given constant. The result set is to be ordered by column x.

SPARQL Query	SELECT ?x ?y ?z
	WHERE
	{
	?record1 j.0:C-x ?x.
	?record1 j.0:C-y ?y.
	?record1 j.0:C-z ?z.
	FILTER (xsd:double (?x)>*number*).
	}
	ORDER BY xsd:double (?x)

Equivalent SQL Query	SELECT x, y, z FROM Table1 WHERE x > *number* ORDER BY x

Table 8.4: Information of Query 4

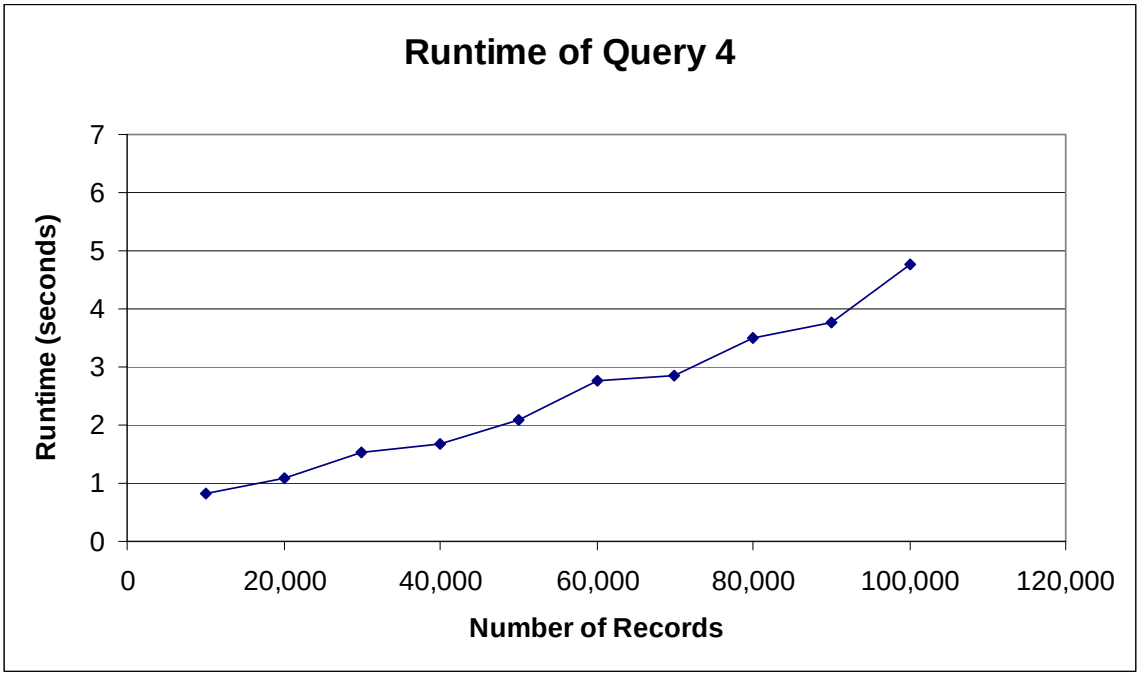

Figure 8.5: Test Results for Query 4

Query 5	
Function	Find the record pairs in Table 1 and Table 2 where the column x value in Table 1 is equal to column y in Table 2. Columns x, y, z from Table 1 and columns a, b, c of Table 2 will be displayed.

SPARQL Query	SELECT ?x ?y ?z ?b ?c
	WHERE
	{
	?record1 j.0:C-x ?x.
	?record1 j.0:C-y ?y.
	?record1 j.0:C-z ?z.
	?record2 j.0:C-a ?x.
	?record2 j.0:C-b ?b.
	?record2 j.0:C-c ?c.
	}

Equivalent SQL Query	SELECT table1.x, table1.y, table1.z, table2.b, table2.c FROM table1 JOIN table2 ON table1.x = table2.a

Table 8.5: Information of Query 5

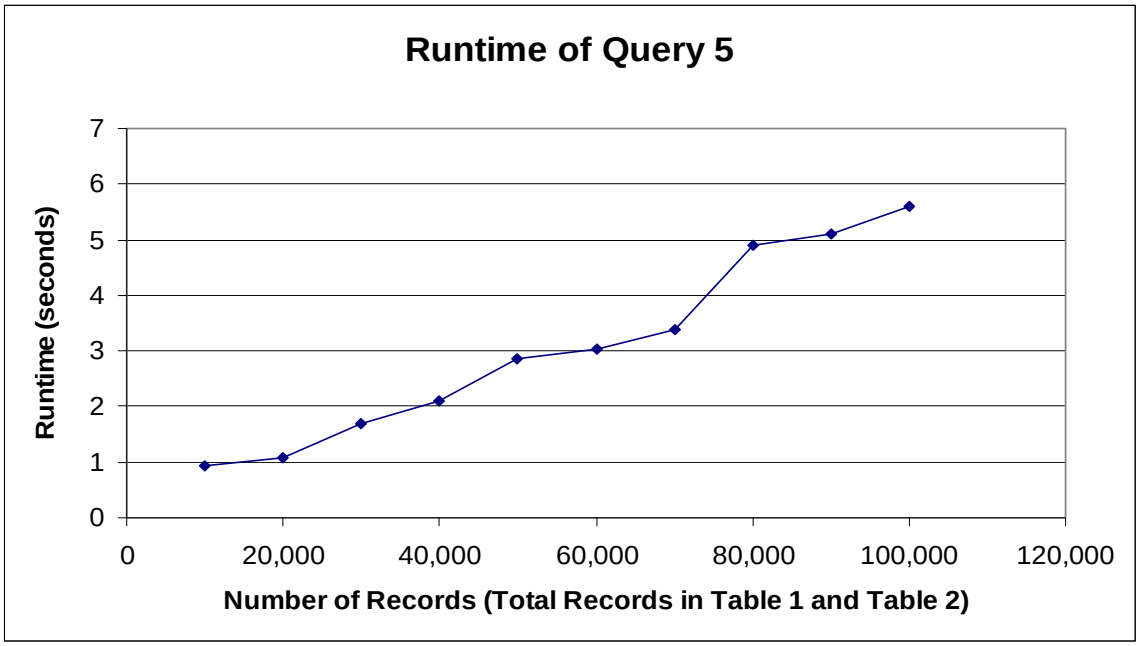

Figure 8.6: Test Results for Query 5

Query 6	
Function	Find the record pairs in Table 1 and Table 2 where the column x value in Table 1 is equal to column y in Table 2 and the value in column x is larger than a given constant. Columns x, y, z of Table 1 and columns a, b, c of Table 2 will be displayed. The result set is ordered by column x.

SPARQL Query	SELECT ?x ?y ?z ?b ?c
	WHERE
	{
	?record1 j.0:C-x ?x.
	?record1 j.0:C-y ?y.
	?record1 j.0:C-z ?z.
	?record2 j.0:C-a ?x.
	?record2 j.0:C-b ?b.
	?record2 j.0:C-c ?c.
	FILTER (xsd:double (?x)>*number*).
	}
	ORDER BY xsd:double (?x)

Equivalent SQL Query	SELECT table1.x, table1.y, table1.z, table2.b, table2.c FROM table1 JOIN table2 ON table1.x = table2.a WHERE table1.x > *number* ORDER BY table1.x

Table 8.6: Information of Query 6

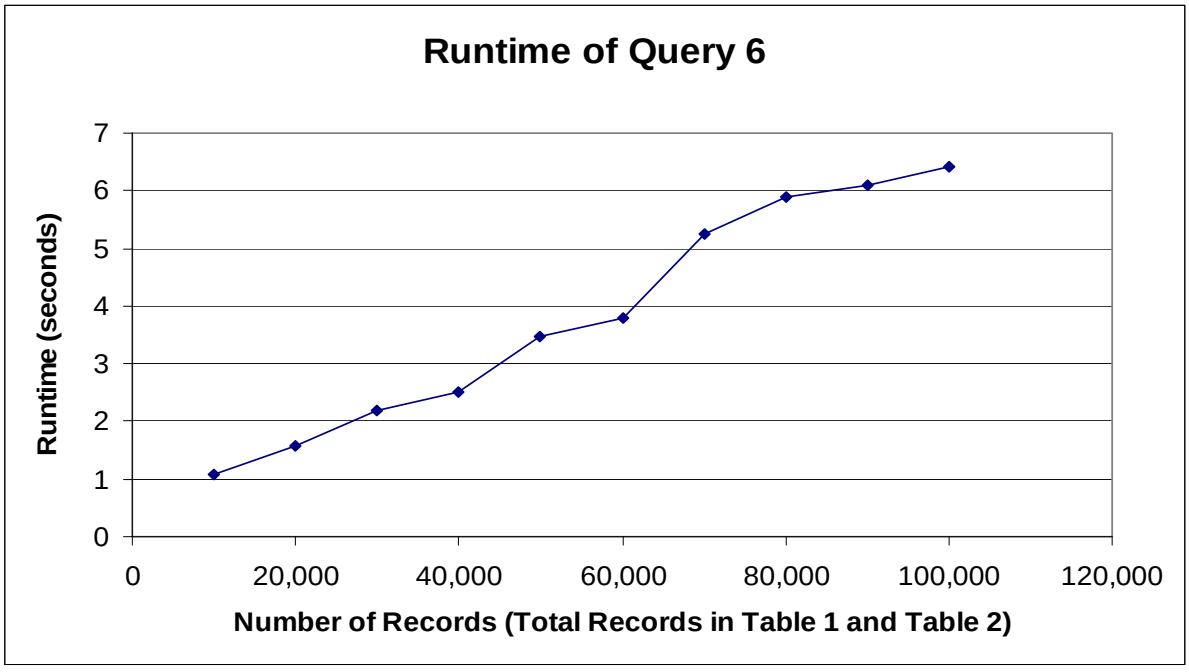

Figure 8.7: Test Results for Query 6

8.3. Result Analysis

The model displayed very good performance loading data (Figure 8.1) and on Query 1 (Figure 8.2). Even the dataset with one million records was handled efficiently. The performance benefited from the streaming query engine of TDB triple store. Instead of waiting for generating the entire result set, the result records can be retrieved dynamically.

However, when additional query options were applied in the SPARQL query, the efficiency of information retrieval declined. In queries 2, 3, and 4, sorting and filtering options were used. Those operations need to be executed after the entire dataset been obtained, which results in a heavier workload. Even though Query 4 is more complicated than Query 3, it has a slightly better performance. This might be due to fact that the filter operations are applied first, generating a smaller result set for sorting.

In queries 5 and 6, the “join on” operation was tested. As expected, the efficiency is worse than working on a single table. The reason is that the “join” operation has to be run on two models simultaneously, the workload of retrieving information from different resources increases runtime.

In summary, when this system works on comparatively smaller datasets, the efficiency appears to be good enough. When larger datasets need to be handled, the advantage of the TDB triple store’s streaming feature can be used. With the outstanding loading and reading speed of data repository, we can use the model to integrate data very quickly, and then deploy the integrated data in a traditional database. With some simple adaptation, the ontology mapping model can also work with relational databases. In this way, the efficiency of searching can be maintained.

Since different triple store implementations have different query mechanisms, in the future, other triple stores will also be implemented and tested. Therefore, the triple store implementation that best fits a specific project will need to be identified.

Chapter 9: Conclusions and Future Work

The system developed in this project has an architecture providing data integration, data management, data mapping, and information retrieval.

To address these challenges, new technologies and approaches, such as the RDF model and SPARQL queries, have been applied and implemented to solve key issues mentioned above. Moreover, the efficiency of this system has also been tested.

With this system, the data stored in heterogeneous databases can be integrated into an RDF data repository, which is described by the metadata model. And then the integrated data can be mapped to a concept in a domain ontology. Through the concepts represented in ontologies, users can readily retrieve information from the system. However, this system still has several drawbacks, which need to be considered and solved in future investigation.

1. The complexity of searching limits the system's capability for supporting large datasets. Future work on designing efficient query templates and optimizing the RDF model is needed.
2. In data integration, the classification of conflicts is still at a theoretical level. However, conflicts in the real world can be complicated. Additional cases of database conflicts need to be studied and analyzed.
3. The operations of database integration mapping and ontology mapping have to be done manually, which results in a heavy workload for users.

With the proposed system architecture, improvements can be applied at various levels in different aspects.

At the RDF model level, there are several approaches to improving efficiency. TDB provides an optimizer that uses information captured in a per-database statistics file generated automatically. This information is used to determine the best execution approach during the execution phase. In addition, the actual data so far retrieved should be taken into account. This can be a practical method to

improving query efficiency. Secondly, since the RDF model has an object-oriented structure, which is totally different from a relational database, further optimizing for the model and query can be tried and tested.

At the data mapping level, much research has been done on automatic database schema mapping [28] [30] [31]. Some of these results provide automatic or semi-automatic mapping algorithms. If those strategies can be implemented in this system, the efficiency of mapping can be improved.

At the information retrieval level, with a defined concept in ontology, more powerful and complicated semantic features can be introduced with RDFS (RDF Schema) or even OWL (Web Ontology Language). Special query strategies will have to be considered and implemented for each kind of relation. Many researches are currently working in this area [29] [32]. If those semantic features can be added to our work, the quality of searching should be improved significantly.

In summary, based on the open architecture of this system, advanced technologies, strategies, and algorithms from other research fields can be integrated to improve capability and performance.

Bibliography

[1] Wikipedia “Data integration” URL: http://en.wikipedia.org/wiki/Data_integration

[2] Alon Halevy, Anand Rajaraman, Joann Ordille (2006) “Data integration: the teenage years” Proceedings of the 32nd international conference on Very large data bases,

[3] Amit P. Sheth , James A. Larson (1990) “Federated database systems for managing distributed, heterogeneous, and autonomous databases” ACM Computing Surveys (CSUR), v.22 n.3, p.183-236

[4] W3C “RDF/XML Syntax Specification” URL:http://www.w3.org/TR/REC-rdf-syntax/

[5] Rod Coffin (2007) “Create Scalable Semantic Applications with Database-Backed RDF Stores” DevX’s Semantic Zone

[6] Michael Grobe (2009)”RDF, Jena, SparQL and the 'Semantic Web'”Proceedings of the ACM SIGUCCS fall conference on User services conference, Pages: 131-138, ISBN:978-1-60558-477-5

[7] Wikipedia, SPARQL, URL: http://en.wikipedia.org/wiki/SPARQL

[8] "W3C Semantic Web Activity News-SPARQL is a Recommendation".(2008) W3.org.

[9] Goksel Aslan, Dennis McLeod (1999) “Semantic heterogeneity resolution in federated databases by metadata implantation and stepwise evolution” The VLDB Journal — The International Journal on Very Large Data Bases, v.8 n.2, p.120-132

[10] Chiang Lee , Chia-Jung Chen , Hongjun Lu (1995) “An aspect of query optimization in multidatabase systems” ACM SIGMOD Record, v.24 n.3, p.28-33

[11] Ee-Peng Lim , Jaideep Srivastava (1993) “Query optimization and processing in federated database systems” Proceedings of the second international conference on Information and knowledge management, p.720-722, November 01-05

[12] Catharine M. Wyss , Edward L. Robertson (2005) “Relational languages for metadata integration” ACM Transactions on Database Systems (TODS), v.30 n.2, p.624-660, June 2005

[13] Vijayan Sugumaran , Veda C. Storey (2006) “The role of domain ontologies in database design: An ontology management and conceptual modeling environment” ACM Transactions on Database Systems (TODS), v.31 n.3, p.1064-1094, September 2006

[14] Wikipedia “Ontology” URL: http://en.wikipedia.org/wiki/Ontology_(information _science)

[15] F. Arvidsson and A. Flycht-Eriksson. (2008) “Ontologies I” URL: http://www.ida.liu.se/~janma/SemWeb/Slides/ontologies1.pdf

[16] Pavel Hruby “Ontology-Based Domain-Driven Design” Microsoft URL: http://www.softmetaware.com/oopsla2005/hruby.pdf

[17] Gregory A. Silver, Osama Al-Haj Hassan, John A. Miller (2007) “From domain ontologies to modeling ontologies to executable simulation models” Winter Simulation Conference, Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come, ISBN:1-4244-1306-0

[18] Xin Wang , Christine W. Chan , Howard J. Hamilton (2002) “Design of knowledge-based systems with the ontology-domain-system approach” Proceedings of the 14th international conference on Software engineering and knowledge engineering, July 15-19, 2002, Ischia, Italy

[19] Graciela Brusa, Ma. Laura Caliusco, Omar Chiotti, (2006) “A process for building a domain ontology: an experience in developing a government budgetary ontology” Proceedings of the second Australasian workshop on Advances in ontologies - Volume 72

[20] Thomas C. Jepsen (2009), “Just What Is an Ontology, Anyway?”, September/October 2009 (vol. 11 no. 5), IT Professional, SSN: 1520-9202

[21] Cancer Research UK (2004) Demonstrations of clinical applications, URL: http://www.openclinical.org/dm_homey.html#

[22] Wikipedia “Client-Server” URL:http://en.wikipedia.org/wiki/Client-server

[23] Jesús Barrasa, Óscar Corcho, Asunción Gómez-Pérez “R2O, an Extensible and Semantically Based Database-to-ontology Mapping Language” URL: http://www.cs.man.ac.uk/~ocorcho/documents/SWDB2004_BarrasaEtAl.pdf

[24] Adriana S. Aparício, Oscar L. M. Farias, Neide dos Santos (2005) “Applying ontologies in the integration of heterogeneous relational databases” Conferences in Research and Practice in Information Technology Series; Vol. 172, Proceedings of the 2005 Australasian Ontology Workshop, ISBN ~ ISSN:1445-1336 , 1-920-68240-6

[25] Dejing Dou , Paea LePendu (2006) “Ontology-based integration for relational databases” Proceedings of the 2006 ACM symposium on Applied computing, April 23-27, 2006, Dijon, France

[26] Kamran Munir , Mohammed Odeh , Richard McClatchey (2008)“Ontology assisted query reformulation using the semantic and assertion capabilities of OWL-DL ontologies” Proceedings of the 2008 international symposium on Database engineering & applications, September 10-12, 2008, Coimbra, Portugal

[27] Kamran Munir, Mohammed Odeh, Richard McClatchey, (2009) “Managing the mappings between domain ontologies and database schemas when formulating relational queries” Proceedings of the 2009 International Database Engineering & Applications Symposium, ISBN:978-1-60558-402-7

[28] Wen-Syan Li, Chris Clifton (2000) “SEMINT: a tool for identifying attribute correspondences in heterogeneous databases using neural networks” Data & Knowledge Engineering, Volume 33 , Issue 1 (April 2000), Pages: 49 - 84 , Year of Publication: 2000, ISSN:0169-023X

[29] Cristian Pérez de Laborda , Stefan Conrad (2005) “Relational.OWL: a data and schema representation format based on OWL” Proceedings of the 2nd Asia-Pacific conference on Conceptual modelling, p.89-96, January 01, 2005, Newcastle, New South Wales, Australia

[30] Yaser Karasneh, Hamidah Ibrahim, Mohamed Othman, Razali Yaakob (2009) “A model for matching and integrating heterogeneous relational biomedical databases schemas” Proceedings of the 2009 International Database Engineering & Applications Symposium, ACM International Conference Proceeding Series

[31] Philip A. Bernstein , Sergey Melnik , Michalis Petropoulos , Christoph Quix (2004) “Industrial-strength schema matching” ACM SIGMOD Record, v.33 n.4, December 2004

[32] Smith B, Ceusters W, Klagges B, Köhler J, Kumar A, Lomax J, Mungall C, Neuhaus F, Rector AL, Rosse C.(2005) “Relations in biomedical ontologies” Genome Biol. 2005;6(5):R46.

[33] The Data Warehousing and Business Intelligence division “The Evolution of ETL-From Hand-coded ETL to Tool-based ETL” @ Vivan Released on 12-Jun-2007

[34] Wikipedia, Triplestore, URL:http://en.wikipedia.org/wiki/Triplestore

[35] Wikipedia, Ontology, URL: http://en.wikipedia.org/wiki/Ontology_(information_science)