RESEARCH ON CONSTRUCTION OF RDF WITH HBASE

Hui Hu

College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China

ABSTRACT

Resource Description Framework (RDF) is designed as a standard metadata model for data interchange on the Internet. Because of machine comprehensibility, it has been successfully used in many areas, such as the intelligent processing of numerous data. While the generation of RDF with relational database (RDB) receives much attention, little effort has been put into the automatic construction of RDF with HBase due to its flexible data structure. Since more data is stored in HBase, it is necessary to extract useful information from HBase. In this paper, we are devoted to construction of RDF with HBase. We put forward formal definitions of RDF and HBase and propose our strategy for generating RDF with HBase. We develop a prototype system to create RDF, and test results demonstrate the feasibility of our method.

KEYWORDS

Semantic Web, RDF, HBase, Construction

1. INTRODUCTION

The vision of the Semantic Web is to establish a semantic framework that allows sharing and reusing data across applications, businesses, and communities [1]. As the core layer of the Semantic Web, RDF is the standard for data representation and exchange recommended by the World Wide Web Consortium. RDF is a metadata model with good machine readability because it adds semantics to the data representation when describing data on the web. RDF allows web applications to share, exchange, and integrate data without losing data semantics [2]. Because of the widespread use of RDF, large amounts of RDF are proliferating. For example, Best Buy and the New York Times use RDF for data storage [3]. But the efficient generation of RDF data remains an open problem. While there have been many efforts to convert relational databases, generating RDF from HBase, a NoSQL database, currently needs more attention.

It is critical to process data in a standard way dealing with massive data with heterogeneous data formats. Because RDF provides a standard, unified semantic manner for information processing, it is necessary to study generating RDF with these data. Numerous studies have been done on RDF generation, primarily concentrating on relational databases and XML. However, as an essential player in the era of Big Data, NoSQL databases have yet to receive much research. Consequently, extracting semantic information from NoSQL databases and constructing RDF is necessary. HBase is an integral part of the NoSQL database. According to Google research, many companies are beginning to choose HBase as their data store. It is an excellent idea to generate RDF from HBase for information processing.
This paper concentrates on constructing RDF with HBase. We give formal representations of HBase and RDF and propose an approach to convert HBase data to RDF based on these representations. And then, We developed an RDF construction program with the proposed transformation method and proved the feasibility of our approach through experiments.

The remaining portions of this essay are structured as follows. The relevant work of RDF construction is introduced in Section 2. Section 3 provides some preliminaries of RDF and HBase. Section 4 details the mapping rules of converting HBase to RDF. The designed system and the processing consequence are shown in Section 5. Section 6 presents the conclusion of this thesis and our future work.

2. RELATED WORK

Many methods exist to build RDF by extracting information from data with another format. This section presents various RDF construction approaches with databases, XML, and JSON.

Due to the widespread use of relational databases (RDB) in many fields, numerous attempts to build RDF using relational databases have been made. In [4], the authors proposed mapping rules and created RDF from MySQL tables. After specifying the mapping relationship between the RDB and RDF models, the authors in [5] proposed the mapping rules to generate RDF with RDB. A recent review [6] specified some tools for translating RDB into RDF. The authors assessed the capabilities of a total of 17 tools. The W3C recommends two methods for creating RDF from RDB. One is direct mapping, and the other is indirect mapping using a mapping language. The construction methods of RDF from RDB have primarily followed these two methods. Some efforts also exist to construct RDF with other database models, like MongoDB [7, 8].

The second is constructing RDF with XML. In [9], the authors propose XSPARQL language, which combines XQuery and SPARQL. With XSPARQL, people can query both XML and RDF and then implement data conversions between the two formats. The authors of [10] suggest using keywords or graphical queries to convert XML into RDF. In [11], the authors suggest a declarative method based on Scala programming language for converting XML to RDF. In [12], to convert XML to RDF, the authors created a template language based on XPath, which is helpful for users unfamiliar with XPath and RDF triples. For converting XML Schema to Shape Expressions (ShEx), an RDF validation language, some mappings from XML Schema to ShEx are proposed in [13].

JSON is utilized frequently because of its benefits of quick parsing and high transmission effectiveness. Therefore, the conversion of JSON data to RDF has received some attention. In [14], to map JSON to RDF, the authors identify the metadata of JSON and align it with domain vocabulary terms. With OWL, the authors of [15] propose the mapping from JSON to RDF and develop a mapping tool. The authors of [16], who focus on the output from SPARQL queries, convert JSON to JSON-LD, a compact RDF format. Concentrating on coverage data representing spatiotemporal data, the authors of [17] convert the data with JSON format to RDF. In [18], the authors encode coverage data in many formats, such as RDF and JSON. For representing RDF, they provide a mapping between the JSON and RDF via JSON-LD.

There have been few attempts to construct RDF using the HBase database in addition to the above. This paper formally defines HBase and RDF and concentrates on RDF generation from the HBase. Furthermore, our method differs from suggestions in [19] that only support the transformation from simple HBase to RDF.
3. PRELIMINARIES

In this section, according to the characteristics of RDF and HBase, we offer formal definitions about HBase and RDF. On the basis of the definitions, it is easy to describe the process of mapping from HBase to RDF.

3.1. Data Model of RDF

RDF is a domain-independent universal description language that does not define domain semantics. The RDF data model consists of a group of RDF statements, represented by the triple denoted as (subject, predicate, object). The triple’s first element denotes the resource, the second is its property, and the third is the attribute value of the resource. RDF uses Universal Resource Identifier (URI) to denote resources.

Depending on the predicate type, the object can be a literal value or resource. To overcome the defect that RDF does not define domain semantics, people use the RDF Schema to define domain semantics. The RDF Schema defines a set of modeling primitives with fixed semantics, such as rdfs: domain [20].

Following is the formal definition of the RDF data model drawn on the research of [21].

Definition 1 (RDF data model): An RDF data model is defined as a 5-tuple: \( RW = (V, E, \Sigma, L, A) \) where:

\[
V = \{V_1, V_2, V_3, \ldots, V_n\} \]

is a finite vertex set in the RDF graph, which can be IRI, literals, or blank nodes.

\[
E = \{E_1, E_2, E_3, \ldots, E_n\} \subseteq V \times V, i \neq j \]

is a finite edge set in the RDF graph.

\[
\Sigma = \{C, DP, OP, D, T\}.
\]

\( C = \{C_1, C_2, C_3, \ldots, C_n\} \) is the finite set of RDF class resource tags,

\( DP = \{DP_1, DP_2, DP_3, \ldots, DP_n\} \) is the finite set of RDF datatype property tags,

\( OP = \{OP_1, OP_2, OP_3, \ldots, OP_n\} \) is the finite set of RDF object property tags,

\( D = \{D_1, D_2, D_3, \ldots, D_n\} \) is the finite set of RDF data tags,

\( T = \{T_1, T_2, T_3, \ldots, T_n\} \) is the finite set of RDF instance tags.

\[
L = \{L_v, L_e\}
\]

is used to assign labels for vertices and edges of an RDF graph. \( L_v : V \rightarrow \Sigma \) is used to assign labels for vertices, and \( L_e : E \rightarrow \Sigma \) is used to assign labels for edges.

\( A \) is an axiom set containing class, attribute, and instance axioms, as shown in Table 1.
An RDF vertex $V_i \in V$ has a corresponding label denoted as $L_e(V_i)$ to indicate the subject or object. The vertex label can be IRI, literal or empty nodes. An edge $(V_i, V_j) \in E$ indicates a directed edge between two vertexes, and its label $L_e(V_i, V_j)$ denotes the triple’s predicate.

There are three types of axioms in Table 1, which are class axioms, attribute axioms, and instance axioms. Let the RDF graph vertex $V_i, V_j \in V$, and its labels are $L_e(V_i), L_e(V_j)$. The vertex with class resource label means RDF Class concept. When edge $E_m = (V_i, V_j)$ exists, its label $L_e(V_i, V_j)$ indicates the RDF predicate that can be a datatype property or object property.

### 3.2. HBase Data Model

As an open-source, scalable, distributed database, HBase is a column-oriented database that differs from relational databases. We can consider the HBase table a multidimensional map indexed by row key, timestamp, and column.

Following content of this section introduce the formal representation of the HBase:

**Definition 2 (HBase database model):** HBase database model $HM$ is represented as tuple $(B, RE, HI)$, where:

- $B = TN \cup CF \cup CQ \cup D$ is the set of essential elements of HBase.
- $TN = \{TN_1, TN_2, TN_3, ..., TN_n\}$ is the set of HBase table names.
- $CF = \{CF_1, CF_2, CF_3, ..., CF_m\}$ is the set of HBase column families. $cf(m)$ specifies the column family in table $tn$.
- $CQ = \{CQ_1, CQ_2, CQ_3, ..., CQ_m\}$ is the set of HBase column qualifiers. $cq(m, cf)$ specifies the column qualifier of column family $cf$ in table $tn$.
- $C = \{C_1, C_2, C_3, ..., C_l\}$ is the set of HBase table columns specified by column family and qualifier. $c(m)$ specifies the column of table $tn$.
- $D$ is a finite set of distinct HBase table column data type. $D(c)$ denotes data type of HBase column.
- $RE$ is a finite set of constraint relation defined by user about HBase database model.
- $RE(TN_1, C, TN_2)$ denotes reference relation from table $TN_1$ to table $TN_2$ by column $C$. 

### Table 1. Axiom set for $A$

<table>
<thead>
<tr>
<th>RDF triple</th>
<th>RDF axiom</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(L_e(V_i) \in \Sigma.C, rdf:type, owl:Class)$</td>
<td>$Type(L_e(V_i), Class)$</td>
</tr>
<tr>
<td>$(L_e(V_i) \in \Sigma.C, L_e(V_i \times V_j) \in \Sigma.OP, L_e(V_j) \in \Sigma.C)$</td>
<td>$ObjectProperty(L_e(V_i),V_j), domain(L_e(V_i)), range(L_e(V_j))$</td>
</tr>
<tr>
<td>$(L_e(V_i) \in \Sigma.C, L_e(V_i \times V_j) \in \Sigma.DP, L_e(V_j) \in \Sigma.C)$</td>
<td>$DatatypeProperty(L_e(V_i),V_j), domain(L_e(V_i)), range(L_e(V_j))$</td>
</tr>
<tr>
<td>$(L_e(V_i) \in \Sigma.T, rdf:type, L_e(V_j) \in \Sigma.C)$</td>
<td>$Individual(L_e(V_i), L_e(V_j))$</td>
</tr>
</tbody>
</table>
denotes reference relation from table $TN_1$ to table $TN_2$ by column family $CF$.

$RE(TN_1,CF,TN_2)$ denotes embed relation in table $TN_1$ through column family $CF$. Users embed entity information into column families rather than storing it in tables.

$HI$ is a set of HBase instances. An HBase instance is a 6-tuple containing a table name, row key, column family, column qualifier, cell value, and cell timestamp. For a HBase instance $hi \in HI$, $hi.tn$ suggests its table name, $hi.rk$ means its row key, $hi.cf$ denotes its column family, $hi.cq$ implies its column qualifier, $hi.v$ suggests its cell value and $hi.ts$ means its timestamp.

4. **MAPPING HBASE DATABASE MODEL TO RDF**

Here Function $\varphi$ is used to map the elements of HBase to RDF.

**Rule 1:** $\forall m \in TN \rightarrow L_\varphi (V_i) = \varphi(m) \in \Sigma.C \land (L_\varphi (V_i). rdf:type,owl:Class)$

Rule 1 maps the HBase table to the RDF vertex with the class label. For example, a table named “employee” is mapped to RDF class $<ns:employee>$.

**Rule 2:** $\forall c \in C(t) \rightarrow \varphi(c) = L_\varphi (V_i \times V_j) \in \Sigma.DP \land \varphi(t) = L_\varphi (V_i) \in \Sigma.C$.

An HBase column in the HBase database model is mapped to an RDF edge with a datatype property label. Table columns in Table 2 such as “personal:name” and “office:phone” are mapped to RDF Datatype Property in Table 3.

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Time Stamp</th>
<th>Column Family: personal</th>
<th>Column Family: office</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001</td>
<td>timestamp1</td>
<td>personal:name=&quot;John&quot;</td>
<td>office:phone=&quot;415-212-5544&quot;</td>
</tr>
<tr>
<td>00001</td>
<td>timestamp1</td>
<td>personal:residence_phone=&quot;415-111111&quot;</td>
<td>office:address=&quot;1021 Market St&quot;</td>
</tr>
<tr>
<td>00001</td>
<td>timestamp2</td>
<td>personal:residence_phone=&quot;415-111-1234&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. A sample data with HBase table employee.

Table 3. Mapping Result of Rule2.

<ns:personal:name> a owl:DatatypeProperty .  
<ns:office:phone> a owl:DatatypeProperty .  
<ns:office:address> a owl:DatatypeProperty .

**Rule 3:**

$\forall m_1,m_2 \in TN, c \in c(m_1), RE(m_1,c,m_2) \rightarrow L_\varphi (V_i \times V_j) = \varphi(c) \in \Sigma.OP \land L_\varphi (V_i) = \varphi(m_1) \in \Sigma.C$ 

$\land L_\varphi (V_j) = \varphi(m_2) \in \Sigma.C$.

Rule 3 maps an HBase relation $RE(m_1,c,m_2)$ between table $m_1$ and table $m_2$ to an RDF edge with an object property label. For example, in Table 4, a user follows another user specified by the column value, then the column is mapped to Object Property as shown in Table 5.
Table 4. A sample data with table user.

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Column Family: follows</th>
</tr>
</thead>
<tbody>
<tr>
<td>AK</td>
<td>follows:1=foo follows:2=bar follows:3=baz</td>
</tr>
<tr>
<td>foo</td>
<td>follows:1=bar follows:2=AK</td>
</tr>
</tbody>
</table>

Table 5. Mapping Result of Rule 3.

<ns:user> a owl:Class .
<ns:follows:1> a owl:ObjectProperty .
<ns:follows:1> rdfs:domain <ns:user> .
<ns:follows:1> rdfs:range <ns:user> .
<ns:follows:2> a owl:ObjectProperty .
<ns:follows:2> rdfs:domain <ns:user> .
<ns:follows:2> rdfs:range <ns:user> .
<ns:follows:3> a owl:ObjectProperty .
<ns:follows:3> rdfs:domain <ns:user> .
<ns:follows:3> rdfs:range <ns:user> .

Rule 4:
∀tn1,tn2∈TN, cf ∈ cf (tn1), RE (tn1, cf ,tn2) → \( L_{\Sigma} (V_i × V_j) = \varphi (cf) ∈ \Sigma.OP \land L_{\varphi} (V_j) = \varphi (tn1) ∈ \Sigma.C \land L_{\varphi} (V_j) = \varphi (tn2) ∈ \Sigma.C. \)

An HBase relation \( RE (tn1, cf ,tn2) \) between table \( tn1 \) and table \( tn2 \) is mapped to the RDF edge with an object property label. For example, in Table 6, one user points to another user by column family qualifier, then the column family “follows” is mapped to the object property as shown in Table 7.

Table 6. A sample data with table user.

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Column Family: follows</th>
</tr>
</thead>
<tbody>
<tr>
<td>AK</td>
<td>follows:foo=1 follows:bar=1 follows:baz=1</td>
</tr>
<tr>
<td>foo</td>
<td>follows:bar=1 follows:AK=1</td>
</tr>
</tbody>
</table>

Table 7. Mapping Result of Rule 4.

<ns:user> a owl:Class .
<ns:follows> a owl:ObjectProperty .
<ns:follows> rdfs:domain <ns:user> .
<ns:follows> rdfs:range <ns:user> .

Rule 5:
∀tn1∈TN, cf ∈ cf (tn1), RE (tn1, cf ) → \( L_{\varphi} (V_i) = \varphi (tn1) ∈ \Sigma.C \land L_{\varphi} (V_j) = \varphi (cf) ∈ \Sigma.C \land L_{\varphi} (V_j) = \varphi (ref - cf) ∈ \Sigma.OP. \)

An HBase relation \( RE (tn1, cf) \) is mapped to the RDF edge with an object property label. This relation means the entity is embedded in the table column family. For example, in Table 8, entity “department” is embedded in the column family “department”, and then the column family is mapped to the RDF class as shown in Table 9.
Table 8. There are sample data of table student.

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Column Family: student</th>
<th>Column Family: department</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>student:name=&quot;bob&quot;</td>
<td>department:dno=5001</td>
</tr>
<tr>
<td>001</td>
<td>student:sex=&quot;M&quot;</td>
<td>department:name=&quot;Computer&quot;</td>
</tr>
<tr>
<td>001</td>
<td></td>
<td>department:header=&quot;Alice&quot;</td>
</tr>
</tbody>
</table>

Table 9. Mapping Result of Rule 5.

*...*
5. SYSTEM IMPLEMENTATION AND EXPERIMENTAL RESULTS

5.1. System Architecture

To validate our construction method of RDF with HBase, we developed a prototype system to extract the HBase data and map these data to RDF. We develop this system with OpenJDK 17.0.1 and JavaFX on a PC (Intel i5-5200U (4) @ 2.700GHz, RAM 8 GB, and ArchLinux system).

The system obtains data by connecting the database and then outputs RDF data. The database parsing, RDF construction, and display modules comprise most of the system, as depicted in Figure 1. Database parsing module analyses the semantic relationships and data information of data based on the HBase database formal definition. The RDF construction module constructs RDF Schema with obtained semantic relationships from the parsing module. And then, this module constructs RDF data with RDF Schema and data information by mapping rules introduced in Section 4. Finally, the display module exhibits the RDF data created by the RDF construction module.

Figure 2 displays the system’s screen snapshot. The GUI of the system contains three display areas. The TextArea on the left exhibits information about the HBase data model, including tables and columns. Moreover, the TextArea (in the upper right) below the label “RDF Schema” shows the corresponding RDF Concept, such as Classes and Properties, in the form of the RDF Turtle. The TextArea (in the lower right) below the label “RDF Individual” exhibits the corresponding RDF individuals.
5.2. Experimental Results and Discussions

We conducted experiments with the HBase data source from the article “Introduction to Hbase Schema Design” created by a Cloudera engineer. Since the paper [19] does not provide the source code of the system for building RDF from HBase data, this section simulates the implementation of the corresponding construction system according to the method given in the paper.
The results of the RDF construction experiments based on HBase data source are shown in Table 11. The first column of the table represents the test metrics of the RDF construction experiment, including the time consumed for construction, the number of RDF classes generated, the number of datatype properties, the number of object properties, the number of domain axioms, the number of range axioms, and the number of RDF triples generated. The second to third columns are the experimental results of the method in this paper (denoted as H2R) and the method proposed in the paper [19] (denoted as OBDI).

Table 11. Experimental results.

<table>
<thead>
<tr>
<th>Test metrics</th>
<th>H2R</th>
<th>OBDI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Construction time (ms)</td>
<td>8934</td>
<td>10142</td>
</tr>
<tr>
<td>RDF classes</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Datatype properties</td>
<td>9</td>
<td>16</td>
</tr>
<tr>
<td>Object properties</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Domain axioms</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>Range axioms</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>RDF triples</td>
<td>680047</td>
<td>190020</td>
</tr>
</tbody>
</table>

Table 11 shows that our method can retain more database information for RDF creation compared to another method. And also, our method has advantages in terms of construction time. Our method constructs more RDF triples per unit time when compared with the other method. While paper [19] only considers the mapping of basic data attributes in HBase database, our approach not only considers the mapping of basic data attributes in HBase, but also pays attention to implicit reference relationships in HBase database, such as column-value based references, column-qualifier based references, and embedded references. Implicit reference relationships in the data are mapped to RDF by parsing user-defined implicit reference relationship documents. Meanwhile, our approach only traverses the HBase data source once to resolve the semantic relationships of HBase data, while the paper [10] traverses the data source twice to resolve the schema information and data information of the HBase data source. Therefore, our method not only retains more database information for constructing RDF compared with another method, but also has advantages in construction efficiency.

6. CONCLUSIONS AND FUTURE WORK

Because of the growth of the Web, more and more people select NoSQL databases to solve the data management problems that come from big data. HBase occupies a part of the market share in NoSQL databases. Hence, the study of transforming HBase data to RDF is conducive to data use and solving the problem of insufficient available RDF data. In this paper, we design a method to convert HBase semantic relationships and data to RDF and develop a system to confirm its feasibility.

In the future, optimizing the implementation system to improve efficiency is the first task. We will also pay attention to expanding the method for more column-oriented databases.

REFERENCES


AUTHORS

Hui Hu is currently a master’s candidate in the College of Computer Science and Technology at the Nanjing University of Aeronautics and Astronautics, China. His research interests include RDF and Semantic Web.

© 2023 By AIRCC Publishing Corporation. This article is published under the Creative Commons Attribution (CC BY) license.