공학석사학위논문

Semantics Preserving MapReduce Process for RDB to RDF Transformation

RDB to RDF 변환을 위한 의미 정보 보존 맵리듀스 처리

2015 년 8 월

서울대학교 대학원

전기컴퓨터공학부

임 경 빈

Abstract

Semantics Preserving MapReduce Process for RDB to RDF Transformation

RDB to RDF 변환을 위한 의미 정보 보존 맵리듀스 처리

Kyung-Bin Lim
Department of Computer Science & Engineering
The Graduate School
Seoul National University

Today, most of the data on the web is stored in relational databases, which is called “deep web”. Semantic web is a movement to the next generation of the web, where all data are augmented with well-defined semantics and linked together in machine-readable format. RDB2RDF approaches have been proposed and standardized by W3C, which publishes relational data to semantic web by converting relational data into RDF formatted data. We propose a system that automatically transforms relational data into RDF data and creates OWL ontology based on the schema of database. Some approaches have been proposed, but most of them did not fully make use of schema information to extract rich semantics, nor did they experimented on large databases for performance. We utilize Hadoop framework in transformation process, which enables distributed system for scalability. We present mapping rules that implements augmented direct mapping to create local ontology with rich semantics. The results show that our system successfully transforms relational data into RDF data with OWL ontology, with satisfactory performance on large-sized databases.

Keywords : Semantic Web, RDB2RDF, MapReduce, Hadoop, RDF

Student Number : 2013-23137

Contents

List of Figures and Tables

Chapter 1

Introduction

After Tim Berners-Lee announced the Semantic Web to be an evolution of the current web, many researches have been conducted to vitalize the Semantic Web by building the Web of Data, which requires the format of RDF. However, the reality is that the majority of the data on the web are still stored in relational databases, which are often called "deep web." It's known that over 70% of the web sites are backed up by the relational databases, which correspond to 500 times of the amount of semantic data established [13]. The amount of the data stored in RDF, which is the standard data model of the Semantic Web, is insufficient for the growth of the Semantic Web. Thus, converting or making data hosted in relational databases accessible to the Semantic Web is essential terms for the Semantic Web's success, and this is often referred to as the "RDB-to-RDF" process. The necessity of transforming relational data to RDF data has been recognized by W3C, and the RDB2RDF Working Group has been established [14].

To accelerate the current Web's evolution toward the Semantic Web, eventually relational data has to be published into semantic data structures. Many researches and papers have suggested techniques to realize the RDB2RDF movement, and there are two different approaches in implementing the process. The first approach is synchronous (real-time) approach, which is also called "wrapper" system [1]. In this approach, the queries are run-time evaluated on demand against the relational data (Figure 1). So, this system keeps virtual RDF version of the relational data, while the actual data is remaining in the relational database. The advantage of the approach is that the data is always up-to-date since the query is evaluated against the original data. However, the disadvantage is that SPARQL query is internally rewritten into SQL in the query time, which would harm the query performance. The Second approach, asynchronous ETL (Extract-Transform-Load) approach [1], is the static transformation of the relational database into RDF data. The "RDB dump" is extracted, transformed to RDF data, and materialized into RDF format as a separate version of data (Figure 2). The disadvantage would be the update on the original relational data will not be propagated to RDF repository, as the name "asynchronous approach" implies. So, it might be needed to dump the relational data into RDF periodically to keep RDF data up-to-date. On the other hand, this approach has advantages of compatibility with existing semantic tools, which means facilitating further processing, analysis, or reasoning. Moreover, queries can be answered without harming the run-time performance, in contrast to the synchronous approach. The ETL process of RDB2RDF is the subject of this paper, and we show an automatic ETL system that extracts semantics data as well as the data.

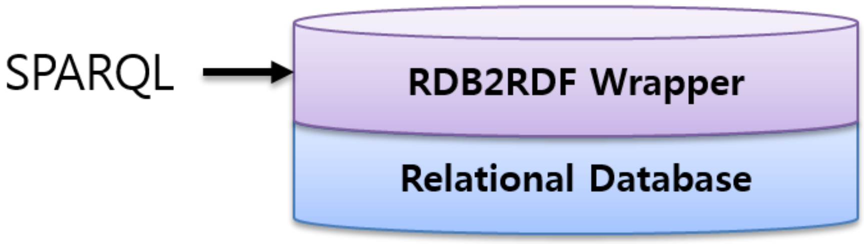


Figure 1 Synchronous "wrapper" approach

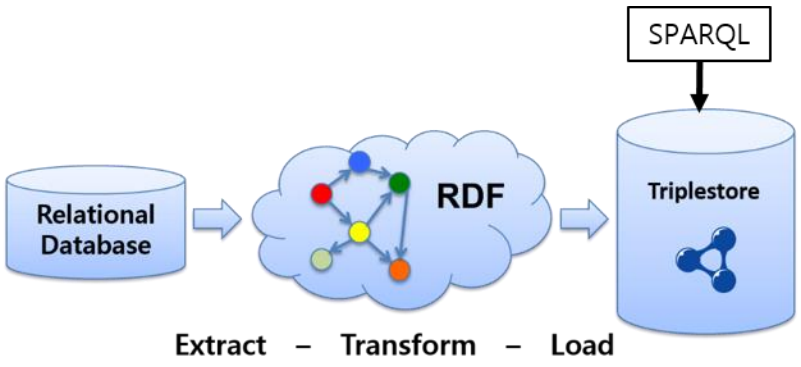


Figure 2 Asynchronous "ETL" approach

As RDB2RDF ETL approach is an area which should be able to work on the legacy relational data, which could be very large and may be executed on a regular basis, the performance of the process should be regarded as a critical issue. Minimizing the downtime of the triplestore each time the RDF dump is materialized for updating the contents is vital [3]. This paper utilizes a new framework for RDB2RDF ETL system, Hadoop, to show improved and scalable performance.

Our contribution is three-fold. First, we implement a semantic ETL system which operates on distributed cluster of machines. To the best of our knowledge, no other works so far have researched distributed approach on RDB2RDF transformation. Second, we utilize schema information from the target database to create extended OWL ontology. Third, we experimented our system on large-sized databases for performance and scalability, and the system showed superior results than other studies.

This paper is organized as follows. Section 2 gives some related works on RDB2RDF approaches and background knowledge for the process. Section 3 shows the mapping rules we implement to create local OWL ontology. Section 4 addresses our approach on the whole process. Preprocessing methods and Hadoop algorithm we implement are demonstrated. Section 5 shows the experimental results we conducted using sample data. Section 6 discusses the experimental results and conclusion.

Chapter 2

Related Work

2.1 Semantic ETL Systems

Many approaches have been experimented to publish the relational data into Semantic RDF data, in both synchronous and asynchronous ways.

Thuy et al. [4] has presented RDB2RDF method that transforms data in all tables in the relational database into RDF. Their system also succeeds in preserving the foreign key and primary key information in the database, and translates it into ontology. They used RDFS ontology language constructs such as rdfs:subClassof, rdfs:subPropertyOf, rdfs:domain, and rdfs:range to improve the resulting ontology. However, RDFS has insufficient constructs to express the semantics that are on the relational database. SQL has some constraints syntax that describes semantics about the domain that could be reflected during transformation process with constructs beyond RDFS language.

Vavliakis et al. [2] has presented RDOTE, a framework for transporting data in relational databases into the Semantic Web. It automatically creates mappings based on the Direct Mapping methods, and allows users to further modify it to map to existing ontologies (Domain Semantics-Driven Mapping). It provides user friendly graphical interface for customizing mappings. They experimented the time performance of their system, and the result showed a logarithmic scale. The experiment has not tested its performance on large-sized databases.

TripleGeo [5] has developed a special-purpose RDB2RDF ETL system. It implemented an open-source ETL utility that can extract geospatial features from various sources including relational databases and transform them into RDF triples for subsequent loading into RDF stores. It directly accesses geographical data either from standard geographic formats or widely used DBMSs. They tested the system against OpenStreetMap dataset. According to the paper they referenced, the result of extracting data from relational databases did not show scalable and good performance results. They discussed about the scalability with increasing data volumes in the conclusion, and mentioned splitting the input into batches and using a parallelization scheme such as MapReduce.

2.2 Hadoop MapReduce

MapReduce [12] is a simple programming model which has been presented by Google. The framework is suitable for processing and generating large datasets, by efficiently exploiting large set of commodity computers and executing process in distributed manner. The technology has well received credits by the areas where large amount of data needs to be handled, and many data mining algorithms are being rewritten in MapReduce-forms to take the advantage of the technology. [19]

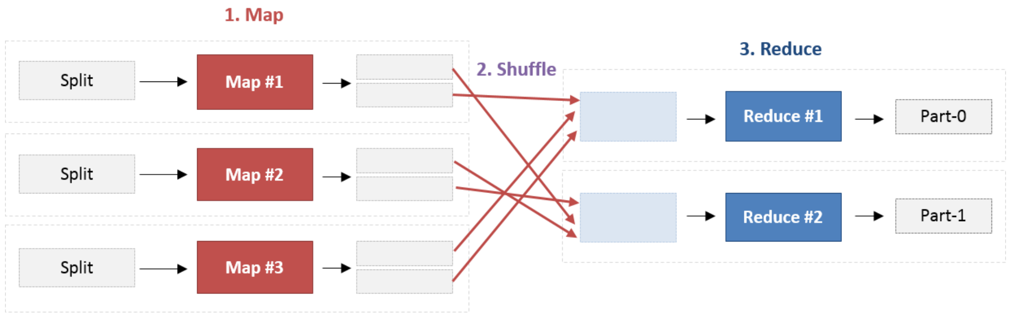


Figure 3 MapReduce Workflow

Hadoop [16] is an open-source framework that provides Hadoop Distributed File System (HDFS) and Hadoop MapReduce. Hadoop Distributed File System is a java-based file system that provides a scalable and reliable data storage system. It provides stable storage by data replication and high fault tolerance. Hadoop MapReduce is one of programming models that Hadoop provides, and implemented based on Google's MapReduce. Hadoop MapReduce comprises of three major steps; Map, Shuffle, and Reduce, as shown in Figure 3. Map produces Key-Value sequence of the input. Shuffle sorts and groups into a sequence of pairs of key-list of values. Reduce operates on the list of values for each key to produce final results.

2.3 Mapping Approaches

RDB to RDF mapping has several different mapping approaches in ways they create RDF representation of databases are described.

First, there is Direct Mapping [11] approach announced by W3C as a standard. Direct Mapping is a default automatic mapping of relational data to RDF, and it takes database schema and data as input and creates an RDF graph. Direct mapping states rules to create URIs of ontology resources, and generate RDF triples using given table and column information. Direct mapping is often used as a starting point where other mapping approaches extend the resulting ontology.

Another approach is Domain Semantics-Driven Mapping [1]. Domain Semantics-Driven Mapping maps relational database to already existing ontologies, when existing ontology and relational database refer to same domain and thus similarity level is high (Figure 5). It is often called manual mapping because it relies on mapping description language which describes the mapping between relational schema and existing ontology. There is standard mapping language R2RML announced by W3C.

Augmented Direct Mapping [1] approach targets to improve the Direct Mapping approach. It applies more rules that can automatically detect and convey domain semantics from relational database design, in automatic manner (Figure 4). Ontologies created from augmented direct mapping is called "local ontology" or "putative ontology", since it is generated by restoring domain semantics based on relational design. Sometimes it is used in semi-automatic way, where domain experts checks the validity of result ontologies.

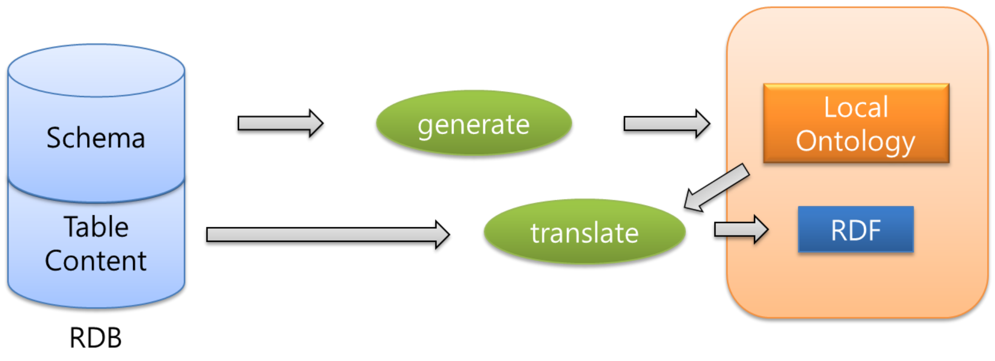


Figure 4 Augmented Direct Mapping

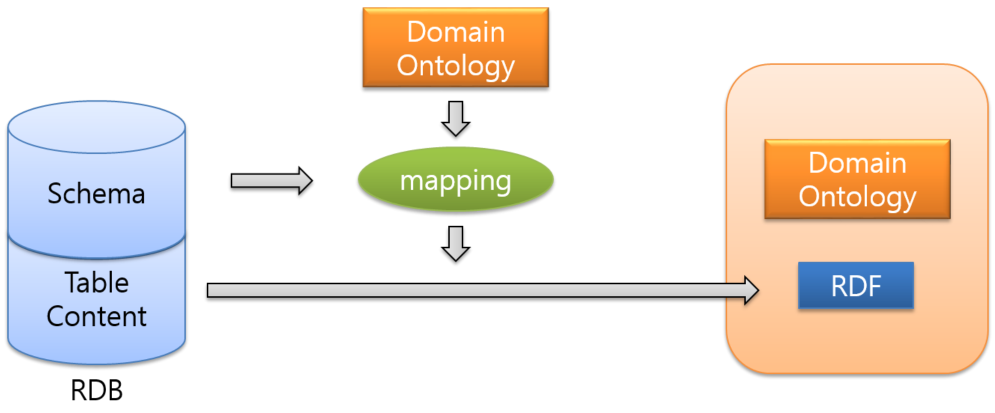


Figure 5 Domain Semantics-Driven Mapping

OWL [9] is a latest standard ontology language recommended by W3C. It facilitates greater machine interpretability and expression power of web content than other ontology representations such as RDFS [10]. OWL offers stronger and extended vocabulary and thus much enhanced inferencing capability. OWL has constructs that can describe some extended metadata on properties, such as cardinality and property characteristics (Figure 6). Thus, our system proposes an automatic ETL system that creates ontology by augmented direct mapping approach with OWL ontology.

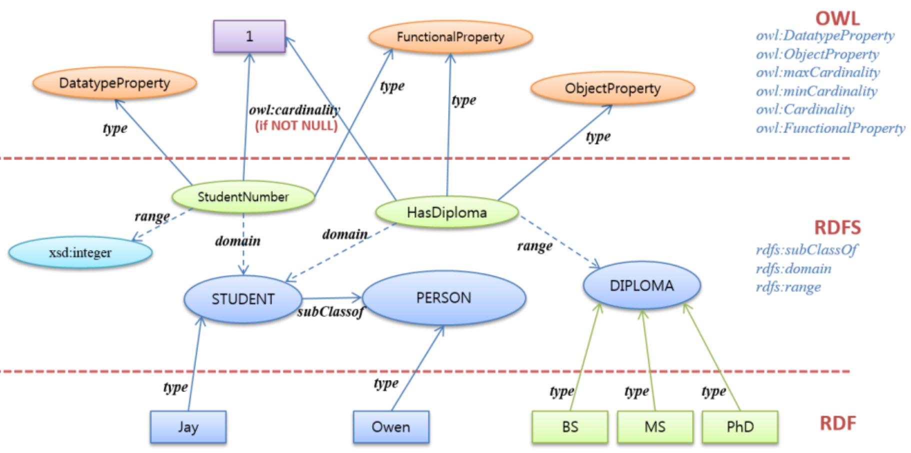


Figure 6 Example OWL ontology

The motivation for generating OWL ontology is explained by the goal of creating local ontology for the target database schema. The goal is to interpret all domain semantics contained in the database schema design, so that resulting ontology can describe the meta-information of the domain as much as possible. However, relational databases have some constraints that cannot be expressed by RDFS constructs. For example, recent version of MySQL database has five constraints that the designer can state on each column. They are NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, and DEFAULT. These constraints express some domain information about entities described by each table, and RDFS has no constructs that can be used for the constraints. Therefore, we use some constructs from OWL language, which can express richer domain information. With OWL constructs, we are able to specify property information that are matched from each SQL constraint.

Chapter 3

Mapping Rules

In this section, we present a set of mapping rules we implement in our system. The set of rules are augmented direct mapping rules, which builds a “putative” ontology through automatic mapping based on the database’s schema information [16]. To extract the semantics that is inherent in relational schema design, we use OWL language constructs in addition to RDF and RDFS constructs. We present augmented direct mapping rules that are used to create OWL ontology. We have eleven rules, five of which are used to map basic ontology structure such as ontology class and property construction. The other six rules are mapping rules we organized based on each SQL constraint. Five basic rules mostly agree with mapping rules that are researched in other studies, such as identifying binary relations or creating object properties. On the other hand, mapping rules on constraints are quite varied, and implicitly applied on other rules.

We define some first order logic predicates to organize mapping rules that our system follows to implement augmented direct mapping.

First, we have some predicates that match conditions on relational database tables, attributes, and constraints.

Rel(r)	r is a relation
BinRel(r, s, t)	r is a binary relation between relations s and t
Attr(x, r)	x is an attribute of r
NonFKey(x, r)	x is a non foreign key attribute in relation r
NotNull(x, r)	x is a attribute in relation r with NOT NULL constraint
Unique(x, r)	x is a attribute in relation r with UNIQUE constraint
PK(x, r)	x is a (single or multiple) primary key of relation r
FK(x, r, y, s)	x is a (single or multiple) foreign key of relation r, referencing y in relation s

Relational Database Predicates

We also have some predicates for OWL ontology, such as conditions of classes and properties.

Class(r)	r is an ontological class
Prop(x, r, s)	r is a rdf property with domain r and range s
DatatypeP(x, r, s)	x is a datatype property with domain r and range s
ObjP(x, r, s)	x is an object property with domain r and range s
FP(x)	x is a functional property
IFP(x)	x is an inverse functional property
Card(x, y)	Property x has cardinality of y
minCard(x, y)	Property x has minimum cardinality of y
maxCard(x, y)	Property x has maximum cardinality of y

Ontology Predicates

We also have some subordinate rules and functions that are used to define some concepts in relational database. These rules are used in our main mapping rules.

fkey(x, r, s)	Returns the foreign key defined on attributes x in r referencing s
AllColumns(r)	Returns all columns of relation r

☞ BinRel(r, s, t) ← Rel(r) ∧ FK(x, r, _, s) ∧ FK(y, r, _, t) ∧ x≠y ∧ x∪y = fkey(_, r, _) ∧ x∪y = AllColumns(r)

☞ NonBinFK(x, r, y, t) ← FK(x, r, y, t) ∧ Rel(r) ∧ Rel(t) ∧ ¬BinRel(r, _, _) ∧ ¬BinRel(t, _, _)

Subordinate rule and functions

We present a sample database schema to show the application of the mapping rules. Consider a relational database schema for a library system (Figure 7). We have five tables. The *USER* table contains data about all the users of the library. The *BORROW* table has all the information about book borrow records with foreign key columns to *USER* and *BOOK* table. The *BOOK* table has information of each book. The *AUTHOR* table describes the information of authors. *BOOKS_BY_AUTHOR* table shows the many-to-many relationship between books and authors by linking *BOOK* table and *AUTHOR* table together. The information below has DDL information of these tables including SQL constraints on the columns. The full version of generated OWL ontology from the sample database can be found in appendix.

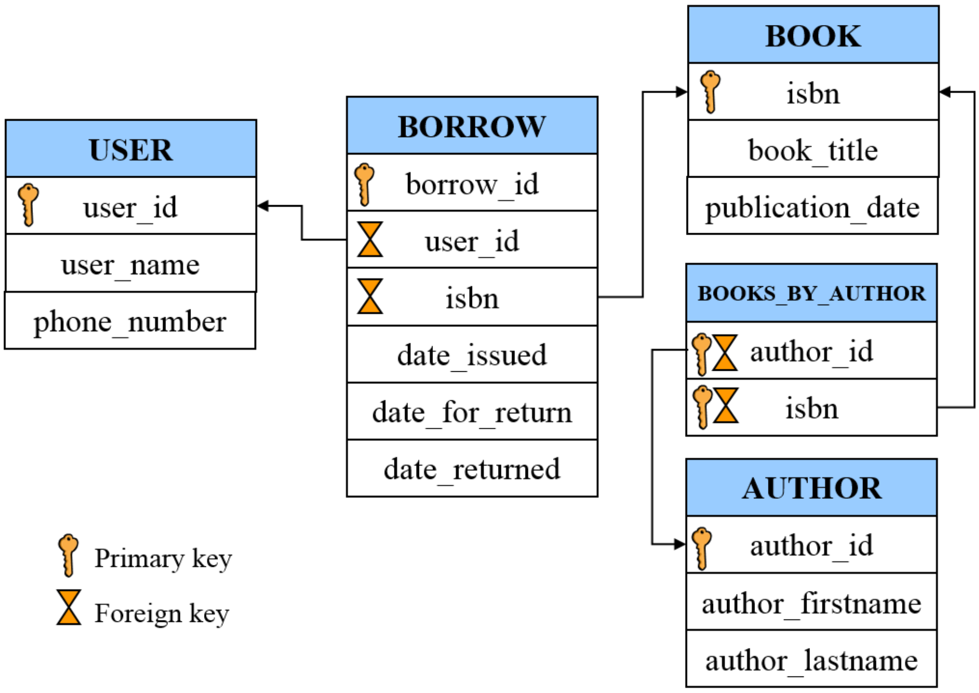


Figure 7 Library System Schema

```
Library System Schema
CREATE TABLE USER {
    user_id integer PRIMARY KEY,
    user_name varchar NOT NULL,
    phone_number integer
}
CREATE TABLE BORROW {
    borrow_id integer PRIMARY KEY,
    user_id integer REFERENCES USER(user_id),
    isbn integer REFERENCES BOOK(isbn),
    date_issued date,
    date_for_return date,
    date_returned date DEFAULT NULL
}
CREATE TABLE BOOK {
    isbn integer PRIMARY KEY,
    book_title varchar UNIQUE,
    publication_date date
}
CREATE TABLE BOOKS_BY_AUTHOR {
    author_id integer REFERENCES BOOK(isbn),
    isbn integer REFERENCES AUTHOR(author_id),
    CONSTRAINT pkey PRIMARY KEY (author_id, isbn)
}
CREATE TABLE AUTHOR {
    author_id integer PRIMARY KEY,
    author_firstname varchar,
    author_lastname varchar
}
```

SQL DDL for library system database schema

With above predicates and subordinate rules, we define set of rules for our automatic augmented direct mapping system. Table 1 shows the rules that are implemented in our system. The rules are expressed in first order logic for clarity.

Table 1 Mapping Rules

General Rule 1.	Class(r) ← Rel(r) ∧ ¬BinRel(r, _, _)
General Rule 2.	DatatypeP(x, r, type(x)) ← NonFKey(x, r)
General Rule 3.	ObjP(r, s, t) ← BinRel(r, s, t) ∧ ¬BinRel(s, _, _) ∧ ¬BinRel(t, _, _)
General Rule 4.	ObjP(x, s, t) ← NonBinFKey(x, s, y, t)
General Rule 5.	SubClass(r, s) ← Rel(r) ∧ Rel(s) ∧ PK(x, r) ∧ FK(x, r, _, s)
Constraint Rule 1.	FunctionalP(x), minCard(x, r, 0), maxCard(x, r, 1) ← AllColumns(x, r)
Constraint Rule 2.	Card(x, r, 1) ← NotNull(x, r)
Constraint Rule 3.	InverseFunctionalP(x) ← Unique(x, r)
Constraint Rule 4.	InverseFunctionalP(x) , Card(x, r, 1), ← PKey(x, r)
Constraint Rule 5.	ObjP(x, s, t) ← NonBinFKey(x, s, y, t)
Constraint Rule 6.	local:Default(x, r, a) ← DEFAULT(x, r, a)

3.1 General Rule 1

General rule 1 is for creating ontology classes. According to Direct Mapping, each relational table is mapped to ontological class. However, this rule rules out some special cases called binary relations. We interpret this case as a many-to-many relationship between the two relations that it is referencing, so these tables are mapped to object properties, according to general rule 3. All other relations are mapped to ontological classes. Therefore, from the sample database, all tables except

BOOK_BY_AUTHOR table are mapped to ontological classes (Figure 8).

3.2 General Rule 2

General rule 2 is for declaring datatype property for non-foreign key columns. Non-foreign key columns have literal values, so the column is stated to be datatype property having its relation as domain and its datatype as range. For example, *user_name* column of *USER* table is mapped to a datatype property with domain *USER* and range string (Figure 8).

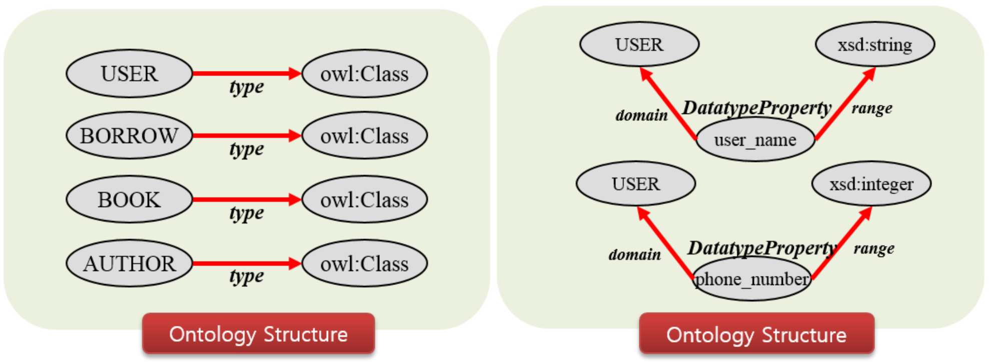


Figure 8 General Rule 1(left) and General Rule 2(right)

3.3 General Rule 3

General rule 3 maps binary relations to object properties. Binary relations are relations that have only two foreign keys and no other attributes. Binary relations are many-to-many relationships between two entities, so we state to be object property having two referenced entities as its domain and range. From the sample database, *BOOKS_BY_AUTHOR* is identified as binary relation and mapped to an object property (Figure 9).

3.4 General Rule 4

General rule 4 is for mapping foreign key columns to object properties. Foreign key columns are columns that reference columns in other tables. According to Direct Mapping, reference triples are created which have other table's row as its object. Therefore the column can be mapped to object property. *user_id* column in *BORROW* table is in this case (Figure 9).

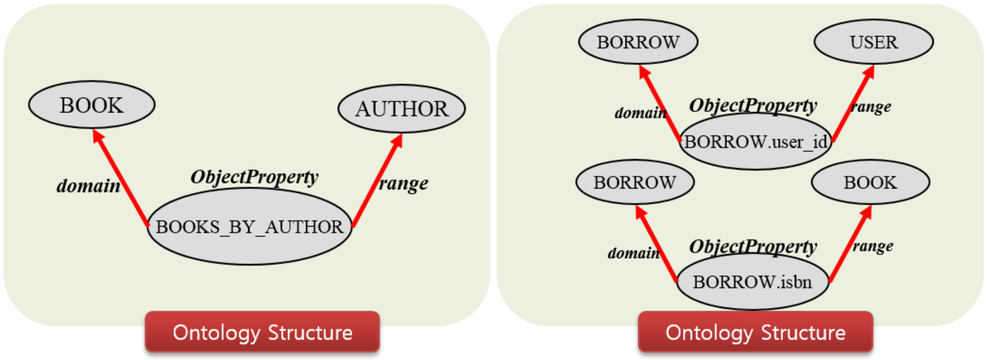


Figure 9 General Rule 3(left) and General Rule 4(right)

3.5 General Rule 5

General rule 5 finds subclass relationships between created ontological classes. When the primary key of a relation is foreign key at the same time, we see the relation as a dependent relation of the referenced relation, and thus declare it as a subclass of referenced relation.

3.6 Constraint Rule 1

The first constraint rule states that for all columns in relational database, it can be stated that the property relevant to the column has minimum cardinality of 0, maximum cardinality of 1, and thus be a functional property (Figure 10). It can be justified that in relational database, each cell has no more than one value (as long as first normal form is kept), so the corresponding property in created ontology can be said to have maximum cardinality of 1. Functional property in OWL ontology is a property that can have only one (unique) value for each instance [9], so it semantically matches the notion of atomicity in relational database.

3.7 Constraint Rule 2

Constraint rule 2 states that if a column has NOT NULL constraint declared, it can be transferred in OWL ontology with cardinality 1. This makes sense because null value corresponds to the lower limit of cardinality 0. With NOT NULL constraint, the possibility of no-value property disappear and we can strictly announce the cardinality to be 1.

From the sample database, *user_name* in *USER* table has NOT NULL constraint and can be stated to have cardinality of 1 (Figure 10).

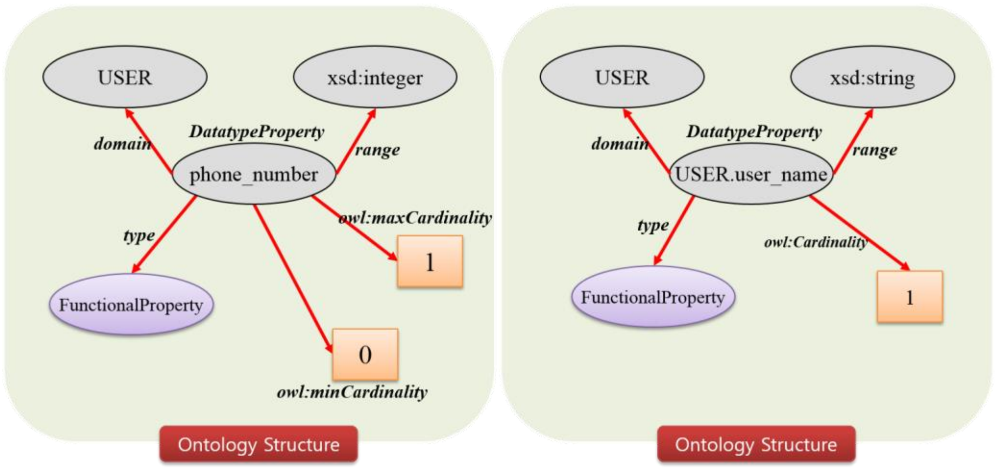


Figure 10 Constraint Rule 1(left) and Constraint Rule 2(right)

3.8 Constraint Rule 3

Third constraint rule maps for another SQL constraint, UNIQUE. UNIQUE constraint in relational database denotes that the column uniquely defines each row, with unique values. OWL language has matching property, Inverse Functional Property. Inverse Functional Property states that the object of a property statement uniquely determines the subject, asserting that there cannot be two distinct instances x1 and x2 such that both pairs (x1, y) (x2, y) are instances of the property. From the sample database, *book_title* in *BOOK* table has UNIQUE constraint and can be stated to be inverse functional property (Figure 11).

3.9 Constraint Rule 4

Constraint Rule 4 covers primary key constraint. Primary key constraint in SQL implicitly states that the column is both NOT NULL and UNIQUE. Therefore, following the reasonings from Constraint Rule 2 and Constraint Rule 3, cardinality of 1 and inverse functional property can be declared (Figure 11).

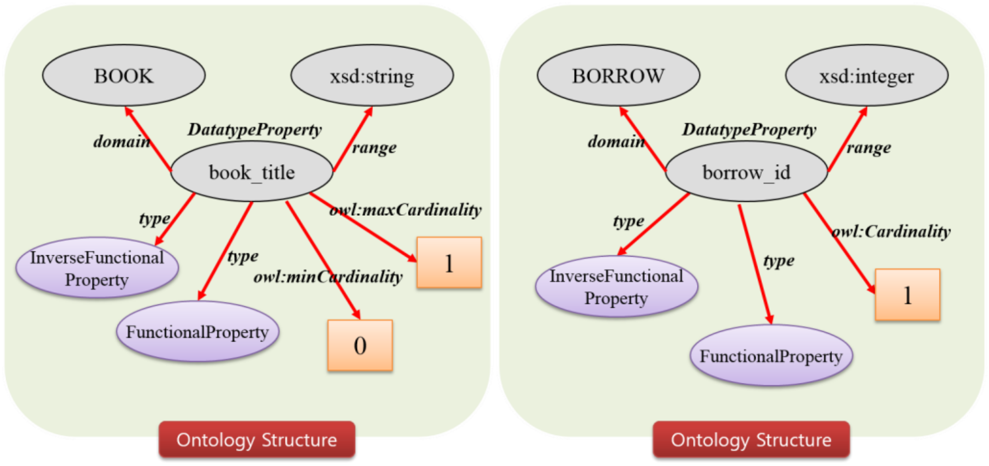


Figure 11 Constraint Rule 3(left) and Constraint Rule 4(right)

3.10 Constraint Rule 5

Constraint Rule 5 is for mapping foreign key constraint. Foreign key constraint references to columns in other table, which are also mapped to an ontology class. Thus, the column links two ontology class entities, so we state that the property as an object property (Figure 12).

3.11 Constraint Rule 6

Constraint Rule 6 takes care of DEFAULT constraint in SQL. As of now, OWL ontology language does not have construct that can describe a default value for a property. On this account, current studies on RDB2RDF mapping leave this constraint out when describing OWL metadata. It is quite disappointing that some SQL semantics cannot be expressed in ontology with standard language, but it is a phenomenon resulting from the difference in expressing power of SQL and ontology. In our system, we take care of this DEFAULT constraint by creating local construct, *local:Default*, which can be used internally for reference (Figure 12). By carrying and knowing that the ontology has such variable would enable creating triples conforming to the constraints on necessity, or replacing to other constructs if standard constructs for default values are announced in the future. From the sample database, *date_returned* column in *BORROW* relation has DEFAULT constraint, so a triple having *local:Default* property is created.

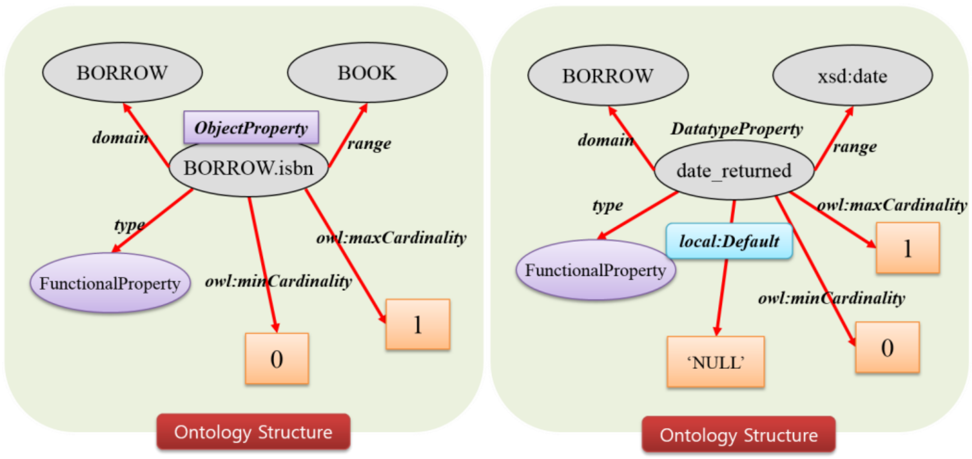


Figure 12 Constraint Rule 5(left) and Constraint Rule 6(right)

3.12 Discussion

In addition to applying the semantic rules, we also applied creating local constructs on other SQL constraints, NOT NULL and UNIQUE. So, we create the constructs *local:Notnull* and *local:Unique* to give internal reference. It may be used in optional ways, such as supporting reverse-restoring the relational schema from the resulting ontology, if necessary. This has a similar concept of reification used in Semantic Web, where they make statements about other statements by adding triples for more information.

There are some variations in other related studies in applying mapping rules, such as basic class structures, inheritance, and N-ary relationships. *Li et al.* [10] integrates the information in several relations into one ontological class, when the relations have equal primary keys. This correctly integrates if vertical partitioning concept is used when designing relational schema, since information on a concept is spread on several relations. The study also has stricter rules on inheritance relationships, because it requires the primary keys have to be equal. However, the two rules conflict with each other, and the user has to decide on which rule to follow when two relations have equal primary keys [18]. *Li et al.* also has special rules on N-ary relationships, where most other works only identify binary relations. This rule can also be controversial, since it likely to produce unnecessary relationships. *Astrova et al.* [21] has some rules specifying property characteristics, such as symmetric property and transitive property, but they are controversial, since they may or may not be correct depending on the domain information [18].

Chapter 4

Our Approach

In this section, we show our approach and architecture of transforming relational data into RDF formatted data. In order to accomplish our goals, we use Hadoop framework for scalability on large-sized data. Hadoop provides a distributed file system as well as the implementation of MapReduce, which is a powerful model for processing large datasets in a distributed environment. Preprocessing on RDB dump data and schema are done for efficient MapReduce process.

4.1 Preprocessing

Our system utilizes several open source libraries and tools. The primary external libraries that are used are:

- MySQL JDBC Connector [6] provides a JDBC driver API that can connect to MySQL databases and enables querying or updating database contents.
- Jena Libraries provide [7] APIs to create and read RDF graphs, and serialize the graphs using various formats such as RDF/XML, N-triples, etc. Jena also offers libraries for creating RDFS or OWL ontologies using memory-based models.

- Apache Elephas [8] is a set of libraries which provides various basic blocks which enable processing and creating RDF data in Hadoop environments.

In order to process RDB data using Hadoop framework, some preprocessing works need to take precedence. First, the dump of relational data needs to be extracted from the databases. Second, extracted RDB dump data need to be pre-processed so that it can be used as Hadoop input. This step brings better efficiency when developing MapReduce algorithm, and overall performance. Our system accepts line-based input data from RDB, with each line corresponds to each row in target relational database. However, the current mapping rules require both row values and the column information corresponding to the values to create literal triples, reference triples, etc. To afford the column information when processing each input line, we apply separate preprocessing and handling on relational schema data. The overall architecture of our system is shown in Figure 13.

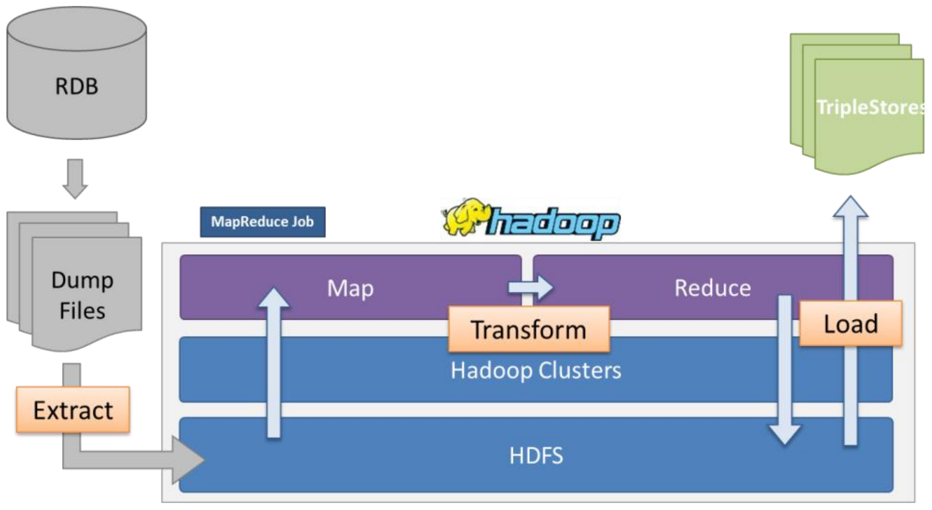


Figure 13 System Architecture

4.1.1 Schema Caching Method

Our system processes relational schema information into a separate file, and passes the file as an input to the distributed cache in Hadoop. When the system connects to the target relational database, it first extracts table names and column information in each table such as column name, type, key information (primary/foreign key), and constraints on columns.

Distributed cache transmits read-only files to each node in distributed network. Therefore, the schema information should be preprocessed so that it takes up least amount of volume to minimize the network congestion. However, caching schema information separately this way enables the system to load the schema information on local memory only once on the beginning of each node, instead of reading same information repeatedly from disk for every key/value input. In order to load cache files in memory, we implemented internal object classes which can store schema information when read from distributed cache in each node.

As shown in the Figure 14, the system creates schema data text file from the database. Each line in the file corresponds to information about each column, and shows whether the column is primary key, foreign key, referenced table/column, and constraints information such as NOT NULL and UNIQUE in tab-separated basis.

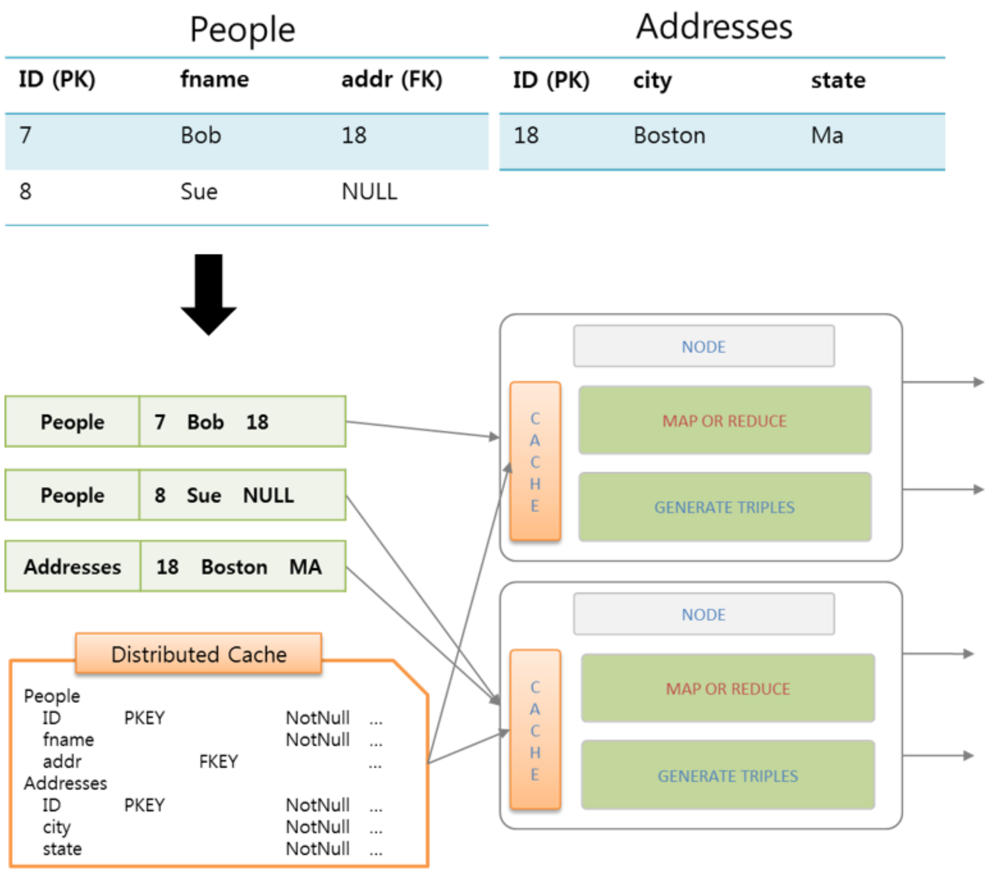
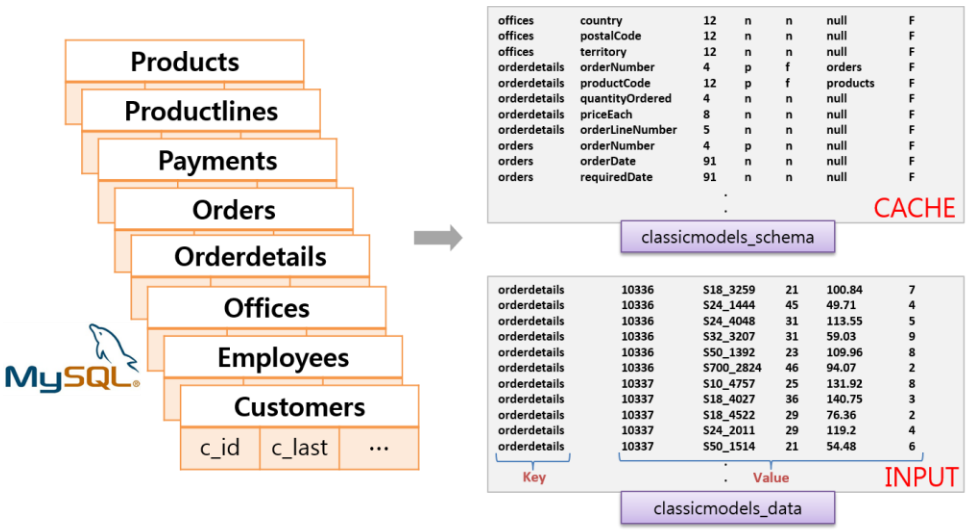


Figure 14 Schema caching method

4.1.2 Relational Data

The content of relational tables are preprocessed in addition to schema file. Since all the column information has been extracted in separate file, data file only contains values in each row along with the table name, which is used as a key in Hadoop input. The data is extracted using JDBC connection to database, which enables querying for each row in each table and written in preferred format in dump file.

As shown in the Figure 15, pre-processed input files for Hadoop are tab-separated text file. Each line represents the values from each row in relational table, with the table name at the beginning.



```
offices        country           12  n  n  null      F
offices        postalCode        12  n  n  null      F
offices        territory         12  n  n  null      F
orderdetails   orderNumber       4   p  f  orders    F
orderdetails   productCode       12  p  f  products  F
orderdetails   quantityOrdered   4   n  n  null      F
orderdetails   priceEach         8   n  n  null      F
orderdetails   orderLineNumber   5   n  n  null      F
orders         orderNumber       4   p  n  null      F
orders         orderDate         91  n  n  null      F
orders         requiredDate      91  n  n  null      F
                          .
                          .
```

```
orderdetails   10336   S18_3259   21   100.84   7
orderdetails   10336   S24_1444   45   49.71    4
orderdetails   10336   S24_4048   31   113.55   5
orderdetails   10336   S32_3207   31   59.03    9
orderdetails   10336   S50_1392   23   109.96   8
orderdetails   10336   S700_2824  46   94.07    2
orderdetails   10337   S10_4757   25   131.92   8
orderdetails   10337   S18_4027   36   140.75   3
orderdetails   10337   S18_4522   29   76.36    2
orderdetails   10337   S24_2011   29   119.2    4
orderdetails   10337   S50_1514   21   54.48    6
                          .
                          .
```

Figure 15 Preprocessing of relational schema and data

4.2 Hadoop Algorithm

In this section, we show the implementation of Hadoop process which transforms preprocessed input file. To make it as simple as possible, we designed a map-only Hadoop job that generates RDF triples in parallel manner.

Each data node in Hadoop job starts with reading distributed cache files into its local storage. In our system, it is carried out in the setup method which is called at the beginning of every map or reduce task for initialization. It reads in the cached files, and creates instances of internal class objects which includes table and column metadata read from each line in the cached file. By having this list of these object instances, each node can accomplish its task independent of other nodes.

According to Direct Mapping methods [11] that are announced by W3C as a standard, three kinds of RDF triples are generated from relational databases. The three types are table triple, literal triple, and reference triple. Table triples can be generated straight from input key/value pair, since input has the table name as the key and primary key information is cached in memory. Literal triple requires row id, column name, and the value, so literal triple can also be generated from input key/value along with column information in memory. To generate reference triple, it is necessary to know which table the column is referencing to, and which column in that table is referenced. To enable generating reference triples, referenced table and column information is included in schema cache file.

As described above, all triples from Direct Mapping approach can be generated in one Map task, by virtue of having common schema information of the entire database loaded in each node's memory. This makes the job a dividable process, where each input pair can be processed independently of other input pairs. Therefore the job could be designed as map-only, which brings efficiency by avoiding the network traffic costs that would have been taken for shuffling and transmitting intermediate values to reducers.

Algorithm 1: Algorithm for generating triples (Direct Mapping)

```
1: class Mapper
2:     method Setup()
3:         C <- LoadSchema()
4:     method Map(Text table, Text V)
5:         if (table == "!")
6:             createOntology(C)
7:         if (table is not BinaryRelation)
8:             T ← generate type triple
9:             Emit(T)
10:        for all words v ∈ V do
11:            if (v is Fkey)
12:                T ← generate reference triple
13:                Emit(T)
14:            else
15:                T ← generate literal triple
                   Emit(T)
```

Algorithm 1 describes how each map task processes cache data and input data to map relational dump to RDF triples. In *Setup( )* method, each node checks for the cached data and calls *LoadSchema( )* method to read in cache file. In Map function, it takes each line in dump data file, and gets the table name as key and list of values for specific row as value. First, algorithms checks if the key equals to predefined symbol (this case '!'), to call *createOntology( )* function. This designates only one mapper specifically to do the job of creating OWL ontology according to augmented direct mapping rules using loaded schema objects.

Chapter 5

Experiment

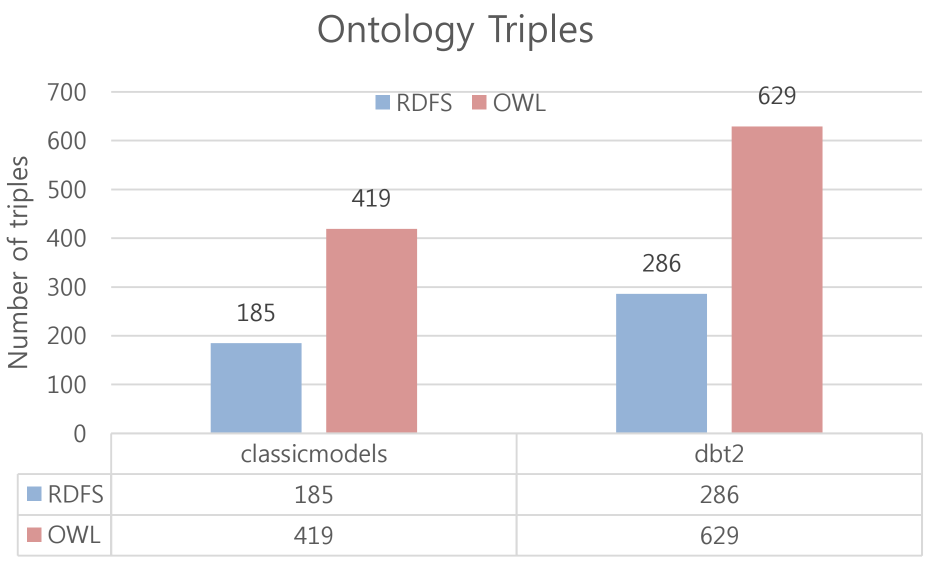
Our Hadoop cluster consists of 9 physical machines; each node equipped with Inter i5 Quad-Core 3.1GHz, 4GB RAM, and 8TB HDD. The cluster runs Ubuntu 10.10 with Hadoop 2.6.0.

The dataset used for the experiment is DBT2 benchmark data. DBT2 is a benchmark for MySQL databases and it mimics an OLTP application for a company owning large amounts of warehouse [20]. It contains transactions to handle New Orders, Order Entry, Order Status, Payment, etc. in addition to arbitrarily building databases with varying number of warehouses.

RDF has several serialization format that are possible when written to a file. Several serialization formats are RDF/XML, N-TRIPLE, N3, etc. The serialization format that was adopted by our experiments are N-Triples, since it is simple, easy-to-parse, and most importantly, line-based format. Our system is able to write in many serialization formats that are supported by Jena library (RDF/XML, N-Triple, N3, Turtle, N-Quads etc.). However, serialization formats such as RDF/XML, are file-based, which means that the whole graph that is to be written has to be generated all and written to file at once. Thus, in such serializations the constructed graph needs to store up all the space in memory and written out in one go at the end. This could be a problem when processing large-sized databases, due to memory limitation. Line-based formats can be written to file in a streaming fashion, relieving the memory problem.

5.1 Ontology Extraction

First, we experimented the amount of ontology triples that are constructed after transforming to RDF data. We ran our system with all the mapping rules presented in section 3.1, and also ran a system which implements only mapping rules that are in the scope of RDFS language. So we compared the number of triples that are extracted.



	classicmodels	dbt2
🟦 RDFS	185	286
🟥 OWL	419	629

Figure 16 Number of Ontology Triples Extracted

Figure 16 shows the number of ontology triples that are generated on two sample databases, classicmodels and DBT2 database. When all constraints rules are applied, generated ontology triples increased by about 120% than only RDFS triples are used. This shows that using OWL constructs generates much richer ontology than other ETL systems such as [4].

5.2 Performance

Figure 17 shows the performance experiment result conducted with varying size of databases. Five DBT2 datasets are prepared, with varying number of warehouses. Precise sizes of the datasets are in Table 2. From the graph, it shows that the Job completion time increases in linear scale, unlike when ran in single node system. In table 2, we can also see that the sizes of outputs and the number of triples generated. However, in the figure, the performance difference is not much overwhelming compared to single node implementation. This attributes to relatively small size of datasets, since Hadoop shows more efficient performance in large-sized batch data.

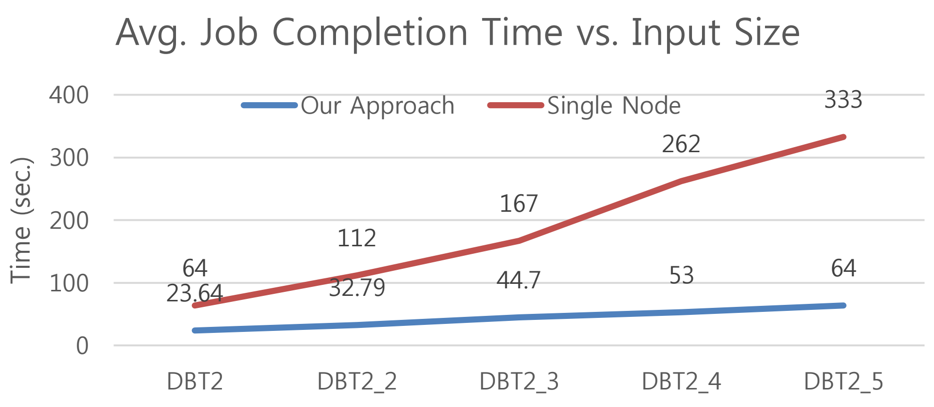


Figure 17 Average job completion time

Table 2 Data size of each database

Input Size	DBT2	DBT2_2	DBT2_3	DBT2_4	DBT2_5
Database Size (MB)	102	193	282	370	461
Output Size (MB)	575	1104	1632	2165	2696
Number of Triples	5,121,505	9,738,278	14,357,620	18,982,849	23,606,636

Figure 18 shows the performance experiments result conducted with larger databases, from 5-warehouse dataset to 25-warehouse dataset. The graph also shows linear increase as the data size grows. The database size, output size, and the number of triples are shown in Table 3. As seen from the results, the completion time with the system takes less time than the completion time of single node in Figure 17, despite of a five-fold increase in data size. According to the result, the system can generate a hundred million RDF triples in reasonable time (2-3 min.).

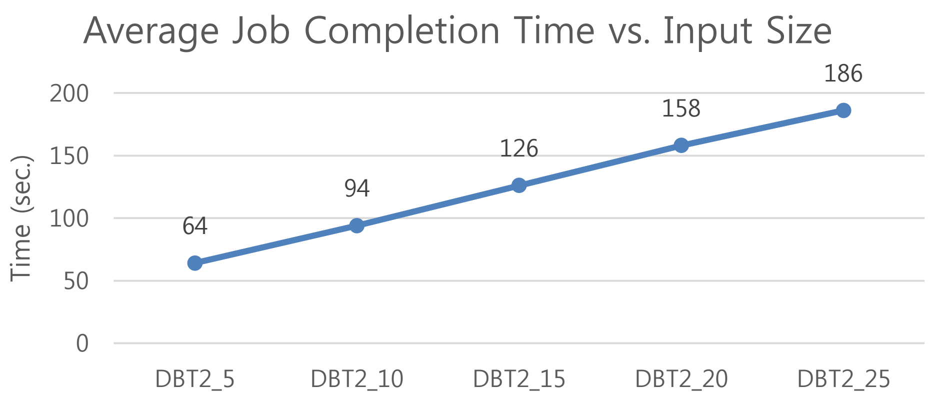


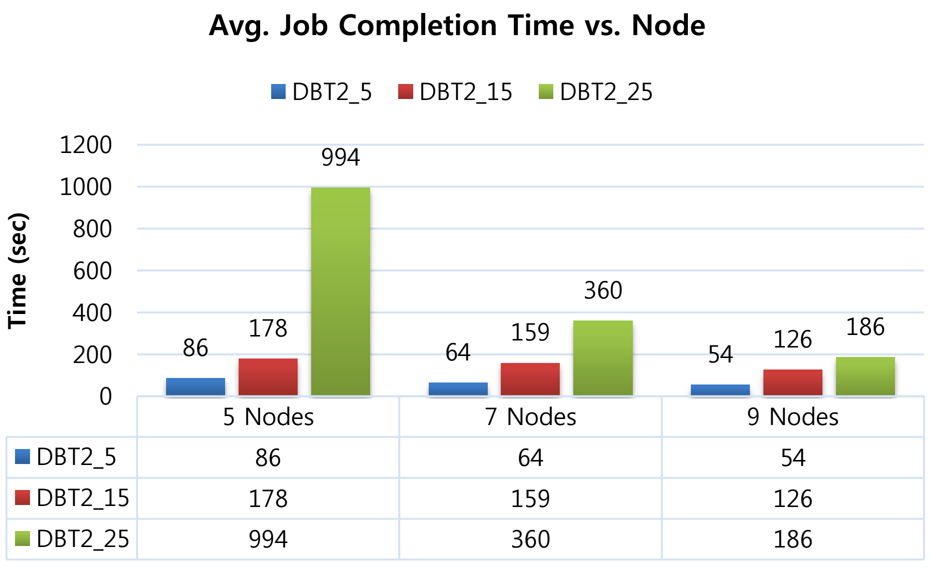
Figure 18 Average job completion time on larger databases

Table 3 Data size of each database

Input Size	DBT2_5	DBT2_10	DBT2_15	DBT2_20	DBT2_25
Database Size (MB)	461	907	1355	1804	2248
Output Size	2696	5354	8015	10708	13383
Number of Triples	23,606,636	46,817,230	70,059,115	93,566,504	116,512,398

5.3 Scalability

Figure 19 shows that the Hadoop-based distributed system is scalable. We tested the system with varying number of nodes to see scalability. We ran on databases with 5, 15, 25 warehouses, and the result shows that adding nodes improves the performance. The result shows that the difference in completion time is much significant with large dataset. With 25-warehouse database, which has size of about 2.3GB, the performance increases about 80% when number of nodes is increased from 4 to 9. The result shows that the ETL system can be extended with more machines as occasion demands, and can show satisfactory performance on large databases.



	5 Nodes	7 Nodes	9 Nodes
🟦 DBT2_5	86	64	54
🟥 DBT2_15	178	159	126
🟩 DBT2_25	994	360	186

Figure 19 Average job completion time with different number of nodes

Chapter 6

Conclusion

Overall, this paper has presented a RDB2RDF ETL system that extracts relational data and schema to create RDF data and OWL ontology. Transforming relational data to RDF data is crucial. It has been called “chicken-and-egg” dilemma that a critical amount of semantic data must be available for novel semantic applications to arise and semantic web’s success [1]. There are several approaches to expose relational data to semantic web, including “wrapper” and ETL approaches. However, current works on RDB2RDF ETL system have not experimented transforming databases in distributed environments and large datasets. There are legacy databases that need to be transformed and integrated into semantic web, and considering the amount of data in the web backed by relational databases, it is indispensable to test RDB2RDF ETL tools in a larger scale.

We have implemented and tested automatic system that transforms relational data into RDF-formatted data. We utilize Hadoop framework to build a distributed system, which exhibits high-performance and scalable results on large-sized databases. Furthermore, the system also makes use of database schema information to create database specific local ontology, fulfilling augmented direct mapping. We presented mapping rules that maps SQL metadata to OWL ontology constructs. The mapping rules include rules that utilize SQL constraints information to create extended ontology. The system generates more ontology information compared to other RDB2RDF ETL tools that implements augmented direct mapping. In implementing the ETL system, we have presented special preprocessing that are necessary to relational data dumps that are to be inputted to Hadoop system. In consideration of the characteristics of relational data and generating triples following direct mapping rules, we designed a special preprocessing step for schema data, and the schema caching method that utilizes Hadoop distributed cache, so that the Hadoop MapReduce algorithm are optimized.

Future work of this research could be planned in several ways. On ontological perspective, mapping rules for more semantics from database schema can be researched. There are other concepts in SQL databases such as SQL triggers. Mapping rules for SQL triggers has been studied in several works, but it has been controversial about the ambiguousness of applying rules on triggers. Connecting to various relational databases other than MySQL is another expansion of the system, since our system only has been tested on MySQL.

Reference

[1] Michel, Franck, Johan Montagnat, and Catherine Faron-Zucker. "A survey of RDB to RDF translation approaches and tools." (2013).

[2] Vavliakis, Konstantinos N., Theofanis K. Grollios, and Pericles A. Mitkas. "RDOTE–Publishing Relational Databases into the Semantic Web." Journal of Systems and Software 86.1 (2013): 89-99.

[3] Konstantinou, Nikolaos, Dimitris Kouis, and Nikolas Mitrou. "Incremental Export of Relational Database Contents into RDF Graphs." Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14). ACM, 2014.

[4] Thuy, Pham Thi Thu, et al. "RDB2RDF: completed transformation from relational database into RDF ontology." Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication. ACM, 2014.

[5] Patroumpas, Kostas, et al. "TripleGeo: an ETL Tool for Transforming Geospatial Data into RDF Triples." EDBT/ICDT Workshops. 2014.

[6] MySQL Connector, Available from: http://dev.mysql.com/doc/connector-j/en/index.html [Accessed 2 May 2015].

[7] Apache Jena, Available from: https://jena.apache.org/index.html [Accessed 2 May 2015].

[8] Apache Jena Elephas, Available from: https://jena.apache.org/documentation/hadoop/ [Accessed 2 May 2015].

[9] W3C Working Group, "OWL Web Ontology Language," Available from: http://www.w3.org/TR/owl-ref/ [Accessed 2 May 2015].

[10] Li, Man, Xiao-Yong Du, and Shan Wang. "Learning ontology from relational database." Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on. Vol. 6. IEEE, 2005.

[11] W3C Working Group, “A Direct Mapping of Relational Data to RDF,” Available from: http://www.w3.org/TR/2011/WD-rdb-direct-mapping-20110324/ [Accessed 2 May 2015].

[12] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.

[13] Tirmizi, Syed Hamid, Juan Sequeda, and Daniel Miranker. "Translating sql applications to the semantic web." Database and Expert Systems Applications. Springer Berlin Heidelberg, 2008.

[14] W3C Working Group, Available from: http://www.w3.org/2001/sw/rdb2rdf/ [Accessed 2 May 2015].

[15] Apache Hadoop, Available from: https://hadoop.apache.org/ [Accessed 2 May 2015].

[16] Sequeda, Juan F., Rudy Depena, and Daniel P. Miranker. "Ultrawrap: Using sql views for rdb2rdf." Proc. of ISWC2009 (2009).

[17] RDB2RDF: Relational Database to RDF, Available from: http://www.rdb2rdf.org/ [Accessed 2 May 2015].

[18] Sequeda, Juan F., et al. "Survey of directly mapping sql databases to the semantic web." Knowledge Engineering Review 26.4 (2011): 445-486.

[19] Husain, Mohammad Farhan, et al. "Storage and retrieval of large rdf graph using hadoop and mapreduce." Cloud Computing. Springer Berlin Heidelberg, 2009. 680-686.

[20] MySQL Benchmark Tool, Available from: https://dev.mysql.com/downloads/benchmarks.html [Accessed 2 May 2015].

[21] Astrova, Irina, Nahum Korda, and Ahto Kalja. "Rule-based transformation of SQL relational databases to OWL ontologies." Proceedings of the 2nd International Conference on Metadata & Semantics Research. 2007.

Appendix

1. Taxonomy of RDB2RDF Data Management [17]

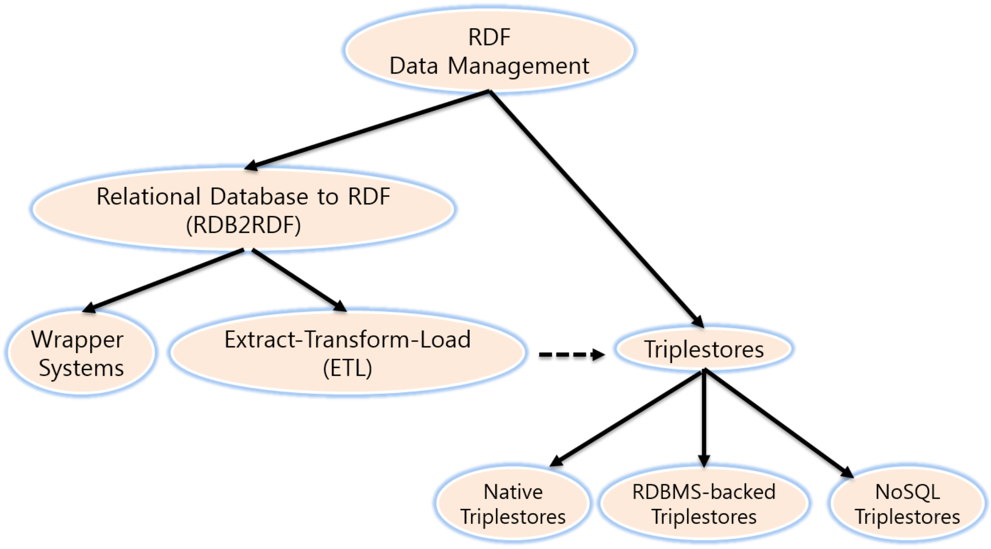

RDF Data Management
Relational Database to RDF (RDB2RDF)
Wrapper Systems
Extract-Transform-Load (ETL)
Triplestores
Native Triplestores
RDBMS-backed Triplestores
NoSQL Triplestores

Figure 20 Taxonomy of RDB2RDF Data Management

2. Three Layers of SQL database vs. OWL ontology [18]

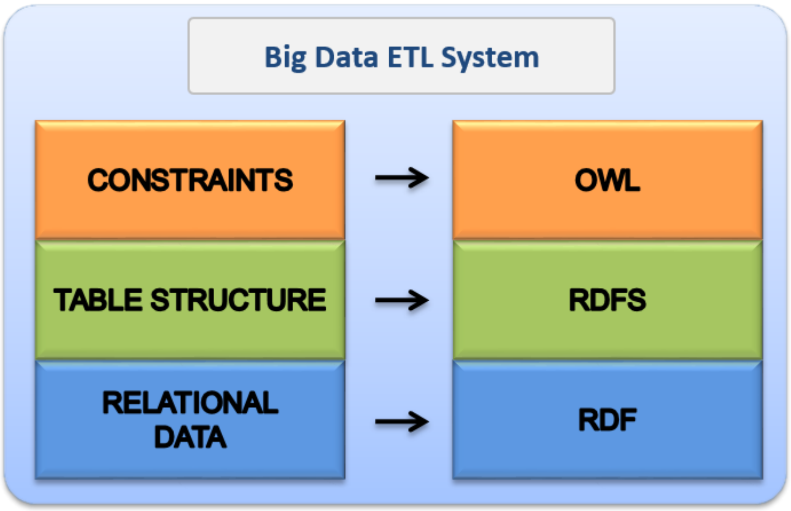


Figure 21 Three Layers of SQL database vs. OWL ontology

3. OWL Constructs

- owl:Cardinality
  - Permits the specification of exactly the number of elements in a relation
  - owl:minCardinality, owl:maxCardinality

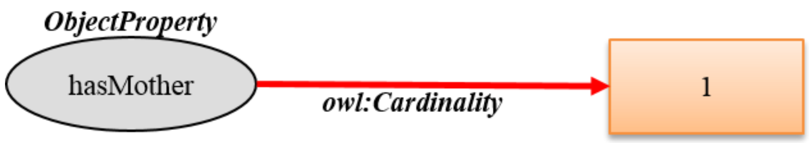


Figure 22 owl:Cardinality

- owl:FunctionalProperty
  - A functional property can have only one value for any individual

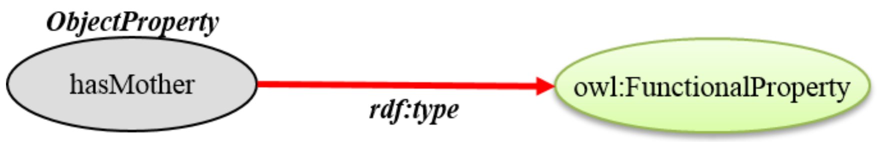


Figure 23 owl:FunctionalProperty

- owl:InverseFunctionalProperty
  - An inverse functional property can have only one individual as a subject for any value

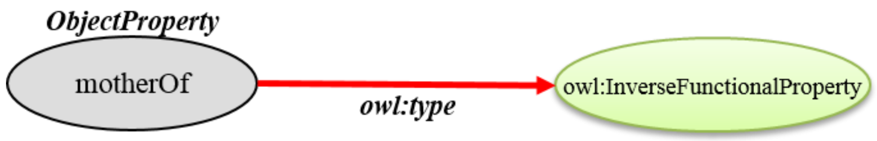


Figure 24 owl:InverseFunctionalProperty

4. Created ontology from sample library system database

Generated Ontology
<USER> <rdf:type> <owl:Class> .
<BORROW> <rdf:type> <owl:Class> .
<BOOK> <rdf:type> <owl:Class> .
<AUTHOR> <rdf:type> <owl:Class> .
<USER.user_id> <rdf:type> <owl:DatatypeProperty> .
<USER.user_id> <rdfs:domain> <USER> .
<USER.user_id> <rdfs:range> <xsd:integer> .
<USER.user_name> <rdf:type> <owl:DatatypeProperty> .

```
<USER.user_name>     <rdfs:domain>     <USER> .
<USER.user_name>     <rdfs:range>     <xsd:string> .
<USER.phone_number>     <rdf:type>     <owl:DatatypeProperty> .
<USER.phone_number >     <rdfs:domain>     <USER> .
<USER.phone_number >     <rdfs:range>     <xsd:integer> .
<BORROW.borrow_id>     <rdf:type>     <owl:DatatypeProperty> .
<BORROW.borrow_id>     <rdfs:domain>     <BORROW> .
<BORROW.borrow_id>     <rdfs:range>     <xsd:integer> .
<BORROW.user_id>     <rdf:type>     <owl:ObjectProperty> .
<BORROW.user_id>     <rdfs:domain>     <BORROW> .
<BORROW.user_id>     <rdfs:range>     <USER> .
<BORROW.isbn>     <rdf:type>     <owl:ObjectProperty> .
<BORROW.isbn>     <rdfs:domain>     <BORROW> .
<BORROW.isbn>     <rdfs:range>     <BOOK> .
<BORROW.date_issued>     <rdf:type>     <owl:DatatypeProperty> .
<BORROW.date_issued>     <rdfs:domain>     <BORROW> .
<BORROW.date_issued>     <rdfs:range>     <xsd:date> .
<BORROW.date_for_return>     <rdf:type>     <owl:DatatypeProperty> .
<BORROW.date_for_return>     <rdfs:domain>     <BORROW> .
<BORROW.date_for_return>     <rdfs:range>     <xsd:date> .
<BORROW.date_returned>     <rdf:type>     <owl:DatatypeProperty> .
<BORROW.date_returned>     <rdfs:domain>     <BORROW> .
<BORROW.date_returned>     <rdfs:range>     <xsd:date> .
<BOOK.isbn>     <rdf:type>     <owl:DatatypeProperty> .
<BOOK.isbn>     <rdfs:domain>     <BOOK> .
<BOOK.isbn>     <rdfs:range>     <xsd:integer> .
<BOOK.book_title>     <rdf:type>     <owl:DatatypeProperty> .
<BOOK.book_title>     <rdfs:domain>     <BOOK> .
<BOOK.book_title>     <rdfs:range>     <xsd:string> .
<BOOK.publication_date>     <rdf:type>     <owl:DatatypeProperty> .
<BOOK.publication_date>     <rdfs:domain>     <BOOK> .
<BOOK.publication_date>     <rdfs:range>     <xsd:date> .
<BOOKS_BY_AUTHOR>     <rdf:type>     <owl:ObjectProperty> .
<BOOKS_BY_AUTHOR>     <rdfs:domain>     <BOOK> .
<BOOKS_BY_AUTHOR>     <rdfs:range>     <AUTHOR> .
<AUTHOR.author_id>     <rdf:type>     <owl:DatatypeProperty> .
```

```
<AUTHOR.author_id>     <rdfs:domain>     <AUTHOR> .
<AUTHOR.author_id>     <rdfs:range>     <xsd:integer> .
<AUTHOR.firstname>     <rdf:type>     <owl:DatatypeProperty> .
<AUTHOR.firstname>     <rdfs:domain>     <AUTHOR> .
<AUTHOR.firstname>     <rdfs:range>     <xsd:string> .
<AUTHOR.lastname>     <rdf:type>     <owl:DatatypeProperty> .
<AUTHOR.lastname>     <rdfs:domain>     <AUTHOR> .
<AUTHOR.lastname>     <rdfs:range>     <xsd:string> .
<USER.user_id>     <rdf:type>     <owl:FunctionalProperty> .
<USER.user_id>     <owl:minCardinality>     "0" .
<USER.user_id>     <owl:maxCardinality>     "1" .
<USER.user_name>     <rdf:type>     <owl:FunctionalProperty> .
<USER.user_name>     <owl:minCardinality>     "0" .
<USER.user_name>     <owl:maxCardinality>     "1" .
<USER.phone_number>     <rdf:type>     <owl:FunctionalProperty> .
<USER.phone_number>     <owl:minCardinality>     "0" .
<USER.phone_number>     <owl:maxCardinality>     "1" .
<BORROW.borrow_id>     <rdf:type>     <owl:FunctionalProperty> .
<BORROW.borrow_id>     <owl:minCardinality>     "0" .
<BORROW.borrow_id>     <owl:maxCardinality>     "1" .
<BORROW.user_id>     <rdf:type>     <owl:FunctionalProperty> .
<BORROW.user_id>     <owl:minCardinality>     "0" .
<BORROW.user_id>     <owl:maxCardinality>     "1" .
<BORROW.isbn>     <rdf:type>     <owl:FunctionalProperty> .
<BORROW.isbn>     <owl:minCardinality>     "0" .
<BORROW.isbn>     <owl:maxCardinality>     "1" .
<BORROW.date_issued>     <rdf:type>     <owl:FunctionalProperty> .
<BORROW.date_issued>     <owl:minCardinality>     "0" .
<BORROW.date_issued>     <owl:maxCardinality>     "1" .
<BORROW.date_for_return>     <rdf:type>     <owl:FunctionalProperty> .
<BORROW.date_for_return>     <owl:minCardinality>     "0" .
<BORROW.date_for_return>     <owl:maxCardinality>     "1" .
<BORROW.date_returned>     <rdf:type>     <owl:FunctionalProperty> .
<BORROW.date_returned>     <owl:minCardinality>     "0" .
<BORROW.date_returned>     <owl:maxCardinality>     "1" .
<BOOK.isbn>     <rdf:type>     <owl:FunctionalProperty> .
```

```
<BOOK.isbn>    <owl:minCardinality>    "0" .
<BOOK.isbn>    <owl:maxCardinality>    "1" .
<BOOK.publication_date>    <rdf:type>    <owl:FunctionalProperty> .
<BOOK.publication_date>    <owl:minCardinality>    "0" .
<BOOK.publication_date>    <owl:maxCardinality>    "1" .
<BOOK.book_title>    <rdf:type>    <owl:FunctionalProperty> .
<BOOK.book_title>    <owl:minCardinality>    "0" .
<BOOK.book_title>    <owl:maxCardinality>    "1" .
<AUTHOR.author_id>    <rdf:type>    <owl:FunctionalProperty> .
<AUTHOR.author_id>    <owl:minCardinality>    "0" .
<AUTHOR.author_id>    <owl:maxCardinality>    "1" .
<AUTHOR.firstname>    <rdf:type>    <owl:FunctionalProperty> .
<AUTHOR.firstname>    <owl:minCardinality>    "0" .
<AUTHOR.firstname>    <owl:maxCardinality>    "1" .
<AUTHOR.lastname>    <rdf:type>    <owl:FunctionalProperty> .
<AUTHOR.lastname>    <owl:minCardinality>    "0" .
<AUTHOR.lastname>    <owl:maxCardinality>    "1" .
<USER.user_name>    <owl:Cardinality>    "1" .
<BOOK.book_title>    <rdf:type>    <owl:InverseFunctionalProperty> .
<USER.user_id>    <owl:Cardinality>    "1" .
<USER.user_id>    <rdf:type>    <owl:InverseFunctionalProperty> .
<BORROW.borrow_id>    <owl:Cardinality>    "1" .
<BORROW.borrow_id>    <rdf:type>    <owl:InverseFunctionalProperty> .
<BOOK.isbn>    <owl:Cardinality>    "1" .
<BOOK.isbn>    <rdf:type>    <owl:InverseFunctionalProperty> .
<AUTHOR.author_id>    <owl:Cardinality>    "1" .
<AUTHOR.author_id>    <rdf:type>    <owl:InverseFunctionalProperty> .
<BORROW.date_returned>    <local:Default>    "null"
```

5. SQL Constraints and their sample DDL

SQL Constraints	SQL DDL

NOT NULL	CREATE TABLE products (
	    product_no integer **NOT NULL**,
	    name text **NOT NULL**,
	    price numeric
	);

UNIQUE	CREATE TABLE products (
	    product_no integer **UNIQUE**,
	    name text,
	    price numeric
	);

PRIMARY KEY	CREATE TABLE products (
	    product_no integer **PRIMARY KEY**,
	    name text,
	    price numeric
	);

FOREIGN KEY	CREATE TABLE orders (
	    order_id integer PRIMARY KEY,
	    product_no integer **REFERENCES** products (product_no),
	    quantity integer
	);

DEFAULT	CREATE TABLE products (
	    product_no integer,
	    name text,
	    price numeric **DEFAULT** 0
	);

요 약

오늘날 웹에는 다양한 종류의 데이터가 존재하고 있으나, 대부분의 데이터는 관계형 데이터베이스 형태로 존재한다. 시맨틱웹은 “의미론적인 웹” 이라는 뜻으로, 분산 환경에서 데이터들을 리소스에 대한 정보와 자원 사이의 관계-의미 정보를 기계가 처리할 수 있는 통일된 포맷으로 통합하여 더 자동화되고 효과적인 데이터 관리 및 공급이 가능한 다음 세대의 웹을 향한 움직임이다. 그러한 시맨틱웹에서 기준으로 지정된 데이터 모델이 RDF 이므로, 시맨틱웹의 성공을 위해서 RDF형태로의 데이터 온톨로지의 구축이 필수적인 요소이다. 이미 W3C에서도 관계형 데이터베이스 형태로 존재하는 많은 데이터들을 RDF형태로 변환하는 RDB2RDF 분야에 대한 기준들을 제시하였다. 본 논문은 자동화된 RDB2RDF 변환 시스템을 연구하였으며, 변환하면서 관계형 데이터베이스의 스키마 정보를 활용하여 도메인에 대한 OWL 온톨로지를 생성하는 연구를 하였다. 그러한 과정에서 하둡 맵리듀스를 활용하여 효율적인 대용량 데이터 분산 병렬 컴퓨팅을 가능케 하였다. 본 논문에는 스키마 정보를 활용하여 온톨로지를 구축하는 맵핑 룰들을 기술하였으며, RDB2RDF 프로세스에 하둡을 활용하는 전처리 및 알고리즘을 제시한다. 결과적으로 본 연구는 관계형 데이터베이스로부터 성공적으로 RDF 데이터 변환과 OWL 온톨로지 구축을 하였고 우수한 성능과 확장성확 보이는 것을 확인하였다.

주요어 : 시맨틱웹, RDB2RDF, 맵리듀스, 하둡, RDF

학 번 : 2013-23137